Package 'tensorsparse'

August 5, 2019

Type Package

Title Multiway Clustering via Tensor Block Models

Version 1.0

Date 2019-08-02

Author Miaoyan Wang, Yuchen Zeng

Maintainer Yuchen Zeng <yzeng58@wisc.edu>

Depends glasso

Imports fields, glmnet, rgl, reshape, mvtnorm, HDCI, clues, parallel, RColorBrewer, viridis, rTensor, methods

Description Implements the multiway sparse clustering approach of Zeng and Wang (2019) <arXiv:1906.03807>.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2019-08-05 13:40:02 UTC

RoxygenNote 6.1.1

Encoding UTF-8

R topics documented:

`Calculate` *Calculate the correct rate of selecting d_1, d_2, d_3*

Description

Calculate the correct rate of the result of sparse.choosekrl().

Usage

```
Calculate(true, results)
```

Arguments

`true`	Vector. The true number of clusters in each mode (eg. c(3,3,3)).
`results`	List. A list consists of 3*1 matrix and each matrix is the estimated c(d_1,d_2,d_3). (The return value of sparse.choosekrl().)

Value

A numeric value. The correct rate of selecting d_1, d_2, d_3

`Calculatekrl` *Summarize the result of estimating the true c(d_1,d_2,d_3) in a simulation.*

Description

Summarize the return objct of sparse.choosekrl().

Usage

```
Calculatekrl(results)
```

Arguments

`results`	List. A list consists of 3-dimensional vectors and each vector is estimated c(d_1,d_2,d_3). (The result returned by sparse.choosekrl().)

Value

`meank` mean estimated d_1 `meanr` mean estimated d_2 `meanl` mean estimated d_3 `sdk` the standard deviation of estimated d_1 `sdr` the standard deviation of estimated d_2 `sdl` the standard deviation of estimated d_3

`cer` *Calculate the clustering error rate (CER)*

Description

Calculate clustering error rate (CER) by comparing the clustering result and the true underlying mean signal tensor.

Usage

```
cer(bires, data)
```

Arguments

`bires`	List. The return value of label2() or classify2() which consists of judgeX, Cs, Ds, Es, objs, mus.
`data`	List. The return value of get.data().

Value

A list which consists of the CER of all three modes.

`choosekrl_bic` *Perform tuning parameter (d_1, d_2, d_3) selection for sparse tensor clustering via BIC criterion*

Description

Select the best d_1, d_2, d_3 to perform clustering. A range of values of d_1, d_2, d_3 is usually considered - value that results in the lowest BIC is selected.

Usage

```
choosekrl_bic(x, k, r, l, lambda = 0, sim.times = 1, method = "L0",
  n.cores = NULL)
```

Arguments

x	a three-dimensional array
k	the range of d_1: a vector, the possible clusters numbers of mode 1
r	the range of d_2: a vector, the possible clusters numbers of mode 2
l	the range of d_3: a vector, the possible clusters numbers of mode 3
lambda	a numeric value. The coefficient of the regularization term.
sim.times	the simulation times when perform clustering of classify2() in label2().
method	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty, "L1" indicating Lasso penalty.
n.cores	the number of cores

Value

a list

estimated_krl a 1*3 matrix which is the estimated c(d_1,d_2,d_3).

BIC a vector which contains the BIC values of all the combination of given range.

---

chooseLambda *Perform tuning parameter (lambda) selection for sparse tensor clustering via BIC criterion*

Description

We assume that d_1, d_2, d_3 are known. A range of values of lambda is usually considered - value that results in the lowest BIC is selected.

Usage

```
chooseLambda(x, k, r, l, lambda = NULL, method = "L0")
```

Arguments

x	a three-dimensional array
k	d_1: the clusters number of mode 1
r	d_2: the clusters number of mode 2
l	d_3: the clusters number of mode 3
lambda	a vector of possible lambda, eg: lambda = c(0,50,100,200,300,400,500,600,700,800,900,1000,1100,1200)
method	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty and "L1" indicating Lasso penalty.

Value

a list lambda the lambda with lowest BIC;

BIC the corresponding BIC for each lambda in the given range;

nonzeromus the number clusters with non-zero mean.

`classify2` *Perform tensor clustering via TBM method*

Description

This function performs sparse clustering on a three-dimensional tensor via TBM method.

Usage

```
classify2(x, k, r, l, lambda = 0, max.iter = 1000, threshold = 1e-15,
  trace = FALSE, Cs.init = NULL, Ds.init = NULL, Es.init = NULL,
  nstart = 20, method = "L0", center = FALSE)
```

Arguments

`x`	a three-dimensional array
`k`	d_1: the clusters number of mode 1
`r`	d_2: the clusters number of mode 2
`l`	d_3: the clusters number of mode 3
`lambda`	a positive numeric value. The coefficient of the regularized term.
`max.iter`	a positive integer. The Maximum times of iteration.
`threshold`	a positive small numeric value which determines whether the algorithm converges or not.
`trace`	logic value. If true, it would print the iteration situation.
`Cs.init`	vector or NULL. Initial clsuter result of mode 1.
`Ds.init`	vector or NULL. Initial clsuter result of mode 2.
`Es.init`	vector or NULL. Initial clsuter result of mode 3.
`nstart`	positive interger. The same as the "nstart" in kmeans().
`method`	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty and "L1" indicating Lasso penalty.
`center`	logic value that indicates whether run "x = x-mean(x)" before performing clustering.

Value

a list `judgeX` estimated underlying mean signal.

`Cs` clustering result of mode 1.

`Ds` clustering result of mode 2.

`Es` clustering result of mode 3.

`mus` estimated underlying mean signal of each cluster.

Examples

```
x = get.data(20,20,20,2,2,2)
classify2(x$x,2,2,2)
```

`classify4` *Perform tensor clustering via TBM method*

Description

This function performs sparse clustering on a three-dimensional tensor via TBM method.

Usage

```
classify4(x, k, r, l, m, lambda = 0, max.iter = 1000,
  threshold = 1e-15, trace = FALSE, Cs.init = NULL, Ds.init = NULL,
  Es.init = NULL, Fs.init = NULL, nstart = 20, method = "L0",
  center = FALSE)
```

Arguments

`x`	a four-dimensional array
`k`	d_1: the clusters number of mode 1
`r`	d_2: the clusters number of mode 2
`l`	d_3: the clusters number of mode 3
`m`	d_4: the clusters number of mode 4
`lambda`	a positive numeric value. The coefficient of the regularized term.
`max.iter`	a positive integer. The Maximum times of iteration.
`threshold`	a positive small numeric value which determines whether the algorithm converges or not.
`trace`	logic value. If true, it would print the iteration situation.
`Cs.init`	vector or NULL. Initial clsuter result of mode 1.
`Ds.init`	vector or NULL. Initial clsuter result of mode 2.
`Es.init`	vector or NULL. Initial clsuter result of mode 3.
`Fs.init`	vector or NULL. Initial clsuter result of mode 4.
`nstart`	positive interger. The same as the "nstart" in kmeans().
`method`	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty and "L1" indicating Lasso penalty.
`center`	logic value that indicates whether run "x = x-mean(x)" before performing clustering.

Value

a list `judgeX` estimated underlying mean signal.

`Cs` clustering result of mode 1.

`Ds` clustering result of mode 2.

`Es` clustering result of mode 3.

`Fs` clustering result of mode 4.

`mus` estimated underlying mean signal of each cluster.

Examples

```
x = get.data4(10,10,10,10,2,2,2,2)
classify4(x$x,2,2,2,2)
```

---

cp_kmeans *Perform tensor clustering via cp decomposition*

Description

Perform tensor clustering via cp decomposition.

Usage

```
cp_kmeans(x, k, r, l, multiplicative = NULL, max.s = NULL)
```

Arguments

x	a three-dimensional array; data tensor
k	d_1: the clusters number of mode 1
r	d_2: the clusters number of mode 2
l	d_3: the clusters number of mode 3
multiplicative	the number of components
max.s	the max value when selecting s

Value

a list with only one element: judgeX

Examples

```
x = get.data(20,20,20,2,2,2)$x
cp_kmeans(x,2,2,2,NULL,2)
```

get.data *Generate a random tensor*

Description

Generate a random tensor.

Usage

```
get.data(n, p, q, k = NULL, r = NULL, l = NULL, error = 3,
  sort = TRUE, sparse.percent = 0, multiplicative = 0,
  center = FALSE, seed = NULL, mumin = -3, mumax = 3)
```

Arguments

n	the dimension of mode 1
p	the dimension of mode 2
q	the dimension of mode 3
k	d_1: the clusters number of mode 1
r	d_2: the clusters number of mode 2
l	d_3: the clusters number of mode 3
error	positive numeric value: the noise in rnorm()
sort	if TRUE, the data belongs to the same clusters would assemble together after sorting
sparse.percent	the proportion of 0 mean in normal case; the proportion of 0 of each vector in multiplicative case
multiplicative	if multiplicative !=0, then it would produce overlapping multiplicative data, the number refers to the number of components.
center	logic value that indicates whether run "x = x-mean(x)" before performing clustering.
seed	default is NULL, otherwise would set seed to corresponding point
mumin	numeric value. The lower bound of mu when sampling mu
mumax	numeric value. The lower bound of mu when sampling mu

Value

a list x the tensor

truthX the tensor before adding the noise

truthCs true distribution in mode 1

truthDs true distribution in mode 2

truthEs true distribution in mode 3

mus the mean signal of all clusters

binaryX the 0-1 tensor (0:the mean signal = 0; 1:the mean signal != 0)

Examples

```
get.data(20,20,20,2,2,2)$x
```

`get.data4` *Generate a random 4th-order tensor*

Description

Generate a random 4th-order tensor.

Usage

```
get.data4(n, p, q, s, k = NULL, r = NULL, l = NULL, m = NULL,
  error = 3, sort = TRUE, sparse.percent = 0, center = FALSE,
  seed = NULL, mumin = -3, mumax = 3)
```

Arguments

`n`	the dimension of mode 1
`p`	the dimension of mode 2
`q`	the dimension of mode 3
`s`	the dimension of mode 4
`k`	d_1: the clusters number of mode 1
`r`	d_2: the clusters number of mode 2
`l`	d_3: the clusters number of mode 3
`m`	d_4: the clusters number of mode 4
`error`	positive numeric value: the noise in rnorm()
`sort`	if TRUE, the data belongs to the same clusters would assemble together after sorting
`sparse.percent`	the proportion of 0 mean in normal case; the proportion of 0 of each vector in multiplicative case
`center`	logic value that indicates whether run "x = x-mean(x)" before performing clustering.
`seed`	default is NULL, otherwise would set seed to corresponding point
`mumin`	numeric value. The lower bound of mu when sampling mu
`mumax`	numeric value. The lower bound of mu when sampling mu

Value

a list x the tensor

truthX the tensor before adding the noise

truthCs true distribution in mode 1

truthDs true distribution in mode 2

truthEs true distribution in mode 3

mus the mean signal of all clusters

binaryX the 0-1 tensor (0:the mean signal = 0; 1:the mean signal != 0)

---

label2 *Perform tensor clustering via TBM method*

Description

Perform tensor clustering with TBM method in a more stable way. Repeat the classify2() many times and select the one with lowest MSE.

Usage

```
label2(x, k, r, l, lambda = 0, max.iter = 1000, threshold = 1e-10,
  sim.times = 1, trace = FALSE, Cs.init = NULL, Ds.init = NULL,
  Es.init = NULL, method = "L0")
```

Arguments

x	a three-dimensional array
k	d_1: the clusters number of mode 1
r	d_2: the clusters number of mode 2
l	d_3: the clusters number of mode 3
lambda	a positive numeric value. The coefficient of the regularized term.
max.iter	a positive integer. The Maximum times of iteration.
threshold	a positive small numeric value which determines whether the algorithm converges or not.
sim.times	the times of calling classify2() with different seeds.
trace	logic value. If true, it would print the iteration situation.
Cs.init	vector or NULL. Initial clsuter result of mode 1.
Ds.init	vector or NULL. Initial clsuter result of mode 2.
Es.init	vector or NULL. Initial clsuter result of mode 3.
method	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty and "L1" indicating Lasso penalty.

Value

a list judgeX estimated underlying mean signal.

Cs clustering result of mode 1.

Ds clustering result of mode 2.

Es clustering result of mode 3.

mus estimated underlying mean signal of each cluster.

Examples

```
x = get.data(20,20,20,2,2,2)$x
label2(x,2,2,2)
```

---

plot_tensor *Draw 3d plot of tensor*

Description

Draw 3d plot of a given tensor.

Usage

```
plot_tensor(tensor)
```

Arguments

tensor	a three-dimensional array

Value

None

Examples

```
x = get.data(30,30,30,3,3,3)$x
plot_tensor(x)
```

sim.choosekrl *Perform simulation: selecting the best d_1, d_2 and d_3*

Description

Simulation: selecting the best d_1, d_2 and d_3.

Usage

```
sim.choosekrl(n, p, q, k, r, l, error = 1, sim.times = 5,
  method = "L0", mode = "bic", seed = TRUE)
```

Arguments

n	the dimension of mode 1
p	the dimension of mode 2
q	the dimension of mode 3
k	d_1: the clusters number of mode 1
r	d_2: the clusters number of mode 2
l	d_3: the clusters number of mode 3
error	positive numeric value. The noise when producing the data
sim.times	positive integer. Simulation times
method	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty and "L1" indicating Lasso penalty.
mode	two options: "bic": selecting cluster numbers by bic; "crossvalidation":selecting cluster numbers by cross validation.
seed	logic value. Whether set seed to each simulation.

Value

A list consists of the estimating result in different iteration.

sim.chooseLambda *Perform simulation: selecting the lambda*

Description

Simulation: selecting the lambda. Select the lambda with lowest BIC while given a certain range.

Usage

```
sim.chooseLambda(n, p, q, k, r, l, sparse, iteration, lambda,
  standarddeviation = 4, center = FALSE, method = "L0")
```

Arguments

n	the dimension of mode 1
p	the dimension of mode 2
q	the dimension of mode 3
k	d_1: the clusters number of mode 1
r	d_2: the clusters number of mode 2
l	d_3: the clusters number of mode 3
sparse	the sparse percent of data
iteration	iteration times
lambda	a vector of possible lambda, for example: lambda = c(0,50,100,200,300,400,500,600,700,800,900,1000,1
standarddeviation	the standard deviation when producing data
center	if True, then x = x- mean(x)
method	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty and "L1" indicating Lasso penalty.

Value

selectedlambda: a vector of selecting lambdas with lowest BIC in each iteration.

---

simulation *Perform simulation: calculating the mean CER over iterations*

Description

Perform simulation: calculating the mean CER over iterations.

Usage

```
simulation(n, p, q, k, r, l, error, lambda, iteration = 1,
  method = "L0")
```

Arguments

n	the dimension of mode 1
p	the dimension of mode 2
q	the dimension of mode 3
k	d_1: the clusters number of mode 1
r	d_2: the clusters number of mode 2
l	d_3: the clusters number of mode 3
error	the standard deviation when producing data
lambda	...
iteration	iteration times
method	two options: "L0", "L1". Two methods use different penalties, where "L1" indicating Lasso penalty.

Value

A list consists of mean CER and standard deviation of CER across iterations.

`sparse.choosekrl` *Perform tuning parameter (d_1, d_2 and d_3) selection for sparse tensor clustering via cross validation*

Description

Select the best d_1, d_2 and d_3 to perform clustering. A range of values of d[1], d[2], d[3] is usually considered.

Usage

```
sparse.choosekrl(x, k, r, l, lambda = 0, percent = 0.2,
  trace = FALSE, nstart = 20, sim.times = 1, method = "L0")
```

Arguments

`x`	a three-dimensional array
`k`	the range of d_1: a vector, the possible clusters numbers of mode 1
`r`	the range of d_2: a vector, the possible clusters numbers of mode 2
`l`	the range of d_3: a vector, the possible clusters numbers of mode 3
`lambda`	a numeric value. The coefficient of the regularization term.
`percent`	a numeric value between 0 and 1
`trace`	trace logic value. If true, it would print the iteration situation.
`nstart`	positive interger. The same as the "nstart" in kmeans().
`sim.times`	the same as label2(): the times of calling classify2() with different seeds.
`method`	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty and "L1" indicating Lasso penalty.

Value

a list `estimated_krl` a 1*3 matrix which is the estimated c(d_1,d_2,d_3).

`results.se` the standard error of each possible combination of the cluster numbers.

`results.mean` the mean residual of each possible combination of the cluster numbers.

`sparse.evaluate` *Evaluate the accuracy of clustering result*

Description

Given the input tensor, perform tensor clustering and evaluate the accuracy of the clustering result.

Usage

```
sparse.evaluate(bires, data, CER = TRUE, show = TRUE)
```

Arguments

`bires`	the return value of label2()
`data`	the return value of get.data()
`CER`	logic value. If true, it would return CER also
`show`	logic value. If true, it would print the result

Value

a list: `sparsityrate`: estimated sparsity rate;

`correctzerorate`: correct zero rate;

`correctonerate`: correct one rate;

`totalincorrectrate`: total incorrect rate;

`cerC`: CER on mode 1;

`cerD`: CER on mode 2;

`cerE`: CER on mode 3;

`cerTotal`: overall CER;

`error`: relative error.

Examples

```
data = get.data(20,20,20,2,2,2,sparse.percent=0.1)
bires = label2(data$x,2,2,2)
sparse.evaluate(bires,data)
```

tensor.calculateBIC *Calculate the BIC of clustering result*

Description

Given the clustering result, calculate the BIC.

Usage

```
tensor.calculateBIC(x, clusterobj, method = "L0", apply = "main")
```

Arguments

x	a three-dimensional array
clusterobj	the return object of label2() or classify2()
method	two options: "L0", "L1". Two methods use different penalties, where "L0" indicating L0 penalty, "L1" indicating Lasso penalty.
apply	"main": apply in the main formula; "cp": apply in the CPD k-means.

Value

a vector [1]BIC, [2]nonzeromus

Examples

```
x = get.data(20,20,20,2,2,2)$x
clusterobj = classify2(x,2,2,2)
tensor.calculateBIC(x,clusterobj)
```

tucker_kmeans *Tensor clustering by performing k-means on tucker decomposition*

Description

Tensor clustering by performing k-means on tucker decomposition.

Usage

```
tucker_kmeans(x, k, r, l)
```

Arguments

x	a three-dimensional array
k	numeric value. The clusters number of mode 1
r	numeric value. The clusters number of mode 2
l	numeric value. The clusters number of mode 3

Value

a list with only one element: judgeX

Examples

```
x = get.data(20,20,20,2,2,2)$x
tucker_kmeans(x,2,2,2)
```

Index