Package 'ccar3'

May 8, 2026

Title Canonical Correlation Analysis via Reduced Rank Regression

Version 0.1.1

Date 2026-04-10

Author Claire Donnat [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7079-8060>),
Elena Tuzhilina [aut] (ORCID: <https://orcid.org/0000-0002-1898-6010>),
Zixuan Wu [aut] (ORCID: <https://orcid.org/0009-0006-4745-0000>)

Maintainer Claire Donnat <cdonnat@uchicago.edu>

Description Canonical correlation analysis (CCA) via reduced-rank regression with support for regularization and cross-validation. Several methods for estimating CCA in high-dimensional settings are implemented. The first set of methods, cca_rrr() (and variants: cca_group_rrr() and cca_graph_rrr()), assumes that one dataset is high-dimensional and the other is low-dimensional, while the second, ecca() (for Efficient CCA) assumes that both datasets are high-dimensional. For both methods, standard l1 regularization as well as group-lasso regularization are available. cca_graph_rrr further supports total variation regularization when there is a known graph structure among the variables of the high-dimensional dataset. In this case, the loadings of the canonical directions of the high-dimensional dataset are assumed to be smooth on the graph. For more details see Donnat and Tuzhilina (2024) <doi:10.48550/arXiv.2405.19539> and Wu, Tuzhilina and Donnat (2025) <doi:10.48550/arXiv.2507.11160>.

Depends R (>= 3.5.0)

Imports methods, magrittr, tidyr, dplyr, foreach, pracma, corpcor, matrixStats, RSpectra

Suggests codetools, SMUT, igraph, testthat (>= 3.0.0), pkgload, rrpack, Matrix, glmnet, CCA, PMA, doParallel, crayon

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Repository CRAN

Date/Publication 2026-04-29 07:00:02 UTC

Contents

`cca_graph_rrr` *Graph-regularized Reduced-Rank Regression for Canonical Correlation Analysis*

Description

Solves a sparse canonical correlation problem using a graph-constrained reduced-rank regression formulation. The problem is solved via an ADMM approach.

Usage

```
cca_graph_rrr(
  X,
  Y,
  Gamma,
  Sx = NULL,
  Sy = NULL,
  Sxy = NULL,
  lambda = 0,
  r,
  standardize = FALSE,
  preprocess = NULL,
  LW_Sy = TRUE,
```

```
  rho = 10,
  niter = 10000,
  thresh = 1e-04,
  thresh_0 = 1e-06,
  verbose = FALSE,
  Gamma_dagger = NULL
)
```

Arguments

X	Matrix of predictors (n x p)
Y	Matrix of responses (n x q)
Gamma	Graph constraint matrix (g x p)
Sx	Optional covariance matrix for X. Kept for backward compatibility; the graph fit now postprocesses directly from X and does not need to form Sx.
Sy	Optional covariance matrix for Y. If NULL, computed similarly; optionally shrunk via Ledoit-Wolf
Sxy	Optional cross-covariance matrix (not currently used)
lambda	Regularization parameter for sparsity
r	Target rank
standardize	Backward-compatible preprocessing flag: TRUE = "scale", FALSE = "center".
preprocess	Preprocessing mode. One of "scale" (center + scale), "center" (center only), or "none".
LW_Sy	Whether to apply Ledoit-Wolf shrinkage to Sy
rho	ADMM penalty parameter
niter	Maximum number of ADMM iterations
thresh	Convergence threshold for ADMM
thresh_0	Threshold for small values in the coefficient matrix (default 1e-6)
verbose	Whether to print diagnostic output
Gamma_dagger	Optional pseudoinverse of Gamma (computed if NULL)

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

cor Canonical covariances

loss The prediction error 1/n * \| XU - YV\|^2

Lambda Canonical correlations

B_opt Estimated reduced-rank coefficient matrix

`cca_graph_rrr_cv` *Graph-regularized Reduced-Rank Regression for Canonical Correlation Analysis with cross validation*

Description

Solves a sparse canonical correlation problem using a graph-constrained reduced-rank regression formulation. The problem is solved via an ADMM approach.

Usage

```
cca_graph_rrr_cv(
  X,
  Y,
  Gamma,
  r = 2,
  lambdas = 10^seq(-3, 1.5, length.out = 10),
  kfolds = 5,
  parallelize = FALSE,
  standardize = TRUE,
  LW_Sy = TRUE,
  preprocess = NULL,
  rho = 10,
  niter = 10000,
  thresh = 1e-04,
  thresh_0 = 1e-06,
  verbose = FALSE,
  Gamma_dagger = NULL,
  nb_cores = NULL
)
```

Arguments

`X`	Matrix of predictors (n x p)
`Y`	Matrix of responses (n x q)
`Gamma`	Graph constraint matrix (g x p)
`r`	Target rank
`lambdas`	Grid of regularization parameters to test for sparsity
`kfolds`	Number of folds for cross-validation
`parallelize`	Whether to parallelize cross-validation
`standardize`	Backward-compatible preprocessing flag: TRUE = `"scale"`, FALSE = `"center"`.
`LW_Sy`	Whether to apply Ledoit-Wolf shrinkage to Sy
`preprocess`	Preprocessing mode. One of `"scale"` (center + scale), `"center"` (center only), or `"none"`.

rho	ADMM penalty parameter
niter	Maximum number of ADMM iterations
thresh	Convergence threshold for ADMM
thresh_0	Threshold for small values in the coefficient matrix (default 1e-6)
verbose	Whether to print diagnostic output
Gamma_dagger	Optional pseudoinverse of Gamma (computed if NULL)
nb_cores	Number of cores to use for parallelization. Defaults to min(kfolds, available cores minus 1).

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

lambda Optimal regularisation parameter lambda chosen by CV

rmse Mean squared error of prediction (as computed in the CV)

cor Canonical covariances

lambda_x Alias of the selected `lambda`

lambda_x_se Foldwise standard error at the selected `lambda`

lambda_y Placeholder for symmetry with two-penalty interfaces

lambda_y_se Placeholder for symmetry with two-penalty interfaces

resultsx Backward-compatible alias of `cv_summary`

cv_summary Data frame with one row per lambda containing mean RMSE and its foldwise standard error

cv_folds Data frame with fold-level RMSE values for each lambda

Lambda Canonical correlations from the final fit

B Estimated reduced-rank coefficient matrix from the final fit

fit Final fit at the selected lambda

`cca_group_rrr` *Group-Sparse Canonical Correlation via Reduced-Rank Regression*

Description

Performs group-sparse reduced-rank regression for CCA using either ADMM or CVXR solvers.

Usage

```
cca_group_rrr(
  X,
  Y,
  groups,
  Sx = NULL,
  Sy = NULL,
  Sxy = NULL,
  lambda = 0,
  r,
  standardize = FALSE,
  preprocess = NULL,
  LW_Sy = TRUE,
  solver = "ADMM",
  rho = 1,
  niter = 10000,
  thresh = 1e-04,
  thresh_0 = 1e-06,
  matrix_free_threshold = 4000L,
  cg_tol = 1e-06,
  cg_maxiter = NULL,
  verbose = FALSE
)
```

Arguments

X	Predictor matrix (n x p)
Y	Response matrix (n x q)
groups	List of index vectors defining groups of predictors
Sx	Optional covariance matrix for X; if NULL computed internally
Sy	Optional covariance matrix for Y; if NULL computed internally
Sxy	Optional cross covariance matrix for X and Y; if NULL computed internally
lambda	Regularization parameter
r	Target rank
standardize	Backward-compatible preprocessing flag: TRUE = "scale", FALSE = "center".
preprocess	Preprocessing mode. One of "scale" (center + scale), "center" (center only), or "none".
LW_Sy	Whether to apply Ledoit-Wolf shrinkage to Sy (default TRUE)
solver	Either "ADMM" or "CVXR". The "CVXR" backend requires the optional package CVXR.
rho	ADMM parameter
niter	Maximum number of ADMM iterations
thresh	Convergence threshold for ADMM
thresh_0	tolerance for declaring entries non-zero

`matrix_free_threshold`
: For ADMM: when both `n` and `p` are at least this value, use a matrix-free conjugate-gradient solve instead of forming a dense linear system.

`cg_tol`
: Relative tolerance for the matrix-free conjugate-gradient solve used in ADMM.

`cg_maxiter`
: Maximum iterations for the matrix-free conjugate-gradient solve. Defaults to `min(p, 1000)`.

`verbose`
: Print diagnostics

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

cor Canonical covariances

loss The prediction error 1/n * \| XU - YV\|^2

Lambda Canonical correlations

B_opt Estimated reduced-rank coefficient matrix

`cca_group_rrr_cv` *Group-Sparse Canonical Correlation via Reduced-Rank Regression with CV*

Description

Performs group-sparse reduced-rank regression for CCA using either ADMM or CVXR solvers.

Usage

```
cca_group_rrr_cv(
  X,
  Y,
  groups,
  r = 2,
  lambdas = 10^seq(-3, 1.5, length.out = 10),
  kfolds = 5,
  parallelize = FALSE,
  standardize = FALSE,
  preprocess = NULL,
  LW_Sy = TRUE,
  solver = "ADMM",
  rho = 1,
  thresh_0 = 0,
  niter = 10000,
  thresh = 1e-04,
```

```
  matrix_free_threshold = 4000L,
  cg_tol = 1e-06,
  cg_maxiter = NULL,
  verbose = FALSE,
  nb_cores = NULL
)
```

Arguments

X	Predictor matrix (n x p)
Y	Response matrix (n x q)
groups	List of index vectors defining groups of predictors
r	Target rank
lambdas	Grid of regularization parameters to try out
kfolds	Nb of folds for the CV procedure
parallelize	Whether to use parallel processing (default is FALSE)
standardize	Backward-compatible preprocessing flag: TRUE = "scale", FALSE = "center".
preprocess	Preprocessing mode. One of "scale" (center + scale), "center" (center only), or "none".
LW_Sy	Whether to apply Ledoit-Wolf shrinkage to Sy (default TRUE)
solver	Either "ADMM" or "CVXR". The "CVXR" backend requires the optional package CVXR.
rho	ADMM parameter
thresh_0	tolerance for declaring entries non-zero
niter	Maximum number of ADMM iterations
thresh	Convergence threshold for ADMM
matrix_free_threshold	For ADMM: when both n and p are at least this value, use a matrix-free conjugate-gradient solve instead of forming a dense linear system.
cg_tol	Relative tolerance for the matrix-free conjugate-gradient solve used in ADMM.
cg_maxiter	Maximum iterations for the matrix-free conjugate-gradient solve. Defaults to min(p, 1000).
verbose	Print diagnostics
nb_cores	Number of cores to use for parallelization (default is all available cores minus 1)

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

lambda Optimal regularisation parameter lambda chosen by CV

rmse Mean squared error of prediction (as computed in the CV)

cor Canonical covariances

lambda_x Alias of the selected `lambda`

lambda_x_se Foldwise standard error at the selected `lambda`

lambda_y Placeholder for symmetry with two-penalty interfaces

lambda_y_se Placeholder for symmetry with two-penalty interfaces

resultsx Backward-compatible alias of `cv_summary`

cv_summary Data frame with one row per lambda containing mean RMSE and its foldwise standard error

cv_folds Data frame with fold-level RMSE values for each lambda

Lambda Canonical correlations from the final fit

B Estimated reduced-rank coefficient matrix from the final fit

fit Final fit at the selected lambda

`cca_rrr` *Canonical Correlation Analysis via Reduced Rank Regression (RRR)*

Description

Estimates canonical directions using various RRR solvers and penalties.

Usage

```
cca_rrr(
  X,
  Y,
  Sx = NULL,
  Sy = NULL,
  lambda = 0,
  r,
  highdim = TRUE,
  solver = "ADMM",
  LW_Sy = TRUE,
  mode = "sqrtm_norm",
  standardize = FALSE,
  preprocess = NULL,
  rho = 1,
  niter = 10000,
  thresh = 1e-04,
  thresh_0 = 0,
  matrix_free_threshold = 4000L,
  cg_tol = 1e-06,
  cg_maxiter = NULL,
  verbose = FALSE
)
```

Arguments

X	Matrix of predictors.
Y	Matrix of responses.
Sx	Optional X covariance matrix.
Sy	Optional Y covariance matrix.
lambda	Regularization parameter.
r	Rank of the solution.
highdim	Boolean for high-dimensional regime.
solver	Solver type: "rrr", "CVX", or "ADMM". The "CVX" backend requires the optional package CVXR.
LW_Sy	Whether to use Ledoit-Wolf shrinkage for Sy.
mode	Mode for postprocessing the RRR solution. One of "sqrtm_norm" (default) or "product_norm". Legacy aliases "new" and "old" are also accepted. The former whitens the canonical variates to have identity covariance, while the latter does not whiten and instead returns the raw SVD factors of the RRR solution. The "product_norm" mode may be more interpretable in some cases but can yield canonical variates with very different scales and is not guaranteed to be numerically stable when the RRR solution is very low-rank or nearly low-rank.
standardize	Backward-compatible preprocessing flag: TRUE = "scale", FALSE = "center".
preprocess	Preprocessing mode. One of "scale" (center + scale), "center" (center only), or "none". Logical values are still accepted for backward compatibility: TRUE uses standardize, FALSE skips preprocessing.
rho	ADMM parameter.
niter	Maximum number of iterations for ADMM.
thresh	Convergence threshold.
thresh_0	For the ADMM solver: Set entries whose absolute value is below this to 0 (default 1e-6).
matrix_free_threshold	For ADMM: when both n and p are at least this value, use a matrix-free conjugate-gradient solve instead of forming a dense linear system.
cg_tol	Relative tolerance for the matrix-free conjugate-gradient solve used in ADMM.
cg_maxiter	Maximum iterations for the matrix-free conjugate-gradient solve. Defaults to min(p, 1000).
verbose	Logical for verbose output.

Value

A list with elements:

- U: Canonical direction matrix for X (p x r)
- V: Canonical direction matrix for Y (q x r)
- cor: Canonical covariances
- loss: The prediction error 1/n * || XU - YV ||^2

cca_rrr_cv *Cross-validated Canonical Correlation Analysis via RRR*

Description

Performs cross-validation to select optimal lambda, fits CCA_rrr. Canonical Correlation Analysis via Reduced Rank Regression (RRR)

Usage

```
cca_rrr_cv(
  X,
  Y,
  r = 2,
  lambdas = 10^seq(-3, 1.5, length.out = 100),
  kfolds = 10,
  solver = "ADMM",
  mode = "sqrtm_norm",
  parallelize = FALSE,
  LW_Sy = TRUE,
  standardize = FALSE,
  preprocess = NULL,
  cv_metric = "mse",
  rho = 1,
  thresh_0 = 0,
  niter = 10000,
  matrix_free_threshold = 4000L,
  cg_tol = 1e-06,
  cg_maxiter = NULL,
  thresh = 1e-04,
  verbose = FALSE,
  nb_cores = NULL
)
```

Arguments

X	Matrix of predictors.
Y	Matrix of responses.
r	Rank of the solution.
lambdas	Sequence of lambda values for cross-validation.
kfolds	Number of folds for cross-validation.
solver	Solver type: "rrr", "CVX", or "ADMM". The "CVX" backend requires the optional package CVXR.

mode	Mode for postprocessing the RRR solution. One of "sqrtm_norm" (default) or "product_norm". Legacy aliases "new" and "old" are also accepted. The former whitens the canonical variates to have identity covariance, while the latter does not whiten and instead returns the raw SVD factors of the RRR solution. The "product_norm" mode may be more interpretable in some cases but can yield canonical variates with very different scales and is not guaranteed to be numerically stable when the RRR solution is very low-rank or nearly low-rank.
parallelize	Logical; should cross-validation be parallelized?
LW_Sy	Whether to use Ledoit-Wolf shrinkage for Sy.
standardize	Backward-compatible preprocessing flag: TRUE = "scale", FALSE = "center".
preprocess	Preprocessing mode. One of "scale" (center + scale), "center" (center only), or "none". Logical values are still accepted for backward compatibility: TRUE uses standardize, FALSE skips preprocessing. Data are preprocessed once up front, and fold fits reuse those transformed matrices without re-centering or re-scaling inside each fold.
cv_metric	Cross-validation metric. Use "mse" to minimize held-out prediction error or "correlation" to maximize held-out association between X %*% U and Y %*% V.
rho	ADMM parameter.
thresh_0	tolerance for declaring entries non-zero
niter	Maximum number of iterations for ADMM.
matrix_free_threshold	For ADMM: when both n and p are at least this value, use a matrix-free conjugate-gradient solve instead of forming a dense linear system.
cg_tol	Relative tolerance for the matrix-free conjugate-gradient solve used in ADMM.
cg_maxiter	Maximum iterations for the matrix-free conjugate-gradient solve. Defaults to min(p, 1000).
thresh	Convergence threshold.
verbose	Logical for verbose output.
nb_cores	Number of cores to use for parallelization. Defaults to min(kfolds, available cores minus 1).

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

lambda Optimal regularisation parameter lambda chosen by CV

rmse Backward-compatible optimization objective. For cv_metric = "mse" this is the held-out mean squared error; for cv_metric = "correlation" it is -cv_score so that smaller still means better.

cv_score Raw held-out cross-validation score averaged across folds.

cv_metric The metric used to score lambdas during cross-validation.

cor Canonical correlations at the selected lambda

lambda_x Alias of the selected `lambda`

lambda_x_se Foldwise standard error at the selected `lambda`

lambda_y Placeholder for symmetry with two-penalty interfaces

lambda_y_se Placeholder for symmetry with two-penalty interfaces

resultsx Backward-compatible alias of `cv_summary`

cv_summary Data frame with one row per lambda containing the mean CV score and its foldwise standard error.

cv_folds Data frame with fold-level CV scores for each lambda.

Lambda Canonical correlations from the final fit

B Estimated reduced-rank coefficient matrix from the final fit

fit Final fit at the selected lambda

`ecca` *Efficient CCA for Two High-Dimensional Views*

Description

Fits sparse canonical directions with an ADMM-based reduced-rank regression formulation tailored to the setting where both views are high-dimensional.

Usage

```
ecca(
  X,
  Y,
  lambda = 0,
  groups = NULL,
  r = 2,
  standardize = FALSE,
  rho = 1,
  B0 = NULL,
  eps = 1e-04,
  maxiter = 500,
  verbose = TRUE,
  epsilon_sv = 1e-08,
  ridge_whiten = 1e-08
)
```

Arguments

X	Predictor matrix (n x p).
Y	Response matrix (n x q).
lambda	Regularization parameter.
groups	Optional group structure for blockwise sparsity.
r	Target rank.
standardize	Whether to scale variables after centering.
rho	ADMM penalty parameter.
B0	Optional warm start for the coefficient matrix.
eps	Convergence tolerance for ADMM.
maxiter	Maximum number of ADMM iterations.
verbose	Whether to print diagnostics.
epsilon_sv	Numerical threshold used to discard near-zero singular values.
ridge_whiten	Ridge added when whitening Gram matrices.

Value

A list containing the estimated canonical directions, canonical correlations, the fitted coefficient matrix, preprocessing metadata, and convergence information.

---

ecca.cv *Cross-Validated Efficient CCA*

Description

Selects a regularization parameter for `ecca()` by cross-validation and refits the final model at the selected value.

Usage

```
ecca.cv(
  X,
  Y,
  lambdas = 0,
  groups = NULL,
  r = 2,
  standardize = FALSE,
  rho = 1,
  B0 = NULL,
  nfold = 5,
  select = "lambda.min",
  eps = 0.001,
  maxiter = 1000,
```

```
  verbose = FALSE,
  maxiter_cv = 300,
  parallel = FALSE,
  nb_cores = NULL,
  set_seed_cv = NULL,
  scoring_method = c("mse", "trace"),
  cv_use_median = FALSE,
  dense = TRUE,
  optimized = FALSE,
  epsilon_sv = 1e-08,
  ridge_whiten = 1e-08
)
```

Arguments

X	Predictor matrix (n x p).
Y	Response matrix (n x q).
lambdas	Candidate regularization values.
groups	Optional group structure for blockwise sparsity.
r	Target rank.
standardize	Whether to scale variables after centering.
rho	ADMM penalty parameter.
B0	Optional warm start for the coefficient matrix.
nfold	Number of cross-validation folds.
select	Selection rule for the final lambda. One of "lambda.min" or "lambda.1se".
eps	Convergence tolerance for the final ADMM refit.
maxiter	Maximum iterations for the final ADMM refit.
verbose	Whether to print diagnostics.
maxiter_cv	Maximum iterations used inside the cross-validation fits.
parallel	Whether to parallelize cross-validation.
nb_cores	Number of worker processes to use when parallel = TRUE.
set_seed_cv	Optional random seed for fold generation.
scoring_method	Cross-validation score to optimize. One of "mse" or "trace".
cv_use_median	Whether to aggregate fold scores with the median instead of the mean.
dense	Retained for backward compatibility.
optimized	Retained for backward compatibility.
epsilon_sv	Numerical threshold used to discard near-zero singular values.
ridge_whiten	Ridge added when whitening Gram matrices.

Value

A list with the final fit, selected lambda, and cross-validation scores when more than one lambda is supplied.

FPR *False Positive Rate (TPR)*

Description

This is a function that compares the structure of two matrices A and B. It outputs the number of entries where A is not zero but Bis. A and B need to have the same number of rows and columns

Usage

```
FPR(A, B, tol = 1e-04)
```

Arguments

A	A matrix.
B	A matrix (assumed to be the ground truth).
tol	tolerance for declaring the entries non zero.

Value

False Positive Rate (nb of values that are non zero in A and zero in B / (nb of values that are non zero in A))

Examples

```
A <- matrix(c(1, 0, 0, 1, 1, 0), nrow = 2)
B <- matrix(c(1, 0, 1, 1, 0, 0), nrow = 2)
FPR(A, B)
```

get_edge_incidence *Return the edge incidence matrix of an igraph graph*

Description

Return the edge incidence matrix of an igraph graph

Usage

```
get_edge_incidence(g, weight = 1)
```

Arguments

g	igraph graph object.
weight	edge weights.

Value

Edge incidence matrix of the graph g, with +weight for the source node and -weight for the target node.

`principal_angles` *Metrics for subspaces*

Description

Calculate principal angles between subspace spanned by the columns of a and the subspace spanned by the columns of b

Usage

```
principal_angles(a, b)
```

Arguments

a	A matrix whose columns span a subspace.
b	A matrix whose columns span a subspace.

Value

a vector of principal angles (in radians)

Examples

```
a <- matrix(rnorm(9), 3, 3)
b <- matrix(rnorm(9), 3, 3)
principal_angles(a, b)
```

`regular_cca` *Function to perform regular (low dimensional) canonical correlation analysis (CCA*

Description

Function to perform regular (low dimensional) canonical correlation analysis (CCA

Usage

```
regular_cca(X, Y, rank)
```

Arguments

X	Matrix of predictors (n x p)
Y	Matrix of responses (n x q)
rank	Number of canonical components to extract

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

cor Canonical covariances

SCCA_Parkhomenko *Function to perform Sparse CCA based on Waaijenborg et al. (2008) REFERENCE Parkhomenko et al. (2009), "Sparse Canonical Correlation Anlaysis with Application to Genomic Data Integration" in Statistical Applications in Genetics and Molecular Biology, Volume 8, Issue 1, Article 1*

Description

Function to perform Sparse CCA based on Waaijenborg et al. (2008) REFERENCE Parkhomenko et al. (2009), "Sparse Canonical Correlation Anlaysis with Application to Genomic Data Integration" in Statistical Applications in Genetics and Molecular Biology, Volume 8, Issue 1, Article 1

Usage

```
SCCA_Parkhomenko(
  x.data,
  y.data,
  n.cv = 5,
  lambda.v.seq = seq(0, 0.2, by = 0.02),
  lambda.u.seq = seq(0, 0.2, by = 0.02),
  Krank = 1,
  standardize = TRUE
)
```

Arguments

x.data	Matrix of predictors (n x p)
y.data	Matrix of responses (n x q)
n.cv	Number of cross-validation folds (default is 5)
lambda.v.seq	Vector of sparsity parameters for Y (default is a sequence from 0 to 1 with step 0.1)

`lambda.u.seq`	Vector of sparsity parameters for X (default is a sequence from 0 to 1 with step 0.1)
`Krank`	Number of canonical components to extract
`standardize`	Standardize (center and scale) the data matrices X and Y (default is TRUE) before analysis

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

cor Canonical correlations

`sinTheta` *SinTheta distance between subspaces*

Description

Calculate the distance spanned by the columns of A and the subspace spanned by the columns of B, defined as ||UU^T - VV^T||_F / sqrt(2)

Usage

```
sinTheta(U, V)
```

Arguments

`U`	A matrix whose columns span a subspace.
`V`	A matrix whose columns span a subspace.

Value

sinTheta distance between the two subspaces spanned by the matrices A and B, defined as ||UU^T - VV^T||_F / sqrt(2)

SparseCCA *Function to perform Sparse CCA based on Wilms and Croux (2018) REFERENCE Wilms, I., & Croux, C. (2018). Sparse canonical correlation analysis using alternating regressions. Journal of Computational and Graphical Statistics, 27(1), 1-10.*

Description

Function to perform Sparse CCA based on Wilms and Croux (2018) REFERENCE Wilms, I., & Croux, C. (2018). Sparse canonical correlation analysis using alternating regressions. Journal of Computational and Graphical Statistics, 27(1), 1-10.

Usage

```
SparseCCA(
  X,
  Y,
  lambdaAseq = seq(from = 1, to = 0.01, by = -0.01),
  lambdaBseq = seq(from = 1, to = 0.01, by = -0.01),
  rank,
  selection.criterion = 1,
  n.cv = 5,
  A.initial = NULL,
  B.initial = NULL,
  max.iter = 20,
  conv = 10^-2,
  standardize = TRUE
)
```

Arguments

X	Matrix of predictors (n x p)
Y	Matrix of responses (n x q)
lambdaAseq	Vector of sparsity parameters for X (default is a sequence from 0 to 1 with step 0.1)
lambdaBseq	Vector of sparsity parameters for Y (default is a sequence from 0 to 1 with step 0.1)
rank	Number of canonical components to extract
selection.criterion	Criterion for selecting the optimal tuning parameter (1 for minimizing difference between test and training sample correlation, 2 for maximizing test sample correlation)
n.cv	Number of cross-validation folds (default is 5)
A.initial	Initial value for the canonical vector A (default is NULL, which uses a canonical ridge solution)

B.initial	Initial value for the canonical vector B (default is NULL, which uses a canonical ridge solution)
max.iter	Maximum number of iterations for convergence (default is 20)
conv	Convergence threshold (default is 1e-2)
standardize	Standardize (center and scale) the data matrices X and Y (default is TRUE) before analysis

Value

A list with elements:

U Canonical direction matrix for X (p x r)

V Canonical direction matrix for Y (q x r)

loss Mean squared error of prediction

cor Canonical covariances

---

sparse_CCA_benchmarks *Additional Benchmarks for Sparse CCA Methods*

Description

Additional Benchmarks for Sparse CCA Methods

Usage

```
sparse_CCA_benchmarks(
  X_train,
  Y_train,
  S = NULL,
  rank = 2,
  kfolds = 5,
  method.type = "FIT_SAR_CV",
  lambdax = 10^seq(from = -3, to = 2, length = 10),
  lambday = c(0, 1e-07, 1e-06, 1e-05),
  standardize = TRUE
)
```

Arguments

X_train	Matrix of predictors (n x p)
Y_train	Matrix of responses (n x q)
S	Optional covariance matrix (default is NULL, which computes it from X_train and Y_train)
rank	Target rank for the CCA (default is 2)
kfolds	Number of cross-validation folds (default is 5)

method.type	Type of method to use for Sparse CCA (default is "FIT_SAR_CV"). Choices include "FIT_SAR_BIC", "FIT_SAR_CV", "Witten_Perm", "Witten.CV", and "SCCA_Parkhomenko".
lambdax	Vector of sparsity parameters for X (default is a sequence from 0 to 1 with step 0.1)
lambday	Vector of sparsity parameters for Y (default is a sequence from 0 to 1 with step 0.1)
standardize	Standardize (center and scale) the data matrices X and Y (default is TRUE) before analysis

Value

A matrix (p+q)x r containing the canonical directions for X and Y.

---

subdistance *Subdistance between subspaces*

Description

Calculate subdistance between subspace spanned by the columns of a and the subspace spanned by the columns of b

Usage

```
subdistance(A, B)
```

Arguments

A	A matrix whose columns span a subspace.
B	A matrix whose columns span a subspace.

Value

subdistance between the two subspaces spanned by the matrices A and B, defined as min(O orthogonal) ||AO-B||_F

TNR *True Negative Rate (TNR)*

Description

This is a function that compares the structure of two matrices A and B. It outputs the number of entries where A and B are both 0. A and B need to have the same number of rows and columns

Usage

```
TNR(A, B, tol = 1e-04)
```

Arguments

A	A matrix.
B	A matrix (assumed to be the ground truth)..
tol	tolerance for declaring the entries non zero.

Value

True Negative Rate (nb of values that are zero in A and zero in B / (nb of values that are zero in A))

TPR *True Positive Rate (TPR)*

Description

This is a function that compares the structure of two matrices A and B. It outputs the number of entries that A and B have in common that are different from zero. A and B need to have the same number of rows and columns

Usage

```
TPR(A, B, tol = 1e-04)
```

Arguments

A	A matrix (the estimator).
B	A matrix (assumed to be the ground truth).
tol	tolerance for declaring the entries non zero.

Value

True Positive Rate (nb of values that are non zero in both A and B / (nb of values that are non zero in A))

Examples

```
A <- matrix(c(1, 0, 0, 1, 1, 0), nrow = 2)
B <- matrix(c(1, 0, 1, 1, 0, 0), nrow = 2)
TPR(A, B)
```

`Witten.CV` *Sparse CCA by Witten and Tibshirani (2009)*

Description

Sparse CCA by Witten and Tibshirani (2009)

Usage

```
Witten.CV(
  X,
  Y,
  n.cv = 5,
  rank,
  lambdax = matrix(seq(from = 0, to = 1, by = 0.1), nrow = 1),
  lambday = matrix(seq(from = 0, to = 1, by = 0.1), nrow = 1),
  standardize = TRUE
)
```

Arguments

`X`	Matrix of predictors (n x p)
`Y`	Matrix of responses (n x q)
`n.cv`	Number of cross-validation folds (default is 5)
`rank`	Number of canonical components to extract
`lambdax`	Vector of sparsity parameters for X (default is a sequence from 0 to 1 with step 0.1)
`lambday`	Vector of sparsity parameters for Y (default is a sequence from 0 to 1 with step 0.1)
`standardize`	Standardize (center and scale) the data matrices X and Y (default is TRUE) before analysis

Value

the appropriate levels of regularisation

Index