# Package 'SuperPCA'

July 26, 2021

**Type** Package

**Title** Supervised Principal Component Analysis

**Version** 0.4.0

**Author** Gen Li <ligen@umich.edu>, Haocheng Ding <haochengding@ufl.edu>, Jiayi Ji <jj2876@caa.columbia.edu>

**Maintainer** Jiayi Ji <jj2876@caa.columbia.edu>

**Description** Dimension reduction of complex data with supervision from auxiliary information. The package contains a series of methods for different data types (e.g., multi-view or multi-way data) including the supervised singular value decomposition (SupSVD), supervised sparse and functional principal component (SupSFPC), supervised integrated factor analysis (SIFA) and supervised PARAFAC/CANDECOMP factorization (SupCP). When auxiliary data are available and potentially affect the intrinsic structure of the data of interest, the methods will accurately recover the underlying low-rank structure by taking into account the supervision from the auxiliary data. For more details, see the paper by Gen Li, <DOI:10.1111/biom.12698>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** false

**Imports** RSpectra, psych, fBasics, R.matlab, glmnet, MASS, matrixStats, timeSeries, stats, matlabr, spls, pracma, matlab

**Depends** Matrix

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2021-07-26 12:30:07 UTC

## R topics documented:

---

kr *Compute a string of Khatri-Rao products*

---

## Description

The function compute a string of Khatri-Rao products. A*B*C*..., "*" denotes the Khatri-Rao product. If the list only contains two matrices, kr() return the Khatri-Rao product of two matrices.

## Usage

```
kr(x)
```

## Arguments

| | |
|---|---|
| x | a list of matrices |

## Value

result of a string of Khatri-Rao products

## Examples

```
#ex1
m1 <- matrix(1:9,3,3)
m2 <- matrix(1:12,4,3)
m3 <- matrix(13:27,5,3)
l1 <- list(m1,m2,m3)
kr(l1)
#ex2
l2 <- list(m1,m3)
kr(l2)
```

## normc *Normaliz the columns of x to a length of 1.*

### Description

Normaliz the columns of x to a length of 1.

### Usage

```
normc(x)
```

### Arguments

| | |
|---|---|
| x | n*p matrix |

### Value

xn normalized result

### Examples

```
#ex1.
m <- matrix(1:4,2,2,byrow=TRUE)
normc(m)
#ex2.
n <- matrix(rnorm(100,10,1),10,10)
normc(n)
```

## Parafac *Performs parafac factorization via ALS*

### Description

Performs parafac factorization via ALS

### Usage

```
Parafac(Y, R)
```

### Arguments

| | |
|---|---|
| Y | Array of dimension m1*m2*...*mK |
| R | Desired rank of the factorization. Defalust is all columns(range=1:R) |

## Value

list with components

| | |
|---|---|
| U: | List of basis vectors for parafac factorization, U[k]: mk*R for k=1,...,K. Columns of U[K] have norm 1 for k>1. |
| SqError: | Vector of squared error for the approximation at each iteration (should be non-increasing) |

## Examples

```
A <- array(stats::rnorm(100*10*10), dim=c(100,10,10))
Parafac(A,4)
```

---

# SIFA *Supervised Integrated Factor Analysis*

---

## Description

Supervised Integrated Factor Analysis

## Usage

```
SIFA(
  X,
  Y,
  r0,
  r,
  relation = "linear",
  sparsity = 0,
  max_niter = 1000,
  convg_thres = 0.01,
  type = "Anp"
)
```

## Arguments

| | |
|---|---|
| X | n*q matrix, centered covariate data |
| Y | 1*K cell array, each cell is a n*pi centered primary data, should roughly contribute equally to joint struct |
| r0 | scalar, prespecified rank of common structure |
| r | 1*K vector, prespecified rank of specific structures |
| relation | 'linear' (default), use linear function to model the relation between U and covariates 'univ_kernel', use kernel methods for single covariates |
| sparsity | 1 (default), when est B0 or B, use LASSO with BIC to select the best tuning, suitable for high |

| | |
|---|---|
| max_niter | default 1000, max number of iteration |
| convg_thres | default 0.01, overall convergence threshold |
| type | "A" or "Anp" or "B" or "Bnp", condition to use |

## Value

list with components

| | |
|---|---|
| B0: | q*r0 matrix, coefficient for joint structure (may be sparse) |
| B: | 1*K cell array, each is a q*ri coefficient matrix (may be sparse) |
| V_joint: | sum(p)*r0 matrix, stacked joint loadings, with orthonormal columns. |
| V_ind: | 1*K array, each is a pi*ri loading matrix, with orthonormal columns |
| se2: | 1*K vector, noise variance for each phenotypic data set |
| Sf0: | r0*r0 matrix, diagonal covariance matrix |
| Sf: | 1*K array, each is a ri*ri diagonal covariance matrix |
| EU: | n*(r0+sum(ri)) matrix, conditional expectation of joint and individual scores |

## Examples

```
## Not run:
r0 <- 2
r <- c(3,3)
V <- matrix(stats::rnorm(10*2),10,2)
Fmatrix <- matrix(MASS::mvrnorm(n=1*500,rep(0,2),matrix(c(9,0,0,4),2,2)),500,2)
E <- matrix(stats::rnorm(500*10,0,3),500,10)
X <- tcrossprod(Fmatrix,V)+E
X <-scale(X,center=TRUE,scale=FALSE)
Y1 <- matrix(stats::rnorm(500*200,0,1),500,200)
Y2 <- matrix(stats::rnorm(500*200,0,1),500,200)
Y <- list(Y1,Y2)
SIFA(X,Y,r0,r,max_niter=200)

## End(Not run)
```

---

SupParafacEM *Using EM algorithm to fit the SupCP model*

---

## Description

Using EM algorithm to fit the SupCP model

## Usage

```
SupParafacEM(
  Y,
  X,
  R,
  AnnealIters = 100,
  ParafacStart = 0,
  max_niter = 1000,
  convg_thres = 10^-3,
  Sf_diag = 1
)
```

## Arguments

| | |
|---|---|
| Y | n*q full column rank reponse matrix(necessarily n>=q) |
| X | n*p1*...*pk design array |
| R | fixed rankd of approximation, R<=min(n,p) |
| AnnealIters | Annealing iterations (default = 100) |
| ParafacStart | binary argument for wether to initialize with Parafac factorization (default = 0) |
| max_niter | maximum number of iterations (default = 1000) |
| convg_thres | convergence threshold for difference in log likelihood (default = 10^-3) |
| Sf_diag | whether Sf is diagnal (default=1,diagnoal) |

## Value

list with components

| | |
|---|---|
| B: | q*r coefficient matrix for U~Y |
| V | list of length K-1. V[k] is a p*r coefficient matrix with columns of norm 1 |
| U: | Conditional expectation of U: n*r |
| se2: | scalar, var(E) |
| Sf: | r*r diagonal matrix, cov(F) |
| rec: | log likelihood for each iteration |

## Examples

```
sigmaF <- diag(c(100,64,36,16,4))
# F matrix n*r
Fmatrix1 <- matrix(MASS::mvrnorm(n=100,rep(0,5),sigmaF),100,5)
U<-Fmatrix1
V1 <- matrix(stats::rnorm(10*5),10,5)
V2 <- matrix(stats::rnorm(10*5),10,5)
L <- list(U,V1,V2)
X <- TensProd(L)
Y <- matrix(stats::rnorm(100*10),100,10)
R <-3
SupParafacEM(Y,X,R)
```

SupPCA *Fit a supervised singular value decomposition (SupSVD) model*

## Description

This function fits the SupSVD model: X=UV' + E, U=YB + F where X is an observed primary data matrix (to be decomposed), U is a latent score matrix, V is a loading matrix, E is measurement noise, Y is an observed auxiliary supervision matrix, B is a coefficient matrix, and F is a random effect matrix.
It is a generalization of principal component analysis (PCA) or singular value decomposition (SVD). It decomposes the primary data matrix X into low-rank components, while taking into account potential supervision from any auxiliary data Y measured on the same samples.

See more details in 2016 JMVA paper "Supervised singular value decomposition and its asymptotic properties" by Gen Li, Dan Yang, Andrew B Nobel and Haipeng Shen.

## Usage

```
SupPCA(Y, X, r)
```

## Arguments

| | |
|---|---|
| Y | n*q (column centered) auxiliary data matrix, rows are samples and columns are stats::variables (must have linearly independent columns to avoid overfitting) |
| X | n*p (column centered) primary data matrix, which we want to decompose. rows are samples (matched with Y) and columns are variables |
| r | positive scalar, prespecified rank (r < min(n,p)) |

## Value

list with components

| | |
|---|---|
| B: | q*r coefficient matrix of Y on the scores of X, maybe sparse if gamma=1 |
| V: | p*r loading matrix of X, with orthonormal columns |
| U: | n*r score matrix of X, conditional expectation of random scores |
| se2: | scalar, variance of measurement error in the primary data X |
| Sf: | r*r diagonal covariance matrix, for random effects (see paper) |

Note: Essentially, U and V are the most important output for dimension reduction purpose as in PCA or SVD.

## Examples

```
r=2
Y <- matrix(rnorm(400,0,1),nrow=100)
B <- c(-1,1,-sqrt(3/2),-1)
B <- cbind(B,c(1,1,-1,sqrt(3/2)))
V <- matrix(rnorm(68*2),68,2)
Fmatrix <- matrix(MASS::mvrnorm(n=1*100,rep(0,2),matrix(c(9,0,0,4),2,2)),100,2)
E <- matrix(rnorm(100*68,0,3),100,68)
Yc <- scale(Y,center=TRUE,scale=FALSE)

# Case 1 (supsvd) X = YBV^T+FV^T+E
X1 <- Y%*%tcrossprod(B,V)+tcrossprod(Fmatrix,V)+E
X1c <- scale(X1,center=TRUE,scale=FALSE)
SupPCA(Yc,X1c,r)
#  Case 2 (PCA) X = FV^T+E
X2 <- tcrossprod(Fmatrix,V)+E
X2c <-scale(X2,center=TRUE,scale=FALSE)
SupPCA(Yc,X2c,r)
# Case 3 (RRR) X = YBV^T+E
X3 <- Y%*%tcrossprod(B,V)+E
X3c <- scale(X3,center=TRUE,scale=FALSE)
SupPCA(Yc,X3c,r)
```

---

SupSFPCA *Supervised Sparse and Functional Principal Component Analysis*

---

## Description

This function conducts supervised sparse and functional principal component analysis by fitting the SupSVD model X=UV' + E U=YB + F where X is an observed primary data matrix (to be decomposed), U is a latent score matrix, V is a loading matrix, E is measurement noise, Y is an observed auxiliary supervision matrix, B is a coefficient matrix, and F is a random effect matrix.

It decomposes the primary data matrix X into low-rank components, while taking into account many different features: 1) potential supervision from any auxiliary data Y measured on the same samples; 2) potential smoothness for loading vectors V (for functional data); 3) sparsity in supervision coefficients B and loadings V (for variable selection).

It is a very general dimension reduction method that subsumes PCA, sparse PCA, functional PCA, supervised PCA, etc as special cases. See more details in 2016 JCGS paper "Supervised sparse and functional principal component analysis" by Gen Li, Haipeng Shen, and Jianhua Z. Huang.

## Usage

```
SupSFPCA(
  Y,
  X,
  r,
```

```
  ind_lam = 1,
  ind_alp = 1,
  ind_gam = 1,
  ind_Omg = 1,
  Omega = 0,
  max_niter = 10^3,
  convg_thres = 10^-6,
  vmax_niter = 10^2,
  vconvg_thres = 10^-4
)
```

## Arguments

| | |
|---|---|
| Y | n*q (column centered) auxiliary data matrix, rows are samples and columns are variables |
| X | n*p (column centered) primary data matrix, which we want to decompose. rows are samples (matched with Y) and columns are variables |
| r | positive scalar, prespecified rank (r should be smaller than n and p) |
| ind_lam | 0 or 1 (default=1, sparse loading), sparsity index for loadings |
| ind_alp | 0 or 1 (default=1, smooth loading), smoothness index for loadings |
| ind_gam | 0 or 1 (default=1, sparse coefficient), sparsity index for supervision coefficients. Note: if gamma is set to be 0, Y must have q<n to avoid overfitting; if gamma is set to be 1, then it can handle high dimensional supervision Y |
| ind_Omg | p*p symmetric positive semi-definite matrix for smoothness penalty (default is for evenly spaced data) Note: only change this if you have unevenly spaced functional data X |
| Omega | ?? |
| max_niter | scalar (default=1E3), max number of overall iterations |
| convg_thres | positive scalar (default=1E-6), overall convergence threshold |
| vmax_niter | scalar (default=1E2), max number of iterations for estimating each loading vector |
| vconvg_thres | positive scalar (default=1E-4), convergence threshold for the proximal gradient descent algorithm for estimating each loading vector |

## Value

list with components

| | |
|---|---|
| B: | q*r coefficient matrix of Y on the scores of X,maybe sparse if gamma=1 |
| V: | p*r loading matrix of X, each column has norm 1, but no strict orthogonality because of sparsity and smoothness. If lambda=1, V is sparse; if alpha=1, each column of V is smooth |
| U: | n*r score matrix of X, conditional expectation of random scores, no strict orthogonality |
| se2: | scalar, variance of measurement error in the primary data X |

`Sf:` r*r diagonal covariance matrix, for random effects (see paper)

Note: Essentially, U and V are the most important output for dimension reduction purpose as in PCA or SVD.

## Examples

```
## Not run:
library(spls)
data(yeast)
r <- 4
ydata <- as.data.frame(yeast[1])
xdata <- as.data.frame(yeast[2])
yc <- scale(ydata,center = TRUE,scale=FALSE)
xc <- scale(xdata,center=TRUE,scale=FALSE)
SupSFPCA(yc,xc,r)

## End(Not run)
```

---

`TensProd` *Compute tensor product over multiple dimensions using Khatri-Rao product*

---

## Description

Compute tensor product over multiple dimensions using Khatri-Rao product

## Usage

```
TensProd(L, range = NA)
```

## Arguments

| | |
|---|---|
| `L` | List of length K, with matrix entries Lk: m_k X R; |
| `range` | Column indices over which to apply tensor product.Default is all columns (range = [1:R]) |

## Value

Ans Array of size m_1 X m_2 ... X m_K, where the [i1,...iK]; entry is the sum of product Lk(i1,r)*...*LK(iK,r) over all r in range.

## Examples

```
L <- list(matrix(rnorm(10*10),10,10),matrix(rnorm(10*10),10,10),matrix(rnorm(1000*10),1000,10))
TensProd(L)
```

# Index