Package ‘JICO’

May 7, 2026

Title Joint and Individual Regression

Version 0.1

Description An R package that implements the JICO algorithm [Wang, P., Wang, H., Li, Q., Shen, D., & Liu, Y. (2024). <Journal of Computational and Graphical Statistics, 33(3), 763-773>].
It aims at solving the multi-group regression problem. The algorithm decomposes the responses from multiple groups into shared and group-specific
components, which are driven by low-
rank approximations of joint and individual structures from the covariates respectively.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports nleqslv, Matrix, MASS, rlist

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Peiyao Wang [aut, cre]

Maintainer Peiyao Wang <peiyaow76@gmail.com>

Repository CRAN

Date/Publication 2025-01-21 09:50:02 UTC

Contents

C2beta 2
continuum 2
continuum.multigroup.iter 3
createFolds 5
cv.continnum.iter 5
DIAG 7
initialize.UDVZ 8
parameter.set.G_2 8

parameter.set.G_3 9
parameter.set.rankA_eq 9
SOLVE 10

Index **11**

`C2beta` *Compute the coefficients from the continuum regression (CR) algorithm*

Description

This function converts the CR algorithm outputs to the regression coefficients

Usage

```
C2beta(X, Y, C, lambda)
```

Arguments

`X`	The input feature matrix
`Y`	The input response vector
`C`	The weight matrix computed from CR algorithm
`lambda`	Deprecated. Regularization parameter if L2 penalization is used for CR. JICO uses zero as default.

Value

A list of regression coefficients to perform the prediction task.

`continuum` *The continuum regression (CR) algorithm*

Description

This function performs an iteration update of the JICO algorithm using the CR algorithm. Details can be found in Appendix B in the JICO paper: https://arxiv.org/pdf/2209.12388.pdf

Usage

```
continuum(
  X,
  Y,
  lambda,
  gam,
  om,
  U_old = matrix(, nrow = nrow(X), ncol = 0),
  D_old = matrix(, nrow = 0, ncol = 0),
  V_old = matrix(, nrow = 0, ncol = 0),
  Z_old = matrix(, nrow = 0, ncol = 0),
  verbose = FALSE
)
```

Arguments

`X`	The input feature matrix
`Y`	The input response vector
`lambda`	Deprecated. Regularization parameter if L2 penalization is used for CR. JICO uses zero as default.
`gam`	The gamma parameter in the CR algorithm. Set gam=0 for OLS model, gam=0.5 for PLS model, gam >= 1e10 for PCR model
`om`	The desired number of weight vectors to obtain in the CR algorithm, i.e. the predefined rank of joint or individual componenet
`U_old`	The given inputs U from the previous JICO iteration step
`D_old`	The given inputs D from the previous JICO iteration step
`V_old`	The given inputs V from the previous JICO iteration step
`Z_old`	The given inputs Z from the previous JICO iteration step
`verbose`	Boolean. If it's desired to print out intermediate outputs

Value

A list of CR outputs that serve as the input for the next JICO iteration

---

`continuum.multigroup.iter` *The Joint and Individual Component Regression (JICO) algorithm*

Description

This function iteratively solves the multi-group regression problem using the JICO algorithm. JICO paper: https://arxiv.org/pdf/2209.12388.pdf

Usage

```
continuum.multigroup.iter(
  X.list,
  Y.list,
  lambda = 0,
  gam,
  rankJ,
  rankA,
  maxiter = 1000,
  conv = 1e-07,
  center.X = TRUE,
  scale.X = TRUE,
  center.Y = TRUE,
  scale.Y = TRUE,
  orthIndiv = FALSE,
  I.initial = NULL,
  sd = 0
)
```

Arguments

`X.list`	The list of feature matrices from multiple groups.
`Y.list`	The list of feature vectors from multiple groups.
`lambda`	Deprecated. Regularization parameter if L2 penalization is used for CR. JICO uses zero as default.
`gam`	The gamma parameter in the CR algorithm. Set gam=0 for OLS model, gam=0.5 for PLS model, gam >= 1e10 for PCR model.
`rankJ`	The rank for the joint component.
`rankA`	The ranks for individual components.
`maxiter`	The maximum number of iterations to conduct before algorithm convergence.
`conv`	The tolerance level for covergence.
`center.X`	Boolean. If X should be preprocessed with centralization.
`scale.X`	Boolean. If X should be preprocessed with scaling.
`center.Y`	Boolean. If Y should be preprocessed with centralization.
`scale.Y`	Boolean. If Y should be preprocessed with scaline.
`orthIndiv`	Boolean. If we impose the orthogonality constraint on individual components.
`I.initial`	The initial values for individual components.
`sd`	The standard deviation used to generate random initial values for individual weight vectors.

Value

The estimated parameters from JICO.

Examples

```
set.seed(76)
X1 = MASS::mvrnorm(50, rep(0, 200), diag(200)) # covariates of the first group
X2 = MASS::mvrnorm(50, rep(0, 200), diag(200)) # covariates of the second group
X.list = list(X1, X2)

Y1 = matrix(stats::rnorm(50)) # responses for the first group
Y2 = matrix(stats::rnorm(50)) # responses for the second group
Y.list = list(Y1, Y2)

ml.JICO = continuum.multigroup.iter(
  X.list, Y.list, gam=1e10, rankJ=1, rankA=c(1, 1),
  maxiter = 300
)
```

`createFolds` *Utility function to create folds for stratified samples*

Description

This function generate data folds for cross validation given stratified samples

Usage

```
createFolds(strat_id, k)
```

Arguments

`strat_id`	A vector of the stratified sample id. E.g. In total of 5 samples, first three from group 1, last two from group 2 -> c(1, 1, 1, 2, 2)
`k`	Number of folds to create.

Value

A list of sample indices in k folds.

`cv.continnum.iter` *Fit JICO with cross-validation to tune hyperparameters*

Description

This function performs K-fold cross validations to select the best tuning parameters for JICO.

Usage

```
cv.continnum.iter(
  X.list,
  Y.list,
  lambda = 0,
  parameter.set,
  nfolds = 10,
  maxiter = 100,
  center.X = TRUE,
  scale.X = TRUE,
  center.Y = TRUE,
  scale.Y = TRUE,
  orthIndiv = FALSE,
  plot = F,
  criteria = c("min", "1se"),
  sd = 0
)
```

Arguments

`X.list`	The list of feature matrices from multiple groups.
`Y.list`	The list of feature vectors from multiple groups.
`lambda`	Deprecated. Regularization parameter if L2 penalization is used for CR. JICO uses zero as default.
`parameter.set`	The set of parameters to be tuned on. Containing choices of rankJ, rankA and gamma.
`nfolds`	number of folds to perform CV
`maxiter`	The maximum number of iterations to conduct before algorithm convergence.
`center.X`	Boolean. If X should be preprocessed with centralization.
`scale.X`	Boolean. If X should be preprocessed with scaling.
`center.Y`	Boolean. If Y should be preprocessed with centralization.
`scale.Y`	Boolean. If Y should be preprocessed with scaline.
`orthIndiv`	Boolean. If we impose the orthogonality constraint on individual components.
`plot`	Boolean. If we want to plot the rMSE vs different parameters
`criteria`	criteria for selecting the best parameter. Use "min" to choose the parameter giving the best performance. Use "1se" to choose the simplest model that gives performance within 1se from the best one.
`sd`	The standard deviation used to generate random initial values for individual weight vectors.

Value

The parameter from the parameter.set that fit the training data the best.

Examples

```
set.seed(76)
X1 = MASS::mvrnorm(50, rep(0, 200), diag(200)) # covariates of the first group
X2 = MASS::mvrnorm(50, rep(0, 200), diag(200)) # covariates of the second group
X.list = list(X1, X2)

Y1 = matrix(stats::rnorm(50)) # responses for the first group
Y2 = matrix(stats::rnorm(50)) # responses for the second group
Y.list = list(Y1, Y2)

cv.parameter.set = parameter.set.G_2(
   maxrankA = 1, maxrankJ = 1, gamma = 1e10
) # enumerate the set of tuning parameters

cv.ml.JICO = cv.continnum.iter(
  X.list, Y.list, parameter.set = cv.parameter.set,
  criteria = "min", nfold = 5, maxiter = 300
) # fit the model and use CV to find the best parameters
```

DIAG *Generate diagonal matrix*

Description

This function returns a diagnoal matrix using the input vector or number as diagonal.

Usage

```
DIAG(e)
```

Arguments

e	Diagonal element. Can be a vector or a number

Value

A square diagonal matrix using the input as diagonal elements

initialize.UDVZ *Helper function to compute the SVD results*

Description

This function computes the SVD results from a given matrix X. This is used as the initialization for the continuum regression.

Usage

```
initialize.UDVZ(X)
```

Arguments

X	The input feature matrix

Value

A list of SVD results that are served as CR algorithm's inputs.

parameter.set.G_2 *Generate parameter sets (G=2)*

Description

This function generate set of hyperparameters when there are two groups.

Usage

```
parameter.set.G_2(maxrankA, maxrankJ, gamma)
```

Arguments

maxrankA	The maximum rank for individual component
maxrankJ	The maximum rank for joint component
gamma	The gamma parameter. Need to be fixed.

Value

A list of hyperparameter candidates

parameter.set.G_3 *Generate parameter sets (G=3)*

Description

This function generate set of hyperparameters when there are three groups.

Usage

```
parameter.set.G_3(maxrankA, maxrankJ, gamma)
```

Arguments

maxrankA	The maximum rank for individual component
maxrankJ	The maximum rank for joint component
gamma	The gamma parameter. Need to be fixed.

Value

A list of hyperparameter candidates

parameter.set.rankA_eq *Generate parameter sets (equal individual ranks)*

Description

This function generate set of hyperparameters when the individual ranks are the same

Usage

```
parameter.set.rankA_eq(G, maxrankA, maxrankJ, gamma.list)
```

Arguments

G	number of groups
maxrankA	The maximum rank for individual component
maxrankJ	The maximum rank for joint component
gamma.list	The list of candidate gammas to be tuned

Value

A list of hyperparameter candidates

SOLVE *Helper function to compute the inverse of input X matrix*

Description

This function computes the general inverse of X when it exists. If X contains a degenerated dimension, return the original X.

Usage

```
SOLVE(x)
```

Arguments

x	The input matrix X

Value

Either the general inverse of X or the X itself

Index

C2beta, 2
continuum, 2
continuum.multigroup.iter, 3
createFolds, 5
cv.continnum.iter, 5

DIAG, 7

initialize.UDVZ, 8

parameter.set.G_2, 8
parameter.set.G_3, 9
parameter.set.rankA_eq, 9

SOLVE, 10