Package 'parallelpam'

July 15, 2024

Type Package

Title Parallel Partitioning-Around-Medoids (PAM) for Big Sets of Data

Version 1.4.3

Date 2024-07-05

Author Juan Domingo [aut, cre] (<https://orcid.org/0000-0003-4728-6256>),
Guillermo Ayala [ctb] (<https://orcid.org/0000-0002-6231-2865>),
Spanish Ministry of Science and Innovation, MCIN/AEI
<doi:10.13039/501100011033> [fnd]

Maintainer Juan Domingo <Juan.Domingo@uv.es>

Description Application of the Partitioning-Around-Medoids (PAM) clustering algorithm described in Schubert, E. and Rousseeuw, P.J.: ``Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms." Information Systems, vol. 101, p. 101804, (2021). <doi:10.1016/j.is.2021.101804>.
It uses a binary format for storing and retrieval of matrices developed for the 'jmatrix' package but the functionality of 'jmatrix'
is included here, so you do not need to install it. Also, it is used by package 'scellpam', so if you have installed it, you do not need
to install this package.
PAM can be applied to sets of data whose dissimilarity matrix can be very big. It has been tested with up to 100.000 points.
It does this with the help of the code developed for other package, 'jmatrix', which allows the matrix not to be loaded in 'R' memory (which
would force it to be of double type) but it gets from disk, which allows using float (or even smaller data types). Moreover, the
dissimilarity matrix is calculated in parallel if the computer has several cores so it can open many threads. The initial part
of the PAM algorithm can be done with the BUILD or LAB algorithms; the BUILD algorithm has been implemented in parallel. The optimization
phase implements the FastPAM1 algorithm, also in parallel. Finally, calculation of silhouette is available and also implemented in parallel.

License GPL (>= 2)

Imports Rcpp (>= 1.0.8), memuse (>= 4.2.1)

LinkingTo Rcpp

RoxygenNote 7.2.3

Encoding UTF-8

Suggests knitr, cluster

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-07-15 18:50:15 UTC

Contents

ApplyPAM *ApplyPAM*

Description

A function to implement the Partitioning-around-medoids algorithm described in
Schubert, E. and Rousseeuw, P.J.: "Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms."
Information Systems, vol. 101, p. 101804, 2021.
doi: https://doi.org/10.1016/j.is.2021.101804
Notice that the actual values of the vectors (instances) are not needed. To recover them, look at the data matrix used to generate the distance matrix.
The number of instances, N, is not passed since dissimilarity matrix is NxN and therefore its size indicates the N value.

Usage

```
ApplyPAM(
  dissim_file,
  k,
  init_method = "BUILD",
  initial_med = NULL,
  max_iter = 1000L,
  nthreads = 0L
)
```

Arguments

`dissim_file` A string with the name of the binary file that contains the symmetric matrix of dissimilarities. Such matrix should have been generated by CalcAndWriteDissimilarityMatrix and it must be a symmetric matrix.

`k` A possitive integer (the desired number of medoids).

`init_method` One of the strings ’PREV’, ’BUILD’ or ’LAB’. See meaning of initialization algorithms BUILD and LAB in the original paper.
’PREV’ should be used exclusively to start the second part of the algorithm (optimization) from a initial set of medoids generated by a former call.
Default: BUILD.

`initial_med` A vector with initial medoids to start optimization. It is to be used only by the ’PREV’ method and it will have been obtained as the first element (L$med) of the two-element list returned by a previous call to this function used in just-initialize mode (max_iter=0).
Default: empty vector.

`max_iter` The maximum number of allowed iterations. 0 means stop immediately after finding initial medoids.
Default: 1000

`nthreads` The number of used threads.
-1 means don’t use threads (serial implementation).
0 means let the program choose according to the number of cores and of points.
Any other number forces this number of threads. Choosing more than the number of available cores is allowed, but discouraged.
Default: 0

Details

With respect to the returned value, L$med has as many components
as requested medoids and L$clasif has as many components as instances.
Medoids are expressed in L$med by its number in the array of points (row in the dissimilarity matrix) starting at 1 (R convention).
L$clasif contains the number of the medoid (i.e.: the cluster) to which each instance has been assigned, according to their order in
L$med (also from 1).
This means that if L$clasif[p] is m, the point p belongs to the
class grouped around medoid L$med[m].
Moreover, if the dissimilarity matrix contains as metadata
(row names) the point names, the returned vector is a R-named vector with such names.

Value

L["med","clasif"] A list of two numeric vectors. See section Details for more information

Examples

```
# Synthetic problem: 10 random seeds with coordinates in [0..20]
# to which random values in [-0.1..0.1] are added
M<-matrix(0,100,500)
rownames(M)<-paste0("rn",c(1:100))
for (i in (1:10))
{
 p<-20*runif(500)
 Rf <- matrix(0.2*(runif(5000)-0.5),nrow=10)
 for (k in (1:10))
 {
  M[10*(i-1)+k,]=p+Rf[k,]
 }
}
tmpfile1=paste0(tempdir(),"/pamtest.bin")
JWriteBin(M,tmpfile1,dtype="float",dmtype="full")
tmpdisfile1=paste0(tempdir(),"/pamDL2.bin")
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype="L2",restype="float",nthreads=0)
L <- ApplyPAM(tmpdisfile1,10,init_method="BUILD")
# Final value of sum of distances to closest medoid
GetTD(L,tmpdisfile1)
# Medoids:
L$med
# Medoid in which each individual has been classified
```

```
n<-names(L$med)
n[L$clasif]
```

CalcAndWriteDissimilarityMatrix *CalcAndWriteDissimilarityMatrix*

Description

Writes a binary symmetric matrix with the dissimilarities between ROWS of the data stored in a binary matrix in the jmatrix/parallelpam package format.
The input matrix of vectors can be a full or a sparse matrix and the algorithm has been modified to calculate faster for sparse matrices.
Output matrix type can be float or double type (but look at the comments in 'Details').

Usage

```
CalcAndWriteDissimilarityMatrix(
  ifname,
  ofname,
  distype = "L2",
  restype = "float",
  comment = "",
  nthreads = 0L
)
```

Arguments

`ifname` A string with the name of the file containing the counts as a binary matrix.

`ofname` A string with the name of the binary output file to contain the symmetric dissimilarity matrix.

`distype` The dissimilarity to be calculated. It must be one of these strings: 'L1', 'L2', 'Pearson', 'Cos' or 'WEuc'.
Respectively: L1 (Manhattan), L2 (Euclidean), Pearson (Pearson dissimilarity), Cos (cosine distance), WEuc (weigthed Euclidean, with inverse-stdevs as weights).
Default: 'L2'.

`restype` The data type of the result. It can be one of the strings 'float' or 'double'. Default: float (and don't change it unless you REALLY need to...).

`comment` Comment to be added to the dissimilary matrix. Default: "" (no comment)

`nthreads` Number of threads to be used for the parallel calculations with this meaning:
-1: don't use threads.
0: let the function choose according to the number of rows and to the number of available cores.
Any possitive number > 1: use that number of threads. You can use even more than cores, but this is discouraged and raises a warning.
Default: 0.

Details

The parameter restype forces the output to be a matrix of either floats or doubles. Precision of float is normally good enough; but if you need double precision (may be because you expect your results to be in a large range, two to three orders of magnitude), change it.
Nevertheless, notice that this at the expense of double memory usage, which is QUADRATIC with the number of individuals (rows) in your input matrix.

Value

No return value, called for side effects (creates a file)

Examples

```
Rf <- matrix(runif(50000),nrow=100)
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",
          comment="Full matrix of floats, 100 rows, 500 columns")
JMatInfo(tmpfile1)
tmpdisfile1=paste0(tempdir(),"/RfullfloatDis.bin")
# Distance file calculated from the matrix stored as full
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype="L2",
                      restype="float",comment="L2 distance matrix from full",nthreads=0)
JMatInfo(tmpdisfile1)
tmpfile2=paste0(tempdir(),"/Rsparsefloat.bin")
JWriteBin(Rf,tmpfile2,dtype="float",dmtype="sparse",
                         comment="Sparse matrix of floats, 100 rows, 500 columns")
JMatInfo(tmpfile2)
# Distance file calculated from the matrix stored as sparse
tmpdisfile2=paste0(tempdir(),"/RsparsefloatDis.bin")
CalcAndWriteDissimilarityMatrix(tmpfile2,tmpdisfile2,distype="L2",
                     restype="float",comment="L2 distance matrix from sparse",nthreads=0)
JMatInfo(tmpdisfile2)
# Read both versions
Dfu<-GetJManyRows(tmpdisfile1,c(1:nrow(Rf)))
Dsp<-GetJManyRows(tmpdisfile2,c(1:nrow(Rf)))
# and compare them
max(Dfu-Dsp)
```

CalculateSilhouette *CalculateSilhouette*

Description

Calculates the silhouette of each point of those classified by a clustering algorithm.

Usage

```
CalculateSilhouette(cl, fdist, nthreads = 0L)
```

Arguments

`cl`	The array of classification with the number of the class to which each point belongs to. This number must be in 1..number_of_classes. This function takes something like the L$clasif array which is the second element of the list returned by ApplyPAM
`fdist`	The binary file containing the symmetric matrix with the dissimilarities between points (usually, generated by a call to CalcAndWriteDissimilarityMatrix)
`nthreads`	The number of used threads for parallel calculation. -1 means don't use threads (serial implementation). 0 means let the program choose according to the number of cores and of points. Any other number forces this number of threads. Choosing more than the number of available cores is allowed, but discouraged. Default: 0

Value

sil Numeric vector with the values of the silhouette for each point, in the same order in which points are in cl.
If cl is a named vector sil will be a named vector, too, with the same names.

Examples

```
# Synthetic problem: 10 random seeds with coordinates in [0..20]
# to which random values in [-0.1..0.1] are added
M<-matrix(0,100,500)
rownames(M)<-paste0("rn",c(1:100))
for (i in (1:10))
{
 p<-20*runif(500)
 Rf <- matrix(0.2*(runif(5000)-0.5),nrow=10)
 for (k in (1:10))
 {
  M[10*(i-1)+k,]=p+Rf[k,]
 }
}
tmpfile1=paste0(tempdir(),"/pamtest.bin")
JWriteBin(M,tmpfile1,dtype="float",dmtype="full")
tmpdisfile1=paste0(tempdir(),"/pamDL2.bin")
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype="L2",restype="float",nthreads=0)
L <- ApplyPAM(tmpdisfile1,10,init_method="BUILD")
sil <- CalculateSilhouette(L$clasif,tmpdisfile1)
# Histogram of the silhouette. In this synthetic problem, almost 1 for all points
hist(sil)
```

`ClassifAsDataFrame` *ClassifAsDataFrame*

Description

Returns the results of the classification returned by ApplyPAM as a R dataframe

Usage

```
ClassifAsDataFrame(L, fdist)
```

Arguments

`L`	The list returned by ApplyPAM with fields L$med and L$clasif with the numbers of the medoids and the classification of each point
`fdist`	The binary file containing the symmetric matrix with the dissimilarities between points (usually, generated by a call to CalcAndWriteDissimilarityMatrix).

Details

The dataframe has three columns: PointName (name of each point), NNPointName (name of the point which is the center of the cluster to which PointName belongs to) and NNDistance (distance between the points PointName and NNPointName). Medoids are identified by the fact that PointName and NNPointName are equal, or equivalently, NNDistance is 0.

Value

Df Dataframe with columns PointName, NNPointName and NNDistance. See Details for description.

Examples

```
# Synthetic problem: 10 random seeds with coordinates in [0..20]
# to which random values in [-0.1..0.1] are added
M<-matrix(0,100,500)
rownames(M)<-paste0("rn",c(1:100))
for (i in (1:10))
{
 p<-20*runif(500)
 Rf <- matrix(0.2*(runif(5000)-0.5),nrow=10)
 for (k in (1:10))
 {
  M[10*(i-1)+k,]=p+Rf[k,]
 }
}
tmpfile1=paste0(tempdir(),"/pamtest.bin")
JWriteBin(M,tmpfile1,dtype="float",dmtype="full")
tmpdisfile1=paste0(tempdir(),"/pamDL2.bin")
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype="L2",restype="float",nthreads=0)
L <- ApplyPAM(tmpdisfile1,10,init_method="BUILD")
df <- ClassifAsDataFrame(L,tmpdisfile1)
df
# Identification of medoids:
which(df[,3]==0)
# Verification they are the same as in L (in different order)
```

```
L$med
```

CsvToJMat *CsvToJMat*

Description

Gets a csv/tsv file and writes to a disk file the binary matrix of counts contained in it in the jmatrix binary format.
First line of the .csv is supposed to have the field names.
First column of each line is supposed to have the row name.
The fields are supposed to be separated by one occurrence of a character-field sepparator (usually, comma or tab) .tsv files can be read with this function, too, setting the csep argument to '\t'

Usage

```
CsvToJMat(
  ifname,
  ofname,
  mtype = "sparse",
  csep = ",",
  ctype = "raw",
  valuetype = "float",
  transpose = FALSE,
  comment = ""
)
```

Arguments

ifname	A string with the name of the .csv/.tsv text file.
ofname	A string with the name of the binary output file.
mtype	A string to indicate the matrix type: 'full', 'sparse' or 'symmetric'. Default: 'sparse'
csep	The character used as separator in the .csv file. Default: ',' (comma) (Set to '\t' for .tsv)
ctype	The string 'raw' or 'log1' to write raw counts or log(counts+1), or the normalized versions, 'rawn' and 'log1n', which normalize ALWAYS BY COLUMNS (before transposition, if requested to transpose). The logarithm is taken base 2. Default: raw
valuetype	The data type to store the matrix. It must be one of the strings 'uint32', 'float' or 'double'. Default: float
transpose	Boolean to indicate if the matrix should be transposed before writing. See Details for a comment about this. Default: FALSE
comment	A comment to be stored with the matrix. Default: "" (no comment)

Details

The parameter transpose has the default value of FALSE. But don't forget to set it to TRUE if you want the cells (which in single cell common practice are by columns) to be written by rows. This will be needed later to calculate the dissimilarity matrix, if this is the next step of your workflow. See help of CalcAndWriteDissimilarityMatrix

Special note for loading symmetric matrices:
If you use this function to load what you expect to be a symmetric matrix from a .csv file, remember that the input table MUST be square, but only the lower-diagonal matrix will be stored, including the main diagonal. The rest of the input table is completely ignored, except to check that there are values in it. It is not checked if the table really represents a symmetric matrix or not.
Furthermore, symmetric matrices can only be loaded in raw mode, i.e.: no normalization is allowed, and they cannot be transposed.

Value

No return value, called for side effects (creates a file)

Examples

```
# Since we have no a .csv file to test, we will generate one with another funcion of this package
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
tmpfile2=paste0(tempdir(),"/Rfullfloat2.bin")
tmpcsvfile1=paste0(tempdir(),"/Rfullfloat.csv")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
JMatToCsv(tmpfile1,tmpcsvfile1)
CsvToJMat(tmpcsvfile1,tmpfile2)
# It can be checked that files Rfullfloat.bin and Rfullfloat2.bin contain the same data
# (even they differ in the comment, which has been eliminated when converting to csv)
```

FilterBySilhouetteQuantile

FilterBySilhouetteQuantile

Description

Takes a silhouette, as returned by CalculateSilhouette, the list of medoids and class assignments, as returned by ApplyPam, a quantile and the matrices of values and dissimilarities and constructs the corresponding matrices clearing off the points whose silhoutte is below the lower quantile, except if they are medoids.

Usage

```
FilterBySilhouetteQuantile(
  s,
  L,
  fallcounts,
  ffilcounts,
  falldissim,
  ffildissim,
  q = 0.2,
  addcom = TRUE
)
```

Arguments

`s`	A numeric vector with the sihouette coefficient of each point in a classification, as returned by CalculateSilhouette.
`L`	A list of two numeric vectors, L$med and L$clasif, obtained normally as the object returned by ApplyPAM.
`fallcounts`	A string with the name of the binary file containing the matrix of data per point. It can be either a full or a sparse matrix.
`ffilcounts`	A string with the name of the binary file that will contain the selected points. It will have the same character (full/sparse) and type of the complete file.
`falldissim`	A string with the name of the binary file containing the dissimilarity matrix of the complete set of points. It must be a symmetric matrix.
`ffildissim`	A string with the name of the binary file that will contain the dissimilarity matrix for the remaining points. It will be a symmetric matrix of.
`q`	Quantile to filter. All points whose silhouette is below this quantile will be filtered out. Default: 0.2
`addcom`	Boolean to indicate if a comment must be appended to the current comment of values and dissimilarity matrices to indicate that they are the result of a filtering process. This comment is automatically generated and contains the value of quantile q. Succesive applications add comments at the end of those already present. Default: TRUE

Details

The renumbering of indices in the returned cluster may seem confusing at first but it was the way of fitting this with the rest of the package. Anyway, notice that if the numeric vectors in the input parameter L were named vectors, the point names are appropriately kept in the result so point identity is preserved. Moreover, if the values and dissimilarity input matrices had row and/or column names, they are preserved in the filtered matrices, too.

Value

Lr["med","clasif"] A list of two numeric vectors.
Lr$med is a modification of the correponding first element of the passed L parameter.

Lr$clasif has as many components as remaining instances.
Since points will have been removed, medoid numbering is modified. Therefore, Lr$med has the NEW index of each medoid in the filtered set.
Lr$clasif contains the number of the medoid (i.e.: the cluster) to which each instance has been assigned, and therefore does not change.
All indexes start at 1 (R convention). Please, see Details section

Examples

```
# Synthetic problem: 10 random seeds with coordinates in [0..20]
# to which random values in [-0.1..0.1] are added
M<-matrix(0,100,500)
rownames(M)<-paste0(”rn”,c(1:100))
for (i in (1:10))
{
 p<-20*runif(500)
 Rf <- matrix(0.2*(runif(5000)-0.5),nrow=10)
 for (k in (1:10))
 {
  M[10*(i-1)+k,]=p+Rf[k,]
 }
}
tmpfile1=paste0(tempdir(),”/pamtest.bin”)
JWriteBin(M,tmpfile1,dtype=”float”,dmtype=”full”)
tmpdisfile1=paste0(tempdir(),”/pamDl2.bin”)
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype=”L2”,restype=”float”,nthreads=0)
L <- ApplyPAM(tmpdisfile1,10,init_method=”BUILD”)
# Which are the medoids
L$med
sil <- CalculateSilhouette(L$clasif,tmpdisfile1)
tmpfiltfile1=paste0(tempdir(),”/pamtestfilt.bin”)
tmpfiltdisfile1=paste0(tempdir(),”/pamDL2filt.bin”)
Lf<-FilterBySilhouetteQuantile(sil,L,tmpfile1,tmpfiltfile1,tmpdisfile1,tmpfiltdisfile1,
                               q=0.4,addcom=TRUE)
# The new medoids are the same points but renumbered, since the L$clasif array has less points
Lf$med
```

FilterBySilhouetteThreshold

FilterBySilhouetteThreshold

Description

Takes a silhouette, as returned by CalculateSilhouette, the list of medoids and class assignments, as returned by ApplyPam, a threshold and the matrices of values and dissimilarities and constructs the corresponding matrices clearing off the points whose silhoutte is below the threshold, except if they are medoids.

Usage

```
FilterBySilhouetteThreshold(
  s,
  L,
  fallcounts,
  ffilcounts,
  falldissim,
  ffildissim,
  thres = 0,
  addcom = TRUE
)
```

Arguments

`s`	A numeric vector with the sihouette coefficient of each point in a classification, as returned by CalculateSilhouette.
`L`	A list of two numeric vectors, L$med and L$clasif, obtained normally as the object returned by ApplyPAM.
`fallcounts`	A string with the name of the binary file containing the matrix of values per point. It can be either a full or a sparse matrix.
`ffilcounts`	A string with the name of the binary file that will contain the selected points. It will have the same character (full/sparse) and type of the complete file.
`falldissim`	A string with the name of the binary file containing the dissimilarity matrix of the complete set of points. It must be a symmetric matrix.
`ffildissim`	A string with the name of the binary file that will contain the dissimilarity matrix for the remaining points. It will be a symmetric matrix.
`thres`	Threshold to filter. All points whose silhouette is below this threshold will be filtered out. Default: 0.0 (remember that silhouette is in [-1..1])
`addcom`	Boolean to indicate if a comment must be appended to the current comment of values and dissimilarity matrices to indicate that they are the result of a filtering process. This comment is automatically generated and contains the value of threshold t. Succesive applications add comments at the end of those already present. Default: TRUE

Details

The renumbering of indices in the returned cluster may seem confusing at first but it was the way of fitting this with the rest of the package. Anyway, notice that if the numeric vectors in the input parameter L were named vectors, the point names are appropriately kept in the result so point identity is preserved. Moreover, if the values and dissimilarity input matrices had row and/or column names, they are preserved in the filtered matrices, too.

Value

Lr["med","clasif"] A list of two numeric vectors.
Lr$med is a modification of the correponding first element of the passed L parameter.

Lr$clasif has as many components as remaining instances.
Since points will have been removed, medoid numbering is modified. Therefore, Lr$med has the NEW index of each medoid in the filtered set.
Lr$clasif contains the number of the medoid (i.e.: the cluster) to which each instance has been assigned, and therefore does not change.
All indexes start at 1 (R convention). Please, see Details section

Examples

```
# Synthetic problem: 10 random seeds with coordinates in [0..20]
# to which random values in [-0.1..0.1] are added
M<-matrix(0,100,500)
rownames(M)<-paste0("rn",c(1:100))
for (i in (1:10))
{
 p<-20*runif(500)
 Rf <- matrix(0.2*(runif(5000)-0.5),nrow=10)
 for (k in (1:10))
 {
  M[10*(i-1)+k,]=p+Rf[k,]
 }
}
tmpfile1=paste0(tempdir(),"/pamtest.bin")
JWriteBin(M,tmpfile1,dtype="float",dmtype="full")
tmpdisfile1=paste0(tempdir(),"/pamDl2.bin")
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype="L2",restype="float",nthreads=0)
L <- ApplyPAM(tmpdisfile1,10,init_method="BUILD")
# Which are the medoids
L$med
sil <- CalculateSilhouette(L$clasif,tmpdisfile1)
tmpfiltfile1=paste0(tempdir(),"/pamtestfilt.bin")
tmpfiltdisfile1=paste0(tempdir(),"/pamDL2filt.bin")
Lf<-FilterBySilhouetteThreshold(sil,L,tmpfile1,tmpfiltfile1,tmpdisfile1,tmpfiltdisfile1,
                               thres=0.4,addcom=TRUE)
# The new medoids are the same points but renumbered, since the L$clasif array has less points
Lf$med
```

FilterJMatByName *FilterJMatByName*

Description

Takes a jmatrix binary file containing a table with rows and columns and filters it by name, eliminating the rows or columns whose whose names are not in certain list

Usage

```
FilterJMatByName(fname, Gn, filname, namesat = "rows")
```

Arguments

`fname` A string with the file name of the original table

`Gn` A list of R strings with the names of the rows or columns that must remain. All others will be filtered out

`filname` A string with the file name of the filtered table

`namesat` The string "rows" or "cols" indicating if the searched names are in the rows or in the columns of the original table. Default: "rows"

Details

If the table has no list of names in the requested dimension (rows or colums), an error is rised.
The row or column names whose names are not found obviosuly cannot remain, and the program rises a warning indicating for which row/column names this happens.
The matrix contained in the filtered file will have the same nature (full or sparse) and the same data type as the original.
This function can be used to filter either by row or by column name, with appropriate usage of parameter namesat

Value

No return value, called for side effects (creates a file)

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
tmpfile2=paste0(tempdir(),"/Rfullfloatrowfilt.bin")
tmpfile3=paste0(tempdir(),"/Rfullfloatrowcolfilt.bin")
tmpcsvfile1=paste0(tempdir(),"/Rfullfloat.csv")
tmpcsvfile3=paste0(tempdir(),"/Rfullfloatrowcolfilt.csv")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
# Let's keep only rows A, C and E
FilterJMatByName(tmpfile1,c("A","C","E"),tmpfile2,namesat="rows")
# and from the result, let's keep only columns b, d and g
FilterJMatByName(tmpfile2,c("b","d","g"),tmpfile3,namesat="cols")
JMatToCsv(tmpfile1,tmpcsvfile1)
JMatToCsv(tmpfile3,tmpcsvfile3)
# You can now compare both ASCII/csv files
```

`GetJCol` *GetJCol*

Description

Returns (as a R numeric vector) the requested column number from the matrix contained in a jmatrix binary file

Usage

```
GetJCol(fname, ncol)
```

Arguments

`fname` String with the file name that contains the binary data.

`ncol` The number of the column to be returned, in R-numbering (from 1)

Value

A numeric vector with the values of elements in the requested column

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
Rf[,3]
vf<-GetJCol(tmpfile1,3)
vf
```

`GetJColByName` *GetJColByName*

Description

Returns (as a R numeric vector) the requested named column from the matrix contained in a jmatrix binary file

Usage

```
GetJColByName(fname, colname)
```

Arguments

`fname` String with the file name that contains the binary data.

`colname` The name of the column to be returned. If the matrix has no column names, or the name is not found, an empty vector is returned

Value

A numeric vector with the values of elements in the requested column

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
Rf[,"c"]
vf<-GetJColByName(tmpfile1,"c")
vf
```

`GetJColNames` *GetJColNames*

Description

Returns a R StringVector with the column names of a matrix stored in the binary format of package jmatrix, if it has them stored.

Usage

```
GetJColNames(fname)
```

Arguments

`fname`	String with the file name that contains the binary data.

Value

A R StringVector with the column names, or the empty vector if the binaryfile has no column names as metadata.

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
cn<-GetJColNames(tmpfile1)
cn
```

GetJManyCols *GetJManyCols*

Description

Returns (as a R numeric matrix) the columns with the requested column numbers from the matrix contained in a jmatrix binary file

Usage

```
GetJManyCols(fname, extcols)
```

Arguments

fname	String with the file name that contains the binary data.
extcols	A numeric vector with the indexes of the columns to be extracted, in R-numbering (from 1)

Value

A numeric matrix with the values of elements in the requested columns

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
vc<-GetJManyCols(tmpfile1,c(1,4))
vc
```

GetJManyColsByNames *GetJManyColsByNames*

Description

Returns (as a R numeric matrix) the columns with the requested column names from the matrix contained in a jmatrix binary file

Usage

```
GetJManyColsByNames(fname, extcolnames)
```

Arguments

`fname`	String with the file name that contains the binary data.
`extcolnames`	A vector of RStrings with the names of the columns to be extracted. If the binary file has no column names, or _any_ of the column names is not present, an empty matrix is returned.

Value

A numeric matrix with the values of elements in the requested columns

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
Rf[,c(1,4)]
vf<-GetJManyColsByNames(tmpfile1,c("a","d"))
vf
```

`GetJManyRows` *GetJManyRows*

Description

Returns (as a R numeric matrix) the rows with the requested row numbers from the matrix contained in a jmatrix binary file

Usage

```
GetJManyRows(fname, extrows)
```

Arguments

`fname`	String with the file name that contains the binary data.
`extrows`	A numeric vector with the indexes of the rows to be extracted, in R-numbering (from 1)

Value

A numeric matrix with the values of elements in the requested rows

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
Rf[c(1,4),]
vc<-GetJManyRows(tmpfile1,c(1,4))
vc
```

GetJManyRowsByNames *GetJManyRowsByNames*

Description

Returns (as a R numeric matrix) the rows with the requested row names from the matrix contained in a jmatrix binary file

Usage

```
GetJManyRowsByNames(fname, extrownames)
```

Arguments

fname	String with the file name that contains the binary data.
extrownames	A vector of RStrings with the names of the rows to be extracted. If the binary file has no row names, or _any_ of the row names is not present, an empty matrix is returned.

Value

A numeric matrix with the values of elements in the requested rows

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
Rf[c("A","C"),]
vf<-GetJManyRowsByNames(tmpfile1,c("A","C"))
vf
```

GetJNames *GetJNames*

Description

Returns a R list of two elements, rownames and colnames, each of them being a R StringVector with the corresponding names

Usage

```
GetJNames(fname)
```

Arguments

fname	String with the file name that contains the binary data.

Value

N["rownames","colnames"]: A list with two elements named rownames and colnames which are R StringVectors. If the binary file has no row or column names as metadata BOTH will be returned as empty vectors, even if one of them exists. If you want to extract only one, use either GetJRowNames or GetJColNames, as appropriate.

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
N<-GetJNames(tmpfile1)
N["rownames"]
N["colnames"]
```

GetJRow *GetJRow*

Description

Returns (as a R numeric vector) the requested row number from the matrix contained in a jmatrix binary file

Usage

```
GetJRow(fname, nrow)
```

Arguments

fname	String with the file name that contains the binary data.
nrow	The number of the row to be returned, in R-numbering (from 1)

Value

A numeric vector with the values of elements in the requested row

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
Rf[3,]
vf<-GetJRow(tmpfile1,3)
vf
```

GetJRowByName *GetJRowByName*

Description

Returns (as a R numeric vector) the requested named row from the matrix contained in a jmatrix binary file

Usage

```
GetJRowByName(fname, rowname)
```

Arguments

fname	String with the file name that contains the binary data.
rowname	The name of the row to be returned. If the matrix has no row names, or the name is not found, an empty vector is returned

Value

A numeric vector with the values of elements in the requested row

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
Rf["C",]
vf<-GetJRowByName(tmpfile1,"C")
vf
```

`GetJRowNames` *GetJRowNames*

Description

Returns a R StringVector with the row names of a matrix stored in the binary format of package jmatrix, if it has them stored.

Usage

```
GetJRowNames(fname)
```

Arguments

`fname`	String with the file name that contains the binary data.

Value

A R StringVector with the row names, or the empty vector if the binary file has no row names as metadata.

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
rn<-GetJRowNames(tmpfile1)
rn
```

GetSubdiag *GetSubdiag*

Description

Takes a symmetric matrix and returns a vector with all its elements under the main diagonal (without those at the diagonal itself) Done as an instrumental function to check the PAM in package cluster. To be removed in final version of the package.

Usage

```
GetSubdiag(fname)
```

Arguments

fname	The name of the file with the dissimilarity matrix in jmatrix binary format.

Value

The vector with the values under the main diagonal, sorted by columns (i.e.: m(2,1) .. m(n,1), m(3,2)..m(n,2),..., m(n-1,n))

Examples

```
Rns <- matrix(runif(49),nrow=7)
Rsym <- 0.5*(Rns+t(Rns))
rownames(Rsym) <- c("A","B","C","D","E","F","G")
colnames(Rsym) <- c("a","b","c","d","e","f","g")
tmpfile1=paste0(tempdir(),"/Rsymfloat.bin")
JWriteBin(Rsym,tmpfile1,dtype="float",dmtype="symmetric")
d<-GetSubdiag(tmpfile1)
Rsym
d
```

GetTD *GetTD*

Description

Function that takes a PAM classification (as returned by ApplyPAM) and the dissimilarity matrix and returns the value of the TD function (sum of dissimilarities between each point and its closest medoid, divided by the number of points). This function is mainly for debugging/internal use.

Usage

```
GetTD(L, dissim_file)
```

Arguments

`L` A list of two numeric vectors, L["med","clasif"], as returned by ApplyPAM (please, consult the help of ApplyPAM for details)

`dissim_file` A string with the name of the binary file that contains the symmetric matrix of dissimilarities. Such matrix should have been generated by CalcAndWriteDissimilarityMatrix.

Value

TD The value of the TD function.

Examples

```
# Synthetic problem: 10 random seeds with coordinates in [0..20]
# to which random values in [-0.1..0.1] are added
M<-matrix(0,100,500)
rownames(M)<-paste0("rn",c(1:100))
for (i in (1:10))
{
 p<-20*runif(500)
 Rf <- matrix(0.2*(runif(5000)-0.5),nrow=10)
 for (k in (1:10))
 {
  M[10*(i-1)+k,]=p+Rf[k,]
 }
}
tmpfile1=paste0(tempdir(),"/pamtest.bin")
tmpdisfile1=paste0(tempdir(),"/pamDL2.bin")
JWriteBin(M,tmpfile1,dtype="float",dmtype="full")
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype="L2",restype="float",nthreads=0)
L <- ApplyPAM(tmpdisfile1,10,init_method="BUILD")
# Final value of sum of distances to closest medoid
GetTD(L,tmpdisfile1)
```

`JMatInfo` *JMatInfo*

Description

Shows in the screen or writes to a file information about a matrix stored in the binary format of package jmatrix

Usage

```
JMatInfo(fname, fres = "")
```

Arguments

fname	String with the file name that contains the binary data.
fres	String with the name of the file to write the information. Default: "" (information is written to the console)

Value

No return value, called for its side effects (writes on screen or creates a file)

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
JMatInfo(tmpfile1)
```

JMatToCsv *JMatToCsv*

Description

Writes a binary matrix in the jmatrix package format as a .csv file. This is mainly for checking/inspection and to load the data from R as read.csv, if the memory of having all data as doubles allows doing such thing.

Usage

```
JMatToCsv(ifile, csvfile, csep = ",", withquotes = FALSE)
```

Arguments

ifile	String with the file name that contains the binary data.
csvfile	String with the file name that will contain the data as csv.
csep	Character used as separator. Default: , (comma)
withquotes	boolean to mark if row and column names in the .csv file must be written surrounded by doble quotes. Default: FALSE

Details

The numbers are written to text with as many decimal places as allowed by its data type (internally obtained with std::numeric_limits<type>::max_digits10)
NOTE ON READING FROM R: to read the .csv files exported by this function you MUST use the R function read.csv (not read.table) AND set its argument row.names to 1, since we always write a first column with the row names, even if the binary matrix does not store them; in this case they are simply "1","2",...

Value

No return value, called for side effects (creates a file)

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
tmpcsvfile1=paste0(tempdir(),"/Rfullfloat.csv")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
JMatToCsv(tmpfile1,tmpcsvfile1)
```

`JWriteBin` *JWriteBin*

Description

Writes a R matrix to a disk file as a binary matrix in the jmatrix format

Usage

```
JWriteBin(M, fname, dtype = "float", dmtype = "full", comment = "")
```

Arguments

`M` The R matrix to be written

`fname` The name of the file to write

`dtype` The data type of the matrix to be written: one of the strings 'short', 'int', 'long', 'float' or 'double'. Default: 'float'

`dmtype` The matrix type: one of the strings 'full', 'sparse' or 'symmetric'. Default: 'full'

`comment` A optional string with the comment to be added as metadata. Default: "" (empty string, no added comment)

Details

Use this function cautiously. Differently to the functions to get one or more rows or columns from the binary file, which book only the memory strictly needed for the vector/matrix and do not load all the binary file in memory, this function books the full matrix in the requested data type and writes it later so with very big matrices you might run out of memory.
Type 'int' is really long int (8-bytes in most modern machines) so using 'int' or 'long' is equivalent.
Type is coerced from double (the internal type of R matrices) to the requested type, which may provoke a loose of precision.
If M is a named-R matrix, row and column names are written as metadata, too.
Also, if you write as symmetric a matrix which is not such, only the lower-diagonal part will be written. The rest of the data will be lost. In this case, if the matrix has row and column names, only row names are written.

Value

No return value, called for side effects (creates a file)

Examples

```
Rf <- matrix(runif(48),nrow=6)
rownames(Rf) <- c("A","B","C","D","E","F")
colnames(Rf) <- c("a","b","c","d","e","f","g","h")
tmpfile1=paste0(tempdir(),"/Rfullfloat.bin")
JWriteBin(Rf,tmpfile1,dtype="float",dmtype="full",comment="Full matrix of floats")
```

NumSilToClusterSil *NumSilToClusterSil*

Description

Takes a silhouette in the form of a NumericVector, as returned by CalculateSilhouette, and returns it as a numeric matrix appropriate to be plotted by the package 'cluster'

Usage

```
NumSilToClusterSil(cl, s)
```

Arguments

cl	The array of classification with the number of the class to which each point belongs to. This number must be in 1..number_of_classes. This function takes something like the L$clasif array which is the second element of the list returned by ApplyPAM
s	The numeric value of the silhouette for each point, with points in the same order as they appear in cl. This is the vector returned by a call to CalculateSilhouette with the same value of parameter cl.

Value

sp A silhouette in the format of the cluster package which is a NumericMatrix with as many rows as points and three columns: cluster, neighbor and sil_width.
Its structure and dimension names are as in package 'cluster', which allows to use it with the silhouette plotting functions of such package
This means you can do library(cluster) followed by plot(NumSilToClusterSil(cl,s)) to get a beatiful plot.

Examples

```
# Synthetic problem: 10 random seeds with coordinates in [0..20]
# to which random values in [-0.1..0.1] are added
M<-matrix(0,100,500)
rownames(M)<-paste0("rn",c(1:100))
for (i in (1:10))
{
 p<-20*runif(500)
 Rf <- matrix(0.2*(runif(5000)-0.5),nrow=10)
 for (k in (1:10))
 {
  M[10*(i-1)+k,]=p+Rf[k,]
 }
}
tmpfile1=paste0(tempdir(),"/pamtest.bin")
JWriteBin(M,tmpfile1,dtype="float",dmtype="full")
tmpdisfile1=paste0(tempdir(),"/pamDL2.bin")
CalcAndWriteDissimilarityMatrix(tmpfile1,tmpdisfile1,distype="L2",restype="float",nthreads=0)
L <- ApplyPAM(tmpdisfile1,10,init_method="BUILD")
sil <- CalculateSilhouette(L$clasif,tmpdisfile1)
sp <- NumSilToClusterSil(L$clasif,sil)
library(cluster)
plot(sp)
```

`ParallelpamSetDebug` *ParallelpamSetDebug*

Description

Sets debugging in parallelpam package to ON (with TRUE) or OFF (with FALSE) for both parts of it.
On package load the default status is OFF.
Setting debugging of any part to ON shows a message. Setting to OFF does not show anything (since debugging is OFF...)

Usage

```
ParallelpamSetDebug(deb = TRUE, debjmat = FALSE)
```

Arguments

`deb`	boolean, TRUE to generate debug messages for the PAM algorithm and silhouette calculation and FALSE to turn them off. Default: true.
`debjmat`	boolean, TRUE to generate debug messages for the jmatrix part inside this package and FALSE to turn them off. Default: false

Value

No return value, called for side effects (internal boolean flag changed)

Examples

```
ParallelpamSetDebug(TRUE,debjmat=TRUE)
ParallelpamSetDebug(TRUE,debjmat=FALSE)
```

Index