Package ‘RGeode’

September 4, 2017

Version 0.1.0
Date 2017-08-24
Author Lorenzo Rimella
Maintainer Lorenzo Rimella <lorenzo.rimella@hotmail.it>
Title Geometric Density Estimation
Description Provides the hybrid Bayesian method Geometric Density Estimation. On the one hand, it scales the dimension of our data, on the other it performs inference. The method is fully described in the paper "Scalable Geometric Density Estimation" by Y. Wang, A. Canale, D. Dunson (2016) <http://proceedings.mlr.press/v51/wang16e.pdf>.
License GPL (>= 2)
Depends R (>= 3.0.0), Rcpp (>= 0.12.11), MASS (>= 7.3-47), stats (>= 3.4.1)
LinkingTo Rcpp
RoxygenNote 6.0.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2017-09-04 13:51:49 UTC

R topics documented:

p ................................................................. 2
randSVD .................................................. 3
rexptr ...................................................... 5
rgammadtr ............................................... 6
rgeode ..................................................... 8

Index 13
Threshold probability (p(t))

Description
The decreasing function for the adaptive pruning.

Usage
p(t, c0, c1)

Arguments

\[ t \quad \text{int} \]
The current iteration at which the probability of an adaptation is calculated.

\[ c0 \quad \text{double} \]
Additive constant at the exponent-

\[ c1 \quad \text{double} \]
Multiplicative constant at the exponent.

Value
p returns the threshold of interest:

\[ p(t) = \exp(c_0) + c_1 \cdot t. \]

Author(s)
L. Rimella, <lorenzo.rimella@hotmail.it>

References
   The implementation of rgammatr is inspired to the Matlab implementation of rexptrunc by Ye Wang.

Examples
\[ t = 10 \]
\[ c0 = -1 \]
\[ c1 = 10 \]
\[ p(t, c0, c1) \]
Randomized SVD

Description

Compute the near-optimal low-rank singular value decomposition (SVD) of a rectangular matrix. The algorithm follows a randomized approach.

Usage

```
randsvd(A, k = NULL, l = NULL, its = 2, sdist = "unif")
```

Arguments

- `A` array_like
  a real/complex input matrix (or data frame), with dimensions \((m,n)\). It is the real/complex matrix being approximated.
- `k` int, optional
  determines the target rank of the low-rank decomposition and should satisfy \(k << \min(m,n)\). Set by default to 6.
- `l` int, optional
  block size of the block Lanczos iterations; \(l\) must be a positive integer greater than \(k\), and defaults \(l = k + 2\).
- `its` int, optional
  number of full iterations of a block Lanczos method to conduct; \(its\) must be a nonnegative integer, and defaults to 2.
- `sdist` str, optional
  Specifies the sampling distribution. 'unif': (default) Uniform '['-1,1']'. 'normal': Normal '~N(0,1)'.

Details

Randomized SVD (randSVD) is a fast algorithm to compute the approximate low-rank SVD of a rectangular \((m,n)\) matrix \(A\) using a probabilistic algorithm. Given the decided rank \(k << n\), rSVD factors the input matrix \(A\) as \(A = U \ast \text{diag}(S) \ast V'\), which is the typical SVD form. Precisely, the columns of \(U\) are orthonormal, as are the columns of \(V\), the entries of \(S\) are all nonnegative, and the only nonzero entries of \(S\) appear in non-increasing order on its diagonal. The dimensions are: \(U\) is \((m,k)\), \(V\) is \((n,k)\), and \(S\) is \((k,k)\), when \(A\) is \((m,n)\).

Increasing \(its\) or \(l\) improves the accuracy of the approximation USV’ to \(A\).

The parameter \(its\) specifies the number of normalized power iterations (subspace iterations) to reduce the approximation error. This is recommended if the singular values decay slowly. In practice 1 or 2 iterations achieve good results, however, computing power iterations increases the computational time. The number of power iterations is set to \(its = 2\) by default.
Value

randSVD returns a list containing the following three components:

- **d**
  - array_like
  - \((k, k)\) matrix in the rank-\(k\) approximation \(USV'\) to \(A\), where \(A\) is \((m, n)\); the entries of \(S\) are all nonnegative, and its only nonzero entries appear in nonincreasing order on the diagonal.

- **u**
  - matrix
  - \((m, k)\) matrix in the rank-\(k\) approximation \(A = U \ast \text{diag}(S) \ast V'\) to \(A\); the columns of \(U\) are orthonormal and are called Left singular vect.

- **v**
  - matrix
  - \((n, k)\) matrix in the rank-\(k\) approximation \(A = U \ast \text{diag}(S) \ast V'\) to \(A\); the columns of \(V\) are orthonormal and are called Right singular vect.

Note

The singular vectors are not unique and only defined up to sign (a constant of modulus one in the complex case). If a left singular vector has its sign changed, changing the sign of the corresponding right vector gives an equivalent decomposition.

Author(s)

L. Rimella, <lorenzo.rimella@hotmail.it>

References

1. N. Halko, P. Martinsson, and J. Tropp.
   "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions" (2009).

2. S. Voronin and P. Martinsson.
   "RSVDPACK: Subroutines for computing partial singular value decompositions via randomized sampling on single core, multi core, and GPU architectures" (2015).

3. N. Benjamin Erichson.
   "Randomized Singular Value Decomposition (rsvd): R package" (2016).
   (available in the CRAN).

4. Nathan Halko, Per-Gunnar Martinsson, and Joel Tropp.
   "Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions" (2009).
   (available at [http://arxiv.org](http://arxiv.org)).

   "A randomized algorithm for principal component analysis" (2009).

The implementation of rand SVD is inspired by the MatLab implementation of RandPCA by M. Tygert.
Examples

```r
# Simulate a general matrix with 1000 rows and 1000 columns
vy = rnorm(1000*1000,0,1)
y = matrix(vy,1000,1000,byrow=TRUE)

# Compute the randsvd for the first hundred components of the matrix y and measure the time
start.time <- Sys.time()
prova1 = randsvd(y,k=100)
Sys.time() - start.time

# Compare with a classical SVD
start.time <- Sys.time()
prova2 = svd(y)
Sys.time() - start.time
```

rexptr

*Random generator for a Truncated Exponential distribution.*

**Description**

Simulate random number from a truncated Exponential distribution.

**Usage**

```r
rexptr(n = 1, lambda = 1, range = NULL)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>int, optional number of simulations.</td>
</tr>
<tr>
<td>lambda</td>
<td>double, optional parameter of the distribution.</td>
</tr>
<tr>
<td>range</td>
<td>array_like, optional domain of the distribution, where we truncate our Exponential. <code>range[0]</code> is the min of the range and <code>range[1]</code> is the max of the range.</td>
</tr>
</tbody>
</table>

**Details**

It provides a way to simulate from a truncated Exponential distribution with given parameter `λ` and the range `range`. This will be used during the posterior sampling in the Gibbs sampler.

**Value**

`rexptr` returns the simulated value of the distribution:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>double it is the simulated value of the truncated Exponential distribution. It will be a value in <code>(range[0], range[1])</code>.</td>
</tr>
</tbody>
</table>
Author(s)
L. Rimella, <lorenzo.rimella@hotmail.it>

References
The implementation of rgammatr is inspired to the Matlab implementation of rexptrunc by Ye Wang.

Examples
```r
#Simulate a truncated Exponential with parameters 0.5 in the range
#5,Inf.
#Set the range:
range<- c(1,Inf)

#Simulate the truncated Gamma
set.seed(123)
vars1<-rexptr(1000,0.5,range)

#Look at the histogram
hist(vars1,freq=FALSE,ylim=c(0,2),xlim = c(0,5))
lines(density(vars1))

#Compare with a non truncated Exponential
set.seed(123)
vars2<-rexp(1000,0.5)

#Compare the two results
lines(density(vars2),col='red')

#Observation: simulate without range is equivalent to simulate from
#rexp(1000,0.5)
```

---

**rgammatr**

*Random generator for a Truncated Gamma distribution.*

Description

Simulate random number from a truncated Gamma distribution.

Usage

```r
rgammatr(n = 1, A = 0, B = 1, range = NULL)
```
**Arguments**

- `n`: int, optional
  number of simulations.
- `A`: double, optional
  shape parameter of the distribution.
- `B`: double, optional
  rate parameter of the distribution.
- `range`: array_like, optional
  domain of the distribution, where we truncate our Gamma. `range(0)` is the min of the range and `range(1)` is the max of the range.

**Details**

It provide a way to simulate from a truncated Gamma distribution with given parameters `A, B` and `range`. This will be used during the posterior sampling in the Gibbs sampler.

**Value**

`rgammatr` returns the simulated value of the distribution:

- `u`: double
  it is the simulated value of the truncated Gamma distribution. It will be a value in `(range(0), range(1))`.

**Author(s)**

L. Rimella, <lorenzo.rimella@hotmail.it>

**References**

   The implementation of `rgammatr` is inspired to the Matlab implementation of `gamrndtruncated` by Ye Wang.

**Examples**

```r
# Simulate a truncated Gamma with parameters 1,2 in the range 1,5.
# Set the range:
range<- c(1,5)

# Simulate the truncated Gamma
set.seed(123)
vars1<-rgammatr(1000,1,2,range)

# Look at the histogram
hist(vars1,freq=FALSE,ylim=c(0,2),xlim = c(0,5))
lines(density(vars1))
```
#Compare with a non truncated Gamma
set.seed(123)
vars2<-rgamma(1000,1,2)

#Compare the two results
lines(density(vars2),col='red')

#Observation: simulate without range is equivalent to simulate from
#rgamma(1000,1,2)

---

**rgeode**

*GEOmetric Density Estimation.*

**Description**

It selects the principal directions of the data and performs inference. Moreover GEODE is also able to handle missing data.

**Usage**

```r
gelog(Y, d = 6, burn = 1000, its = 2000, tol = 0.01, atau = 1/20,
       asigma = 1/2, bsigma = 1/2, starttime = NULL, stoptime = NULL,
       fast = TRUE, c0 = -1, c1 = -0.005)
```

**Arguments**

- `Y`: array_like
  a real input matrix (or data frame), with dimensions \((n, D)\). It is the real matrix of data.
- `d`: int, optional
  it is the conservative upper bound for the dimension \(D\). We are confident that the real dimension is smaller than it.
- `burn`: int, optional
  number of burn-in to perform in our Gibbs sampler. It represents also the stopping time that stop the choice of the principal axes.
- `its`: int, optional
  number of iterations that must be performed after the burn-in.
- `tol`: double, optional
  threshold for adaptively removing redundant dimensions. It is used compared with the ratio: \(\frac{\alpha^2(t)}{\max \alpha^2(t)}\).
- `atau`: double, optional
  The parameter \(\alpha_\tau\) of the truncated Exponential (the prior for \(\tau_j\)).
- `asigma`: double, optional
  The shape parameter \(\alpha_\sigma\) of the truncated Gamma (the prior for \(\sigma^2\)).
rgeode
double, optional
The rate parameter $b_\sigma$ of the truncated Gamma (the prior for $\sigma^2$).

starttime int, optional
starting time for adaptive pruning. It must be less then the number of burn-in.

stoptime int, optional
stop time for adaptive pruning. It must be less then the number of burn-in.

fast bool, optional
If TRUE it is run using fast d-rank SVD. Otherwise it uses the classical SVD.

c0 double, optional
Additive constant for the exponent of the pruning step.

c1 double, optional
Multiplicative constant for the exponent of the pruning step.

Details

GEOmetric Density Estimation (rgeode) is a fast algorithm performing inference on normally distributed data. It is essentially divided in two principal steps:

- Selection of the principal axes of the data.
- Adaptive Gibbs sampler with the creation of a set of samples from the full conditional posteriors of the parameters of interest, which enable us to perform inference.

It takes in inputs several quantities. A rectangular $(N, D)$ matrix $Y$, on which we will run a Fast rank $d$ SVD. The conservative upper bound of the true dimension of our data $d$. A set of tuning parameters. We remark that the choice of the conservative upper bound $d$ must be such that $d > p$, with $p$ real dimension, and $d << D$.

Value

rgeode returns a list containing the following components:

- **Ind** array_like
  The chose principal axes.

- **u** matrix
  Containing the sample from the full conditional posterior of $u_j$s. We store each iteration on the columns.

- **tau** matrix
  Containing the sample from the full conditional posterior of $tau_j$s.

- **sigmaS** array_like
  Containing the sample from the full conditional posterior of $sigma$.

- **W** matrix
  Containing the principal singular vectors.

- **Miss** list
  Containing all the informations about missing data. If there are not missing data this output is not provide.
  - **id_m** array
    It contains the set of rows with missing data.
- **pos_m** list
  It contains the set of missing data positions for each row with missing values.

- **yms** list
  The list contained the pseudo-observation substituting our missing data. Each element of the list represents the simulated data for that time.

**Note**

The part related to the missing data is filled only in the case in which we have missing data.

**Author(s)**

L. Rimella, <lorenzo.rimella@hotmail.it>

**References**


**Examples**

```r
library(MASS)
library(RGeode)

# WITHOUT MISSING DATA
# Define the dataset
D= 200
n= 500
d= 10
d_true= 3

set.seed(321)

mu_true= runif(d_true, -3, 10)
Sigma_true= matrix(0,d_true,d_true)
diag(Sigma_true)= c(runif(d_true, 10, 100))

W_true = svd(matrix(rnorm(D*d_true, 0, 1), d_true, D))$v
sigma_true = abs(runif(1,0,1))

mu= W_true%*%mu_true
C= W_true %*% Sigma_true %*% t(W_true)+ sigma_true* diag(D)

y= mvrnorm(n, mu, C)
```
# GEODE: Without missing data

```
start.time <- Sys.time()
GEODE <- rgeode(Y = y, d)
Sys.time() - start.time
```

# SIGMAS

```
#plot(seq(110,3000,by=1),GEODE$sigmaS[110:3000],ty='l',col=2, 
# xlab='Iteration', ylab='sigma^2', main='Simulation of sigma^2')
#abline(v=800,lwd= 2, col= 'blue')
#legend('bottomright',c('Posterior of sigma^2', 'Stopping time'), 
# lwd=c(1,2),col=c(2,4),cex=0.55, border='black', box.lwd=3)
```

# WITH MISSING DATA

```
#Insert NaN
n_m = 5 #number of data vectors containing missing features 
d_m = 1  #number of missing features

data_miss= sample(seq(1,n),n_m)
features= sample(seq(1,D), d_m)
for(i in 2:n_m)
{
   features= rbind(features, sample(seq(1,D), d_m))
}
for(i in 1:length(data_miss))
{
   if(i==length(data_miss))
   {
      y[data_miss[i],features[i,][-1]]= NaN 
   } else
   {
      y[data_miss[i],features[i,]]= NaN 
   }
}
```

# GEODE: With missing data

```
set.seed(321))
start.time <- Sys.time()
GEODE <- rgeode(Y = y, d)
Sys.time() - start.time
```
```r
# SIGMAS
# plot(seq(110,3000,by=1),GEODE$sigmaS[110:3000],ty='l',col=2,
#     xlab='Iteration', ylab='sigma^2', main='Simulation of sigma^2')
# abline(v=800,lwd=2, col='blue')
# legend('bottomright', c('Posterior of sigma^2', 'Stopping time'),
#         lwd=c(1,2),col=c(2,4),cex=0.55, border='black', box.lwd=3)
```

Index

p, 2

randSVD, 3
rexptr, 5
rgammatr, 6
rgeode, 8