The caMassClass Package

February 16, 2008

Version 1.6

Date Feb 01 2007

Title Processing & Classification of Protein Mass Spectra (SELDI) Data

Author Jarek Tuszynski <jaroslaw.w.tuszynski@saic.com>

Maintainer Jarek Tuszynski <jaroslaw.w.tuszynski@saic.com>

Depends R (>= 2.0.0), PROcess, e1071, nnet, rpart, caTools, XML, digest, MASS

Description Functions for processing and classification of protein mass spectra (SELDI) data. Also includes support for mzXML Files.

License The caMassClass Software License, Version 1.0 (See COPYING file or "http://ncicb.nci.nih.gov/download/camassclasslicense.jsp")

R topics documented:

`caMassClass-package`

Processing and Classification of Protein Mass Spectra Data

Description

Functions for processing and classification of protein mass spectra data. Includes support for I/O in mzXML and CSV formats.

Details

Package:	caMassClass
Version:	1.5
Date:	2006-04-11
Depends:	R (>= 2.0.0), PROcess, e1071, nnet, rpart, caTools, XML, digest
License:	The caMassClass Software License, Version 1.0 (See COPYING file or `http://ncicb.nci.nih.gov/download/camassclasslicense.jsp`)
URL:	`http://ncicb.nci.nih.gov/download/index.jsp`

Index of Preprocessing Functions:

`msc.project.read`	Read and Manage a Batch of Protein Mass Spectra
`msc.project.run`	Read and Preprocess Protein Mass Spectra
`msc.preprocess.run`	Preprocessing Pipeline of Protein Mass Spectra
`msc.baseline.subtract`	Baseline Subtraction for Mass Spectra Data
`msc.mass.cut`	Remove Low Mass Portion of the Mass Spectra Data.
`msc.mass.adjust`	Perform Normalization and Mass Drift Adjustment
`msc.peaks.find`	Find Peaks of Mass Spectra
`msc.peaks.clust`	Clusters Peaks of Mass Spectra (low level)
`msc.peaks.align`	Align Peaks of Mass Spectra into Biomarker Matrix
`msc.biomarkers.fill`	Fill Empty Spaces in Biomarker Matrix
`msc.copies.merge`	Merge Multiple Copies of Mass Spectra Samples
`msc.sample.correlation`	Sample Correlation (low level)

Index of Classification Functions:

`msc.classifier.test`	Test a Classifier through Cross-validation
`msc.features.remove`	Remove Highly Correlated Features

msc.features.scale	Scale Classification Data
msc.features.select	Reduce Number of Features Prior to Classification
msc.classifier.run	Train and Test Chosen Classifier.

Index of I/O Functions:

msc.rawMS.read.csv	Read Protein Mass Spectra from CSV files
msc.rawMS.read.mzXML	Read raw Mass Spectra from mzXML Files
msc.rawMS.write.mzXML	Write raw Mass Spectra to mzXML Files
msc.peaks.read.csv	Read Mass Spectra Peaks in CSV Format
msc.peaks.write.csv	Write Mass Spectra Peaks in CSV Format
msc.peaks.read.mzXML	Read peaks heights and positions from mzXML File
msc.peaks.write.mzXML	Write peaks heights and positions to mzXML File
msc.biomarkers.read.csv	Read biomarker matrix in CSV format
msc.biomarkers.write.csv	Write biomarker matrix in CSV format
msc.project.read	Read and Manage a Batch of Protein Mass Spectra Data
read.mzXML	Read mzXML Files into generic data structure
write.mzXML	Write mzXML Files from generic data structure

Author(s)

Jarek Tuszynski <jaroslaw.w.tuszynski@saic.com>

See Also

See additional documentation in "R/library/caMassClass/doc" directory.

Se also Packages: PROcess (http://www.bioconductor.org/packages/bioc/1.8/html/PROcess.html), xcms (http://www.bioconductor.org/packages/bioc/1.8/html/xcms.html), ppc (CRAN).

---

msc.baseline.subtract

Baseline Subtraction for Mass Spectra Data

Description

Perform baseline subtraction on batch of mass spectra data

Usage

```
msc.baseline.subtract(X, ...)
```

Arguments

X	Spectrum data either in matrix format [nFeatures × nSamples] or in 3D array format [nFeatures × nSamples × nCopies]. Row names (rownames(X) store M/Z mass of each row/feature.
...	Parameters to be passed to bslnoff function from **PROcess** library. See details for explanation of breaks, qntl, and bw. Boolean parameter plot can be used to plot results.

Details

Perform baseline subtraction for every sample in a batch of data, using bslnoff function from **PROcess** library. The bslnoff function splits spectrum into breaks number of exponentially growing regions. Baseline is calculated by applying quantile(...,probs=qntl) to each region and smoothing the results using loess(..., span=bw, degree=1) function.

Value

Data in the same format and size as input variable X but with the subtracted baseline.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Part of msc.preprocess.run and msc.project.run pipelines.
- Previous step in the pipeline was msc.project.read and msc.rawMS.read.csv
- Next step in the pipeline is msc.mass.cut
- This function uses bslnoff (from **PROcess** library) which is a single-spectrum baseline removal function implemented using loess function.
- Function rmBaseline (from **PROcess** library) can read all CSV files in directory and remove their baselines.

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run msc.baseline.subtract using 3D input
# this data had baseline removed already so little change is expected
Y = msc.baseline.subtract(X)
avr = mean(abs(X-Y))
cat("Data Size: ", dim(X), " average change :", avr, "\n")
stopifnot(avr<0.3)

# test on data provided in PROcess package (2D input)
# this is "raw" data, so large changes are expected
directory  = system.file("Test", package = "PROcess")
```

```
X = msc.rawMS.read.csv(directory)
Y = msc.baseline.subtract(X, plot=TRUE)
avr = mean(abs(X-Y))
cat("Data Size: ", dim(X), " average change :", avr, "\n")
stopifnot(avr>7.5)
```

`msc.biomarkers.read.csv & msc.biomarkers.write.csv`

Read and Write biomarker matrix in CSV format

Description

Functions to read and write CSV (comma separated values) text files containing biomarkers (aligned peaks) in the format used by Ciphergen's biomarker file, with spectra (samples) as rows, and biomarkers as columns (features).

Usage

```
X = msc.biomarkers.read.csv(fname, mzXML.record=FALSE)
msc.biomarkers.write.csv(X, fname)
```

Arguments

`fname`	either a character string naming a file or a connection.
`X`	biomarker data in form of a 2D matrix (nFeatures $\times$ nSamples) or 3D array (nFeatures $\times$ nSamples $\times$ nCopies. Notice that this data is in format which is a transpose of data in CSV file.
`mzXML.record`	should mzXML record be created to store mata-data (input file names)?

Value

Function `msc.biomarkers.read.csv` returns peak information data frame. See argument `X` above. If `mzXML.record` was set to true than mzXML record with input file names will be attached to `X` as "`mzXML`" attribute.

Function `msc.biomarkers.write.csv` does not return anything.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

`msc.biomarkers.fill`, `msc.rawMS.read.csv`

Examples

```
example("msc.peaks.align", verbose=FALSE) # create biomarkers data
X = Y$Bmrks   #  biomarkers data is stored in variable 'Y$Bmrks'
msc.biomarkers.write.csv(X, "biomarkers.csv")
Y = msc.biomarkers.read.csv("biomarkers.csv", mzXML.record=TRUE)
stopifnot( all(X==Y, na.rm=TRUE) )
mzXML = attr(Y,"mzXML")
strsplit(mzXML$parentFile, '\n')      # show mzXML$parentFile record
file.remove("biomarkers.csv")
```

`msc.biomarkers.fill`

Fill Empty Spaces in Biomarker Matrix

Description

Fill empty spaces (NA's) in biomarker matrix created by `msc.peaks.align`

Usage

```
msc.biomarkers.fill( X, Bmrks, BinBounds, FillType=0.9)
```

Arguments

`X`	Spectrum data either in matrix format [nFeatures × nSamples] or in 3D array format [nFeatures × nSamples × nCopies]. Row names `(rownames(X))` store M/Z mass of each row.
`Bmrks`	biomarker matrix containing one sample per column and one biomarker per row
`BinBounds`	position (mass) of left-most and right-most peak in each bin
`FillType`	how to fill empty spaces in biomarker data? • if `0<=FillType<=1` than fill spaces with `quantile(probs=FillType)`. For example: if `FillType=1/2` than medium will be used, if `FillType=1` than maximum value will be used, if `FillType=0.9` than maximum will be used after discarding 10% of "outliers" • if `FillType<0` than empty spaces will not be filled and NA's will remain • if `FillType==2` than X value closest to the center of the bin will be used • if `FillType==3` empty spaces will be set to zero

Details

This function attempts to correct a problem which is a side-effect of `msc.peaks.align` function. Namely numerous NA's in biomarker data, each time when some peak was found only in some of the samples. `msc.peaks.align` already removed the most problematic features using `SampFrac` variable, but likely a lot of NA's remain and they can cause problem for some classification algorithms.

Value

Data in the same format and size as `Bmrks`

Note

The whole idea of filling spaces in biomarker matrix is a little bit suspect since we are mixing proverbial apples and oranges. However, it might be better than the other options of filling empty spaces with zeros or keeping NA's.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Input comes most likely from: `msc.peaks.align`, or from Ciphergen's software
- Output can be processed further by `msc.copies.merge`
- Biomarkers matrix can be read and written by `msc.biomarkers.read.csv` and `msc.biomarkers.write.cs`
- Function `pk2bmkr` from **PROcess** package also perform similar function.

Examples

```
# load 'X' and 'Y' calculated in example("msc.peaks.align")
example("msc.peaks.align")
nNA = sum(is.na(Y$Bmrk))
cat( "dim(Y$Bmrk)=", dim(Y$Bmrk), "; number of NA's is ", nNA,"\n")
stopifnot(nNA==232)

# run msc.biomarkers.fill
Z = msc.biomarkers.fill( X, Y$Bmrks, Y$BinBounds)
nNA = sum(is.na(Z))
cat( "dim(Z)=", dim(Z), "; number of NA's is ", nNA,"\n")
stopifnot( dim(Z)==c(22, 20, 2) )
stopifnot(nNA==0)

# run msc.biomarkers.fill with other FillType
Z = msc.biomarkers.fill( X, Y$Bmrks, Y$BinBounds, FillType=2)
```

`msc.classifier.run` *Train and Test Chosen Classifier.*

Description

Common interface for training and testing several standard classifiers. Includes feature selection and feature scaling steps. Allows to specify that some test samples are multiple copies of the same sample, and should return the same label.

Usage

```
msc.classifier.run( xtrain, ytrain, xtest, ret.prob=FALSE,
    RemCorrCol=0, KeepCol=0, prior=1, same.sample=NULL,
    ScaleType=c("none", "min-max", "avr-std", "med-mad"),
    method=c("svm", "nnet", "lda", "qda", "LogitBoost", "rpart"), ...)
```

Arguments

`xtrain`	A matrix or data frame with training data. Rows contain samples and columns contain features/variables
`ytrain`	Class labels for the training data samples. A response vector with one label for each row/component of `x`. Can be either a factor, string or a numeric vector.
`xtest`	A matrix or data frame with test data. Rows contain samples and columns contain features/variables
`ret.prob`	if set to TRUE than the a-posterior probabilities for each class are returned as attribute called "probabilities".
`same.sample`	optional parameter which allows to specify that some (or all) test samples have multiple copies which should be used to predict a single label for all of them. Can be either a factor, string or a numeric vector, with unique values for different samples and identical values for copies of the same sample.
`RemCorrCol`	If non-zero than some of the highly correlated columns are removed using `msc.features.remove` function with `ccMin=RemCorrCol`.
`KeepCol`	If non-zero than columns with low AUC are removed. • if `KeepCol` smaller than 0.5 - do nothing • if `KeepCol` in between [0.5, 1] - keep columns with AUC bigger than `KeepCol` • if `KeepCol` bigger than one - keep top "`KeepCol`" number of columns
`ScaleType`	Optional parameter, if provided than following types are recognized • "none" - no scaling is performed • "min-max" - data minimum is mapped to 0 and maximum is mapped to 1 • "avr-std" - data is mapped to zero mean and unit variance • "med-mad" - data is mapped to zero median and unit mad (median absolute deviation)
`prior`	class weights. following types are recognized • `prior==1` - all samples in all classes have equal weight (default) • `prior==2` - all classes have equal weight • `prior` is a vector - a named vector of weights for the different classes, used for asymmetric class sizes.
`method`	classifier to be used. Following ones are recognized (followed by some parameters that could be passed through ...: • "svm" - see `svm` from **e1071** package. Possible parameters: `cost`, `gamma` • "nnet" - see `nnet` from **nnet** package. Possible parameters: `size`, `decay`, `maxit`

- "LogitBoost" - see `LogitBoost` from **caTools** package Possible parameter: `nIter`
- "lda" - see `lda` from **MASS** package. Possible parameters: `method`
- "qda" - see `qda` from **MASS** package. Possible parameters: `method`
- "rpart" - see `rpart` from **rpart** package. Possible parameters: `minsplit`, `cp`, `maxdepth`

`...` Additional parameters to be passed to classifiers. See `method` for suggestions.

Details

This function performs the following steps:

- Remove highly correlated columns and columns with low AUC with `msc.features.select` function
- Scale each feature separately using `msc.features.scale` function
- Train chosen classifier using `xtrain` and `ytrain`
- Predict labels of `xtest` using trained model
- If `same.sample` variable is given than synchronize predicted labels in such a way that all copies of the same sample return the same label.
- Return labels. If `ret.prob=TRUE` then return a-posterior probabilities as well.

Value

Predicted class labels for each sample in `xtest`. If `ret.prob=TRUE` than the a-posterior probabilities of each sample belonging to each class are returned as attribute called "probabilities". The returned probabilities do not take into account `same.sample` variable, used to synchronize predicted labels.

Note

This function is not fully tested and might be changed in future versions

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

References

- "A Practical Guide to Support Vector Classification" by Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin (http://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf)

See Also

- Used by `msc.classifier.test` function.
- Best classifier parameter set can be found by `tune` function from **e1071** package.
- Uses `msc.features.select` and `msc.features.scale` functions.
- Uses variety of classification algorithms: `svm`, `nnet`, `LogitBoost`, `lda`, `qda`, `rpart`

Examples

```
data(iris)
mask  = sample.split(iris[,5], SplitRatio=1/4) # very few points to train
xtrain = iris[ mask,-5]  # use output of sample.split to ...
xtest  = iris[!mask,-5]  # create train and test subsets
ytrain = iris[ mask, 5]
ytest  = iris[!mask, 5]
table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="svm") )
table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="nnet") )
table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="lda") )
table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="qda") )
table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="LogitBoost") )

a=table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="LogitBoost") )
stopifnot(  sum(diag(a))==length(ytrain) )
```

---

msc.classifier.test

Test a Classifier through Cross-validation

Description

Test classifier through cross-validation. Common interface for cross-validation of several standard classifiers. Includes feature selection and feature scaling steps. Allows to specify that some test samples are multiple copies of the same sample, and should return the same label.

Usage

```
msc.classifier.test( X, Y, iters=50, SplitRatio=2/3, verbose=FALSE,
    RemCorrCol=0, KeepCol=0, prior=1, same.sample=NULL,
    ScaleType=c("none", "min-max", "avr-std", "med-mad"),
    method=c("svm", "nnet", "lda", "qda", "LogitBoost", "rpart"), ...)
```

Arguments

X	A matrix or data frame with training/testing data. Rows contain samples and columns contain features/variables
Y	Class labels for the training data samples. A response vector with one label for each row/component of x. Can be either a factor, string or a numeric vector. Labels with 'NA' value signify test data-set.
iters	Number of iterations. Each iteration consist of splitting the data into train and test sets, performing the classification and storing results
SplitRatio	Splitting ratio used to divide available data during cross-validation: • if (0<=SplitRatio<1) then SplitRatio fraction of samples will be used for training and the rest for validation.

- if (SplitRatio==1) leave-one-out cross-validation. All but one samples will used for training, and validation will be done using single sample per iteration.
- if (SplitRatio>1) then SplitRatio number of samples to be used for training and the rest for validation.

RemCorrCol	See msc.classifier.run.
KeepCol	See msc.classifier.run.
ScaleType	See msc.classifier.run.
prior	See msc.classifier.run.
same.sample	See msc.classifier.run.
method	See msc.classifier.run.
verbose	boolean flag turns debugging printouts on.
...	Additional parameters to be passed to classifiers. See method for suggestions.

Details

This function follows standard cross-validation steps:

- Class labels Y are used to divide data X into train set (with known labels) and test set (labels set to NA and will be calculated)
- For number of iterations repeat the following steps of cross-validation:
  - split train data into temporary train and test sets using sample.split function from **caTools** package.
  - train and test the chosen classifier using temporary train and test data sets and msc.classifier.run function
- Calculate the overall performance of the classifier
- Train the classifier using the whole train data set (all labeled samples)
- Use this classifier to predict values of the whole test data set (all samples without label - NA.)

Value

Y	Predicted class labels. If there were any unknown samples in input data, marked by NA's in input Y, than output Y will only hold prediction of those samples, otherwise prediction will be made for all samples.
Res	Holds fraction of correct prediction during cross-validation for each iteration. mean(Res) will give you average accuracy.
Tabl	Contingency table of predictions shows all the input label compared to output labels

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Input comes most likely from `msc.preprocess.run` and/or `msc.project.run` functions.
- Uses `sample.split`, `msc.classifier.run`, `msc.features.select` and `msc.features.scale` functions.
- Best classifier parameter set can be found by `tune` function from **e1071** package.
- Uses variety of classification algorithms: `svm`, `nnet`, `LogitBoost`, `lda`, `qda`, `rpart`

Examples

```
data(iris)
A = msc.classifier.test(iris[,-5],iris[,5], method="LogitBoost", nIter=2)
print(A)
cat("correct classification in",100*mean(A$Res),"+-",100*sd(A$Res),"percent of cases\n")
stopifnot( mean(A$Res)<89 )
```

`msc.copies.merge` *Merge Multiple Copies of Mass Spectra Samples*

Description

Protein mass spectra (SELDI) samples are sometimes scanned multiple times in order to reduce hardware or software based errors. `msc.copies.merge` function is used to merge, concatenate, and/or average all of those copies together in preparation for classification.

Usage

```
msc.copies.merge( X, mergeType, PeaksOnly=TRUE)
```

Arguments

`X`	Spectrum data in 3D array format [nFeatures × nSamples × nCopies]. Row names (`rownames(X)`) store M/Z mass of each row. If X is in matrix format [nFeatures × nSamples] nothing will be done.
`mergeType`	an integer variable in [0,11] range, telling how to merge samples and what to do with bad copies: • 0 - do nothing • add 1 - if all original copies are to be concatenated as separate samples • add 2 - if copies are to be averaged and the average added as a separate sample • add 4 - if for each sample the worst copy is to be deleted • add 8 - if for each sample in case of large differences between copies, a single bad copy of a sample is to be replaced with the best copy. Not to be used with previous option. See details.
`PeaksOnly`	This variable is being passed to function `msc.sample.correlation`. Set it to `TRUE` in case of raw spectra and switch to `FALSE` in case of data where only peaks (biomarkers) are present.

Details

Quality of a sample is measured by calculating for each copy of each sample two variables: inner correlation (average correlation between multiple copies of the same sample) and outer correlation (average correlation between each sample and every other sample within the same copy). Inner correlation measures how similar copies are to each other and outer correlation measures how similar each copy is to everybody else. For example in case of experiment using SELDI technology to distinguish cancerous samples and non-cancerous samples one can assume that most of the proteins present in both cancerous and non-cancerous samples will be the same. In that case one will expect high correlation between samples and even higher correlation between copies of the same sample

if `mergeType/4` (`mergeType %/% 4`) is

- 0 - all copies are kept
- 1 - if inner correlation is smaller than outer correlation, or in other words, if a signature is more similar to other signatures than to other copies of the same signature, than there is some problem with that signature. In that case that bad signature can be replaced with the best copy of the signature.
- 2 - rate each copy of each sample using `score=outer_correlation + inner_correlation` measure. Delete worst copy.

Option 2 is more suitable in case of data with a lot of copies, when we can afford dropping one copy. Option 1 is designed to patch the most serious problems with the data.

There are also four merging options, if `mergeType mod 4` (`mergeType %% 4`) is

- 0 - no merging is done to the data and it is left as 3D array
- 1 - all copies are concatenated `X = cbind(X[,,1], X[,,2], ..., X[,,nCopy])` so they seem as separate samples
- 2 - all copies are averaged `X = (X[,,1] + X[,,2] + ... + X[,,nCopy])/nCopy)`
- 3 - all copies are first averaged and than concatenated with extra average copy `X = cbind(X[,,1], X[,,2], ..., X[,,nCopy], Xavr)`

In preparation for classification one can use multiple copies in several ways: option 2 above improves (one hopes) accuracy of each sample, while options 1 and 3 increase number of samples available during classification. So the choice is: do we want a lot of samples during classification or fewer, better samples?

The best option of `mergeType` depends on kind of data.

- 0 if data has single copy.
- 1+2+4 will produce the largest number of samples since we will keep all the copies and an average of all the copies
- 2+8 will produce single most accurate sample from multiple copies (usually if more than 2 copies are present) since we will delete outliers before averaging all the copies

Value

Return matrix containing features as rows and samples as columns, unless `mergeType` is 0,4, or 8 when no merging is done and data is returned in same or similar format as the input format.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.mass.adjust` or peak finding functions: `msc.peaks.find`, `msc.peaks.align`, and `msc.biomarkers.fill`
- Next step in the pipeline is data classification `msc.classifier.test`
- Uses `msc.sample.correlation`

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run msc.copies.merge
Y = msc.copies.merge(X, 1+2+4)
colnames(Y)
stopifnot( dim(Y)==c(11883,60) )
```

---

msc.features.remove

Remove Highly Correlated Features

Description

Remove Highly Correlated Features. The function checks neighbor features looking for highly correlated ones and removes one of them. Used in order to drop dimensionality of the data.

Usage

```
msc.features.remove(Data, Auc, ccMin=0.9, verbose=FALSE)
```

Arguments

`Data`	Data containing one sample per row and one feature per column.
`Auc`	A measure of usefulness of each column/feature, used to choose which one of two highly correlated columns to remove. Usually a measure of discrimination power of each feature as measured by `colAUC`, student t-test or other method. See details.
`ccMin`	Minimum correlation coefficient of "highly correlated" columns.
`verbose`	Boolean flag turns debugging printouts on.

Details

If `colAUC` was used and there were more than two classes present than `Auc` is a matrix with multiple measurements for each feature. In such a case `Auc = apply(Auc, 2, mean)` is run in order to extract a single measure per feature. If other measures are desired, like `Auc = apply(Auc, 2, max)`, than they should be called beforehand.

Value

Vector of column indexes to be kept.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Used by `msc.classifier.test` and `msc.features.select` functions.
- Uses `colAUC`

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

X = t(X[,,1])
auc = colAUC(X,SampleLabels)
quantile(auc)
cidx = msc.features.remove(X, auc, verbose=TRUE)
Y = X[,cidx]
stopifnot( dim(Y)==c(20, 3516) )
stopifnot( abs(mean(auc)-0.64)<0.01 )
```

`msc.features.scale` *Scale Classification Data*

Description

Scale features of the data to be used for classification. Scaling factors are extracted from each column/feature of the train data-set and applied to both train and test sets.

Usage

```
msc.features.scale( xtrain, xtest, type = c("min-max", "avr-std", "med-mad"))
```

Arguments

xtrain	A matrix or data frame with train data. Rows contain samples and columns contain features/variables
xtest	A matrix or data frame with test data. Rows contain samples and columns contain features/variables
type	Following types are recognized • "min-max" - data minimum is mapped to 0 and maximum is mapped to 1 • "avr-std" - data is mapped to zero mean and unit variance • "med-mad" - data is mapped to zero median and unit mad (median absolute deviation)

Details

Many classification algorithms perform better if input data is scaled beforehand. Some of them perform scaling internally (for example svm), but many don't. For some it makes no difference (for example rpart or LogitBoost).

In case xtrain contains NA values or infinities all non-finite numbers are omitted from scaling parameter calculations.

Value

xtrain	A matrix or data frame with scaled train data.
xtest	A matrix or data frame with scaled test data.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

Used by msc.classifier.test and msc.features.select functions.

Examples

```
library(e1071)
data(iris)
mask   = sample.split(iris[,5], SplitRatio=1/4) # very few points to train
xtrain = iris[ mask,-5]  # use output of sample.split to ...
xtest  = iris[!mask,-5]  # create train and test subsets
ytrain = iris[ mask, 5]
ytest  = iris[!mask, 5]
x = msc.features.scale(xtrain, xtest)
model = svm(x$xtrain, ytrain, scale=FALSE)
print(a <- table(predict(model, x$xtest), ytest) )
model = svm(xtrain, ytrain, scale=FALSE)
print(b <- table(predict(model, xtest), ytest) )
stopifnot( sum(diag(a))<sum(diag(b)) )
```

`msc.features.select` *Reduce Number of Features Prior to Classification*

Description

Select subset of individual features that are potentially most useful for classification.

Usage

```
msc.features.select( x, y, RemCorrCol=0.98, KeepCol=0.6)
```

Arguments

`x`	A matrix or data frame with training data. Rows contain samples and columns contain features/variables
`y`	Class labels for the training data samples. A response vector with one label for each row/component of `x`. Can be either a factor, string or a numeric vector.
`RemCorrCol`	If non-zero than some of the highly correlated columns are removed using `msc.features.remove` function with `ccMin=RemCorrCol`.
`KeepCol`	If non-zero than columns with low AUC are removed. • if `KeepCol` smaller than 0.5 - do nothing • if `KeepCol` in between [0.5, 1] - keep columns with AUC bigger than `KeepCol` • if `KeepCol` bigger than one - keep top `KeepCol` number of columns

Details

This function reduces number of features in the data prior to classification, using following steps:

- calculate AUC measure for each feature using `colAUC`
- remove some of the highly correlated neighboring columns using `msc.features.remove` function.
- remove columns with low AUC

This function finds subset of individual features that are potentially most useful for classification, and each feature is rated individually. However, often set of two or more very poor individual features can produce a superior classifier. So, this function should be used with care. I found it very useful when classifying raw protein mass spectra (SELDI) data, for reducing dimensionality of the data from 10 000's to 100's prior of classification, instead of peak-finding (see `msc.peaks.find`).

Value

Vector of column indexes to be kept.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Used by `msc.classifier.test` function.
- Uses `colAUC`, `msc.features.remove` and `msc.features.scale` functions.

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")
X = t(X[,,1])

cidx = msc.features.select(X, SampleLabels, KeepCol=0.8)
auc  = colAUC(X[,cidx], SampleLabels)
cat(length(cidx),"features were selected out of",ncol(X),
    "; min(auc)=",min(auc),"; mean(auc)=",mean(auc),"\n")
stopifnot( length(cidx)==612, min(auc)>0.8 )

cidx = msc.features.select(X, SampleLabels, KeepCol=400)
auc  = colAUC(X[,cidx], SampleLabels)
cat(length(cidx),"features were selected out of",ncol(X),
    "; min(auc)=",min(auc),"; mean(auc)=",mean(auc),"\n")
stopifnot( length(cidx)==400, min(auc)>0.8 )
```

---

msc.mass.adjust *Perform Normalization and Mass Drift Adjustment for Mass Spectra Data.*

Description

Perform normalization and mass drift adjustment for protein mass spectra (for example SELDI) data. Process also refered to as removal of "phase variation" in MS data by peak alignment, "profile alignment", "mass calibration"

Usage

```
msc.mass.adjust(X, scalePar=2, shiftPar=0.0005, AvrSamp=0)
msc.mass.adjust.calc(X, scalePar=2, shiftPar=0.0005, AvrSamp=0)
msc.mass.adjust.apply(X, shiftX, scaleY, shiftY)
```

Arguments

X	Spectrum data either in matrix format [nFeatures × nSamples] or in 3D array format [nFeatures × nSamples × nCopies]. Row names (rownames(X)) store M/Z mass of each row.
scalePar	Controls scaling (normalization): 1 means that afterwards all samples will have the same mean, 2 means that afterwards all samples will have the same mean and medium (default)
shiftPar	Controls mass adjustment. Shifting sample has to improve correlation by at least that amount to be considered. Designed to prevent shifts based on "improvement" on order of magnitude of machine accuracy. If set to too large will turn off shifting. Default = 0.0005.
AvrSamp	Is used to normalize test set the same way train set was normalized. Test set is processed using AvrSamp array that was one of the outputs from train-set mass-adjustment. See examples.
shiftX	matrix [nSamp × nCopy] - integer number of positions a sample should be shifted to the right (+) or left (-). Output from msc.mass.adjust.calc and input to msc.mass.adjust.apply.
scaleY	matrix [nSamp × nCopy] - multiply each sample in order to normalize it. Output from msc.mass.adjust.calc and input to msc.mass.adjust.apply.
shiftY	matrix [nSamp × nCopy] - subtract this number from scaled sample (if matching medians). Output from msc.mass.adjust.calc and input to msc.mass.adjust.apply.

Details

Mass adjustment assumes that SELDI data has some error associated with inaccuracy of setting the starting point of time measurement (x-axis origin or zero M/Z value). We try to correct this error by allowing the samples to shift a few time-steps to the left or to the right, if that will help with cross-correlation with other samples. The function performs the following steps

- normalize all samples in such a way as to make their means (and optionally medians) the same
- if multiple copies exist than
  - align multiple copies of each sample to each other
  - temporarily merge multiple copies of each sample to create a "super-sample" vector with more features
- align each sample to the mean of all samples
- recalculate mean of all samples and repeat above step

msc.mass.adjust function was split into two parts (one to calculate parameters and one to apply them) in order to give users more flexibility and information about what is done to the data. This split allows inspection, plotting and/or modification of shiftX, shiftY, scaleY parameters before data is modified. For example one can set shiftX to zero to perform normalization without mass adjustment or set shiftY to zero and scaleY to one to perform mass adjustment without normalization. Three function provided are:

- msc.mass.adjust.calc - calculates and returns all the normalization and mass drift adjustment parameters

- `msc.mass.adjust.apply` - performs normalization and mass drift adjustment using pre-calculated parameters
- `msc.mass.adjust` - simple interface version of above 2 functions

Value

Functions `msc.mass.adjust` and `msc.mass.adjust.apply` return modified spectra in the same format and size as `X`. Functions `msc.mass.adjust.calc` returns list containing the following:

`shiftX`	matrix [nSamp × nCopy] - integer number of positions sample should be shifted to the right (+) or left (-)
`scaleY`	matrix [nSamp × nCopy] - multiply each sample in order to normalize it
`shiftY`	matrix [nSamp × nCopy] - subtract this number from scaled sample (if matching mediums)
`AvrSamp`	Use AvrSamp returned from train-set mass-adjustment to process test-set

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

References

Description of more elaborate algorithm for similar purpose can be found in Lin S., Haney R., Campa M., Fitzgerald M., Patz E.; *"Characterizing phase variations in MALDI-TOF data and correcting them by peak alignment"*; Cancer Informatics 2005: 1(1) 32-40

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline is `msc.mass.cut`
- Next step in the pipeline is either `msc.peaks.find` or `msc.copies.merge`

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run on 3D input data using long syntax
out = msc.mass.adjust.calc (X)
Y   = msc.mass.adjust.apply(X, out$ShiftX, out$ScaleY, out$ShiftY)
stopifnot(  mean(out$ShiftX)==-0.15, abs(mean(out$ScaleY)-0.98)<0.01 )

# check what happened to means
Z   = cbind(colMeans(X), colMeans(Y))
colnames(Z) = c("copy 1 before", "copy 2 before", "copy 1 after", "copy 2 after" )
cat("Sample means after and after:\n")
Z
```

```
# check what happen to sample correlation
A = msc.sample.correlation(X, PeaksOnly=TRUE)
B = msc.sample.correlation(Y, PeaksOnly=TRUE)
cat("Mean corelation between two copies of the same sample:\n")
cat(" before: ", mean(A$innerCor)," after: ", mean(B$innerCor), "\n")
cat("Mean corelation between unrelated samples:\n")
cat(" before: ", mean(A$outerCor)," after: ", mean(B$outerCor), "\n")

# run on 2D input data using short syntax
# check what happened to means and medians
Y = msc.mass.adjust(X[,,1], scalePar=2)
Z = cbind(colMeans(X[,,1]), apply(X[,,1],2,median), colMeans(Y), apply(Y,2,median))
colnames(Z) = c("means before", "medians before", "means after", "medians after" )
Z
Y = msc.mass.adjust(X[,,1], scalePar=1)
Z = cbind(colMeans(X[,,1]), apply(X[,,1],2,median), colMeans(Y), apply(Y,2,median))
colnames(Z) = c("means before", "medians before", "means after", "medians after" )
Z

# mass adjustment for train and test sets, where test set is normalized in
# the same way as train set was
Xtrain = X[, 1:10,]
Xtest  = X[,11:20,]
out    = msc.mass.adjust.calc (Xtrain);
Xtrain = msc.mass.adjust.apply(Xtrain, out$ShiftX, out$ScaleY, out$ShiftY)
out    = msc.mass.adjust.calc (Xtest , AvrSamp=out$AvrSamp);
Xtest  = msc.mass.adjust.apply(Xtest , out$ShiftX, out$ScaleY, out$ShiftY)
```

`msc.mass.cut` *Remove Low Mass Portion of the Mass Spectra Data.*

Description

Remove low-mass portion of the protein mass spectra (SELDI) data.

Usage

```
msc.mass.cut( X, MinMass=3000)
```

Arguments

`X`	Spectrum data either in matrix format [nFeatures × nSamples] or in 3D array format [nFeatures × nSamples × nCopies]. Row names (`rownames(X)`) store M/Z mass of each row.
`MinMass`	Minimum mass threshold. All data below that mass will be deleted

Details

Low-mass portion of the protein mass spectra is removed since it is not expected to have any biological information, and it has large enough amplitude variations that can skew normalization process. This function also removes all the masses (features) where the values in all the samples are identical. That happens sometimes when the ends of the samples are set to zero.

Value

Data in the similar format as input variable `X` but likely with fewer features.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.baseline.subtract`
- Next step in the pipeline is `msc.mass.adjust`

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run in 3D input
Y = msc.mass.cut( X, MinMass=3000)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")
stopifnot( nrow(Y)==9377 )

# test on data provided in PROcess package (2D input)
directory  = system.file("Test", package = "PROcess")
X = msc.rawMS.read.csv(directory)
Y = msc.mass.cut( X, MinMass=4000)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")
stopifnot( nrow(Y)==7439 )
```

`msc.peaks.read.csv & msc.peaks.write.csv`

Read and Write Mass Spectra Peaks in CSV Format

Description

Functions to read and write CSV (comma separated values) text files containing peaks in the format used by Ciphergen's peak file.

Usage

```
X = msc.peaks.read.csv(fname, mzXML.record=FALSE)
msc.peaks.write.csv(X, fname)
```

Arguments

X	Peak information. A data-frame in the same format as returned by msc.peaks.find, containing five components: • Spectrum.Tag - sample name of each peak • Spectrum. - sample number of each peak • Peak. - peak number within each sample • Intensity - peak height (intensity) • Substance.Mass - x-axis position, or corresponding mass of the peak measured in M/Z
fname	either a character string naming a file or a connection.
mzXML.record	should mzXML record be created to store mata-data (input file names)?

Value

Function msc.peaks.read.csv returns peak information data frame. See argument X above. If mzXML.record was set to true than mzXML record with input file names will be attached to X as "mzXML" attribute.

Function msc.peaks.write.csv does not return anything.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

msc.peaks.find and msc.peaks.align

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# Find Peaks and save them
Peaks = msc.peaks.find(X) # create peak data
msc.peaks.write.csv(Peaks, "peaks.csv")
Pks = msc.peaks.read.csv("peaks.csv", mzXML.record=TRUE)
stopifnot(Pks==Peaks)
mzXML = attr(Pks,"mzXML")
strsplit(mzXML$parentFile, '\n')      # show mzXML$parentFile record
# Suggestion: inspect 'peaks.csv' using any text editor
file.remove("peaks.csv")
```

`msc.peaks.read.mzXML & msc.peaks.write.mzXML`
Read / write calculated peaks heights and positions to/from mzXML Files

Description

Functions to read and write mzXML files containing peaks stored in the format used by peak finding functions.

Usage

```
msc.peaks.write.mzXML(scans, filename, mzXML=NULL, ...)
msc.peaks.read.mzXML(filename, wipe=TRUE)
```

Arguments

`scans`	Peak information to be stored in mzXML file. A data-frame in the similar to format as returned by `msc.peaks.find`, containing four components: • Spectrum. - sample number of each peak (the same for all peaks from the same sample) • Peak. - peak number within each sample • Intensity - peak height (intensity) • Substance.Mass - x-axis position, or corresponding mass of the peak measured in M/Z
`filename`	character string with name of the file.
`mzXML`	class storing partially parsed mzXML data
`wipe`	Should all scans that were returned be also deleted (wiped) from mzXML record? Set to `TRUE` by default to minimize memory use.
`...`	additional parameters to be passed to `write.mzXML` function (precision)

Details

Functions `read.mzXML` and `write.mzXML` use very general data type to communicate with `mzXML` files. Functions `msc.rawMS.read.mzXML` and `msc.rawMS.write.mzXML` allow passing information using data format specialized for storing peak data.

Value

Function `msc.peaks.read.mzXML` returns data in the same format as `msc.peaks.write.mzXML` input parameter `scans`. In addition, object of type `mzXML` is attached as "`mzXML`" attribute. See `read.mzXML` for details.

Functions `msc.peaks.write.mzXML` do not return anything.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- `read.mzXML`, `write.mzXML` are more general mzXML file reader/writer.
- `msc.rawMS.read.mzXML` & `msc.rawMS.write.mzXML` functions also read/write mzXML file, but use different data format.
- `msc.peaks.read.csv` & `msc.peaks.write.mzXML` function can read/write peak data using CSV files.

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
directory  = system.file("Test", package = "caMassClass")
X = msc.rawMS.read.csv(directory, "IMAC_.*csv", mzXML.record=TRUE)

# Find Peaks and save them
Peaks = msc.peaks.find(X) # create peak data
msc.peaks.write.mzXML(Peaks, "peaks.mzXML", mzXML=attr(X,"mzXML"),
                      precision='64')
Output = msc.peaks.read.mzXML("peaks.mzXML")
stopifnot(Output$Substance.Mass == Peaks$Substance.Mass,
          Output$Intensity      == Peaks$Intensity,
          Output$Spectrum.      == Peaks$Spectrum.,
          Output$Peak.          == Peaks$Peak.)
mzXML = attr(Output,"mzXML")
strsplit(mzXML$parentFile, '\n')      # show mzXML$parentFile record
# Suggestion: inspect 'peaks.mzXML' using any text editor
file.remove("peaks.mzXML")
```

`msc.peaks.align` *Align Peaks of Mass Spectra into a "Biomarker" Matrix*

Description

Align peaks from multiple protein mass spectra (SELDI) samples into a single "biomarker" matrix

Usage

```
msc.peaks.align(Peaks, SampFrac=0.3, BinSize=c(0.002, 0.008), ...)
msc.peaks.alignment(S,  M,  H, Tag=0, SampFrac=0.3, BinSize=c(0.002, 0.008), ...)
```

Arguments

Peaks	Peak information. Could have two formats: a filename where to find the data, or the data itself. In the first case, Peaks is string containing path to a file saved by msc.peaks.find, getPeaks (from **PROcess** package), or by other software. In the second case, it is a data-frame in the same format as returned by msc.peaks.find. A third way to pass the same input data is through use of S, M, H and Tag variables (described below) used by msc.peaks.alignment function.
S	Peak sample number. Unique number of the sample the peak belongs to. Likely to come from Peaks$Spectrum..
M	Peak center mass. Position of the peak on the x-axis. Likely to come from Peaks$Substance.Mass.
H	Peak height. Likely to come from Peaks$Intensity.
Tag	Peak sample name. Unique name of the sample the peak belongs to. Likely to come from Peaks$Spectrum.Tag. Optional since is used only to set column-names of output data.
SampFrac	After peak alignment, bins with fewer peaks than SampFrac*nSamp are removed.
BinSize	Upper and lower bound of bin-sizes, based on expected experimental variation in the mass (m/z) values. Size of any bin is measured as (R-L)/mean(R,L) where L and R are masses (m/z values) of left and right boundaries. All resulting bin sizes will all be between BinSize[1] and BinSize[2]. Since SELDI data is often assumed to have ± 3% mass drift than a good bin size is twice that number (0.006). Same as BinSize variable in msc.peaks.clust, except for default.
...	Two additional parameters that can be passed to msc.peaks.clust are mostly for expert users fine-tuning the code: • tol - gaps bigger than tol*max(gap) are assumed to be the same size as the largest gap. See details. • verbose - boolean flag turns debugging printouts on.

Details

Two interfaces were provided to the same function:

- msc.peaks.alignment is a lower level function with more detailed inputs and outputs. Possibly easier to customize for other purposes than processing SELDI data.
- msc.peaks.align is a higher level function with simpler interface customized for processing SELDI data.

This function aligns peaks from different samples into bins in such a way as to satisfy constraints in following order:

- bin sizes are in between BinSize[1] and BinSize[2]
- no two peaks from the same sample are present in the same bin

- bins are split in such a way as to minimize bin size and maximize spaces between bins
- if there are multiple, equally good, ways to split a bin than bin is split in such a way as to minimize number of repeats on each smaller sub-bin

The algorithm used does the following:

- Store mass and sample number of each peak into an array
- Concatenate arrays from all samples and sort them according to mass
- Group sets of peaks into subsets (bins). Each subset will consist of peaks from different spectra that have similar mass. That is done by puting all peaks into a single bin and recursively going through the following steps:
  - Check size of the current bin: if it is too small than we are done, if it is too big than it will be split and if it is already in the desired range than it will be split only if multiple peaks from the same sample are present.
  - If bin needs to be split than find the biggest gap between peaks
  - If multiple gaps were found with the same size as the largest gap (or within `tol` tolerance from it) than minimizes number of multiple peaks from the same sample after cut
  - Divide the bin into two sub-bins: to the left and to the right of the biggest gap
  - Recursively repeat the above four steps for both sub-bins
- Store peaks into 2D array (bins by samples)
- Remove bins with fewer peaks than `SampFrac*nSamp`

The algorithm for peak alignment is described as recursive algorithm but the actual implementation uses internal stack, instead in order to increase speed.

Value

`Bmrks`	Biomarker matrix containing one sample per column and one biomarker per row. If a given sample does not have a peak in some bin than `NA` is inserted.
`BinBounds`	Mass of left-most and right-most peak in the bin

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

References

The initial version of this function started as implementation of algorithm described on webpage of Virginia Prostate Center (at Virginia Medical School) documenting their PeakMiner Software. See `http://www.evms.edu/vpc/seldi/peakminer.pdf`

See Also

- Input comes most Likely from: `msc.peaks.find`, `getPeaks` (from **PROcess** package), or Ciphergen's software
- Output can be processed further by: `msc.biomarkers.fill` or `msc.copies.merge`
- Part of `msc.preprocess.run` pipeline

- Uses msc.peaks.clust function to do most of the work
- Uses msc.peaks.read.csv function to read peak file
- Uses msc.biomarkers.write.csv function to save results
- Function pk2bmkr from **PROcess** package performs similar function.

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# Find and Align peaks
Peaks = msc.peaks.find(X)
cat(nrow(Peaks), "peaks were found in", Peaks[nrow(Peaks),2], "files.\n")
Y = msc.peaks.align(Peaks)
print( t(Y$Bmrks) , na.print=".",  digits=2)
stopifnot( dim(Y$Bmrks)==c(22, 40) )
```

---

msc.peaks.clust *Clusters Peaks of Mass Spectra*

Description

Clusters peaks from multiple protein mass spectra (SELDI) samples

Usage

```
msc.peaks.clust(dM, S, BinSize=c(0,sum(dM)), tol=0.97, verbose=FALSE)
```

Arguments

S	Peak sample number, used to identify the spectrum the peak come from.
dM	Distance between sorted peak positions (masses, m/z).
BinSize	Upper and lower bound of bin-sizes, based on expected experimental variation in the mass (m/z) values. Size of any bin is measured as (R-L)/mean(R,L) where L and R are masses (m/z values) of left and right boundaries. All resulting bin sizes will be between BinSize[1] and BinSize[2]. Default is c(0,sum(dM)) which ensures that no BinSizes is not being used.
tol	gaps bigger than tol*max(gap) are assumed to be the same size as the largest gap. See details.
verbose	boolean flag turns debugging printouts on.

Details

This is a low level function used by `msc.peaks.alignment` and not intended to be directly used by many users. However it might be useful for other code developers. It clusters peaks from different samples into bins in such a way as to satisfy constraints in following order:

- bin sizes are in between `BinSize[1]` and `BinSize[2]`
- no two peaks from the same sample are present in the same bin
- bins are split in such a way as to minimize bin size and maximize spaces between bins
- if there are multiple, equally good, ways to split a bin than bin is split in such a way as to minimize number of repeats on each smaller sub-bin

Value

The output is binary array of the same size as `dM` and `S` where left boundaries of each clusters-bin (biomarker) are marked

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

References

The initial version of this function started as implementation of algorithm described on webpage of Virginia Prostate Center (at Virginia Medical School) documenting their PeakMiner Software. See http://www.evms.edu/vpc/seldi/peakminer.pdf

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.peaks.find`
- Next step in the pipeline is `msc.peaks.align` and `msc.biomarkers.fill`
- Part of `msc.peaks.align` function

Examples

```
# example with simple made up data (18 peaks, 3 samples)
M = c(1,5,8,12,17,22, 3,5,7,11,14,25, 1, 5, 7,10,17,21) # peak position/mass
S = rep(1:3, each=6)                # peak's sample number
 idx = sort(M, index=TRUE)$ix       # sort peaks by mass
M   = M[idx]                        # sorted mass
S   = S[idx]                        # arrange sample numbers in the same order
bin = msc.peaks.clust(diff(M), S, verbose=TRUE)
rbind(S,M,bin)                      # show results

# use the results to align peaks into biomarkers matrix
Bmrks = matrix(NA,sum(bin),max(S)) # init feature (biomarker) matrix
bin   = cumsum(bin)                # find bin numbers for each peak in S array
for (j in 1:length(S))             # Bmrks usually store height H of each peak
  Bmrks[bin[j], S[j]] = M[j];      # but in this example it will be mass
```

```
Bmrks
stopifnot( dim(Bmrks)==c(7,3) )
stopifnot( sum(is.na(Bmrks[5,]))==2 )
```

`msc.peaks.find` *Find Peaks of Mass Spectra*

Description

Find Peaks in a Batch of Protein Mass Spectra (SELDI) Data.

Usage

```
msc.peaks.find(X, SNR=2, span=c(81,11), zerothresh=0.9)
```

Arguments

`X`	Spectrum data either in matrix format [nFeatures × nSamples] or in 3D array format [nFeatures × nSamples × nCopies]. Row names (`rownames(X)`) store M/Z mass of each row.
`SNR`	signal to noise ratio (z-score) criterion for peak detection. Similar to `SoN` variable in `isPeak` from **PROcess** package.
`span`	two moving window widths. Smaller one will be used for smoothing and local maxima finding. Larger one will be used for local variance estimation. Similar to `span` and `sm.span` variables in `isPeak` from **PROcess** package.
`zerothresh`	Intensity threshold criterion for peak detection. Positive numbers in range [0,1), like default 0.9, will be used to calculate a single threshold used for all samples using `quantile(X,zerothresh)` equation. Negative numbers in range (-1, 0) will be used to calculate threshold for each single sample `i` using `quantile(X[i,],-zerothresh)`. Similar to `zerothrsh` variable in `isPeak` from **PROcess** package.

Details

Peak finding is done using the following algorithm:

```
x      =   X[j,]
thresh =   if(zerothresh>=0) quantile(X,zerothresh) else quantile(x,-zerothresh)
sig    =   runmean(x, span[2])
rMax   =   runmax (x, span[2])
rAvr   =   runmed (x, span[1])
rStd   =   runmad (x, span[1], center=rAvr)
peak   =   (rMax == x) & (sig > thresh) & (sig-rAvr > SNR*rStd)
```

What means that a peak have to meet the following criteria to be classified as a peak:

- be a local maxima in `span[2]` neighborhood

- smoothed sample (sig) is above user defined threshold zerothresh
- locally calculated z-score (see http://mathworld.wolfram.com/z-Score.html) of the signal is above user defined signal-to-noise ratio

It is very similar to the isPeak and getPeaks functions from **PROcess** library (ver 1.3.2) written by Xiaochun Li. For example getPeaks(X, PeakFile, SoN=SNR, span=span[1], sm.span=span[2], zerothrsh=zerothresh, area.w=0.003, ratio=0) would give very similar results as msc.peaks.find the differences include: speed (msc.peaks.find uses much faster C-level code), different use of signal-to-noise-ratio variable, and msc.peaks.find does not do or use area calculations.

Value

A data frame, in the same format as data saved in peakinfofile, have five components:

Spectrum.Tag	sample name of each peak
Spectrum.	sample number of each peak
Peak.	peak number within each sample
Intensity	peak height (intensity)
Substance.Mass	x-axis position, or corresponding mass of the peak measured in M/Z, which were extracted from row names of the X matrix.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Part of msc.preprocess.run and msc.project.run pipelines.
- Previous step in the pipeline was msc.mass.adjust
- Functions msc.peaks.align or pk2bmkr can be used to align peaks from different samples in order to find biomarkers.
- Peak data can be read and writen by msc.peaks.read.csv and msc.peaks.write.csv.
- Functions isPeak and getPeaks from **PROcess** package are very similar.
- Uses runmax, runmean, runmed, runmad functions.

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")
Peaks = msc.peaks.find(X) # Find Peaks
cat(nrow(Peaks), "peaks were found in", Peaks[nrow(Peaks),2], "files.\n")
stopifnot( nrow(Peaks)==823 )

# work directly with data from the input files
directory  = system.file("Test", package = "caMassClass")
```

```
X = msc.rawMS.read.csv(directory, "IMAC_normal_.*csv")
Peaks = msc.peaks.find(X) # Find Peaks
cat(nrow(Peaks), "peaks were found in", Peaks[nrow(Peaks),2], "files.\n")
stopifnot( nrow(Peaks)==424 )
```

`msc.preprocess.run` *Preprocessing Pipeline of Protein Mass Spectra*

Description

Pipeline for preprocessing protein mass spectra (SELDI) data before classification.

Usage

```
msc.preprocess.run ( X, mzXML=NULL,
    baseline.removal = 0,
      breaks=200, qntl=0, bw=0.005,                    # bslnoff
    min.mass = 3000,                                   # msc.mass.cut
    mass.drift.adjustment = 1,
      shiftPar=0.0005,                                 # msc.mass.adjust
    peak.extraction = 0,
     PeakFile=0, SNR=2, span=c(81,11), zerothresh=0.9, # msc.peaks.find
     BmrkFile=0, BinSize=c(0.002, 0.008), tol=0.97,    # msc.peaks.align
     FlBmFile=0, FillType=0.9,                         # msc.biomarkers.fill
    merge.copies = 1,                                  # msc.copies.merge
    verbose = TRUE)
```

Arguments

`X`	Spectrum data either in matrix format [nFeatures × nSamples] or in 3D array format [nFeatures × nSamples × nCopies]. Row names `(rownames(X)` store M/Z mass of each row.
`mzXML`	optional record of experimental setup and processing so far, beeing prepared for possible output as mzXML file.
`baseline.removal`	Remove baseline from each spectrum? (boolean or 0/1 integer). See function `msc.baseline.subtract` and `bslnoff` from **PROcess** library for other parameters that can be passed: **`breaks`**, **`qntl`** and **`bw`**.
`min.mass`	Cutting place when removing data corresponding to low masses (m/z). See function `msc.mass.cut` for details.
`mass.drift.adjustment`	Controls mass drift adjustment and scaling. If 0 than no mass adjustment or scaling will be performed; otherwise, it is passed to `msc.mass.adjust` function as `scalePar`. Because of that: 1 means that afterwards all samples will have the same mean, 2 means that afterwards all samples will have the same mean and medium. See function `msc.mass.adjust` for details and additional parameter **`shiftPar`** that can be passed.

`peak.extraction`
: Perform peak extraction and alignment, or keep on working with the raw spectra? (boolean or 0/1 integer). See following functions for other parameters that can be passed:
  - `msc.peaks.find` - see parameters: **`PeakFile`**, **`SNR`**, **`span`** and **`zerothresh`**
  - `msc.peaks.align` - see parameters: **`BmrkFile`**, **`BinSize`** and **`tol`**
  - `msc.biomarkers.fill` - see parameters: **`FlBmFile`** and **`FillType`**

  Especially filenames to store intermediate results.

`PeakFile`
: Optional filename, storing peak finding results. If provided than CSV file will be created in the same format as Ciphergen's peak-info file, with following columns of data: "Spectrum.Tag", "Spectrum.", "Peak.", "Intensity" and "Substance.Mass".

`BmrkFile`
: Optional filename, storing peak alignment results. If provided than CSV file will be created in the same format as Ciphergen's biomarker file, with spectra (samples) as rows, and biomarkers as columns (features).

`FlBmFile`
: Optional filename, storing results of `msc.biomarkers.fill`. If provided than CSV file will be created in the same format as Ciphergen's biomarker file, with spectra (samples) as rows, and biomarkers as columns.

`merge.copies`
: In case multiple copies of data exist should they be merged and how? Passed to `msc.copies.merge` function as `mergeType` variable. See that function for more details.

`verbose`
: Boolean flag turns debugging printouts on.

`breaks`
: parameter to be passed to `bslnoff` function from **PROcess** library by `msc.baseline.subtract`

`qntl`
: parameter to be passed to `bslnoff` function from **PROcess** library by `msc.baseline.subtract`

`bw`
: parameter to be passed to `bslnoff` function from **PROcess** library by `msc.baseline.subtract`

`shiftPar`
: parameter to be passed to `msc.mass.adjust`

`SNR`
: parameter to be passed to `msc.peaks.find`

`span`
: parameter to be passed to `msc.peaks.find`

`zerothresh`
: parameter to be passed to `msc.peaks.find`

`BinSize`
: parameter to be passed to `msc.peaks.align`

`tol`
: parameter to be passed to `msc.peaks.align`

`FillType`
: parameter to be passed to `msc.biomarkers.fill`

Details

Function containing several pre-processing steps preparing protein mass spectra (SELDI) data for classification. This function is a "pipeline" performing several operations, all of which do not need class label information. Any and all steps are optional and can be skipped:

- Remove baseline from each spectrum, using `msc.baseline.subtract` and `bslnoff` from **PROcess** library.
- Remove data corresponding to low masses (m/z), using `msc.mass.cut`.
- Adjust for mass drift and normalize data, using `msc.mass.adjust`.
- Find peaks and align them into "biomarker" matrix, using `msc.peaks.find`, `msc.peaks.align` and `msc.biomarkers.fill`.
- Merge multiple copies of data, using `msc.copies.merge`.

Value

Return matrix containing features as rows and samples as columns, unless `merge.copies` was 0,4, or 8 when no merging is done and data is returned in same or similar format as the input format [nFeatures × nSamples × nCopies]. Row names `(rownames(X)` store M/Z mass of each row. If `mzXML` input argument was not null than updated version of mzXML record will be outputted as "`mzXML`"attribute of `X`.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Input data likely come from `msc.project.read` or `msc.rawMS.read.csv` functions
- As mentioned above function uses the following lower level functions: `msc.baseline.subtract`, `bslnoff` from **PROcess** library, `msc.mass.cut`, `msc.mass.adjust`, `msc.peaks.find`, `msc.peaks.align`, `msc.biomarkers.fill`, and `msc.copies.merge`.
- Output data can be latter used for classification using `msc.classifier.test` function

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata") # load data: X & mzXML

# run preprocess with peak extraction
Y = msc.preprocess.run(X, mzXML=mzXML, peak.extraction=1,
  PeakFile="peaks_IMAC.mzXML", BmrkFile="bmrk_IMAC.csv")
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")
stopifnot( dim(Y)==c(25, 40) )        # make sure it is what's expected
YmzXML = attr(Y, "mzXML")
strsplit(YmzXML$dataProcessing, '\n') # show mzXML$dataProcessing record
# inspect by hand output files: "peaks_IMAC.mzXML" & "bmrk_IMAC.csv"

# run preprocess with no peak extraction
Y = msc.preprocess.run(X, mzXML=mzXML)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")
stopifnot( dim(Y)==c(9377, 40) )      # make sure it is what's expected
YmzXML = attr(Y, "mzXML")
strsplit(YmzXML$dataProcessing, '\n') # show mzXML$dataProcessing record
```

`msc.project.read` *Read and Manage a Batch of Protein Mass Spectra*

Description

Read and manage a batch of protein mass spectra (SELDI) files where files could contain multiple spectra taken from the same sample, or multiple experiments performed on the same sample.

Usage

```
msc.project.read(ProjectFile, directory.out=NULL)
```

Arguments

`ProjectFile`	Path and name of text file in Excel's CSV format storing information about a batch of Mass Spectra data files. Alternative input format is a table equivalent to such CSV file. See details.
`directory.out`	Optional character vector with name of directory where output files will be saved. Use "/" slashes in directory name. By default the directory containing `ProjectFile` and all Mass Spectra files is used, and this argument is provided in case that directory is read-only and user have to choose a different directory.

Details

Function `msc.project.read` allows to user to manage large batches of Mass Spectra files, especially when multiple copies of each sample are present. The `ProjectFile` contains all the information about the project. An example format might be:

Name,	**Class**,	**IMAC1**,	**IMAC2**,	**WCX1**,	**WCX2**
r0008,	1,	Nr/imac_r0008.csv,	Nr/imac_r0008(2).csv,	Nr/wcx_r0008.csv,	Nr/wcx_r0008(2).csv
r0012,	1,	Nr/imac_r0012.csv,	Nr/imac_r0012(2).csv,	Nr/wcx_r0012.csv,	Nr/wcx_r0012(2).csv
r0014,	1,	Nr/imac_r0014.csv,	Nr/imac_r0014(2).csv,	Nr/wcx_r0014.csv,	Nr/wcx_r0014(2).csv
r0021,	2,	Ca/imac_r0021.csv,	Ca/imac_r0021(2).csv,	Ca/wcx_r0021.csv,	Ca/wcx_r0021(2).csv
r0022,	2,	Ca/imac_r0022.csv,	Ca/imac_r0022(2).csv,	Ca/wcx_r0022.csv,	Ca/wcx_r0022(2).csv
r0024,	2,	Ca/imac_r0024.csv,	Ca/imac_r0024(2).csv,	Ca/wcx_r0024.csv,	Ca/wcx_r0024(2).csv
r0027,	2,	Ca/imac_r0027.csv,	Ca/imac_r0027(2).csv,	Ca/wcx_r0027.csv,	Ca/wcx_r0027(2).csv

`ProjectFile` always has the following format:

- column 1 - unique name for each sample - Those names will be used in the program to identify the samples
- column 2 - class label for each sample - in the classification part of the code those labels will be used as a response vector (target values). Usually a factor for classification, but could be a unique number for regression.
- columns 3+ - file path (from `directory`) for each file in the project. If `ProjectFile` has more than 3 columns than multiple copies of the same sample are present. In that case column labels (IMAC1, IMAC2, WCX1, WCX2) become important, since they distinguish between equivalent copies taken under the same conditions and copies taken under different conditions. In our example both kinds of copies exist: files in columns IMAC1 and IMAC2 contain two copies of spectra collected using Ciphergen's IMAC ProteinChip array and files in columns WCX1 and WCX2 used WCX array. The labels of those columns are expected to use letters as labels for different copies and numbers to mark multiple identical copies.

File names in `ProjectFile` could be compressed using zip and gzip file compression. They can also be saved in CSV or in mzXML file formats. For example if individual file name is in the format:

- "dir/a.csv" - uncompressed file 'a.csv' in directory 'dir'
- "dir/b.zip/a.csv" - file 'a.csv' within zipped file 'b.zip'
- "dir/a.csv.gz" - gziped individual file
- "dir/a.mzxml/3" - sample number 3 from file 'a.mzxml'

Value

List of .Rdata files storing data that was just read. Each file contains either 2D data (if only one copy of the the data existed) or 3D data (if multiple copies of the data existed). Multiple files are produced if multiple experiments were performed under different conditions. In above example two files will be produced: Data_IMAC.Rdata and Data_WCX.Rdata.

Each file will contain the following objects: `X`, `SampleLabels`, `mzXML`. At the moment, matadata related to the individual scans (stored in `mzXML$scan`) is stored only for single copy data.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- `msc.rawMS.read.csv` is a lower level function useful for data that does not have multiple copies,
- `msc.preprocess.run` is usually used to process output of this function.
- `read.files` from **PROcess** library can read a single SELDI file and `rmBaseline` can read in a directory of files and subtract their baselines.
- `ppc.read.raw.batch` and `ppc.read.raw.nobatch` from **ppc** library can also read SELDI files, assuming correct directory structure.

Examples

```
#================================================
# test reading project file with only CVS files
#================================================
# find name of example project file
directory = system.file("Test", package = "caMassClass") # input directory
ProjectFile = file.path(directory,"InputFiles.csv")      # full name
# read $ save the project data
FileName1 = msc.project.read(ProjectFile, '.')
cat("File ",FileName1," was created\n")
# load and inspect the project data
load(FileName1)
stopifnot( dim(X)==c(11883,20,2) )     # make sure it is what's expected
strsplit(mzXML$parentFile, '\n')       # show mzXML$parentFile record
X1 = X                                 # make a copy of X for future use

#================================================
# test reading project file with mzXML files
#================================================
# save X in mzXML format
```

```
msc.rawMS.write.mzXML(X, "rawMS32.mzXML",  precision="32") # save X as mzXML
# create new project table
ProjectTable = c(colnames(X), SampleLabels, paste("rawMS32.mzXML", 1:40, sep='/'))
dim(ProjectTable) = c(20,4)
colnames(ProjectTable) = c("SampleName", "Class", "Temp1", "Temp2")
print(ProjectTable)
# read $ save the project data
FileName2 = msc.project.read(ProjectTable, '.')
cat("File ",FileName2," was created\n")
# compare results
load(FileName2) # load data: X & SampleLabels
stopifnot(max(abs(X-X1))<1e-5)
file.remove(FileName2) # delete temporary files
```

`msc.project.run` *Read and Preprocess Protein Mass Spectra*

Description

Read and preprocess protein mass spectra (SELDI) files where files could contain multiple copies of spectra taken from the same sample, or spectra from multiple experiments performed on the same sample.

Usage

```
msc.project.run(ProjectFile, directory.out=NULL, verbose = TRUE, ...)
```

Arguments

`ProjectFile`	path and name of text file in Excel's CSV format which is used to store information about a batch of Mass Spectra data files. See details.
`directory.out`	Optional character vector with name of directory where output files will be saved. Use "/" slashes in directory name. By default the directory containing `ProjectFile` and all Mass Spectra files is used, and this argument is provided in case that directory is read-only and user have to choose a different directory.
`verbose`	boolean flag turns debugging printouts on.
`...`	parameters to be passed to `msc.preprocess.run`

Details

High level processing of protein mass spectra (SELDI) data. `msc.project.read` supports projects with multiple sets of spectra taken under different experimental condition. Those sets will be called *batches*. With that in mind, following steps are performed:

- msc.project.read(ProjectFile, directory) is called which reads and saves different batches of mass spectra (SELDI) data into separate files. List of those files is saved in temporary "RInputFiles.csv" file. In future calls to msc.project.run, if above file exist than msc.project.read is not called again.
- Each batch of data is loaded and preprocessed by calls to msc.preprocess.run. All the required parameters have to be passed through "..." mechanism.
- In case of multiple batches of data results are rbinded

Value

X	Spectrum data either in matrix format [nFeatures × nSamples]. Row names (rownames(X) store M/Z mass of each row merged with batch name
SampleLabels	Class label for each sample as read from msc.project.read
SameSample	array of the same length as number of columns in X indicating samples that are multiple copies that came from the same physical sample, and should be the same, if not for noise.
mxXML	Experiment information in mzXML file format. Included only if input data was read from mzXML files, and NULL otherwise.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

msc.project.read, msc.preprocess.run

Examples

```
directory  = system.file("Test", package = "caMassClass")
ProjectFile = file.path(directory,"InputFiles.csv")
Data = msc.project.run(ProjectFile, '.',
    baseline.removal=0, mass.drift.adjustment=1, min.mass=3000,
    peak.extraction=1, merge.copies=7, shiftPar=0.0004)
stopifnot( dim(Data$X)==c(25,60) )
strsplit(Data$mzXML$parentFile, '\n')      # show mzXML$parentFile record
strsplit(Data$mzXML$dataProcessing, '\n')  # show mzXML$dataProcessing record
```

---

msc.rawMS.read.csv *Read Protein Mass Spectra from CSV files*

Description

Read multiple protein mass spectra (SELDI) files, listed in FileList, from a given directory and combine them into a single data structure. Files are in CSV format, possibly compresses. Data is stored as a matrix one file per column.

Usage

```
msc.rawMS.read.csv(directory=".", FileList="\.csv", mzXML.record=FALSE)
```

Arguments

directory	a character vector with name of directory where all the files can be found. Use "/" slashes in directory name. The default corresponds to the working directory getwd().
FileList	List of files to read. List can be in the following formats: • single string - a regular expression (see regex) to be used in selecting files to read, for example "\.csv" • list - list of file names to be read The last format also support file zip and gzip file compression. For example if individual file name is in the format: • "dir/a.csv" - uncompressed file 'a.csv' in directory 'dir' • "dir/b.zip/a.csv" - file 'a.csv' within zipped file 'b.zip' • "dir/a.csv.gz" - gziped individual file
mzXML.record	should mzXML record be created to store mata-data (input file names)?

Details

All files should be in Excel's CSV format (table in text format: 1 row per line, comma delaminated columns). Each file is assumed to have two columns, in case of SELDI data: column 1 (x-axis) is mass/charge (M/Z), and column 2 (y-axis) is spectrum intensity. All files are assumed to have identical first (M/Z) column.

Value

Data structure containing all the data read from the files, in form of a 2D matrix (nFeatures $\times$ nSamples). If mzXML.record was set to true than mzXML record with input file names will be attached to X as "mzXML" attribute.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Part of msc.project.run pipeline.
- msc.project.read gives user much more flexibility in defining the meaning of the data to be read.
- msc.preprocess.run is often used as a next step in the process
- read.files from **PROcess** library can read a single SELDI file and rmBaseline can read in a directory of files and subtract their baselines.
- ppc.read.raw.batch and ppc.read.raw.nobatch from **ppc** library can also read SELDI files, assuming correct directory structure.

Examples

```
# example of mode "single string" FileList
directory  = system.file("Test", package = "caMassClass")
X = msc.rawMS.read.csv(directory, "IMAC_normal_.*csv")
stopifnot ( dim(X) == c(11883, 20) ) # make sure it is what's expected

# example of explicit 1D FileList
ProjectFile = file.path(directory,"InputFiles.csv")
FileList = read.csv(file=ProjectFile, comment.char = "")
FileList[,3]
X = msc.rawMS.read.csv(directory, FileList=FileList[,3], mzXML.record=TRUE)
stopifnot ( dim(X) == c(11883, 20) ) # make sure it is what's expected
mzXML = attr(X,"mzXML")
strsplit(mzXML$parentFile, '\n')      # show mzXML$parentFile record

# example using data provided in PROcess package
directory  = system.file("Test", package = "PROcess")
X = msc.rawMS.read.csv(directory)
msc.baseline.subtract(X, plot=TRUE) # used here to plot results
dim(X)
```

---

msc.rawMS.read.mzXML & msc.rawMS.write.mzXML

Read / write raw Mass Spectra to/from mzXML Files

Description

Read / write raw protein mass spectra to/from mzXML Files

Usage

```
msc.rawMS.write.mzXML(scans, filename, mzXML=NULL, ...)
msc.rawMS.read.mzXML(input, scanIdx=NULL, wipe=TRUE)
```

Arguments

scans	data to be stored in mzXML file, in form of a 2D matrix (nFeatures × nSamples) or 3D array (nFeatures × nSamples × nCopies).
filename	character string with name of the file (connection)
mzXML	class storing partially parsed mzXML data
input	Either mzXML object, or character string with name of the file (connection)
scanIdx	List of scans to return. Optional. By default all will be returned, but one can choose only a subset using this argument.
wipe	Should all scans that were returned be also deleted (wiped) from mzXML record? Set to TRUE by default to minimize memory use.
...	additional parameters to be passed to write.mzXML function

Value

Function `msc.rawMS.read.mzXML` returns data in the matrix format (nFeatures × nSamples) `rownames(scan)` storing masses (M/Z) of each feature. In addition, object of type `mzXML` is attached as "`mzXML`" attribute. See `read.mzXML` for details.

Functions `msc.rawMS.write.mzXML` do not return anything.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- `read.mzXML`, `write.mzXML` are more general mzXML file reader/writer.
- `msc.peaks.read.mzXML`, `msc.peaks.write.mzXML` functions also read/write mzXML file, but use different data format.
- `msc.rawMS.read.csv` function can read raw MS files from CSV files.

Examples

```
# load "Data_IMAC.Rdata" file containing raw MS spectra 'X'
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# save raw MS data as mzXML using 32-bit precision
msc.rawMS.write.mzXML(X, "rawMS32.mzXML",  precision="32")
Y = msc.rawMS.read.mzXML("rawMS32.mzXML")
dim(Y) = c(nrow(Y), ncol(Y)/2, 2)
stopifnot(all.equal(X,Y, tolerance=1e-5, check.attributes = FALSE)) # about the same

# save raw MS data as mzXML using 64-bit precision
msc.rawMS.write.mzXML(X, "rawMS64.mzXML",  precision="64")
Y = msc.rawMS.read.mzXML("rawMS64.mzXML")
dim(Y) = c(nrow(Y), ncol(Y)/2, 2)
stopifnot(X==Y)                          # exactly the same

# Suggestion: inspect 'rawMS32.mzXML' and 'rawMS64.mzXML' using a text editor
file.remove("rawMS32.mzXML") # delete temporary files
file.remove("rawMS64.mzXML")
```

`msc.sample.correlation` *Sample Correlation*

Description

Calculates correlations between different samples and correlations between different copies of the same sample

Usage

```
msc.sample.correlation(X, PeaksOnly=FALSE)
```

Arguments

X	Spectrum data either in matrix format [nFeatures × nSamples] or in 3D array format [nFeatures × nSamples × nCopies]. Row names (rownames(X)) store M/Z mass of each row.
PeaksOnly	Should only peaks be used in calculating the correlation? In case of raw mass spectra data it does not make much sense to calculate correlation of "valleys" between peaks so one can set this flag to TRUE and only points above sample mean will be used.

Details

Function calculates for each copy of each sample two variables:

- inner correlation - average correlation between multiple copies of the same sample. Inner correlation measures how similar copies are to each other. For example innerCor[iSamp,iCopy] measures average correlation between X[,iSamp,iCopy] and all other copies of that sample. In case of one copy of the data innerCor is set to one. In case of two copies innerCor[iSamp,1] = innerCor[iSamp,2] = cor(X[,iSamp,1],X[,iSamp,2]). In case of 3 copies innerCor[iSamp,1] = (cor(X[,iSamp,1],X[,iSamp,2]) + cor(X[,iSamp,1],X[,iSamp,3]))/ ,etc.
- outer correlation - average correlation between each sample and every other sample within the same copy. Outer correlation measures how similar each copy is to everybody else.

Value

Returns list with two components: innerCor and outerCor both of size [nSamples × nCopies].

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

See Also

- Part of msc.preprocess.run and msc.project.run pipelines.
- Used by msc.copies.merge function.
- Uses cor function

Examples

```
 # load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run in 3D input data using long syntax
out = msc.mass.adjust.calc (X);
```

```
Y   = msc.mass.adjust.apply(X, out$ShiftX, out$ScaleY, out$ShiftY)

# check what happen to sample correlation
A = msc.sample.correlation(X, PeaksOnly=TRUE)
B = msc.sample.correlation(Y, PeaksOnly=TRUE)
cat("Mean corelation between two copies of the same sample:\n")
cat(" before: ", ai<-mean(A$innerCor)," after: ", bi<-mean(B$innerCor), "\n")
cat("Mean corelation between unrelated samples:\n")
cat(" before: ", ao<-mean(A$outerCor)," after: ", bo<-mean(B$outerCor), "\n")
stopifnot(ao<bo, ai<bi, bo<bi, abs(bi-0.91)<0.01, abs(ao-0.75)<0.01)
```

`read.mzXML & write.mzXML`

Read and Write mzXML Files

Description

Read and write protein mass spectra data to/from mzXML files.

Usage

```
mzXML = new.mzXML()
mzXML = read.mzXML(filename)
write.mzXML(mzXML, filename, precision=c('32', '64'))
```

Arguments

`mzXML`	class storing partially parsed mzXML data
`filename`	character string with name of the file (connection)
`precision`	precision to be used in saving `scan` data. Save double (floating point) array using 32 or 64 bits?

Details

The main task of `read.mzXML` and `write.mzXML` functions is to extract and save scan data of mzXML files. In addition attempt is made to keep all other sections of mzXML file as unparsed XML code, so the data can be extracted latter or saved into new mzXML files. Those unparsed sections are stored as XML text

Value

Function `read.mzXML` returns object of type `mzXML`, containing:

`scan`	List of Mass Spectra scans. Each element of the list contain the following elements:

- `peaks` - intensities or peaks of the scan

- mass - masses (m/z) corresponding to peaks. Vectors mass and peaks have the same length.
- num - scan number
- parentNum - scan number of parent scan in case of recursively stored scans (msLevel>1)
- msLevel - 1- means MS scan, 2- means MS/MS scan, etc.
- header - xml code of <scan> header might contain other useful attributes
- maldi - optional - acquisition dependent properties of a MALDI experiment
- scanOrigin - optional - name of parent file(?)
- precursorMz - optional - information about the precursor ion
- nameValue - optional - properties of the scan not included elsewhere

All optional elements contain unparsed XML code, if corresponding sections are present, or NULL. See mzXML schema and documentation for more details

header — Stores header of <mzXML> section containing information about namespace and schema file location.

msInstrument — General information about the MS instrument. Stored as XML.

parentFile — Path to all the ancestor files. Stored as XML.

dataProcessing — Description of any data manipulation. Stored as XML.

separation — Information about the separation technique. Stored as XML.

spotting — Acquisition independent properties of a MALDI experiment. Stored as XML.

indexOffset — Offset of the index element. Either 0 or a vector.

Function new.mzXML returns the same object as read.mzXML but with all fields equal to NULL. Function write.mzXML does not return anything.

Author(s)

Jarek Tuszynski (SAIC) ⟨jaroslaw.w.tuszynski@saic.com⟩

References

Definition of mzXML format: http://tools.proteomecenter.org/mzXMLschema.php

Documentation of mzXML format: http://sashimi.sourceforge.net/schema_revision/mzXML_2.1/Doc/mzXML_2.1_tutorial.pdf

More Documentation of mzXML format: http://sashimi.sourceforge.net/software_glossolalia.html

ReadmzXML software http://tools.proteomecenter.org/readmzXML.php

See Also

For reading XML files see xmlTreeParse from **XML**.

Other R function related to mzXML format: xcmsRaw from **xcms** BioConductor package.

Examples

```
directory = system.file("Test", package = "caMassClass")
FileName = file.path(directory,"test1.xml")
xml = read.mzXML(FileName)
xml

# test reading/writing
write.mzXML(xml, "temp.xml")
xml2 = read.mzXML("temp.xml")
file.remove("temp.xml")
stopifnot(all(xml$scan[[1]]$peaks == xml2$scan[[1]]$peaks))
stopifnot(xml$msInstrument == xml2$msInstrument)

# extracting scan data from the output
FileName = file.path(directory,"test2.xml")
xml  = read.mzXML(FileName)
plot(xml$scan[[1]]$mass, xml$scan[[1]]$peaks, type="l")

# extracting data from unparsed sections
tree = xmlTreeParse(xml$msInstrument, asText=TRUE, asTree=TRUE)
x = xmlRoot(tree)
xmlName(x)
xmlAttrs(x[["msManufacturer"]]) ["value"]
xmlAttrs(x[["software"]])
```

Index