Package 'PMA'

February 7, 2024

Type Package

Title Penalized Multivariate Analysis

Version 1.2-3

Date 2024-02-06

URL https://github.com/bnaras/PMA

BugReports https://github.com/bnaras/PMA/issues

Description Performs Penalized Multivariate Analysis: a penalized matrix decomposition, sparse principal components analysis, and sparse canonical correlation analysis, described in Witten, Tibshirani and Hastie (2009) <doi:10.1093/biostatistics/kxp008> and Witten and Tibshirani (2009) Extensions of sparse canonical correlation analysis, with applications to genomic data <doi:10.2202/1544-6115.1470>.

Depends R (>= 2.10)

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.3

Imports utils

NeedsCompilation yes

Author Daniela Witten [aut],
Rob Tibshirani [aut],
Sam Gross [aut],
Balasubramanian Narasimhan [cre, aut]

Maintainer Balasubramanian Narasimhan <naras@stanford.edu>

Repository CRAN

Date/Publication 2024-02-06 23:10:02 UTC

R topics documented:

`PMA-package` *Penalized Multivariate Analysis*

Description

This package is called **PMA**, for __P__enalized __M__ultivariate __A__nalysis. It implements three methods: A penalized matrix decomposition, sparse principal components analysis, and sparse canonical correlations analysis. All are described in the reference below. The main functions are: `PMD`, `CCA` and `SPC`.

Details

The first, `PMD`, performs a penalized matrix decomposition. `CCA` performs sparse canonical correlation analysis. `SPC` performs sparse principal components analysis.

There also are cross-validation functions for tuning parameter selection for each of the above methods: `SPC.cv`, `PMD.cv`, `CCA.permute`. And `PlotCGH` produces nice plots for DNA copy number data.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) doi:10.1093/biostatistics/kxp008.

CCA *Perform sparse canonical correlation analysis using the penalized matrix decomposition.*

Description

Given matrices X and Z, which represent two sets of features on the same set of samples, find sparse u and v such that u'X'Zv is large. For X and Z, the samples are on the rows and the features are on the columns. X and Z must have same number of rows, but may (and usually will) have different numbers of columns. The columns of X and/or Z can be unordered or ordered. If unordered, then a lasso penalty will be used to obtain the corresponding canonical vector. If ordered, then a fused lasso penalty will be used; this will result in smoothness.

Usage

```
CCA(
  x,
  z,
  typex = c("standard", "ordered"),
  typez = c("standard", "ordered"),
  penaltyx = NULL,
  penaltyz = NULL,
  K = 1,
  niter = 15,
  v = NULL,
  trace = TRUE,
  standardize = TRUE,
  xnames = colnames(x),
  znames = colnames(z),
  chromx = NULL,
  chromz = NULL,
  upos = FALSE,
  uneg = FALSE,
  vpos = FALSE,
  vneg = FALSE,
  outcome = NULL,
  y = NULL,
  cens = NULL
)
```

Arguments

x	Data matrix; samples are rows and columns are features. Cannot contain missing values.
z	Data matrix; samples are rows and columns are features. Cannot contain missing values.

typex	Are the columns of x unordered (type="standard") or ordered (type="ordered")? If "standard", then a lasso penalty is applied to u, to enforce sparsity. If "ordered" (generally used for CGH data), then a fused lasso penalty is applied, to enforce both sparsity and smoothness.
typez	Are the columns of z unordered (type="standard") or ordered (type="ordered")? If "standard", then a lasso penalty is applied to v, to enforce sparsity. If "ordered" (generally used for CGH data), then a fused lasso penalty is applied, to enforce both sparsity and smoothness.
penaltyx	The penalty to be applied to the matrix x, i.e. the penalty that results in the canonical vector u. If typex is "standard" then the L1 bound on u is penaltyx*sqrt(ncol(x)). In this case penaltyx must be between 0 and 1 (larger L1 bound corresponds to less penalization). If "ordered" then it's the fused lasso penalty lambda, which must be non-negative (larger lambda corresponds to more penalization).
penaltyz	The penalty to be applied to the matrix z, i.e. the penalty that results in the canonical vector v. If typez is "standard" then the L1 bound on v is penaltyz*sqrt(ncol(z)). In this case penaltyz must be between 0 and 1 (larger L1 bound corresponds to less penalization). If "ordered" then it's the fused lasso penalty lambda, which must be non-negative (larger lambda corresponds to more penalization).
K	The number of u's and v's desired; that is, the number of canonical vectors to be obtained.
niter	How many iterations should be performed? Default is 15.
v	The first K columns of the v matrix of the SVD of X'Z. If NULL, then the SVD of X'Z will be computed inside the CCA function. However, if you plan to run this function multiple times, then save a copy of this argument so that it does not need to be re-computed (since that process can be time-consuming if X and Z both have high dimension).
trace	Print out progress?
standardize	Should the columns of x and z be centered (to have mean zero) and scaled (to have standard deviation 1)? Default is TRUE.
xnames	An optional vector of column names for x, defaults to `colnames(x)`
znames	An optional vector of column names for z, defaults to `colnames(z)`
chromx	Used only if typex is "ordered"; allows user to specify a vector of length ncol(x) giving the chromosomal location of each CGH spot. This is so that smoothness will be enforced within each chromosome, but not between chromosomes.
chromz	Used only if typez is "ordered"; allows user to specify a vector of length ncol(z) giving the chromosomal location of each CGH spot. This is so that smoothness will be enforced within each chromosome, but not between chromosomes.
upos	If TRUE, then require elements of u to be positive. FALSE by default. Can only be used if type is "standard".
uneg	If TRUE, then require elements of u to be negative. FALSE by default. Can only be used if type is "standard".
vpos	If TRUE, require elements of v to be positive. FALSE by default. Can only be used if type is "standard".

vneg — If TRUE, require elements of v to be negative. FALSE by default. Can only be used if type is "standard".

outcome — If you would like to incorporate a phenotype into CCA analysis - that is, you wish to find features that are correlated across the two data sets and also correlated with a phenotype - then use one of "survival", "multiclass", or "quantitative" to indicate outcome type. Default is NULL.

y — If outcome is not NULL, then this is a vector of phenotypes - one for each row of x and z. If outcome is "survival" then these are survival times; must be non-negative. If outcome is "multiclass" then these are class labels (1,2,3,...). Default NULL.

cens — If outcome is "survival" then these are censoring statuses for each observation. 1 is complete, 0 is censored. Default NULL.

Details

This function is useful for performing an integrative analysis of two sets of measurements taken on the same set of samples: for instance, gene expression and CGH measurements on the same set of patients. It takes in two data sets, called x and z, each of which have (the same set of) samples on the rows. If z is a matrix of CGH data with *ordered* CGH spots on the columns, then use typez="ordered". If z consists of unordered columns, then use typez="standard". Similarly for typex.

This function performs the penalized matrix decomposition on the data matrix $X'Z$. Therefore, the results should be the same as running the PMD function on t(x)\ using the CCA function is much faster because it avoids computation of $X'Z$.

The CCA criterion is as follows: find unit vectors u and v such that $u'X'Zv$ is maximized subject to constraints on u and v. If typex="standard" and typez="standard" then the constraints on u and v are lasso (L_1). If typex="ordered" then the constraint on u is a fused lasso penalty (promoting sparsity and smoothness). Similarly if typez="ordered".

When type x is "standard": the L1 bound of u is penaltyx*sqrt(ncol(x)).

When typex is "ordered": penaltyx controls the amount of sparsity and smoothness in u, via the fused lasso penalty: $lambda sum_j |u_j| + lambda sum_j |u_j - u_(j-1)|$. If NULL, then it will be chosen adaptively from the data.

Value

u — u is output. If you asked for multiple factors then each column of u is a factor. u has dimension nxK if you asked for K factors.

v — v is output. If you asked for multiple factors then each column of v is a factor. v has dimension pxK if you asked for K factors.

d — A vector of length K, which can alternatively be computed as the diagonal of the matrix $u'X'Zv$.

v.init — The first K factors of the v matrix of the SVD of x'z. This is saved in case this function will be re-run later.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

PMD,CCA.permute

Examples

```
# first, do CCA with type="standard"
# A simple simulated example
set.seed(3189)
u <- matrix(c(rep(1,25),rep(0,75)),ncol=1)
v1 <- matrix(c(rep(1,50),rep(0,450)),ncol=1)
v2 <- matrix(c(rep(0,50),rep(1,50),rep(0,900)),ncol=1)
x <- u%*%t(v1) + matrix(rnorm(100*500),ncol=500)
z <- u%*%t(v2) + matrix(rnorm(100*1000),ncol=1000)
# Can run CCA with default settings, and can get e.g. 3 components
out <- CCA(x,z,typex="standard",typez="standard",K=3)
print(out,verbose=TRUE) # To get less output, just print(out)
# Or can use CCA.permute to choose optimal parameter values
perm.out <- CCA.permute(x,z,typex="standard",typez="standard",nperms=7)
print(perm.out)
plot(perm.out)
out <- CCA(x,z,typex="standard",typez="standard",K=1,
   penaltyx=perm.out$bestpenaltyx,penaltyz=perm.out$bestpenaltyz,
   v=perm.out$v.init)
print(out)


##### The remaining examples are commented out, but uncomment to run: ######

# Not run, to save time:
## Not run:
## Now try CCA with a constraint that elements of u must be negative and
## elements of v must be positive:
perm.out <- CCA.permute(x,z,typex="standard",typez="standard",nperms=7,
penaltyxs=seq(.1,.7,len=10), penaltyzs=seq(.1,.7,len=10), uneg=TRUE, vpos=TRUE)
print(perm.out)
plot(perm.out)
out <- CCA(x,z,typex="standard",typez="standard",K=1,
   penaltyx=perm.out$bestpenaltyx,penaltyz=perm.out$bestpenaltyz,
            v=perm.out$v.init, uneg=TRUE, vpos=TRUE)
print(out)


## Suppose we also have a quantitative outcome, y, and we want to find
## features in x and z that are correlated with each other and with the
```

```
## outcome:
y <- rnorm(nrow(x))
perm.out <- CCA.permute(x,z,typex="standard",typez="standard",
outcome="quantitative",y=y, nperms=6)
print(perm.out)
out<-CCA(x,z,typex="standard",typez="standard",outcome="quantitative",
 y=y,penaltyx=perm.out$bestpenaltyx,penaltyz=perm.out$bestpenaltyz)
print(out)

## now, do CCA with type="ordered"
## Example involving the breast cancer data: gene expression + CGH
set.seed(22)
breastdata <- download_breast_data()
with(breastdata, {
dna <- t(dna)
rna <- t(rna)
perm.out <- CCA.permute(x=rna,z=dna[,chrom==1],typex="standard",
        typez="ordered",nperms=5,penaltyxs=seq(.02,.7,len=10))
## We run CCA using all gene exp. data, but CGH data on chrom 1 only.
print(perm.out)
plot(perm.out)
out <- CCA(x=rna,z=dna[,chrom==1], typex="standard", typez="ordered",
   penaltyx=perm.out$bestpenaltyx,
           v=perm.out$v.init, penaltyz=perm.out$bestpenaltyz,
           xnames=substr(genedesc,1,20),
           znames=paste("Pos", sep="", nuc[chrom==1]))
# Save time by inputting  lambda and v
print(out) # could do print(out,verbose=TRUE)
print(genechr[out$u!=0]) # Cool! The genes associated w/ gain or loss
## on chrom 1 are located on chrom 1!!
par(mfrow=c(1,1))
PlotCGH(out$v, nuc=nuc[chrom==1], chrom=chrom[chrom==1],
main="Regions of gain/loss on Chrom 1 assoc'd with gene expression")
} )

## End(Not run)
```

`CCA.permute` *Select tuning parameters for sparse canonical correlation analysis using the penalized matrix decomposition.*

Description

This function can be used to automatically select tuning parameters for sparse CCA using the penalized matrix decompostion. For each data set x and z, two types are possible: (1) type "standard", which does not assume any ordering of the columns of the data set, and (2) type "ordered", which assumes that columns of the data set are ordered and thus that corresponding canonical vector should be both sparse and smooth (e.g. CGH data).

Usage

```
CCA.permute(
  x,
  z,
  typex = c("standard", "ordered"),
  typez = c("standard", "ordered"),
  penaltyxs = NULL,
  penaltyzs = NULL,
  niter = 3,
  v = NULL,
  trace = TRUE,
  nperms = 25,
  standardize = TRUE,
  chromx = NULL,
  chromz = NULL,
  upos = FALSE,
  uneg = FALSE,
  vpos = FALSE,
  vneg = FALSE,
  outcome = NULL,
  y = NULL,
  cens = NULL
)
```

Arguments

x	Data matrix; samples are rows and columns are features.
z	Data matrix; samples are rows and columns are features. Note that x and z must have the same number of rows, but may (and generally will) have different numbers of columns.
typex	Are the columns of x unordered (type="standard") or ordered (type="ordered")? If "standard", then a lasso penalty is applied to v, to enforce sparsity. If "ordered" (generally used for CGH data), then a fused lasso penalty is applied, to enforce both sparsity and smoothness.
typez	Are the columns of z unordered (type="standard") or ordered (type="ordered")? If "standard", then a lasso penalty is applied to v, to enforce sparsity. If "ordered" (generally used for CGH data), then a fused lasso penalty is applied, to enforce both sparsity and smoothness.
penaltyxs	The set of x penalties to be considered. If typex="standard", then the L1 bound on u is penaltyxs*sqrt(ncol(x)). If "ordered", then it's the lambda for the fused lasso penalty. The user can specify a single value or a vector of values. If penaltyxs is a vector and penaltyzs is a vector, then the vectors must have the same length. If NULL, then the software will automatically choose a single lambda value if type is "ordered", or a grid of (L1 bounds)/sqrt(ncol(x)) if type is "standard".
penaltyzs	The set of z penalties to be considered. If typez="standard", then the L1 bound on v is penaltyzs*sqrt(ncol(z)). If "ordered", then it's the lambda for the fused

lasso penalty. The user can specify a single value or a vector of values. If penaltyzs is a vector and penaltyzs is a vector, then the vectors must have the same length. If NULL, then the software will automatically choose a single lambda value if type is "ordered", or a grid of (L1 bounds)/sqrt(ncol(z)) if type is "standard".

`niter` How many iterations should be performed each time CCA is called? Default is 3, since an approximate estimate of u and v is acceptable in this case, and otherwise this function can be quite time-consuming.

`v` The first K columns of the v matrix of the SVD of X'Z. If NULL, then the SVD of X'Z will be computed inside this function. However, if you plan to run this function multiple times, then save a copy of this argument so that it does not need to be re-computed (since that process can be time-consuming if X and Z both have high dimension).

`trace` Print out progress?

`nperms` How many times should the data be permuted? Default is 25. A large value of nperms is very important here, since the formula for computing the z-statistics requires a standard deviation estimate for the correlations obtained via permutation, which will not be accurate if nperms is very small.

`standardize` Should the columns of X and Z be centered (to have mean zero) and scaled (to have standard deviation 1)? Default is TRUE.

`chromx` Used only if typex="ordered"; a vector of length ncol(x) that allows you to specify which chromosome each CGH spot is on. If NULL, then it is assumed that all CGH spots are on same chromosome.

`chromz` Used only if typex="ordered"; a vector of length ncol(z) that allows you to specify which chromosome each CGH spot is on. If NULL, then it is assumed that all CGH spots are on same chromosome.

`upos` If TRUE, then require all elements of u to be positive in sign. Default is FALSE. Can only be used if type is standard.

`uneg` If TRUE, then require all elements of u to be negative in sign. Default is FALSE. Can only be used if type is standard.

`vpos` If TRUE, then require all elements of v to be positive in sign. Default is FALSE. Can only be used if type is standard.

`vneg` If TRUE, then require all elements of v to be negative in sign. Default is FALSE. Can only be used if type is standard.

`outcome` If you would like to incorporate a phenotype into CCA analysis - that is, you wish to find features that are correlated across the two data sets and also correlated with a phenotype - then use one of "survival", "multiclass", or "quantitative" to indicate outcome type. Default is NULL.

`y` If outcome is not NULL, then this is a vector of phenotypes - one for each row of x and z. If outcome is "survival" then these are survival times; must be non-negative. If outcome is "multiclass" then these are class labels. Default NULL.

`cens` If outcome is "survival" then these are censoring statuses for each observation. 1 is complete, 0 is censored. Default NULL.

Details

For X and Z, the samples are on the rows and the features are on the columns.

The tuning parameters are selected using a permutation scheme. For each candidate tuning parameter value, the following is performed: (1) The samples in X are randomly permuted nperms times, to obtain matrices $X*_1,X*_2,...$. (2) Sparse CCA is run on each permuted data set $(X*_i,Z)$ to obtain factors $(u*_i, v*_i)$. (3) Sparse CCA is run on the original data (X,Z) to obtain factors u and v. (4) Compute $c*_i=cor(X*_i u*_i,Z v*_i)$ and $c=cor(Xu,Zv)$. (5) Use Fisher's transformation to convert these correlations into random variables that are approximately normally distributed. Let Fisher(c) denote the Fisher transformation of c. (6) Compute a z-statistic for Fisher(c), using $(Fisher(c)-mean(Fisher(c*)))/sd(Fisher(c*))$. The larger the z-statistic, the "better" the corresponding tuning parameter value.

This function also gives the p-value for each pair of canonical variates (u,v) resulting from a given tuning parameter value. This p-value is computed as the fraction of $c*_i$'s that exceed c (using the notation of the previous paragraph).

Using this function, only the first left and right canonical variates are considered in selection of the tuning parameter.

Note that x and z must have same number of rows. This function performs just a one-dimensional search in tuning parameter space, even if penaltyxs and penaltyzs both are vectors: the pairs `(penaltyxs[1],penaltyzs[1])`, `(penaltyxs[2],penaltyzs[2])`,.... are considered.

Value

`zstat`	The vector of z-statistics, one per element of sumabss.
`pvals`	The vector of p-values, one per element of sumabss.
`bestpenaltyx`	The x penalty that resulted in the highest z-statistic.
`bestpenaltyz`	The z penalty that resulted in the highest z-statistic.
`cors`	The value of cor(Xu,Zv) obtained for each value of sumabss.
`corperms`	The nperms values of cor(X*u*,Zv*) obtained for each value of sumabss, where X* indicates the X matrix with permuted rows, and u* and v* are the output of CCA using data (X*,Z).
`ft.cors`	The result of applying Fisher transformation to cors.
`ft.corperms`	The result of applying Fisher transformation to corperms.
`nnonzerous`	Number of non-zero u's resulting from applying CCA to data (X,Z) for each value of sumabss.
`nnonzerouv`	Number of non-zero v's resulting from applying CCA to data (X,Z) for each value of sumabss.
`v.init`	The first factor of the v matrix of the SVD of x'z. This is saved in case this function (or the CCA function) will be re-run later.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

PMD,CCA

Examples

```
# See examples in CCA function
```

`download_breast_data` *Download and return the breast data*

Description

Breast cancer gene expression + DNA copy number data set from Chin et. al. and used in Witten, et. al. See references below.

This data set consists of gene expression and DNA copy number measurements on a set of 89 samples. The data set can be used to perform integrative analysis of gene expression and DNA copy number data, as in . That is, we can look for sets of genes that are associated with regions of chromosomal gain/loss.

Missing values were imputed using 5-nearest neighbors (see library `pamr`).

Usage

```
download_breast_data(url = "https://tibshirani.su.domains/PMA/breastdata.rda")
```

Arguments

`url`	source, default `"https://tibshirani.su.domains/PMA/breastdata.rda"`

Value

a list containing the following elements:

- `dna`: a 2149x89 matrix of CGH spots x Samples
- `rna`: a 19672x89 matrix of Genes x Samples
- `chrom`: a 2149-vector of chromosomal location of each CGH spot
- `nuc`: a 2149-vector of nucleotide position for each CGH spot
- `gene`: a 19672-vector wiith an accession number for each gene
- `genenames`: a 19672-vector with a name for each gene
- `genechr`: a 19672-vector with a chromosomal location for each gene
- `genedesc`: a 19672-vector with a description for each gene
- `genepos`: a 19672-vector with a nucleotide position for each gene.

References

Chin K., et. al. (2006) doi:10.1016/j.ccr.2006.10.009.

Witten D. M., Tibshirani R., and Hastie, T. (2009) doi:10.1093/biostatistics/kxp008.

`MultiCCA` *Perform sparse multiple canonical correlation analysis.*

Description

Given matrices $X1,...,XK$, which represent K sets of features on the same set of samples, find sparse $w1,...,wK$ such that $sum_(i<j) (wi' Xi' Xj wj)$ is large. If the columns of Xk are ordered (and type="ordered") then wk will also be smooth. For $X1,...,XK$, the samples are on the rows and the features are on the columns. $X1,...,XK$ must have same number of rows, but may (and usually will) have different numbers of columns.

Usage

```
MultiCCA(
  xlist,
  penalty = NULL,
  ws = NULL,
  niter = 25,
  type = "standard",
  ncomponents = 1,
  trace = TRUE,
  standardize = TRUE
)
```

Arguments

`xlist`	A list of length K, where K is the number of data sets on which to perform sparse multiple CCA. Data set k should be a matrix of dimension $n x p_k$ where p_k is the number of features in data set k.
`penalty`	The penalty terms to be used. Can be a single value (if the same penalty term is to be applied to each data set) or a K-vector, indicating a different penalty term for each data set. There are 2 possible interpretations for the penalty terms: If type="standard" then this is an L1 bound on wk, and it must be between 1 and $sqrt(p_k)$ (p_k is the number of features in matrix Xk). If type="ordered" then this is the parameter for the fused lasso penalty on wk.
`ws`	A list of length K. The kth element contains the first ncomponents columns of the v matrix of the SVD of Xk. If NULL, then the SVD of $X1,...,XK$ will be computed inside the MultiCCA function. However, if you plan to run this function multiple times, then save a copy of this argument so that it does not need to be re-computed.
`niter`	How many iterations should be performed? Default is 25.

type	Are the columns of $x1,...,xK$ unordered (type="standard") or ordered (type="ordered")? If "standard", then a lasso penalty is applied to v, to enforce sparsity. If "ordered" (generally used for CGH data), then a fused lasso penalty is applied, to enforce both sparsity and smoothness. This argument can be a vector of length K (if different data sets are of different types) or it can be a single value "ordered"/"standard" (if all data sets are of the same type).
ncomponents	How many factors do you want? Default is 1.
trace	Print out progress?
standardize	Should the columns of $X1,...,XK$ be centered (to have mean zero) and scaled (to have standard deviation 1)? Default is TRUE.

Value

ws	A list of length K, containg the sparse canonical variates found (element k is a $p_k x ncomponents$ matrix).
ws.init	A list of length K containing the initial values of ws used, by default these are the v vector of the svd of matrix Xk.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

MultiCCA.permute,CCA, CCA.permute

Examples

```
# Generate 3 data sets so that first 25 features are correlated across
# the data sets...
set.seed(123)
u <- matrix(rnorm(50),ncol=1)
v1 <- matrix(c(rep(.5,25),rep(0,75)),ncol=1)
v2 <- matrix(c(rep(1,25),rep(0,25)),ncol=1)
v3 <- matrix(c(rep(.5,25),rep(0,175)),ncol=1)

x1 <- u%*%t(v1) + matrix(rnorm(50*100),ncol=100)
x2 <- u%*%t(v2) + matrix(rnorm(50*50),ncol=50)
x3 <- u%*%t(v3) + matrix(rnorm(50*200),ncol=200)

xlist <- list(x1, x2, x3)

# Run MultiCCA.permute w/o specifying values of tuning parameters to
# try.
# The function will choose the lambda for the ordered data set.
# Then permutations will be used to select optimal sum(abs(w)) for
```

```
# standard data sets.
# We assume that x1 is standard, x2 is ordered, x3 is standard:
perm.out <- MultiCCA.permute(xlist, type=c("standard", "ordered",
"standard"))
print(perm.out)
plot(perm.out)
out <- MultiCCA(xlist, type=c("standard", "ordered", "standard"),
penalty=perm.out$bestpenalties, ncomponents=2, ws=perm.out$ws.init)
print(out)
# Or if you want to specify tuning parameters by hand:
# this time, assume all data sets are standard:
perm.out <- MultiCCA.permute(xlist, type="standard",
penalties=cbind(c(1.1,1.1,1.1),c(2,3,4),c(5,7,10)), ws=perm.out$ws.init)
print(perm.out)
plot(perm.out)

# Making use of the fact that the features are ordered:
out <- MultiCCA(xlist, type="ordered", penalty=.6)
par(mfrow=c(3,1))
PlotCGH(out$ws[[1]], chrom=rep(1,ncol(x1)))
PlotCGH(out$ws[[2]], chrom=rep(2,ncol(x2)))
PlotCGH(out$ws[[3]], chrom=rep(3,ncol(x3)))
```

`MultiCCA.permute` *Select tuning parameters for sparse multiple canonical correlation analysis using the penalized matrix decomposition.*

Description

This function can be used to automatically select tuning parameters for sparse multiple CCA. This is the analog of sparse CCA, when >2 data sets are available. Each data set may have features of type="standard" or type="ordered" (e.g. CGH data). Assume that there are K data sets, called $X1,...,XK$.

Usage

```
MultiCCA.permute(
  xlist,
  penalties = NULL,
  ws = NULL,
  type = "standard",
  nperms = 10,
  niter = 3,
  trace = TRUE,
  standardize = TRUE
)
```

Arguments

xlist	A list of length K, where K is the number of data sets on which to perform sparse multiple CCA. Data set k should be a matrix of dimension $n x p_k$ where p_k is the number of features in data set k.
penalties	The penalty terms to be considered in the cross-validation. If the same penalty term is desired for each data set, then this should be a vector of length equal to the number of penalty terms to be considered. If different penalty terms are desired for each data set, then this should be a matrix with rows equal to the number of data sets, and columns equal to the number of penalty terms to be considered. For a given data set Xk, if type is "standard" then the penalty term should be a number between 1 and $sqrt(p_k)$ (the number of features in data set k); it is a L1 bound on wk. If type is "ordered", on the other hand, the penalty term is of the form lambda in the fused lasso penalty. Therefore, the interpretation of the argument depends on whether type is "ordered" or "standard" for this data set.
ws	A list of length K; the kth element contanis the first ncomponents columns of the v matrix of the SVD of Xk. If NULL, then the SVD of Xk will be computed inside this function. However, if you plan to run this function multiple times, then save a copy of this argument so that it does not need to be re-computed.
type	A K-vector containing elements "standard" or "ordered" - or a single value. If a single value, then it is assumed that all elements are the same (either "standard" or "ordered"). If columns of v are ordered (e.g. CGH spots ordered along the chromosome) then "ordered", otherwise use "standard". "standard" will result in a lasso (L_1) penalty on v, which will result in smoothness. "ordered" will result in a fused lasso penalty on v, yielding both sparsity and smoothness.
nperms	How many times should the data be permuted? Default is 25. A large value of nperms is very important here, since the formula for computing the z-statistics requires a standard deviation estimate for the correlations obtained via permutation, which will not be accurate if nperms is very small.
niter	How many iterations should be performed each time CCA is called? Default is 3, since an approximate estimate of u and v is acceptable in this case, and otherwise this function can be quite time-consuming.
trace	Print out progress?
standardize	Should the columns of X and Z be centered (to have mean zero) and scaled (to have standard deviation 1)? Default is TRUE.

Details

The tuning parameters are selected using a permutation scheme. For each candidate tuning parameter value, the following is performed: (1) Repeat the following n times, for n large: (a) The samples in $(X1,...,XK)$ are randomly permuted to obtain data sets $(X1*,...,XK*)$. (b) Sparse multiple CCA is run on the permuted data sets $(X1*,...,XK*)$ to get canonical variates $(w1*,...,wK*)$. (c) Record $t* = sum_(i<j) Cor(Xi* wi*, Xj* wj*)$. (2) Sparse CCA is run on the original data $(X1,...,XK)$ to obtain canonical variates $(w1,...,wK)$. (3) Record $t = sum_(i<j) Cor(Xi wi, Xj wj)$. (4) The resulting p-value is given by $mean(t* > t)$; that is, the fraction of permuted

totals that exceed the total on the real data. Then, choose the tuning parameter value that gives the smallest value in Step 4.

This function only selets tuning parameters for the FIRST sparse multiple CCA factors.

Note that $x1,...,xK$ must have same number of rows. This function performs just a one-dimensional search in tuning parameter space.

Value

`zstat`	The vector of z-statistics, one per element of penalties.
`pvals`	The vector of p-values, one per element of penalties.
`bestpenalties`	The best set of penalties (the one with the highest zstat).
`cors`	The value of $sum_(j<k) cor(Xk wk, Xj wj)$ obtained for each value of penalties.
`corperms`	The nperms values of $sum_(j<k) cor(Xk* wk*, Xj* wj*)$ obtained for each value of penalties, where Xk* indicates the Xk matrix with permuted rows, and wk* is the canonical variate corresponding to the permuted data.
`ws.init`	Initial values used for ws in sparse multiple CCA algorithm.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

MultiCCA, CCA.permute, CCA

Examples

```
# See examples in MultiCCA function
```

`PlotCGH` *Plot CGH data*

Description

Given a vector of gains/losses at CGH spots, this makes a plot of gain/loss on each chromosome.

Usage

```
PlotCGH(array, chrom = NULL, nuc = NULL, main = "", scaleEachChrom = TRUE)
```

Arguments

array	A vector containing the chromosomal location of each CGH spot.
chrom	A numeric vector of the same length as "array"; its values should indicate the chromosome that each CGH spot is on (for instance, for human genomic data, values of chrom should range from 1 to 24). If NULL, then it is assumed that all elements of 'array' are on the same chromosome.
nuc	A numeric vector of same length as "array", indicating the nucleotide position of each CGH spot. If NULL, then the function assumes that each CGH spot corresponds to a consecutive position. E.g. if there are 200 CGH spots on chromosome 1, then they are located at positions 1,2,...,199,200.
main	Give your plot a title.
scaleEachChrom	Default is TRUE. This means that each chromosomes CGH spots are divided by 1.1 times the max of the CGH spots on that chromosome. This way, the CGH spots on each chromosome of the plot are as big as possible (i.e. easy to see). If FALSE, then all of the CGH spots are divided by 1.1 times the max of ALL the CGH spots. This means that on some chromosomes CGH spots might be hard to see, but has the advantage that now relative magnitudes of CGH spots on different chromosomes can be seen from figure.

Details

This function makes a plot of regions of genomic gain/loss.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

PMD, PMD.cv, CCA, CCA.permute

Examples

```
## Not run:
# Use breast data
breastdata <- download_breast_data()
with(breastdata, {
# dna contains CGH data and chrom contains chromosome of each CGH spot;
# nuc contains position of each CGH spot.
dna <- t(dna)
ch1 <- which(chrom == 1)
PlotCGH(dna[1,],chrom=chrom,nuc=nuc,main="Sample 1: All Chromosomes")
PlotCGH(dna[1,ch1], chrom=chrom[ch1], nuc=nuc[ch1],
main= "Sample 1: Chrom 1")
chlt3 = which(chrom <= 3)
PlotCGH(dna[1,chlt3], chrom=chrom[chlt3], nuc=nuc[chlt3],
```

```
 main="Sample 1: Chroms 1, 2, and 3")
} )

## End(Not run)
```

PMD *Get a penalized matrix decomposition for a data matrix.*

Description

Performs a penalized matrix decomposition for a data matrix. Finds factors u and v that summarize the data matrix well. u and v will both be sparse, and v can optionally also be smooth.

Usage

```
PMD(
  x,
  type = c("standard", "ordered"),
  sumabs = 0.4,
  sumabsu = 5,
  sumabsv = NULL,
  lambda = NULL,
  niter = 20,
  K = 1,
  v = NULL,
  trace = TRUE,
  center = TRUE,
  chrom = NULL,
  rnames = NULL,
  cnames = NULL,
  upos = FALSE,
  uneg = FALSE,
  vpos = FALSE,
  vneg = FALSE
)
```

Arguments

x	Data matrix of dimension $n x p$, which can contain NA for missing values.
type	"standard" or "ordered": Do we want v to simply be sparse, or should it also be smooth? If the columns of x are ordered (e.g. CGH spots along a chromosome) then choose "ordered". Default is "standard". If "standard", then the PMD function will make use of sumabs OR sumabsu&sumabsv. If "ordered", then the function will make use of sumabsu and lambda.
sumabs	Used only if type is "standard". A measure of sparsity for u and v vectors, between 0 and 1. When sumabs is specified, and sumabsu and sumabsv are NULL, then sumabsu is set to $sqrt(n)*sumabs$ and sumabsv is set to $sqrt(p)*sumabs$.

If sumabs is specified, then sumabsu and sumabsv should be NULL. Or if sumabsu and sumabsv are specified, then sumabs should be NULL.

`sumabsu` Used for types "ordered" AND "standard". How sparse do you want u to be? This is the sum of absolute values of elements of u. It must be between 1 and the square root of the number of rows in data matrix. The smaller it is, the sparser u will be.

`sumabsv` Used only if type is "standard". How sparse do you want v to be? This is the sum of absolute values of elements of v. It must be between 1 and square root of number of columns of data. The smaller it is, the sparser v will be.

`lambda` Used only if type is "ordered". This is the tuning parameter for the fused lasso penalty on v, which takes the form $lambda ||v||*1* + *lambda* |*v_j* - *v*(j-1)|$. $lambda$ must be non-negative. If NULL, then it is chosen adaptively from the data.

`niter` How many iterations should be performed. It is best to run at least 20 of so. Default is 20.

`K` The number of factors in the PMD to be returned; default is 1.

`v` The first right singular vector(s) of the data. (If missing data is present, then the missing values are imputed before the singular vectors are calculated.) v is used as the initial value for the iterative PMD algorithm. If x is large, then this step can be time-consuming; therefore, if PMD is to be run multiple times, then v should be computed once and saved.

`trace` Print out progress as iterations are performed? Default is TRUE.

`center` Subtract out mean of x? Default is TRUE.

`chrom` If type is "ordered", then this gives the option to specify that some columns of x (corresponding to CGH spots) are on different chromosomes. Then v will be sparse, and smooth *within* each chromosome but not *between* chromosomes. Length of chrom should equal number of columns of x, and each entry in chrom should be a number corresponding to which chromosome the CGH spot is on.

`rnames` An optional vector containing a name for each row of x.

`cnames` An optional vector containing a name for each column of x.

`upos` Constrain the elements of u to be positive? TRUE or FALSE.

`uneg` Constrain the elements of u to be negative? TRUE or FALSE.

`vpos` Constrain the elements of v to be positive? TRUE or FALSE. Cannot be used if type is "ordered".

`vneg` Constrain the elements of v to be negative? TRUE or FALSE. Cannot be used if type is "ordered."

Details

The criterion for the PMD is as follows: we seek vectors u and v such that $u'Xv$ is large, subject to $||u||_2=1, ||v||_2=1$ and additional penalties on u and v. These additional penalties are as follows: If type is "standard", then lasso (L_1) penalties (promoting sparsity) are placed on u and v. If type is "ordered", then lasso penalty is placed on u and a fused lasso penalty (promoting sparsity and smoothness) is placed on v.

If type is "standard", then arguments sumabs OR sumabsu&sumabsv are used. If type is "ordered", then sumabsu AND lambda are used. Sumabsu is the bound of absolute value of elements of u. Sumabsv is bound of absolute value of elements of v. If sumabs is given, then sumabsu is set to sqrt(nrow(x))*sumabs and sumabsv is set to sqrt(ncol(x))*sumabs. $lambda$ is the parameter for the fused lasso penalty on v when type is "ordered": $lambda(||v||*1* + *sum_j* |*v_j* - *v*(j-1))$.

Value

u	u is output. If you asked for multiple factors then each column of u is a factor. u has dimension nxK if you asked for K factors.
v	v is output. If you asked for multiple factors then each column of v is a factor. v has dimension pxK if you asked for K factors.
d	d is output. Computationally, $d=u'Xv$ where u and v are the sparse factors output by the PMD function and X is the data matrix input to the PMD function. When K=1, the residuals of the rank-1 PMD are given by $X - duv'$.
v.init	The first right singular vector(s) of the data; these are returned to save on computation time if PMD will be run again.
meanx	Mean of x that was subtracted out before PMD was performed.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

PMD.cv, SPC

Examples

```
# Try PMD with L1 penalty on rows and columns: type="standard"
# A simple simulated example
set.seed(1)
# Our data is a rank-one matrix, plus noise. The underlying components
# contain 50 and 75 non-zero elements, respectively.
u <- matrix(c(rnorm(50), rep(0,150)),
ncol=1)
v <- matrix(c(rnorm(75),rep(0,225)), ncol=1)
x <- u%*%t(v)+
matrix(rnorm(200*300),ncol=300)
# We can use cross-validation to try to find optimal value of sumabs
cv.out <- PMD.cv(x, type="standard", sumabss=seq(0.1, 0.6, len=20))
print(cv.out)
plot(cv.out)
# The optimal value of sumabs is 0.4157, but we can get within one
# standard error of that CV error using sumabs=0.337, which corresponds to
```

```
# an average of 45.8 and 71.8 non-zero elements in each component - pretty
# close to the true model.
# We can fit the model corresponding to the lowest cross-validation error:
out <- PMD(x, type="standard", sumabs=cv.out$bestsumabs, K=1, v=cv.out$v.init)
print(out)
par(mfrow=c(2,2))
par(mar=c(2,2,2,2))
plot(out$u[,1], main="Est. u")
plot(out$v[,1], main="Est. v")
plot(u, main="True u")
plot(v, main="True v")
# And if we want to control sumabsu and sumabsv separately, we can do
# that too. Let's get 2 components while we're at it:
out2 <- PMD(x, type="standard",  K=2, sumabsu=6, sumabsv=8, v=out$v.init,
cnames=paste("v", sep=" ", 1:ncol(x)), rnames=paste("u", sep=" ", 1:nrow(x)))
print(out2)

# Now check out PMD with L1 penalty on rows and fused lasso penalty on
# columns: type="ordered". We'll use the Chin et al (2006) Cancer Cell
# data set; try "?breastdata" for more info.
## Not run:
breastdata <- download_breast_data()
with(breastdata, {
# dna contains CGH data and chrom contains chromosome of each CGH spot;
# nuc contains position of each CGH spot.
dna <- t(dna) # Need samples on rows and CGH spots on columns
# First, look for shared regions of gain/loss on chromosome 1.
# Use cross-validation to choose tuning parameter value
par(mar=c(2,2,2,2))
ch1 = which(chrom == 1)
cv.out <- PMD.cv(dna[, ch1],type="ordered",chrom=chrom[ch1],
nuc=nuc[ch1],
sumabsus=seq(1, sqrt(nrow(dna)), len=15))
print(cv.out)
plot(cv.out)
out <- PMD(dna[,chrom==1],type="ordered",
sumabsu=cv.out$bestsumabsu,chrom=chrom[chrom==1],K=1,v=cv.out$v.init,
cnames=paste("Pos",sep=""",
nuc[chrom==1]), rnames=paste("Sample", sep=" ", 1:nrow(dna)))
print(out, verbose=TRUE)
# Which samples actually have that region of gain/loss?
par(mfrow=c(3,1))
par(mar=c(2,2,2,2))
PlotCGH(dna[which.min(out$u[,1]),chrom==1],chrom=chrom[chrom==1],
main=paste(paste(paste("Sample ", sep="", which.min(out$u[,1])),
sep="; u=", round(min(out$u[,1]),3))),nuc=nuc[chrom==1])
PlotCGH(dna[88,chrom==1], chrom=chrom[chrom==1],
main=paste("Sample 88; u=", sep="", round(out$u[88,1],3)),
nuc=nuc[chrom==1])
PlotCGH(out$v[,1],chrom=chrom[chrom==1], main="V",nuc=nuc[chrom==1])
} )

## End(Not run)
```

PMD.cv *Do tuning parameter selection for PMD via cross-validation*

Description

Performs cross-validation to select tuning parameters for rank-1 PMD, the penalized matrix decomposition for a data matrix.

Usage

```
PMD.cv(
  x,
  type = c("standard", "ordered"),
  sumabss = seq(0.1, 0.7, len = 10),
  sumabsus = NULL,
  lambda = NULL,
  nfolds = 5,
  niter = 5,
  v = NULL,
  chrom = NULL,
  nuc = NULL,
  trace = TRUE,
  center = TRUE,
  upos = FALSE,
  uneg = FALSE,
  vpos = FALSE,
  vneg = FALSE
)
```

Arguments

x	Data matrix of dimension $n x p$, which can contain NA for missing values.
type	"standard" or "ordered": Do we want v to simply be sparse, or should it also be smooth? If the columns of x are ordered (e.g. CGH spots along a chromosome) then choose "ordered". Default is "standard". If "standard", then the PMD function will make use of sumabs OR sumabsu&sumabsv. If "ordered", then the function will make use of sumabsu and lambda.
sumabss	Used only if type is "standard". A vector of sumabs values to be used. Sumabs is a measure of sparsity for u and v vectors, between 0 and 1. When sumabss is specified, and sumabsus and sumabsvs are NULL, then sumabsus is set to $sqrt(n)*sumabss$ and sumabsvs is set at $sqrt(p)*sumabss$. If sumabss is specified, then sumabsus and sumabsvs should be NULL. Or if sumabsus and sumabsvs are specified, then sumabss should be NULL.
sumabsus	Used only for type "ordered". A vector of sumabsu values to be used. Sumabsu measures sparseness of u - it is the sum of absolute values of elements of u. Must be between 1 and sqrt(n).

lambda	Used only if type is "ordered". This is the tuning parameter for the fused lasso penalty on v, which takes the form $lambda ||v||*1* + *lambda* |*v_j* - *v*(j-1)|$. $lambda$ must be non-negative. If NULL, then it is chosen adaptively from the data.
nfolds	How many cross-validation folds should be performed? Default is 5.
niter	How many iterations should be performed. For speed, only 5 are performed by default.
v	The first right singular vector(s) of the data. (If missing data is present, then the missing values are imputed before the singular vectors are calculated.) v is used as the initial value for the iterative PMD algorithm. If x is large, then this step can be time-consuming; therefore, if PMD is to be run multiple times, then v should be computed once and saved.
chrom	If type is "ordered", then this gives the option to specify that some columns of x (corresponding to CGH spots) are on different chromosomes. Then v will be sparse, and smooth *within* each chromosome but not *between* chromosomes. Length of chrom should equal number of columns of x, and each entry in chrom should be a number corresponding to which chromosome the CGH spot is on.
nuc	If type is "ordered", can specify the nucleotide position of each CGH spot (column of x), to be used in plotting. If NULL, then it is assumed that CGH spots are equally spaced.
trace	Print out progress as iterations are performed? Default is TRUE.
center	Subtract out mean of x? Default is TRUE
upos	Constrain the elements of u to be positive? TRUE or FALSE.
uneg	Constrain the elements of u to be negative? TRUE or FALSE.
vpos	Constrain the elements of v to be positive? TRUE or FALSE. Cannot be used if type is "ordered".
vneg	Constrain the elements of v to be negative? TRUE or FALSE. Cannot be used if type is "ordered."

Details

If type is "standard", then lasso (L_1) penalties (promoting sparsity) are placed on u and v. If type is "ordered", then lasso penalty is placed on u and a fused lasso penalty (promoting sparsity and smoothness) is placed on v.

Cross-validation of the rank-1 PMD is performed over sumabss (if type is "standard") or over sumabsus (if type is "ordered"). If type is "ordered", then lambda is chosen from the data without cross-validation.

The cross-validation works as follows: Some percent of the elements of x is removed at random from the data matrix. The PMD is performed for a range of tuning parameter values on this partially-missing data matrix; then, missing values are imputed using the decomposition obtained. The value of the tuning parameter that results in the lowest sum of squared errors of the missing values if "best".

To do cross-validation on the rank-2 PMD, first the rank-1 PMD should be computed, and then this function should be performed on the residuals, given by $x-udv'$.

Value

cv	Average sum of squared errors obtained over cross-validation folds.
cv.error	Standard error of average sum of squared errors obtained over cross-validation folds.
bestsumabs	If type="standard", then value of sumabss resulting in smallest CV error is returned.
bestsumabsu	If type="ordered", then value of sumabsus resulting in smallest CV error is returned.
v.init	The first right singular vector(s) of the data; these are returned to save on computation time if PMD will be run again.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

PMD, SPC

Examples

```
# See examples in PMD help file
```

SPC *Perform sparse principal component analysis*

Description

Performs sparse principal components analysis by applying PMD to a data matrix with lasso (L_1) penalty on the columns and no penalty on the rows.

Usage

```
SPC(
  x,
  sumabsv = 4,
  niter = 20,
  K = 1,
  orth = FALSE,
  trace = TRUE,
  v = NULL,
  center = TRUE,
  cnames = NULL,
```

```
  vpos = FALSE,
  vneg = FALSE,
  compute.pve = TRUE
)
```

Arguments

x	Data matrix of dimension $n x p$, which can contain NA for missing values. We are interested in finding sparse principal components of dimension p.
sumabsv	How sparse do you want v to be? This is the sum of absolute values of elements of v. It must be between 1 and square root of number of columns of data. The smaller it is, the sparser v will be.
niter	How many iterations should be performed. It is best to run at least 20 of so. Default is 20.
K	The number of factors in the PMD to be returned; default is 1.
orth	If TRUE, then use method of Section 3.2 of Witten, Tibshirani and Hastie (2008) to obtain multiple sparse principal components. Default is FALSE.
trace	Print out progress as iterations are performed? Default is TRUE.
v	The first right singular vector(s) of the data. (If missing data is present, then the missing values are imputed before the singular vectors are calculated.) v is used as the initial value for the iterative PMD(L_1, L_1) algorithm. If x is large, then this step can be time-consuming; therefore, if PMD is to be run multiple times, then v should be computed once and saved.
center	Subtract out mean of x? Default is TRUE
cnames	An optional vector containing a name for each column.
vpos	Constrain the elements of v to be positive? TRUE or FALSE.
vneg	Constrain the elements of v to be negative? TRUE or FALSE.
compute.pve	Compute percent variance explained? Default TRUE. If not needed, then choose FALSE to save time.

Details

PMD(x,sumabsu=sqrt(nrow(x)), sumabsv=3, K=1) and SPC(x,sumabsv=3, K=1) give the same result, since the SPC method is simply PMD with an L1 penalty on the columns and no penalty on the rows.

In Witten, Tibshirani, and Hastie (2008), two methods are presented for obtaining multiple factors for SPC. The methods are as follows:

(1) If one has already obtained factors $k-1$ factors then oen can compute residuals by subtracting out these factors. Then u_k and v_k can be obtained by applying the SPC/PMD algorithm to the residuals.

(2) One can require that u_k be orthogonal to u_i's with $i<k$; the method is slightly more complicated, and is explained in WT&H(2008).

Method 1 is performed by running SPC with option orth=FALSE (the default) and Method 2 is performed using option orth=TRUE. Note that Methods 1 and 2 always give identical results for the first component, and often given quite similar results for later components.

Value

u — u is output. If you asked for multiple factors then each column of u is a factor. u has dimension nxK if you asked for K factors.

v — v is output. These are the sparse principal components. If you asked for multiple factors then each column of v is a factor. v has dimension pxK if you asked for K factors.

d — d is output; it is the diagonal of the matrix D in the penalized matrix decomposition. In the case of the rank-1 decomposition, it is given in the formulation $||X-duv'||_F^2$ subject to $||u||_1 <= sumabsu$, $||v||_1 <= sumabsv$. Computationally, $d=u'Xv$ where u and v are the sparse factors output by the PMD function and X is the data matrix input to the PMD function.

prop.var.explained — A vector containing the proportion of variance explained by the first 1, 2, ..., K sparse principal components obtaineds. Formula for proportion of variance explained is on page 20 of Shen & Huang (2008), Journal of Multivariate Analysis 99: 1015-1034.

v.init — The first right singular vector(s) of the data; these are returned to save on computation time if PMD will be run again.

meanx — Mean of x that was subtracted out before SPC was performed.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

SPC.cv, PMD, PMD.cv

Examples

```
# A simple simulated example
#NOT RUN
#set.seed(1)
#u <- matrix(c(rnorm(50), rep(0,150)),ncol=1)
#v <- matrix(c(rnorm(75),rep(0,225)), ncol=1)
#x <- u%*%t(v)+matrix(rnorm(200*300),ncol=300)
## Perform Sparse PCA - that is, decompose a matrix w/o penalty on rows
## and w/ L1 penalty on columns
## First, we perform sparse PCA and get 4 components, but we do not
## require subsequent components to be orthogonal to previous components
#out <- SPC(x,sumabsv=3, K=4)
#print(out,verbose=TRUE)
## We could have selected sumabsv by cross-validation, using function SPC.cv
```

```
## Now, we do sparse PCA using method in Section 3.2 of WT&H(2008) for getting
## multiple components - that is, we require components to be orthogonal
#out.orth <- SPC(x,sumabsv=3, K=4, orth=TRUE)
#print(out.orth,verbose=TRUE)
#par(mfrow=c(1,1))
#plot(out$u[,1], out.orth$u[,1], xlab="", ylab="")
## Note that the first components w/ and w/o orth option are identical,
## since the orth option only affects the way that subsequent components
## are found
#print(round(t(out$u)%*%out$u,4)) # not orthogonal
#print(round(t(out.orth$u)%*%out.orth$u,4)) # orthogonal
#
## Use SPC.cv to choose tuning parameters:
#cv.out <- SPC.cv(x)
#print(cv.out)
#plot(cv.out)
#out <- SPC(x, sumabsv=cv.out$bestsumabsv)
#print(out)
## or we could do
#out <- SPC(x, sumabsv=cv.out$bestsumabsv1se)
#print(out)
#
#
```

SPC.cv *Perform cross-validation on sparse principal component analysis*

Description

Selects tuning parameter for the sparse principal component analysis method of Witten, Tibshirani, and Hastie (2008), which involves applying PMD to a data matrix with lasso (L_1) penalty on the columns and no penalty on the rows. The tuning parameter controls the sum of absolute values - or L_1 norm - of the elements of the sparse principal component.

Usage

```
SPC.cv(
  x,
  sumabsvs = seq(1.2, 5, len = 10),
  nfolds = 5,
  niter = 5,
  v = NULL,
  trace = TRUE,
  orth = FALSE,
  center = TRUE,
  vpos = FALSE,
  vneg = FALSE
)
```

Arguments

x	Data matrix of dimension $n x p$, which can contain NA for missing values. We are interested in finding sparse principal components of dimension p.
sumabsvs	Range of sumabsv values to be considered in cross-validation. Sumabsv is the sum of absolute values of elements of v. It must be between 1 and square root of number of columns of data. The smaller it is, the sparser v will be.
nfolds	Number of cross-validation folds performed.
niter	How many iterations should be performed. By default, perform only 5 for speed reasons.
v	The first right singular vector(s) of the data. (If missing data is present, then the missing values are imputed before the singular vectors are calculated.) v is used as the initial value for the iterative PMD(L_1, L_1) algorithm. If x is large, then this step can be time-consuming; therefore, if PMD is to be run multiple times, then v should be computed once and saved.
trace	Print out progress as iterations are performed? Default is TRUE.
orth	If TRUE, then use method of Section 3.2 of Witten, Tibshirani and Hastie (2008) to obtain multiple sparse principal components. Default is FALSE.
center	Subtract out mean of x? Default is TRUE
vpos	Constrain elements of v to be positive? Default is FALSE.
vneg	Constrain elements of v to be negative? Default is FALSE.

Details

This method only performs cross-validation for the first sparse principal component. It does so by performing the following steps nfolds times: (1) replace a fraction of the data with missing values, (2) perform SPC on this new data matrix using a range of tuning parameter values, each time getting a rank-1 approximationg udv' where v is sparse, (3) measure the mean squared error of the rank-1 estimate of the missing values created in step 1.

Then, the selected tuning parameter value is that which resulted in the lowest average mean squared error in step 3.

In order to perform cross-validation for the second sparse principal component, apply this function to $X-udv'$ where udv' are the output of running SPC on the raw data X.

Value

cv	Average sum of squared errors that results for each tuning parameter value.
cv.error	Standard error of the average sum of squared error that results for each tuning parameter value.
bestsumabsv	Value of sumabsv that resulted in lowest CV error.
nonzerovs	Average number of non-zero elements of v for each candidate value of sumabsvs.
v.init	Initial value of v that was passed in. Or, if that was NULL, then first right singular vector of X.
bestsumabsv1se	The smallest value of sumabsv that is within 1 standard error of smallest CV error.

References

Witten D. M., Tibshirani R., and Hastie, T. (2009) *A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis*, *Biostatistics, Gol 10 (3), 515-534, Jul 2009*

See Also

SPC, PMD, PMD.cv

Examples

```
#NOT RUN
## A simple simulated example
#set.seed(1)
#u <- matrix(c(rnorm(50), rep(0,150)),ncol=1)
#v <- matrix(c(rnorm(75),rep(0,225)), ncol=1)
#x <- u%*%t(v)+matrix(rnorm(200*300),ncol=300)
## Perform Sparse PCA - that is, decompose a matrix w/o penalty on rows
## and w/ L1 penalty on columns
## First, we perform sparse PCA and get 4 components, but we do not
## require subsequent components to be orthogonal to previous components
#cv.out <- SPC.cv(x, sumabsvs=seq(1.2, sqrt(ncol(x)), len=6))
#print(cv.out)
#plot(cv.out)
#out <- SPC(x,sumabsv=cv.out$bestsumabs, K=4) # could use
## cv.out$bestsumabvsv1se instead
#print(out,verbose=TRUE)
## Now, we do sparse PCA using method in Section 3.2 of WT&H(2008) for getting
## multiple components - that is, we require components to be orthogonal
#cv.out <- SPC.cv(x, sumabsvs=seq(1.2, sqrt(ncol(x)), len=6), orth=TRUE)
#print(cv.out)
#plot(cv.out)
#out.orth <- SPC(x,sumabsv=cv.out$bestsumabsv, K=4, orth=TRUE)
#print(out.orth,verbose=TRUE)
#par(mfrow=c(1,1))
#plot(out$u[,1], out.orth$u[,1], xlab="", ylab="")
#
#
```

Index