

# Package ‘tensorregress’

June 20, 2021

**Title** Supervised Tensor Decomposition with Side Information

**Version** 4.0

**Date** 2021-06-03

**Description** Implement the alternating algorithm for supervised tensor decomposition with interactive side information.

**Imports** pracma,speedglm,MASS,methods

**Maintainer** Jiaxin Hu <jhu267@wisc.edu>

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**NeedsCompilation** no

**Author** Jiaxin Hu [aut, cre, cph], Zhuoyan Xu [aut, cph], Chan-woo Lee [aut, cph], Miaoyan Wang [aut, cph]

**RoxygenNote** 7.1.1

**Depends** R (>= 2.10)

## R topics documented:

as.tensor . . . . .	2
dim-methods . . . . .	2
fold . . . . .	3
HCP . . . . .	4
hosvd . . . . .	4
kronecker_list . . . . .	5
nations . . . . .	6
rand_tensor . . . . .	6
sele_rank . . . . .	7
sim_data . . . . .	8
Tensor-class . . . . .	10
tensor_regress . . . . .	10
ttl . . . . .	12
ttm . . . . .	13
tucker . . . . .	14
unfold-methods . . . . .	15
<b>Index</b>	<b>17</b>

---

as.tensor	<i>Tensor Conversion</i>
-----------	--------------------------

---

**Description**

Create a **Tensor-class** object from an array, matrix, or vector.

**Usage**

```
as.tensor(x, drop = FALSE)
```

**Arguments**

x	an instance of array, matrix, or vector
drop	whether or not modes of 1 should be dropped

**Value**

a **Tensor-class** object

**Examples**

```
#From vector
vec <- runif(100); vecT <- as.tensor(vec); vecT
#From matrix
mat <- matrix(runif(1000),nrow=100,ncol=10)
matT <- as.tensor(mat); matT
#From array
indices <- c(10,20,30,40)
arr <- array(runif(prod(indices)), dim = indices)
arrT <- as.tensor(arr); arrT
```

---

dim-methods	<i>Mode Getter for Tensor</i>
-------------	-------------------------------

---

**Description**

Return the vector of modes from a tensor

**Usage**

```
## S4 method for signature 'Tensor'
dim(x)
```

**Arguments**

x	the Tensor instance
---	---------------------

**Details**

```
dim(x)
```

**Value**

an integer vector of the modes associated with x

**Examples**

```
tnsr <- rand_tensor()
dim(tnsr)
```

---

fold

*General Folding of Matrix*


---

**Description**

General folding of a matrix into a Tensor. This is designed to be the inverse function to [unfold-methods](#), with the same ordering of the indices. This amounts to following: if we were to unfold a Tensor using a set of `row_idx` and `col_idx`, then we can fold the resulting matrix back into the original Tensor using the same `row_idx` and `col_idx`.

**Usage**

```
fold(mat, row_idx = NULL, col_idx = NULL, modes = NULL)
```

**Arguments**

<code>mat</code>	matrix to be folded into a Tensor
<code>row_idx</code>	the indices of the modes that are mapped onto the row space
<code>col_idx</code>	the indices of the modes that are mapped onto the column space
<code>modes</code>	the modes of the output Tensor

**Details**

This function uses `aperm` as the primary workhorse.

**Value**

Tensor object with modes given by `modes`

**References**

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009, Vol. 51, No. 3 (September 2009), pp. 455-500. URL: <https://www.jstor.org/stable/25662308>.

**See Also**

[unfold-methods](#)

**Examples**

```
tnsr <- new('Tensor', 3L, c(3L, 4L, 5L), data = runif(60))
matT3 <- unfold(tnsr, row_idx = 2, col_idx = c(3, 1))
identical(fold(matT3, row_idx = 2, col_idx = c(3, 1), modes = c(3, 4, 5)), tnsr)
```

HCP

*HCP data***Description**

The array "tensor" is a  $68 \times 68 \times 136$  binary tensor consisting of structural connectivity patterns among 68 brain regions for 136 individuals from Human Connectome Project (HCP). All the individual images were preprocessed following a standard pipeline (Zhang et al., 2018), and the brain was parcellated to 68 regions-of-interest following the Desikan atlas (Desikan et al., 2006). The tensor entries encode the presence or absence of fiber connections between those 68 brain regions for each of the 136 individuals. The data frame "attr" is a  $136 \times 573$  matrix consisting of 573 personal features for 136 individuals. The full list of covariates can be found at: <https://wiki.humanconnectome.org/display/PublicData/>

**Usage**

```
data(HCP)
```

**Format**

A list. Includes a 68-68-136 binary array named "tensor" and a 136-573 data frame named "attr".

hosvd

*(Truncated-)Higher-order SVD***Description**

Higher-order SVD of a K-Tensor. Write the K-Tensor as a (m-mode) product of a core Tensor (possibly smaller modes) and K orthogonal factor matrices. Truncations can be specified via ranks (making them smaller than the original modes of the K-Tensor will result in a truncation). For the mathematical details on HOSVD, consult Lathauwer et. al. (2000).

**Usage**

```
hosvd(tnsr, ranks = NULL)
```

**Arguments**

tnsr	Tensor with K modes
ranks	a vector of desired modes in the output core tensor, default is tnsr@modes

**Details**

A progress bar is included to help monitor operations on large tensors.

**Value**

a list containing the following:

`Z` core tensor with modes specified by ranks

`U` a list of orthogonal matrices, one for each mode

`est` estimate of `tnsr` after compression

`fnorm_resid` the Frobenius norm of the error `fnorm(est-tnsr)` - if there was no truncation, then this is on the order of `mach_eps * fnorm`.

**Note**

The length of ranks must match `tnsr@num_modes`.

**References**

L. Lathauwer, B.Moor, J. Vandewalle, "A multilinear singular value decomposition". Journal of Matrix Analysis and Applications 2000, Vol. 21, No. 4, pp. 1253–1278.

**See Also**

[tucker](#)

**Examples**

```
tnsr <- rand_tensor(c(6,7,8))
hosvdD <- hosvd(tnsr)
hosvdD$fnorm_resid
hosvdD2 <- hosvd(tnsr, ranks=c(3,3,4))
hosvdD2$fnorm_resid
```

---

kronecker\_list

*List Kronecker Product*


---

**Description**

Returns the Kronecker product from a list of matrices or vectors. Commonly used for n-mode products and various Tensor decompositions.

**Usage**

```
kronecker_list(L)
```

**Arguments**

`L` list of matrices or vectors

**Value**

matrix that is the Kronecker product

### Examples

```
smalllize <- list('mat1' = matrix(runif(12),ncol=4),
  'mat2' = matrix(runif(12),ncol=4),
  'mat3' = matrix(runif(12),ncol=4))
dim(kronecker_list(smalllize))
```

---

nations

*nations data*

---

### Description

The array "R" is a  $14 \times 14 \times 56$  binary tensor consisting of 56 political relations of 14 countries between 1950 and 1965. The tensor entry indicates the presence or absence of a political action, such as "treaties", "sends tourists to", between the nations. Please set the diagonal elements  $Y(i,i,k) = 0$  in the analysis. The matrix "cov" is a  $14 \times 6$  matrix describing a few important country attributes, e.g. whether a nation is actively involved in medicine NGO, law NGO, or belongs to a catholic nation, etc.

### Usage

```
data(nations)
```

### Format

A list. Includes a 14-14-56 binary array named "R" and a 14-6 matrix named "cov".

---

rand\_tensor

*Tensor with Random Entries*

---

### Description

Generate a Tensor with specified modes with iid normal(0,1) entries.

### Usage

```
rand_tensor(modes = c(3, 4, 5), drop = FALSE)
```

### Arguments

modes	the modes of the output Tensor
drop	whether or not modes equal to 1 should be dropped

### Value

a Tensor object with modes given by modes

### Note

Default rand\_tensor() generates a 3-Tensor with modes c(3,4,5).

**Examples**

```

rand_tensor()
rand_tensor(c(4,4,4))
rand_tensor(c(10,2,1),TRUE)

```

sele\_rank

*Rank selection***Description**

Estimate the Tucker rank of tensor decomposition based on BIC criterion. The choice of BIC aims to balance between the goodness-of-fit for the data and the degree of freedom in the population model.

**Usage**

```

sele_rank(
  tsr,
  X_covar1 = NULL,
  X_covar2 = NULL,
  X_covar3 = NULL,
  rank_range,
  niter = 10,
  cons = "non",
  lambda = 0.1,
  alpha = 1,
  solver = "CG",
  dist,
  initial = c("random", "QR_tucker")
)

```

**Arguments**

tsr	response tensor with 3 modes
X_covar1	side information on first mode
X_covar2	side information on second mode
X_covar3	side information on third mode
rank_range	a matrix containing rank candidates on each row
niter	max number of iterations if update does not convergence
cons	the constraint method, "non" for without constraint, "vanilla" for global scale down at each iteration, "penalty" for adding log-barrier penalty to object function.
lambda	penalty coefficient for "penalty" constraint
alpha	max norm constraint on linear predictor
solver	solver for solving object function when using "penalty" constraint, see "details"
dist	distribution of response tensor, see "details"
initial	initialization of the alternating optimiation, "random" for random initialization, "QR_tucker" for deterministic initialization using tucker decomposition

## Details

For rank selection, recommend using non-constraint version.

Constraint penalty adds log-barrier regularizer to general object function (negative log-likelihood). The main function uses solver in function "optim" to solve the objective function. The "solver" passes to the argument "method" in function "optim".

dist specifies three distributions of response tensor: binary, poisson and normal distributions.

## Value

a list containing the following:

rank a vector with selected rank with minimal BIC

result a matrix containing rank candidate and its loglikelihood and BIC on each row

## Examples

```
seed=24
dist='binary'
data=sim_data(seed, whole_shape = c(20,20,20),
core_shape=c(3,3,3),p=c(5,5,5),dist=dist, dup=5, signal=4)
rank_range = rbind(c(3,3,3),c(3,3,2),c(3,2,2),c(2,2,2),c(3,2,3))
re = sele_rank(data$tsr[[1]],data$X_covar1,data$X_covar2,data$X_covar3,
rank_range = rank_range,niter=10,cons = 'non',dist = dist)
```

---

sim\_data

*Simulation of supervised tensor decomposition models*

---

## Description

Generate tensor data with multiple side information matrices under different simulation models, specifically for tensors with 3 modes

## Usage

```
sim_data(
  seed = NA,
  whole_shape = c(20, 20, 20),
  core_shape = c(3, 3, 3),
  p = c(3, 3, 0),
  dist,
  dup,
  signal,
  block = rep(FALSE, 3),
  ortho = FALSE
)
```



## Arguments

seed	a random seed for generating data
whole_shape	a vector containing dimension of the tensor
core_shape	a vector containing Tucker rank of the tensor decomposition
p	a vector containing numbers of side information on each mode, see "details"
dist	distribution of response tensor, see "details"
dup	number of simulated tensors from the same linear predictor
signal	a scalar controlling the max norm of the linear predictor
block	a vector containing boolean variables, see "details"
ortho	if "TRUE", generate side information matrices with orthogonal columns; if "FALSE" (default), generate side information matrices with gaussian entries

## Details

By default non-positive entry in p indicates no covariate on the corresponding mode of the tensor.

dist specifies three distributions of response tensor: binary, poisson or normal distribution.

block specifies whether the coefficient factor matrix is a membership matrix, set to TRUE when utilizing the stochastic block model

## Value

a list containing the following:

tsr a list of simulated tensors, with the number of replicates specified by dup

X\_covar1 a matrix, side information on first mode

X\_covar2 a matrix, side information on second mode

X\_covar3 a matrix, side information on third mode

W a list of orthogonal factor matrices - one for each mode, with the number of columns given by core\_shape

G an array, core tensor with size specified by core\_shape

C\_ts an array, coefficient tensor, Tucker product of G,A,B,C

U an array, linear predictor,i.e. Tucker product of C\_ts,X\_covar1,X\_covar2,X\_covar3

## Examples

```
seed = 34
dist = 'binary'
data=sim_data(seed, whole_shape = c(20,20,20), core_shape=c(3,3,3),
p=c(5,5,5),dist=dist, dup=5, signal=4)
```

---

Tensor-class	<i>S4 Class for a Tensor</i>
--------------	------------------------------

---

### Description

An S4 class for a tensor with arbitrary number of modes. The Tensor class extends the base "array" class to include additional tensor manipulation (folding, unfolding, reshaping, subsetting) as well as a formal class definition that enables more explicit tensor algebra.

### Slots

**num\_modes** number of modes (integer)  
**modes** vector of modes (integer), aka sizes/extents/dimensions  
**data** actual data of the tensor, which can be 'array' or 'vector'

### Note

All of the decompositions and regression models in this package require a Tensor input.

### Author(s)

James Li <jamesyli@gmail.com>

### References

James Li, Jacob Bien, Martin T. Wells (2018). rTensor: An R Package for Multidimensional Array (Tensor) Unfolding, Multiplication, and Decomposition. Journal of Statistical Software, Vol. 87, No. 10, 1-31. URL: <http://www.jstatsoft.org/v087/i10/>.

### See Also

[as.tensor](#)

---

tensor_regress	<i>Supervised Tensor Decomposition with Interactive Side Information</i>
----------------	--

---

### Description

Supervised tensor decomposition with interactive side information on multiple modes. Main function in the package. The function takes a response tensor, multiple side information matrices, and a desired Tucker rank as input. The output is a rank-constrained M-estimate of the core tensor and factor matrices.

**Usage**

```

tensor_regress(
  tsr,
  X_covar1 = NULL,
  X_covar2 = NULL,
  X_covar3 = NULL,
  core_shape,
  niter = 20,
  cons = c("non", "vanilla", "penalty"),
  lambda = 0.1,
  alpha = 1,
  solver = "CG",
  dist = c("binary", "poisson", "normal"),
  traj_long = FALSE,
  initial = c("random", "QR_tucker")
)

```

**Arguments**

tsr	response tensor with 3 modes
X_covar1	side information on first mode
X_covar2	side information on second mode
X_covar3	side information on third mode
core_shape	the Tucker rank of the tensor decomposition
niter	max number of iterations if update does not convergence
cons	the constraint method, "non" for without constraint, "vanilla" for global scale down at each iteration, "penalty" for adding log-barrier penalty to object function
lambda	penalty coefficient for "penalty" constraint
alpha	max norm constraint on linear predictor
solver	solver for solving object function when using "penalty" constraint, see "details"
dist	distribution of the response tensor, see "details"
traj_long	if "TRUE", set the minimal iteration number to 8; if "FALSE", set the minimal iteration number to 0
initial	initialization of the alternating optimiation, "random" for random initialization, "QR_tucker" for deterministic initialization using tucker decomposition

**Details**

Constraint penalty adds log-barrier regularizer to general object function (negative log-likelihood). The main function uses solver in function "optim" to solve the objective function. The "solver" passes to the argument "method" in function "optim".

dist specifies three distributions of response tensor: binary, poisson and normal distribution.

If dist is set to "normal" and initial is set to "QR\_tucker", then the function returns the results after initialization.

**Value**

a list containing the following:

W a list of orthogonal factor matrices - one for each mode, with the number of columns given by core\_shape

G an array, core tensor with the size specified by core\_shape

C\_ts an array, coefficient tensor, Tucker product of G,A,B,C

U linear predictor,i.e. Tucker product of C\_ts,X\_covar1,X\_covar2,X\_covar3

lg1k a vector containing loglikelihood at convergence

sigma a scalar, estimated error variance (for Gaussian tensor) or dispersion parameter (for Bernoulli and Poisson tensors)

violate a vector listing whether each iteration violates the max norm constraint on the linear predictor, 1 indicates violation

**Examples**

```
seed = 34
dist = 'binary'
data=sim_data(seed, whole_shape = c(20,20,20), core_shape=c(3,3,3),
p=c(5,5,5),dist=dist, dup=5, signal=4)
re = tensor_regress(data$tsr[[1]],data$X_covar1,data$X_covar2,data$X_covar3,
core_shape=c(3,3,3),niter=10, cons = 'non', dist = dist)
```

---

ttl

---

*Tensor Times List*


---

**Description**

Contracted (m-Mode) product between a Tensor of arbitrary number of modes and a list of matrices. The result is folded back into Tensor.

**Usage**

```
ttl(tnsr, list_mat, ms = NULL)
```

**Arguments**

tnsr	Tensor object with K modes
list_mat	a list of matrices
ms	a vector of modes to contract on (order should match the order of list_mat)

**Details**

Performs ttm repeated for a single Tensor and a list of matrices on multiple modes. For instance, suppose we want to do multiply a Tensor object tnsr with three matrices mat1, mat2, mat3 on modes 1, 2, and 3. We could do ttm(ttm(ttm(tnsr,mat1,1),mat2,2),3), or we could do ttl(tnsr,list(mat1,mat2,mat3),c(1,2,3)). The order of the matrices in the list should obviously match the order of the modes. This is a common operation for various Tensor decompositions such as CP and Tucker. For the math on the m-Mode Product, see Kolda and Bader (2009).

**Value**

Tensor object with K modes

**Note**

The returned Tensor does not drop any modes equal to 1.

**References**

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009, Vol. 51, No. 3 (September 2009), pp. 455-500. URL: <https://www.jstor.org/stable/25662308>

**See Also**

[ttm](#)

**Examples**

```
tnsr <- new('Tensor', 3L, c(3L, 4L, 5L), data=runif(60))
lizt <- list('mat1' = matrix(runif(30), ncol=3),
'mat2' = matrix(runif(40), ncol=4),
'mat3' = matrix(runif(50), ncol=5))
ttl(tnsr, lizt, ms=c(1, 2, 3))
```

---

ttm

---

*Tensor Matrix Product (m-Mode Product)*


---

**Description**

Contracted (m-Mode) product between a Tensor of arbitrary number of modes and a matrix. The result is folded back into Tensor.

**Usage**

```
ttm(tnsr, mat, m = NULL)
```

**Arguments**

tnsr	Tensor object with K modes
mat	input matrix with same number columns as the mth mode of tnsr
m	the mode to contract on

**Details**

By definition, the number of columns in mat must match the mth mode of tnsr. For the math on the m-Mode Product, see Kolda and Bader (2009).

**Value**

a Tensor object with K modes

**Note**

The `m`th mode of `tnsr` must match the number of columns in `mat`. By default, the returned Tensor does not drop any modes equal to 1.

**References**

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009, Vol. 51, No. 3 (September 2009), pp. 455-500. URL: <https://www.jstor.org/stable/25662308>

**See Also**

[ttl](#)

**Examples**

```
tnsr <- new('Tensor', 3L, c(3L, 4L, 5L), data = runif(60))
mat <- matrix(runif(50), ncol = 5)
ttm(tnsr, mat, m = 3)
```

---

tucker

---

*Tucker Decomposition*


---

**Description**

The Tucker decomposition of a tensor. Approximates a  $K$ -Tensor using a  $n$ -mode product of a core tensor (with modes specified by ranks) with orthogonal factor matrices. If there is no truncation in all the modes (i.e. `ranks = tnsr@modes`), then this is the same as the HOSVD, [hosvd](#). This is an iterative algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten below `tol`, or the `max_iter` number of iterations has been reached. For more details on the Tucker decomposition, consult Kolda and Bader (2009).

**Usage**

```
tucker(tnsr, ranks = NULL, max_iter = 25, tol = 1e-05)
```

**Arguments**

<code>tnsr</code>	Tensor with $K$ modes
<code>ranks</code>	a vector of the modes of the output core Tensor
<code>max_iter</code>	maximum number of iterations if error stays above <code>tol</code>
<code>tol</code>	relative Frobenius norm error tolerance

**Details**

Uses the Alternating Least Squares (ALS) estimation procedure also known as Higher-Order Orthogonal Iteration (HOOI). Initialized using a (Truncated-)HOSVD. A progress bar is included to help monitor operations on large tensors.

**Value**

a list containing the following:

`Z` the core tensor, with modes specified by ranks

`U` a list of orthogonal factor matrices - one for each mode, with the number of columns of the matrices given by ranks

`conv` whether or not  $\text{resid} < \text{tol}$  by the last iteration

`est` estimate of `tnsr` after compression

`norm_percent` the percent of Frobenius norm explained by the approximation

`fnorm_resid` the Frobenius norm of the error  $\text{fnorm}(\text{est} - \text{tnsr})$

`all_resids` vector containing the Frobenius norm of error for all the iterations

**Note**

The length of ranks must match `tnsr@num_modes`.

**References**

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009, Vol. 51, No. 3 (September 2009), pp. 455-500. URL: <https://www.jstor.org/stable/25662308>

**See Also**

[hosvd](#)

**Examples**

```
tnsr <- rand_tensor(c(4,4,4,4))
tuckerD <- tucker(tnsr,ranks=c(2,2,2,2))
tuckerD$conv
tuckerD$norm_percent
plot(tuckerD$all_resids)
```

**Description**

Unfolds the tensor into a matrix, with the modes in `rs` onto the rows and modes in `cs` onto the columns. Note that `c(rs,cs)` must have the same elements (order doesn't matter) as `x@modes`. Within the rows and columns, the order of the unfolding is determined by the order of the modes. This convention is consistent with Kolda and Bader (2009).

**Usage**

```
unfold(tnsr, row_idx, col_idx)
```

**Arguments**

<code>tnsr</code>	the Tensor instance
<code>row_idx</code>	the indices of the modes to map onto the row space
<code>col_idx</code>	the indices of the modes to map onto the column space

**Details**

```
unfold(tnsr,row_idx=NULL,col_idx=NULL)
```

**Value**

matrix with `prod(row_idx)` rows and `prod(col_idx)` columns

**References**

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009, Vol. 51, No. 3 (September 2009), pp. 455-500. URL: <https://www.jstor.org/stable/25662308>.

**Examples**

```
tnsr <- rand_tensor()
matT3<-unfold(tnsr,row_idx=2,col_idx=c(3,1))
```



# Index

## \* datasets

HCP, [4](#)

nations, [6](#)

as.tensor, [2](#), [10](#)

dim, Tensor-method (dim-methods), [2](#)

dim-methods, [2](#)

fold, [3](#)

HCP, [4](#)

hosvd, [4](#), [14](#), [15](#)

kronecker\_list, [5](#)

nations, [6](#)

rand\_tensor, [6](#)

sele\_rank, [7](#)

sim\_data, [8](#)

Tensor (Tensor-class), [10](#)

Tensor-class, [10](#)

tensor\_regress, [10](#)

ttl, [12](#), [14](#)

ttm, [13](#), [13](#)

tucker, [5](#), [14](#)

unfold (unfold-methods), [15](#)

unfold, Tensor-method (unfold-methods),

[15](#)

unfold-methods, [15](#)