Package 'Cardinal'

October 12, 2016

Type Package

Title A mass spectrometry imaging toolbox for statistical analysis

Version 1.4.0

Date 2015-1-12

Author Kyle D. Bemis <kbemis@purdue.edu>

Maintainer Kyle D. Bemis <kbemis@purdue.edu>

Description Implements statistical & computational tools for analyzing mass spectrometry imaging datasets, including methods for efficient pre-processing, spatial segmentation, and classification.

License Artistic-2.0

Depends BiocGenerics, Biobase, graphics, methods, stats, ProtGenerics

Imports grid, irlba, lattice, signal, sp, stats4, utils

Suggests BiocStyle, testthat

biocViews Software, Infrastructure, Proteomics, Lipidomics, Normalization, MassSpectrometry, ImagingMassSpectrometry, Clustering, Classification

URL http://www.cardinalmsi.org

NeedsCompilation yes

R topics documented:

`Cardinal-package` *Mass spectrometry imaging tools*

Description

Implements statistical & computational tools for analyzing mass spectrometry imaging datasets, including methods for efficient pre-processing, spatial segmentation, and classification.

Details

Cardinal provides an abstracted interface to manipulating mass spectrometry imaging datasets, simplifying most of the basic programmatic tasks encountered during the statistical analysis of imaging data. These include image manipulation and processing of both images and mass spectra, and dynamic plotting of both.

While pre-processing steps including normalization, baseline correction, and peak-picking are provided, the core functionality of the package is statistical analysis. The package includes classification and clustering methods based on nearest shrunken centroids, as well as traditional tools like PCA and PLS.

Type `vignette("Cardinal-demo")` for a brief walkthrough of common workflows.

To view other vignettes, type `browseVignettes("Cardinal")`.

Author(s)

Kyle D. Bemis

Maintainer: Kyle D. Bemis <kbemis@purdue.edu>

---

batchProcess-methods *Batch Pre-Processing on an Imaging Dataset*

Description

Batch apply multiple pre-processing steps on an imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet'
batchProcess(object,
    normalize = NULL,
    smoothSignal = NULL,
    reduceBaseline = NULL,
    peakPick = NULL,
    ...,
    layout,
    pixel = pixels(object),
    plot = FALSE)
```

Arguments

`object`	An object of class `MSImageSet`.
`normalize`	Either 'TRUE' or a `list` of arguments to be passed to the `normalize` method. Use 'FALSE' or 'NULL' to skip this pre-processing step.
`smoothSignal`	Either 'TRUE' or a `list` of arguments to be passed to the `smoothSignal` method. Use 'FALSE' or 'NULL' to skip this pre-processing step.

reduceBaseline	Either 'TRUE' or a list of arguments to be passed to the reduceBaseline method. Use 'FALSE' or 'NULL' to skip this pre-processing step.
peakPick	Either 'TRUE' or a list of arguments to be passed to the peakPick method. Use 'FALSE' or 'NULL' to skip this pre-processing step.
layout	The layout of the plots, given by a length 2 numeric as c(ncol, nrow).
pixel	The pixels to process. If less than the extent of the dataset, this will result in a subset of the data being processed.
plot	Plot the pre-processing step for each pixel while it is being processed?
...	Ignored.

Details

When performing batch pre-processing, the mean spectrum is also calculated and returned as part of the 'featureData' of the result.

Internally, pixelApply is used to apply the pre-processing steps, as with other pre-processing methods.

Value

An object of class MSImageSet with the processed spectra.

Author(s)

Kyle D. Bemis

See Also

MSImageSet, normalize, smoothSignal, reduceBaseline, peakPick, pixelApply

Examples

```
data <- generateImage(as="MSImageSet")

batchProcess(data, normalize=TRUE, smoothSignal=TRUE,
    reduceBaseline=TRUE, peakPick=TRUE,
    layout=c(2,2), plot=interactive())

batchProcess(data, normalize=TRUE,
    reduceBaseline=list(blocks=200), peakPick=list(SNR=12),
    layout=c(1,3), plot=interactive())
```

Binmat-class *On-Disk Matrix Class Using On-Demand Disk Reads*

Description

The Binmat class implements on-disk matrices with efficient access to columns. Values within each column are contiguously stored in a binary file on disk. The columns themselves need not be stored contiguously. Only the accessed elements of the matrix are loaded into memory.

Usage

```
## Instance creation
Binmat(
    files,
    nrow, ncol,
    offsets = 0,
    extents = rep(nrow, ncol),
    datatype = c("16-bit integer",
        "32-bit integer",
        "64-bit integer",
        "32-bit float",
        "64-bit float"),
    dimnames = NULL,
    ...)

## Additional methods documented below
```

Arguments

files	The file(s) where the matrix is stored.
nrow	The number of rows in the on-disk matrix.
ncol	The number of columns in the on-disk matrix.
offsets	The positions of the first value of each column in number of bytes from the beginning of the file.
extents	The length of each column.
datatype	The binary data type.
dimnames	The 'dimnames' giving the dimension names for the matrix, analogous to the 'dimnames' attribute of an ordinary R matrix. This must be a list of length 2 or NULL.
...	Additional arguments passed to the constructor.

Slots

`files`: A `factor` giving the full file paths of the binary files storing the matrix (or matrices) on disk. Length must be equal to the number of columns.

`offsets`: A `numeric` vector giving the positions of the first value of each column in number of bytes from the beginning of the file.

`extents`: A `numeric` vector giving the length of each column.

`datatype`: A `factor` vector giving the binary data types of each element of the matrix (or matrices) on disk. Length must be equal to the number of columns.

`dim`: A length 2 integer vector analogous to the 'dim' attribute of an ordinary R matrix.

`dimnames`: A length 2 `list` analogous to the 'dimnames' attribute of an ordinary R matrix.

`.__classVersion__`: A `Versions` object describing the version of the class used to created the instance. Intended for developer use.

Extends

`Versioned`

Creating Objects

`Binmat` instances are usually created through `Binmat()`.

Methods

Standard generic methods:

`dim(x), dim(x) <- value`: Return or set the dimensions of the on-disk matrix.

`dimnames(x), dimnames(x) <- value`: Return or set the 'dimnames' of the on-disk matrix.

`colnames(x), colnames(x) <- value`: Return or set the column names of the on-disk matrix.

`rownames(x), rownames(x) <- value`: Return or set the row names of the on-disk matrix.

`ncol`: Return the number of columns in the on-disk matrix.

`nrow`: Return the number of columns in the on-disk matrix.

`cbind`: Combine on-disk matrices by columns.

`rbind`: Not allowed for on-disk matrices. (Always returns an error.)

`Binmat[i, j, ..., drop]`: Access elements in the on-disk matrix. A Binmat on-disk matrix can be indexed like an ordinary R matrix. Note however that linear indexing is not supported. Assignment is not currently allowed.

Author(s)

Kyle D. Bemis

See Also

`matrix`, `Hashmat`, `SImageSet`

Examples

```
## Not run:
## Create an Binmat object
Binmat("path/to/file.extension")

## End(Not run)
```

`coord-methods` *Retrieve Pixel Coordinates from iSets*

Description

These generic functions accesses pixel coordinates stored in an object derived from `iSet`. The `coordinates` method is an *alias* for `coord`.

Usage

```
coord(object)
coord(object) <- value

coordinates(object)
coordinates(object) <- value

coordLabels(object)
coordLabels(object) <- value
```

Arguments

`object`	An object, possible derived from `iSet`.
`value`	Value to be assigned to the corresponding object.

Value

`coord` returns a `data.frame` with each row containing coordinates for an individual pixel. `coordLabels` retrieves the coordinate labels.

Author(s)

Kyle D. Bemis

See Also

`iSet`, `SImageSet`, `MSImageSet`

coregister-methods *Coregister Images*

Description

Coregister images of an imaging dataset. Currently this is only used to coregister the class assignments for clustering methods, but additional functionality may be added in the future for 3D experiments and registration of optical images.

Usage

```
## S4 method for signature 'SpatialShrunkenCentroids,missing'
coregister(object, ref, ...)

## S4 method for signature 'SpatialKMeans,missing'
coregister(object, ref, ...)
```

Arguments

object	An imaging dataset.
ref	A reference for the coregistration.
...	Ignored.

Value

A new imaging dataset of the same class with coregistered images.

Author(s)

Kyle D. Bemis

See Also

spatialShrunkenCentroids

cvApply-methods *Apply Cross-Validated Analysis to Imaging Datasets*

Description

Apply an existing or a user-specified function over imaging datasets.

Usage

```
## S4 method for signature 'SImageSet'
cvApply(.x, .y, .fun, .fold = sample, ...)
```

Arguments

.x	An object of class SImageSet.
.y	An appropriate response variable.
.fun	The function to be used for the analyses.
.fold	A variable determining the cross-validation folds. By default, this will set to 'sample' from pixelData(.x), to ensure that whole samples are left out during the cross-validation. This argument is evaluated in pixelData(.x).
...	Additional arguments passed to .fun.

Details

This method is designed to be used with the provided classification methods, but can also be used with user-provided functions and methods as long as they fulfill certain expectations.

The function or method passed to '.fun' must take at least two arguments: the first argument must be a object derived from SImageSet, and the second argument must be the response variable. The function should return an object of a class derived from ResultSet, which should have a predict method that takes arguments 'newx' and 'newy'.

Value

An object of class 'CrossValidated', which is derived from ResultSet.

Author(s)

Kyle D. Bemis

See Also

PLS, OPLS, spatialShrunkenCentroids

---

generateImage *Generate a Simulated Image*

Description

Generates a simulated image of spectral signals.

Usage

```
generateImage(data = factor(1),
    coord = expand.grid(
        x = 1:max(1, nrow(data)),
        y = 1:max(1, ncol(data))),
    peaks = length(levels(as.factor(data))),
    delta = 10,
    as = c("SImageSet", "MSImageSet"),
    ...)
```

Arguments

data	Either a factor or an integer matrix. If a factor is used, the coord argument should be specified with data to indicate the arrangement of regions in the image. If a matrix is given, coord should not be specified. The image will automatically be generated with different regions corresponding to unique integers in the matrix.
coord	A data.frame with columns representing the spatial dimensions. Each row provides a spatial coordinate for the location of an element of data if data is a factor.
peaks	The number of peaks in the signal.
delta	The effect size of the difference between peaks differentiating different regions in the image (as specified by data).
as	Should the output object be an SImageSet or MSImageSet?
...	Additional arguments to pass to generateSpectrum.

Value

An SImageSet or an MSImageSet.

Author(s)

Kyle D. Bemis

See Also

generateSpectrum

Examples

```
data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
 NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
 1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
 1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)

set.seed(1)
x <- generateImage(data)

plot(x, pixel=1)
image(x, feature=1)

coord <- expand.grid(x=1:nrow(data), y=1:ncol(data))

data2 <- as.factor(data[is.finite(data)])
coord2 <- coord[is.finite(data),]

set.seed(1)
x2 <- generateImage(data=data, coord=coord, as="MSImageSet")
```

```
plot(x, pixel=1)
image(x2, feature=1)
```

---

generateSpectrum *Generate a Simulated Spectrum*

Description

Generates a simulated spectral signal, or multiple such signals, with peaks of specified intensities.

Usage

```
generateSpectrum(n, peaks = 100,
    range = c(1001, 20000),
    centers = seq(
        from = range[1] + diff(range) / (peaks + 1),
        to = range[2] - diff(range) / (peaks + 1),
        length.out = peaks),
    intensities = runif(peaks, min=0.1, max=1),
    step = diff(range)/1e3,
    resolution = 500,
    noise = 0.05,
    sd = 0.1,
    baseline = 2000,
    auc = TRUE)
```

Arguments

n	The number of signals to simulate.
peaks	The number of peaks in the signal.
range	A pair of numbers specifying the range of continues feature values at which the signal is measured.
centers	The values of the singal feature at which peaks occur.
intensities	The values of the intensities of the peaks, which could either be heights of the peaks or their area under the curve.
step	The step size between measurements in the feature space.
resolution	The instrument resolution. This affects the width of the peaks. Higher resolutions produce sharper peaks.
noise	A value without scale that indicates the amount of noise in the signal.
sd	Standard deviation of the intensities of the peaks.
baseline	A value without scale that indicates the shape and size of the baseline.
auc	Should the peak heights be influenced by the area under the curve? This reflects fragmentation and limited accuracy at higher mass ranges. If 'FALSE' then the peak heights correspond directly to the provided intensities.

Value

A `list` with elements:

- `x`: `numeric`, a numeric vector of signal intensities
- `t`: `numeric`, a numeric vector of signal features

Author(s)

Kyle D. Bemis

See Also

`generateImage`

Examples

```
s <- generateSpectrum(1)
plot(x ~ t, type="l", data=s)

s <- generateSpectrum(1, centers=c(2000,3000), resolution=10, baseline=3000)
plot(x ~ t, type="l", data=s)

s <- generateSpectrum(1, peaks=2, auc=FALSE, baseline=0)
plot(x ~ t, type="l", data=s)
```

Hashmat-class *Sparse Matrix Class Using Lists as Hash Tables*

Description

The Hashmat class implements compressed sparse column (CSC) style matrices using R `list` objects as the columns. The implementation is unique in that it allows re-assignment of the keys describing the rows, allowing for arbitrary re-ordering of rows and row-wise elements. This is useful for storing sparse signals, such as processed spectra.

Usage

```
## Instance creation
Hashmat(data = NA, nrow = 1, ncol = 1, byrow=FALSE,
    dimnames = NULL, ...)

## Additional methods documented below
```

Arguments

data	A matrix or a vector. If data is a matrix, then a sparse matrix is construced from matrix directly and other arguments (except for dimnames) are ignored. If data is a vector, then the behavior is the same as for ordinary matrix construction.
nrow	The number of rows in the sparse matrix.
ncol	The number of columns in the sparse matrix.
byrow	If 'FALSE', the matrix is filled by columns. If 'TRUE', it is filled by rows.
dimnames	The 'dimnames' giving the dimension names for the matrix, analogous to the 'dimnames' attribute of an ordinary R matrix. This must be a list of length 2 or NULL.
...	Additional arguments passed to the constructor.

Slots

data: A list with vectors corresponding columns of the sparse matrix, whose elements are its non-zero elements.

keys: A character vector providing the keys that determine the rows of the non-zero elements of the matrix.

dim: A length 2 integer vector analogous to the 'dim' attribute of an ordinary R matrix.

dimnames: A length 2 list analogous to the 'dimnames' attribute of an ordinary R matrix.

.__classVersion__: A Versions object describing the version of the class used to created the instance. Intended for developer use.

Extends

Versioned

Creating Objects

Hashmat instances are usually created through Hashmat().

Methods

Class-specific methods:

pData(object), pData(object)<-: Access or set the list of numeric vectors storing the column-vectors of the sparse matrix directly.

keys(object), keys(object)<-: Access of set the keys for the row elements. If this is a character, it sets the keys slot directly, and hence the 'dim' is also changed. If this is a list, then the list should have length equal to the number of rows, and each element should be an integer vector of length equal to the number of non-zero row elements for the respective column. The vectors are used to index the keys slot and set the key names of the vectors, and hence change or reorder the row elements.

Standard generic methods:

`combine(x, y, ...)`: Combines two Hashmat objects. See the `combine` method for matrices for details of how the Hashmat sparse matrices are combined. The behavior is identical, except when filling in missing elements in non-shared rows and columns, the resulting Hashmat object will have zeroes instead of NAs.

`dim(x), dim(x) <- value`: Return or set the dimensions of the sparse matrix.

`dimnames(x), dimnames(x) <- value`: Return or set the 'dimnames' of the sparse matrix.

`colnames(x), colnames(x) <- value`: Return or set the column names of the sparse matrix.

`rownames(x), rownames(x) <- value`: Return or set the row names of the sparse matrix.

`ncol`: Return the number of columns in the sparse matrix.

`nrow`: Return the number of columns in the sparse matrix.

`cbind`: Combine sparse matrices by columns. The keys used to resolve the rows must match between matrices.

`rbind`: Not allowed for sparse matrices. (Always returns an error.)

`Hashmat[i, j, ..., drop], Hashmat[i, j, ...] <- value`: Access and assign elements in the sparse matrix. A Hashmat sparse matrix can be indexed like an ordinary R matrix. Note however that linear indexing is not supported. Use `drop = NA` to return a subset of the same class as the object.

Author(s)

Kyle D. Bemis

See Also

`matrix`, `Binmat`, `SImageSet`

Examples

```
## Create an Hashmat object
Hashmat()

## Using a list of elements and keys
dmat1 <- diag(3)
smat1 <- Hashmat(dmat1)
all.equal(smat1[], dmat1, check.attr=FALSE)

## Filling an empty sparse matrix
smat2 <- Hashmat(nrow=1000, ncol=1000)
smat2[500,] <- rep(1, 1000)

dmat2 <- matrix(nrow=1000, ncol=1000)
dmat2[500,] <- rep(1, 1000)

print(object.size(dmat2), units="Mb")
print(object.size(smat2), units="Mb") # Much smaller

all.equal(dmat2[500,], smat2[500,], , check.attr=FALSE)
```

IAnnotatedDataFrame-class

Class Containing Measured Variables and Their Meta-Data Description for Imaging Experiments

Description

An IAnnotatedDataFrame is an extension of an AnnotatedDataFrame as defined in the 'Biobase' package modified to reflect that individual rows in data represent pixels rather than samples, and many pixels will come from a single sample. Additionally, it keeps track of the coordinates of the pixel represented by each row.

Usage

```
## Instance creation
IAnnotatedDataFrame(data, varMetadata,
dimLabels=c("pixelNames", "pixelColumns"),
...)

## Additional methods documented below
```

Arguments

data	A data.frame of the pixels (rows) and measured variables (columns). Omitting this will yield an empty IAnnotatedDataFrame with zero rows.
varMetadata	A data.frame with columns describing the measured variables in data. Generated automatically if missing.
dimLabels	Aesthetic labels for the rows and columns in the show method.
...	Additional arguments passed to the initialize method.

Details

The key difference between a IAnnotatedDataFrame and a AnnotatedDataFrame is that an IAnnotatedDataFrame makes a distinction between samples and pixels, recognizing that rows belong to pixels, many of which may belong to the same sample. Therefore, data contains a required column called 'sample', which indicates the sample to which the pixel belongs, and varMetadata contains an additional required column called 'labelType', which indicates whether a variable is a spatial dimensions ('dim') or a phenotype ('pheno') or a sample ('sample'). The 'labelType' of the 'sample' variable depends on the structure of the experiment. See below for details.

The 'labelType' for 'sample' will be 'sample' in the case of a 2D imaging experiment with a single sample. The 'labelType' for 'sample' will be 'dim' in the case of a 2D imaging experiment with multiple samples, since the 'sample' will be acting as a proxy spatial coordinate. Note however that in this case, the result of a call to coordLabels will *not* include 'sample'.

It is possible to compare the results of names(coord(object)) and coordLabels(object) to distinguish between coordinate types that should be considered independent. It will be assumed

a spatial relationship exists for all variables returned by `coordLabels(object)`, but this is not necessarily true for all variables returned by `names(coord(object))`. This is required, because every row in the `data.frame` returned by `coord(object)` should be unique and correspond to a unique pixel.

The suggested structure for 3D imaging experiments is to create an additional variable solely to refer to the spatial dimension (e.g., 'z') and treat it separately from the 'sample'. Therefore, in a 3D imaging experiment with a single sample, the 'labelType' for 'sample' would be 'sample'.

Slots

`data`**:** Object of class `data.frame` containing pixels (rows) and measured variables (columns). Contains at least one column named 'sample' which is a `factor` and gives the sample names for each pixel. The sample names can be set using `sampleNames<-`. Inherited from AnnotatedDataFrame.

`varMetadata`**:** Object of class `data.frame` with number of rows equal to the number of columns in `data`. Contains at least two columns, one named 'labelDescription' giving a textual description of each variable, and an additional one named 'labelType' describing the type of variable. The 'labelType' is a factor with levels `"dim"`, `"sample"`, `"pheno"`. Inherited from AnnotatedDataFrame

`dimLabels`**:** Object of class `character` of length 2 that provides labels for the rows and columns in the `show` method. Inherited from AnnotatedDataFrame.

`.__classVersion__`**:** A `Versions` object describing the version of the class used to created the instance. Intended for developer use.

Extends

Class `AnnotatedDataFrame`, directly. Class `Versioned`, by class "AnnotatedDataFrame", distance 2.

Creating Objects

`IAnnotatedDataFrame` instances are usually created through `IAnnotatedDataFrame()`.

Methods

Class-specific methods:

`sampleNames(object)`, `sampleNames(object)<-`**:** Return or set the sample names in the object, as determined by the factor levels of the 'sample' variable in `data`.

`pixelNames(object)`, `pixelNames(object)<-`**:** Return or set the pixel names (the rows of `data`).

`coordLabels(object)`, `coordLabels(object)<-`**:** Return or set the names of the pixel coodinates. These are the subset of `varLabels(object) for which the corresponding variables have a 'labelType` Note that this will *never* include 'sample', even if the 'sample' variable has type 'dim'. (See details.)

`coord(object)`, `coord(object)<-`**:** Return or set the coodinates. This is a `data.frame` containing the subset of columns of `data` for which the variables have a 'labelType' of 'dim'.

Standard generic methods:

`combine(x, y, ...)`: Combine two or more `IAnnotatedDataFrame` objects. The objects are combined similarly to 'rbind' for `data.frame` objects. Pixels coordinates are checked for uniqueness. The 'varLabels' and 'varMetadata' must match.

Author(s)

Kyle D. Bemis

See Also

`AnnotatedDataFrame`, `iSet`, `SImageSet` `MSImageSet`

Examples

```
## Create an IAnnotatedDataFrame object
IAnnotatedDataFrame()

## Simple IAnnotatedDataFrame
df1 <- IAnnotatedDataFrame(data=expand.grid(x=1:3, y=1:3),
varMetadata=data.frame(labelType=c("dim", "dim")))
pData(df1)
varMetadata(df1)

# Example of possible experiment data
coord <- expand.grid(x=1:3, y=1:3)
df2 <- IAnnotatedDataFrame(data=
data.frame(rbind(coord, coord), sample=factor(rep(1:2, each=nrow(coord)))),
varMetadata=data.frame(labelType=c("dim", "dim")))
df2$diagnosis <- factor(rbinom(nrow(df2), 1, 0.5), labels=c("normal", "cancer"))
varMetadata(df2)["diagnosis", "labelDescription"] <- "disease pathology"
df2[["time", labelDescription="time measured"]] <- rep(date(), nrow(df2))
pData(df2)
varMetadata(df2)

# Change labels and pixel coord
coordLabels(df2) <- c("x1", "x2")
pixelNames(df2) <- paste("p", 1:nrow(df2), sep="")
sampleNames(df2) <- c("subject A", "subject B")
coord(df2) <- coord(df2)[nrow(df2):1,]
pData(df2)
```

---

image-methods *Plot the Pixel-Space of an Imaging Dataset*

Description

Create and display plots in the pixel space of an imaging dataset. This uses a formula interface inspired by the `lattice` graphics package.

Usage

```
## S4 method for signature 'SImageSet'
image(x, formula = ~ x * y,
    feature,
    feature.groups,
    groups = NULL,
    superpose = FALSE,
    strip = TRUE,
    key = FALSE,
    fun = mean,
    normalize.image = c("none", "linear"),
    contrast.enhance = c("none", "suppression", "histogram"),
    smooth.image = c("none", "gaussian", "adaptive"),
    ...,
    xlab,
    xlim,
    ylab,
    ylim,
    zlab,
    zlim,
    layout,
    asp = 1,
    col = rainbow(nlevels(feature.groups)),
    col.regions = intensity.colors(100),
    colorkey = !is3d,
    subset = TRUE,
    lattice = FALSE)

## S4 method for signature 'MSImageSet'
image(x, formula = ~ x * y,
    feature = features(x, mz=mz),
    feature.groups,
    mz,
    plusminus,
    ...)

## S4 method for signature 'ResultSet'
image(x, formula,
    model = pData(modelData(x)),
    feature,
    feature.groups,
    superpose = TRUE,
    strip = TRUE,
    key = superpose,
    ...,
    column,
    col = if (superpose) rainbow(nlevels(feature.groups)) else "black",
    lattice = FALSE)
```

```
## S4 method for signature 'CrossValidated'
image(x, fold = 1:length(x), layout, ...)

## S4 method for signature 'PCA'
image(x, formula = substitute(mode ~ x * y),
    mode = "scores",
    ...)

## S4 method for signature 'PLS'
image(x, formula = substitute(mode ~ x * y),
    mode = c("fitted", "scores", "y"),
    ...)

## S4 method for signature 'OPLS'
image(x, formula = substitute(mode ~ x * y),
    mode = c("fitted", "scores", "Oscores", "y"),
    ...)

## S4 method for signature 'SpatialShrunkenCentroids'
image(x, formula = substitute(mode ~ x * y),
    mode = c("probabilities", "classes", "scores"),
    ...)

## S4 method for signature 'SpatialKMeans'
image(x, formula = substitute(mode ~ x * y),
    mode = "cluster",
    ...)

## S4 method for signature 'SImageSet'
image3D(x, formula = ~ x * y * z, ...)
```

Arguments

x	An imaging dataset.
formula	A formula of the form 'z ~ x * y | g1 * g2 * ...' (or equivalently, 'z ~ x + y | g1 + g2 + ...'), indicating a LHS 'y' (on the y-axis) versus a RHS 'x' (on the x-axis) and conditioning variables 'g1, g2, ...'. Usually, the LHS is not supplied, and the formula is of the form '~ x * y | g1 * g2 * ...', and the y-axis is implicityl assumed to be the feature vectors corresponding to each pixel in the imaging dataset specified by the object 'x'. However, a variable evaluating to a vector of pixel values, or a sequence of such variables, can also be supplied. The RHS is evaluated in `pData(x)` and should provide values for the xy-axes. These must be spatial coordinates. The conditioning variables are evaluated in `fData(x)`. These can be specified in the formula as 'g1 * g2 * ...'. The argument 'feature.groups' allows an alternate

way to specify a single conditioning variable. Conditioning variables specified using the formula interface will always appear on separate plots. This can be combined with 'superpose = TRUE' to both overlay plots based on a conditioning variable and use conditioning variables to create separate plots.

`model` A vector or `list` specifying which fitted model to plot. If this is a vector, it should give a subset of the rows of `modelData(x)` to use for plotting. Otherwise, it should be a list giving the values of parameters in `modelData(x)`.

`feature` The feature or vector of features for which to plot the image. This is an expression that evaluates to a logical or integer indexing vector.

`feature.groups` An alternative way to express a single conditioning variable. This is a variable or expression to be evaluated in `fData(x)`, expected to act as a grouping variable for the features specified by 'feature', typically used to distinguish different groups or ranges of features. Pixel vectors of images from features in the same feature group will have 'fun' applied over them; 'fun' will be applied to each feature group separately, usually for averaging. If 'superpose = FALSE' then these appear on separate plots.

`groups` A variable or expression to be evaluated in `pData(x)`, expected to act as a grouping variable for the pixel regions in the image(s) to be plotted, typically used to distinguish different image regions by varying graphical parameters like color and line type. By default, if 'superpose = FALSE', these appear overlaid on the same plot.

`superpose` Should feature vectors from different feature groups specified by 'feature.groups' be superposed on the same plot? If 'TRUE' then the 'groups' argument is ignored.

`strip` Should strip labels indicating the plotting group be plotting along with the each panel? Passed to 'strip' in `levelplot` is 'lattice = TRUE'.

`key` A logical, or `list` containing components to be used as a key for the plot. This is passed to 'key' in `levelplot` if 'lattice = TRUE'.

`fun` A function to apply over pixel vectors of images grouped together by 'feature.groups'. By default, this is used for averaging over features.

`normalize.image`
Normalization function to be applied to each image. The function can be user-supplied, of one of 'none' or 'linear'. The 'linear' normalization method normalized each image to the same intensity range using a linear transformation.

`contrast.enhance`
Contrast enhancement function to be applied to each image. The function can be user-supplied, or one of 'none', 'histogram', or 'suppression'. The 'histogram' equalization method flatterns the distribution of intensities. The hotspot 'suppression' method uses thresholding to reduce the intensities of hotspots.

`smooth.image` Image smoothing function to be applied to each image. The function can be user-supplied, or one of 'none', 'gaussian', or 'adaptive'. The 'gaussian' smoothing method smooths images with a simple gaussian kernel. The 'adaptive' method uses bilateral filtering to preserve edges.

`xlab` Character or expression giving the label for the x-axis.

`ylab` Character or expression giving the label for the y-axis.

zlab	Character or expression giving the label for the z-axis. (Only used for plotting 3D images.)
xlim	A numeric vector of length 2 giving the left and right limits for the x-axis.
ylim	A numeric vector of length 2 giving the top and bottom limits for the y-axis.
zlim	A numeric vector of length 2 giving the lower and upper limits for the z-axis (i.e., the range of colors to be plotted).
layout	The layout of the plots, given by a length 2 numeric as `c(ncol, nrow)`. This is passed to `levelplot` if 'lattice = TRUE'. For base graphics, this defaults to one plot per page.
asp	The aspect ratio of the plot.
col	A specification for the default plotting color(s) for groups.
col.regions	The default plotting color(s) for the z-axis of image intensities.
colorkey	Should a coloykey describing the z-axis be drawn with the plot?
subset	An expression that evaluates to a logical or integer indexing vector to be evaluated in `pData(x)`.
lattice	Should lattice graphics be used to create the plot?
...	additional arguments passed to the underlying `plot` functions.
mz	The m/z value for which to plot the ion image.
plusminus	If specified, a window of m/z values surrounding the one given by `coord` will be included in the plot with `fun` applied over them, and this indicates the range of the window on either side.
fold	What folds of the cross-validation should be plotted.
mode	What kind of results should be plotted. This is the name of the object to plot in the `ResultSet` object.
column	What columns of the results should be plotted. If the results are a matrix, this corresponds to the columns to be plotted, which can be indicated either by numeric index or by name.

Note

For objects derived from class SImageSet, calling `image3D(x)` is equivalent to `image(x, ~ x * y * z)`.

Author(s)

Kyle D. Bemis

See Also

`plot-methods`, `select-methods`

Examples

```
data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
 NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
 1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
 1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)

mycol <- gradient.colors(100, "red", "black")

set.seed(1)
sset <- generateImage(data, range=c(1000,5000), centers=c(3000,4000), resolution=100)

pData(sset)$pg <- factor(data[is.finite(data)], labels=c("black", "red"))
fData(sset)$fg <- factor(rep("bg", nrow(fData(sset))), levels=c("bg", "black", "red"))
fData(sset)$fg[2950 < fData(sset)$t & fData(sset)$t < 3050] <- "black"
fData(sset)$fg[3950 < fData(sset)$t & fData(sset)$t < 4050] <- "red"

image(sset, feature=1, col=mycol)

image(sset, feature=fData(sset)$fg=="black", col=mycol)

image(sset, feature=fData(sset)$fg=="red", col=mycol)

image(sset, ~ x * y | fg, feature=1:nrow(sset), lattice=TRUE, col=mycol)

image(sset, feature=1:nrow(sset), feature.groups=fg, lattice=TRUE, col=mycol)

set.seed(1)
msset <- generateImage(data, range=c(1000,5000), centers=c(3000,4000), resolution=100, as="MSImageSet")

image(msset, mz=3000, col=mycol)

image(msset, mz=4000, col=mycol)

image(msset, mz=3500, plusminus=500, col=mycol)
```

`ImageData-class` *Class Containing Arrays of Imaging Data*

Description

A container class for holding imaging data, designed to contain one or more arrays in an immutable environment. It is assumed that the first dimension of each array corresponds to the features.

Note that only visible objects (names not beginning with '.') are checked for validity; however, *all* objects are copied if any elements in the `data` slot are modified when `data` is an "immutableEnvironment".

Usage

```
## Instance creation
ImageData(...,
    data = new.env(parent=emptyenv()),
    storageMode = c("immutableEnvironment",
        "lockedEnvironment", "environment"))

## Additional methods documented below
```

Arguments

`...`	Named arguments that are passed to the `initialize` method for instantiating the object. These must be arrays or array-like objects with an equal number of dimensions. They will be assigned into the environment in the `data` slot.
`data`	An environment in which to assign the previously named variables.
`storageMode`	The storage mode to use for the `ImageData` object for the environment in the `data` slot. This must be one of `"immutableEnvironment"`, `"lockedEnvironment"`, or `"environment"`. See documentation on the `storageMode` slot below for more details.

Slots

`data`**:** An `environment` which may contain one or more arrays with an equal number of dimensions. It is assumed that the first dimension corresponds to the features.

`storageMode`**:** A `character` which is one of `"immutableEnvironment"`, `"lockedEnvironment"`, or `"environment"`. The values `"lockedEnvironment"` and `"environment"` behave as described in the documentation of `AssayData`. An `"immutableEnvironment"` uses a locked environment while retaining R's typical copy-on-write behavior. Whenever an object in an immutable environment is modified, a new environment is created for the `data` slot, and all objects copied into it. This allows usual R functional semantics while avoiding copying of large objects when other slots are modified.

`.__classVersion__`**:** A `Versions` object describing the version of the class used to created the instance. Intended for developer use.

Extends

`Versioned`

Creating Objects

`ImageData` instances are usually created through `ImageData()`.

Methods

Class-specific methods:

`storageMode(object)`, `storageMode(object)<-`**:** Return or set the storage mode. See documentation on the `storageMode` slot above for more details.

Standard generic methods:

`initialize`: Initialize an `ImageData` object. Called by `new`. Not to be used by the user.

`validObject`: Validity-check that the arrays in the `data` slot environment are all of equal number of dimensions, and the storage mode is a valid value.

`combine(x, y, ...)`: Combine two or more `ImageData` objects. All elements must have matching names, and are combined with calls to `combine`. Higher dimensional arrays are combined using the same rules as for matrices. (See `combine` for more details.)

`annotatedDataFrameFrom(object)`: Returns an `IAnnotatedDataFrame` with columns for the dimensions of the elements of `data`. All dimensions must be named (determined by the `rownames(dims(object))`). It is assumed that the first dimension corresponds to the features, and is not used as a dimension in the returned `IAnnotatedDataFrame`. Additional arguments (`byrow`, ...) are ignored.

`dims`: A matrix with each column corresponding to the dimensions of an element in the `data` slot.

`names(x)`, `names(x)<-`: Access or replace the array names of the elements contained in the `data` slot environment.

`ImageData[[name]]`, `ImageData[[name]] <- value`: Access or replace an element named "`name`" in the environment in the `data` slot.

Author(s)

Kyle D. Bemis

See Also

`AssayData`, `SImageData`, `SImageSet`, `MSImageSet`

Examples

```
## Create an ImageData object
ImageData()

idata <- ImageData(data0=matrix(1:4, nrow=2))
idata[["data0"]]

# Immutable environments in ImageData objects
storageMode(idata) <- "lockedEnvironment"
try(idata[["data0"]][,1] <- c(10,11)) # Fails

storageMode(idata) <- "immutableEnvironment"
try(idata[["data0"]][,1] <- c(10,11)) # Succeeds

# Test copy-on-write for immutable environments
idata2 <- idata
idata2[["data0"]] <- matrix(5:8, nrow=2)
idata[["data0"]] == idata2[["data0"]] # False
```

imageData-methods *Retrieve Image Data from iSets*

Description

These generic functions image data (typically spectra) stored in an object derived from iSet.

Usage

```
imageData(object)
imageData(object) <- value
iData(object)
iData(object) <- value

spectra(object, ...)
spectra(object) <- value

peaks(object, ...)
peaks(object) <- value

mzData(object)
mzData(object) <- value

peakData(object)
peakData(object) <- value
```

Arguments

object	An object, possible derived from iSet.
value	Value to be assigned to the corresponding object.
...	Additional arguments (ignored).

Value

imageData returns an object containing both image data and metadata, usually an object derived from ImageData. iData returns only the image data in a matrix-like object with the rows corresponding to features and the columns corresponding to pixels. spectra is an *alias* for iData for use with MSImageSet objects. mzData and peakData are used for retrieving both peak data and metadata from peak-picked objects. peaks retrieves peak cubes from peak-picked objects.

Author(s)

Kyle D. Bemis

See Also

iSet, SImageSet, MSImageSet

`intensity.colors` *Color Palettes for Imaging*

Description

Create a vector of n continuous colors.

Usage

```
intensity.colors(n, alpha=1)
risk.colors(n, alpha=1)
gradient.colors(n, start="white", end="black", alpha=1)
alpha.colors(n, col="red", alpha.power=2, alpha=(seq_len(n)/n)^alpha.power)
```

Arguments

`n`	the number of colors
`alpha`	a vector of alpha values between 0 and 1
`start`	the starting color value
`end`	the ending color value
`col`	the color(s) to expand with transparency
`alpha.power`	how the alpha should ramp as it increases

Value

A pallete of colors.

Author(s)

Kyle D. Bemis

Examples

```
col <- intensity.colors(100^2)
if ( interactive() ) {
image(matrix(1:(100^2), nrow=100), col=col)
}
```

iSet-class *Class to Contain High-Throughput Imaging Experiment Data and Metadata*

Description

A container class for data from high-throughput imaging experiments and associated metadata. Classes derived from from iSet contain one or more arrays or array-like objects with an equal number of dimensions as imageData elements. It is assumed that the first dimension of each such element corresponds to the data features, and all other dimensions are described by associated coordinates in the pixelData slot. Otherwise, derived classes are responsible for managing how the elements of imageData behave and their relationship with the rows of pixelData and featureData.

The MSImageSet class for mass spectrometry imaging experiments is the primary derived class of iSet. Its parent class SImageSet is another derived class for more general images.

This class is based on the eSet virtual class from Biobase. However, the iSet class contains an imageData slot which is an 'immutableEnvironment' that preserves copy-on-write behavior for iSet derived classes, but only copying elements of imageData when that slot specifically is modified. In addition pixelData is an IAnnotatedDataFrame that stores pixel information such as pixel coordinates in addition to phenotypic data.

Slots

imageData**:** An instance of ImageData, which stores one or more array or array-like objects of equal number of dimensions as elements in an 'immutableEnvironment'. This slot preserves copy-on-write behavior when it is modified specifically, but is pass-by-reference otherwise, for memory efficiency.

pixelData**:** Contains pixel information in an IAnnotatedDataFrame. This includes both pixel coordinates and phenotypic and sample data. Its rows correspond to individual pixels, many of which may belong to the same sample. Apart a requirement on columns describing the pixel coordinates, it is left to derived classes to decide the relationship to elements of imageData.

featureData**:** Contains variables describing features. It Is left to derived classes to decide the relationship to elements of imageData.

experimentData**:** Contains details of experimental methods. Should be an object of a derived class of MIAxE.

protocolData**:** Contains variables describing the generation of the samples in pixelData.

.__classVersion__**:** A Versions object describing the version of the class used to created the instance. Intended for developer use.

Extends

VersionedBiobase, directly. Versioned, by class "VersionedBiobase", distance 2.

Creating Objects

iSet is a virtual class. No instances can be created.

Methods

Class-specific methods:

`sampleNames(object)`, `sampleNames(object) <- value`: Access and set the sample names in the `pixelData` and `protocolData` slots.

`featureNames(object)`, `featureNames(object) <- value`: Access and set the feature names in the `featureData` slot.

`pixelNames(object)`, `pixelNames(object) <- value`: Access and set the pixel names in the `pixelData` slot.

`coordLabels(object)`, `coordLabels(object) <- value`: Access and set the coordinate names described by the coordinate variables in the `pixelData` slot. Note that this does *not* set or get coordinate names with a `labelType` of `sample`, regardless of whether they are currently being used to describe coordinates or not. Therefore, checking `coordLabels(object)` versus `names(coord(object))` is a simple way of checking whether a dataset is 2D or 3D.

`coord(object)`, `coord(object)<-`: Return or set the coodinates. This is a `data.frame` containing the subset of columns of `data` for which the variables have a 'labelType' of 'dim'.

`imageData(object)`, `imageData(object) <- value`: Access and set the `imageData` slot.

`pixelData(object)`, `pixelData(object) <- value`: Access and set the `pixelData` slot.

`pData(object)`, `pData(object) <- value`: Access and set the pixel information.

`varMetadata(object)`, `varMetadata(object) <- value`: Access and set the metadata describing the variables in `pData`.

`varLabels(object)`, `varLabels(object) <- value`: Access and set the variable labels in `pixelData`.

`featureData(object)`, `featureData(object) <- value`: Access and set the `featureData` slot.

`fData(object)`, `fData(object) <- value`: Access and set the feature information.

`fvarMetadata(object)`, `fvarMetadata(object) <- value`: Access and set the metadata describing the features in `fData`.

`fvarLabels(object)`, `fvarLabels(object) <- value`: Access and set the feature labels in `featureData`.

`features(object, ...)`: Access the feature indices (rows in `featureData`) corresponding to variables in `featureData`.

`pixels(object, ...)`: Access the pixel indices (rows in `pixelData`) corresponding to variables in `pixelData`.

`experimentData(object)`, `experimentData(object) <-`: Access and set the `experimentData` slot.

`protocolData(object)`, `protocolData(object) <-`: Access and set the `protocolData` slot.

`storageMode(object)`, `storageMode(object)<-`: Return or set the storage mode of the `imageData` slot. See documentation on the `storageMode` slot above for more details.

Standard generic methods:

`initialize`: Initialize a object of an `iSet` derived class. Called by `new`. Not to be used by the user.

`validObject`: Checks that there exist columns in `pixelData` describing the pixel coordinates, cooresponding to the dimensions of the elements of `imageData`. For every named dimension of the arrays on `imageData` there must be a `pData` column describing its pixel coordinates. Also checks that the `sampleNames` match between `pixelData` and `protocolData`.

`combine(x, y, ...)`: Combine two or more `iSet` objects. To be combined, iSets must have identical `featureData` and distinct `pixelNames` and `sampleNames`. All elements of `imageData` must have matching names. Elements of `imageData` are combined by calls for `combine`.

`dim`: The dimensions of the object, as determined by the number of features (rows in featureData) and the number of pixels (rows in pixelData). This may differ from the dimensions returned by `dims(object)` (which corresponds to the arrays in `data`) or returned by `dim(imageData(object))`. See SImageSet for an example where this is the case, due to its use of a "virtual" datacube.

`dims`: A matrix with each column corresponding to the dimensions of an element in the `data` slot.

`iSet$name, iSet$name <- value`: Access and set the `name` column in `pixelData`.

`iSet[[i, ...]], iSet[[i, ...]] <- value`: Access and set the column `i` (character or numeric index) in `pixelData`. The ... argument can include named variables (especially 'labelDescription') to be added to the `varMetadata`.

Author(s)

Kyle D. Bemis

See Also

eSet, SImageSet, MSImageSet

Examples

```
## Cannot create an iSet object
try(new("iSet"))

## Create an iSet derived class
MyImageSet <- setClass("MyImageSet", contains="iSet")
MyImageSet()

removeClass("MyImageSet")
```

MIAPE-Imaging-class *Class for Storing Mass Spectrometry Imaging Experiment Information*

Description

The Minimum Information About a Proteomics Experiment for MS Imaging. The current implementation is based on the imzML specification.

Slots

`name`: Object of class `character` containing the experimenter name

`lab`: Object of class `character` containing the laboratory where the experiment was conducted.

`contact`: Object of class `character` containing contact information for lab and/or experimenter.

`title`: Object of class `character` containing a single-sentence experiment title.

`abstract`: Object of class `character` containing an abstract describing the experiment.

`url`: Object of class `character` containing a URL for the experiment.

`pubMedIds`: Object of class `character` listing strings of PubMed identifiers of papers relevant to the dataset.

`samples`: Object of class `list` containing information about the samples.

`preprocessing`: Object of class `list` containing information about the pre-processing steps used on the raw data from this experiment.

`other`: Object of class `list` containing other information for which none of the above slots does not applies.

`specimenOrigin`: Object of class `character` describing the specimen origin (institution, ...).

`specimenType`: Object of class `character` describing the specimen type (species, organ, ...).

`stainingMethod`: Object of class `character` describing the staining method, if any, applied to the sample (H&E, ...).

`tissueThickness`: Object of class `numeric` giving the tissue thickness in micrometers (um).

`tissueWash`: Object of class `character` describing the wash method (spray, dipping, ...).

`embeddingMethod`: Object of class `character` describing the embedding method (if any); this could be paraffin, ...

`inSituChemistry`: Object of class `character` describing any on-sample chemistry (tryptic digest, ...)

`matrixApplication`: Object of class `character` describing how the matrix was applied, if applicable

`pixelSize`: Object of class `numeric` describing the size of the pixels in micrometers (um).

`instrumentModel`: Object of class `character` indicating the instrument model used to generate the data.

`instrumentVendor`: Object of class `character` indicating the mass spectrometer vendor.

`massAnalyzerType`: Object of class `character` describing the mass analyzer type (LTQ, TOF, ...).

`ionizationType`: Object of class `character` describing the ionization type (MALDI, DESI, ...).

`scanPolarity`: Object of class `character` describing the polarity (negative or positive).

`softwareName`: Object of class `character` with the control and/or analysis software name.

`softwareVersion`: Object of class `character` with the version of the control and/or analysis software.

`scanType`: Object of class `character` describing the scan type. This must be either 'horizontal line scan' or 'vertical line scan'. See the imzML specifications for more details.

`scanPattern`: Object of class `character` describing the scan type. This must be one of 'flyback', 'meandering', or 'random access'. See the imzML specifications for more details.

`scanDirection`: Object of class `character` describing the scan type. This must be one of 'bottom up', 'left right', 'right left', or 'top down'. See the imzML specifications for more details.

`lineScanDirection`: Object of class `character` describing the scan type. This must be one of 'linescan bottom up', 'linescan left right', 'linescan right left', or 'linescan top down'. See the imzML specifications for more details.

`imageShape`: Object of class `character` describing the image shape (rectangular, free form, ...). See the imzML specifications for more details.

Extends

Class MIAxE, directly, Class Versioned, by class "MIAxE", distance 2.

Creating Objects

MIAPE-Imaging instances can be created through new("MIAPE-Imaging"). In general, instances should not be created by the user, but are automatically generated when reading an external file to create an MSImageSet object, and then modified through the accessor and setter methods if necessary.

Methods

Class-specific methods:

msiInfo: Displays 'MIAPE-Imaging' information.

abstract: An accessor function for abstract.

expinfo: An accessor function for name, lab, contact, title, and url.

notes(object), notes(object) <- value: Accessor functions for other. notes(object) <- character *appends* character to notes; use notes(object) <- list to replace the notes entirely.

otherInfo: An accessor function for other.

preproc: An accessor function for preprocessing.

pubMedIds(object), pubMedIds(object) <- value: Accessor function for pubMedIds.

samples: An accessor function for samples.

specimenOrigin(object), specimenOrigin(object) <- value: Accessor and setter function for specimenOrigin.

specimenType(object), specimenType(object) <- value: Accessor and setter function for specimenType.

stainingMethod(object), stainingMethod(object) <- value: Accessor and setter function for stainingMethod.

tissueThickness(object), tissueThickness(object) <- value: Accessor and setter function for tissueThickness.

tissueWash(object), tissueWash(object) <- value: Accessor and setter function for tissueWash.

embeddingMethod(object), embeddingMethod(object) <- value: Accessor and setter function for embeddingMethod.

inSituChemistry(object), inSituChemistry(object) <- value: Accessor and setter function for inSituChemistry.

matrixApplication(object), matrixApplication(object) <- value: Accessor and setter function for matrixApplication.

pixelSize(object), pixelSize(object) <- value: Accessor and setter function for pixelSize.

instrumentModel(object), instrumentModel(object) <- value: Accessor and setter function for instrumentModel.

instrumentVendor(object), instrumentVendor(object) <- value: Accessor and setter function for instrumentVendor.

`massAnalyzerType(object)`, `massAnalyzerType(object) <- value`: Accessor and setter function for `massAnalyzerType`.

`ionizationType(object)`, `ionizationType(object) <- value`: Accessor and setter function for `ionizationType`.

`scanPolarity(object)`, `scanPolarity(object) <- value`: Accessor and setter function for `scanPolarity`.

`softwareName(object)`, `softwareName(object) <- value`: Accessor and setter function for `softwareName`.

`softwareVersion(object)`, `softwareVersion(object) <- value`: Accessor and setter function for `softwareVersion`.

`scanType(object)`, `scanType(object) <- value`: Accessor and setter function for `scanType`.

`scanPattern(object)`, `scanPattern(object) <- value`: Accessor and setter function for `scanPattern`.

`scanDirection(object)`, `scanDirection(object) <- value`: Accessor and setter function for `scanDirection`.

`lineScanDirection(object)`, `lineScanDirection(object) <- value`: Accessor and setter function for `lineScanDirection`.

`imageShape(object)`, `imageShape(object) <- value`: Accessor and setter function for `imageShape`.

Standard generic methods:

`show`: Displays object content.

`combine(x, y, ...)`: Combine two or more `MIAPE-Imaging` objects.

Author(s)

Kyle D. Bemis

References

Schramm T, Hester A, Klinkert I, Both J-P, Heeren RMA, Brunelle A, Laprevote O, Desbenoit N, Robbe M-F, Stoeckli M, Spengler B, Rompp A (2012) imzML - A common data format for the flexible exchange and processing of mass spectrometry imaging data. Journal of Proteomics 75 (16):5106-5110. doi:10.1016/j.jprot.2012.07.026

See Also

`MIAxE`, `MSImageSet`

Examples

```
showClass("MIAPE-Imaging")
```

MSImageData-class *Class Containing Mass Spectrometry Image Data*

Description

A container class for mass spectrometry imaging data. This is an extension of the SImageData class, which adds methods specific for the extraction and replacement of mass spectral peaks.

Usage

```
## Instance creation
MSImageData(
    data = Hashmat(nrow=0, ncol=0),
    coord = expand.grid(
        x = seq_len(ncol(data)),
        y = seq_len(ifelse(ncol(data) > 0, 1, 0))),
    storageMode = "immutableEnvironment",
    positionArray = generatePositionArray(coord),
    dimnames = NULL,
    ...)

## Additional methods documented below
```

Arguments

data	A matrix-like object with number of rows equal to the number of features and number of columns equal to the number of non-missing pixels. Each column should be a feature vector. Alternatively, a multidimensional array that represents the datacube with the first dimension as the features can also be supplied. Additional dimensions could be the spatial dimensions of the image, for example.
coord	A data.frame with columns representing the spatial dimensions. Each row provides a spatial coordinate for the location of a feature vector corresponding to a column in data. This argument is ignored if data is a multidimensional array rather than a matrix.
storageMode	The storage mode to use for the MSImageData object for the environment in the data slot. Only "immutableEnvironment" is allowed for MSImageData. See documentation on the storageMode slot below for more details.
positionArray	The positionArray for the imaging data. This should not normally be specified the user, since it is generated automatically from the coord argument, unless for some reason coord is not specified.
dimnames	A list of length two, giving the feature names and pixel names in that order. If missing, this is taken from the 'dimnames' of the data argument.
...	Additional Named arguments that are passed to the initialize method for instantiating the object. These must be matrices or matrix-like objects of equal dimension to data. They will be assigned into the environment in the data slot.

Slots

`data`: An `environment` which contains at least one element named `"iData"`, and possibly containing an element named `"peakData"` and `"mzData"`. The `"peakData"` element contains the intensities of the peak cube in a sparse matrix format. The `"mzData"` element contians the m/z values of the peaks in a sparse matrix format. All of these matrices have been aligned for that their dimensions reflect only the shared peaks, possibly across multiple datasets. They have been aligned from a call to `peakAlign`.

`coord`: An `data.frame` with rows giving the spatial coordinates of the pixels corresponding to the columns of `"iData"`.

`positionArray`: An `array` with dimensions equal to the spatial dimensions of the image, which stores the column numbers of the feature vectors corresponding to the pixels in the `"iData"` element of the `data` slot. This allows re-construction of the imaging "datacube" on-the-fly.

`dim`: A length 2 integer vector analogous to the 'dim' attribute of an ordinary R matrix.

`dimnames`: A length 2 `list` analogous to the 'dimnames' attribute of an ordinary R matrix.

`storageMode`: A `character` which is one of `"immutableEnvironment"`, `"lockedEnvironment"`, or `"environment"`. The values `"lockedEnvironment"` and `"environment"` behave as described in the documentation of `AssayData`. An `"immutableEnvironment"` uses a locked environment while retaining R's typical copy-on-write behavior. Whenever an object in an immutable environment is modified, a new environment is created for the `data` slot, and all objects copied into it. This allows usual R functional semantics while avoiding copying of large objects when other slots are modified.

`.__classVersion__`: A `Versions` object describing the version of the class used to created the instance. Intended for developer use.

Extends

`Versioned`

Creating Objects

`MSImageData` instances are usually created through `MSImageData()`.

Methods

Class-specific methods:

`iData(object)`, `iData(object)<-`: Return or set the matrix of image intensities. Columns should correspond to feature vectors, and rows should correspond to pixel vectors.

`peakData(object)`, `peakData(object)<-`: Return or set the sparse matrix of peak intensities if it exists.

`mzData(object)`, `mzData(object)<-`: Return or set the sparse matrix of peak m/z values if it exists.

`coord(object)`, `coord(object)<-`: Return or set the coodinates. This is a `data.frame` with each row corresponding to the spatial coordinates of a pixel.

`positionArray(object)`, `positionArray(object)<-`: Return or set the `positionArray` slot. When setting, this should be an array returned by a call to `generatePositionArray`.

`featureNames(object)`, `featureNames(object) <- value`: Access and set feature names (names of the rows of the intensity matrix).

`pixelNames(object)`, `pixelNames(object) <- value`: Access and set the pixel names (names of the columns of the intensity matrix).

`storageMode(object)`, `storageMode(object)<-`: Return or set the storage mode. See documentation on the `storageMode` slot above for more details.

Standard generic methods:

`combine(x, y, ...)`: Combine two or more `MSImageData` objects. Elements must be matrix-like objects and are combined column-wise with a call to 'cbind'. The numbers of rows must match, but otherwise no checking of row or column names is performed. The pixel coordinates are checked for uniqueness.

`dim`: Return the dimensions of the (virtual) datacube. This is equal to the number of features (the number of rows in the matrix returned by `iData`) and the dimensions of the `positionArray` slot. For a standard imaging dataset, that is the number features followed by the spatial dimensions of the image.

`dims`: A matrix where each column corresponds to the dimensions of the (virtual) datacubes stored as elements in the `data` slot. See above for how the dimensions are calculated.

`MSImageData[i, j, ..., drop]`: Access intensities in the (virtual) imaging datacube. The datacube is reconstructed on-the-fly. The object can be indexed like any ordinary array with number of dimensions equal to `dim(object)`. Use `drop = NA` to return a subset of the same class as the object.

Author(s)

Kyle D. Bemis

See Also

ImageData, SImageData, SImageSet, MSImageSet

Examples

```
## Create an MSImageData object
MSImageData()

## Using a P x N matrix
data1 <- matrix(1:27, nrow=3)
coord <- expand.grid(x=1:3, y=1:3)
sdata1 <- MSImageData(data1, coord)
sdata1[] # extract data as array

## Using a P x X x Y array
data2 <- array(1:27, dim=c(3,3,3))
sdata2 <- MSImageData(data2)
sdata2[] # should be identical to above

# Missing data from some pixels
data3 <- matrix(1:9, nrow=3)
```

```
sdata3 <- MSImageData(data3, coord[c(1,5,9),])

dim(sdata3) # presents as an array
iData(sdata3) # stored as matrix
sdata3[] # recontruct the datacube

iData(sdata3)[,1] <- 101:103 # assign using iData()
sdata3[] # can only assign into matrix representation

## Sparse feature vectors
data4 <- Hashmat(nrow=9, ncol=9)
sdata4 <- MSImageData(data4, coord)
iData(sdata4)[] <- diag(9)
sdata4[1,,]
```

`MSImageProcess-class` *Class Containing Mass Spectral Pre-Processing Information*

Description

A class containing information about mass spectral pre-processsing operations. These should not usually be set by the user, and are automatically updated when processing methods are applied.

Slots

`files`**:** Object of class `character` storing the file paths to the raw data files used to create the dataset.

`normalization`**:** Object of class `character` describing any normalization applied to the dataset.

`smoothing`**:** Object of class `character` describing any smoothing applied to the dataset.

`baselineReduction`**:** Object of class `character` describing baseline correction applied to the dataset.

`spectrumRepresentation`**:** Object of class `character` describing the spectrum type (profile or centroid).

`peakPicking`**:** Object of class `character` describing the peak picking applied to the dataset (area or height).

`centroided`**:** Object of class `logical` describing whether the data have been centroided.

`history`**:** Object of class `list` containing specific information about the function calls applied to the `MSImageSet` `object` to produce the current instance and their parameters.

`CardinalVersion`**:** Object of class `character` indicating the version of Cardinal.

`.__classVersion__`**:** Object of class `Versions` indicating the version of the `MSImageProcess` instance. Intended for developer use.

Extends

Class `Versioned`, directly.

Creating Objects

`MSImageProcess` instances can be created through `new("MSImageProcess")`. In general, instances should not be created by the user, but are automatically generated by processing methods applied to `MSImageSet` objects.

Methods

Class-specific methods:

`prochistory(object)`, `prochistory(object) <- value`**:** Accessor and setter for the history of methods applied to the experiment dataset.

`files(object)`, `files(object) <- value`**:** Accessor and setter function for `files`.

`normalization(object)`, `normalization(object) <- value`**:** Accessor and setter function for `normalization`.

`smoothing(object)`, `smoothing(object) <- value`**:** Accessor and setter function for `smoothing`.

`baselineReduction(object)`, `baselineReduction(object) <- value`**:** Accessor and setter function for `baselineReduction`.

`spectrumRepresentation(object)`, `spectrumRepresentation(object) <- value`**:** Accessor and setter function for `spectrumRepresentation`.

`peakPicking(object)`, `peakPicking(object) <- value`**:** Accessor and setter function for `peakPicking`.

`centroided(object)`, `centroided(object) <- value`**:** Accessor and setter function for `centroided`.

Standard generic methods:

`show`**:** Displays object content.

`combine(x, y, ...)`**:** Combine two or more `MSImageProcess` objects.

Author(s)

Kyle D. Bemis

See Also

`MSImageSet`

Examples

```
showClass("MSImageProcess")
```

MSImageSet-class *Class to Contain Mass Spectrometry Imaging Experiment Data*

Description

Container for mass spectrometry imaging experimental data and metadata. MSImageSet is derived from iSet through SImageSet. It extends these classes with information about the processing and analysis, requiring MIAPE-Imaging in its experimentData slot.

Usage

```
## Instance creation
MSImageSet(
    spectra = Hashmat(nrow=0, ncol=0),
    mz = seq_len(dim(spectra)[1]),
    coord = expand.grid(
        x = seq_len(prod(dim(spectra)[-1])),
        y = seq_len(ifelse(prod(dim(spectra)[-1]) > 0, 1, 0))),
    imageData = MSImageData(data=spectra, coord=coord),
    pixelData = IAnnotatedDataFrame(
        data=coord,
        varMetadata=data.frame(labelType=rep("dim", ncol(coord)))),
    featureData = AnnotatedDataFrame(
        data=data.frame(mz=mz)),
    processingData = new("MSImageProcess"),
    protocolData = AnnotatedDataFrame(
        data=data.frame(row.names=sampleNames(pixelData))),
    experimentData = new("MIAPE-Imaging"),
    ...)

## Additional methods documented below
```

Arguments

spectra	A matrix-like object with number of rows equal to the number of features and number of columns equal to the number of non-missing pixels. Each column should be a mass spectrum. Alternatively, a multidimensional array that represents the datacube with the first dimension as the features (m/z values) can also be supplied. Additional dimensions could be the spatial dimensions of the image, for example.
mz	A numeric vector representing the mass-to-charge ratio features (m/z values) corresponding to the rows in the spectra matrix. Must be strictly increasing or decreasing.
coord	A data.frame with columns representing the spatial dimensions. Each row provides a spatial coordinate for the location of a mass spectrum corresponding to a column in spectra. This argument is ignored if spectra is a multidimensional array rather than a matrix.

imageData	An object of class SImageData that will contain the imaging mass spectra. Usually constructed through the spectra and coord arguments.
pixelData	An object of class IAnnotatedDataFrame giving the information about the pixels including coordinates of the data in imageData.
featureData	An object of class AnnotatedDataFrame giving information about the data features. Requires a column named "mz".
processingData	An object of class MSImageProcess giving information about the pre-processing steps applied to the spectra.
protocolData	An object of class AnnotatedDataFrame giving information about the samples. It must have one row for each of the sampleNames in pixelData.
experimentData	An object derived from class MIAxE giving information about the imaging experiment.
...	Additional arguments passed to the initializer.

Slots

imageData: An instance of SImageData, which stores one or more matrices of equal number of dimensions as elements in an 'immutableEnvironment'. This slot preserves copy-on-write behavior when it is modified specifically, but is pass-by-reference otherwise, for memory efficiency.

pixelData: Contains pixel information in an IAnnotatedDataFrame. This includes both pixel coordinates and phenotypic and sample data. Its rows correspond to the columns in imageData.

featureData: Contains variables describing features. Its rows correspond to the rows in imageData in an IAnnotatedDataFrame.

processingData: Contains details about the pre-processing steps that have been applied to the spectra. An object of class MSImageProcess.

experimentData: Contains details of experimental methods. Must be MIAPE-Imaging.

protocolData: Contains variables describing the generation of the samples in pixelData in an IAnnotatedDataFrame.

.__classVersion__: A Versions object describing the version of the class used to created the instance. Intended for developer use.

Extends

SImageSet, directly. iSet, by class "SImageSet", distance 1. VersionedBiobase, by class "iSet", distance 2. Versioned, by class "VersionedBiobase", distance 3.

Creating Objects

MSImageSet instances can be created through MSImageSet(), but are more commonly created through reading of external data files.

Methods

Class-specific methods:

`spectra(object), spectra(object) <- value`: Access and set the mass spectra in `imageData`. This is a matrix-like object with rows corresponding to features and columns corresponding to pixels, so that each column of the returned object is a mass spectrum.

`peaks(object), peaks(object) <- value`: Access and set the peaks in `imageData` if peak picking have been performed. This is a shortcut for `peakData(imageData(object))`. These are the unaligned peaks. Aligned peaks (if they exist) are accesed by spectra(object).

`mz(object), mz(object) <- value`: Returns and sets the common m/z values of the mass spectra in the dataset. This is a required column of `featureData`.

`features(object, ..., mz)`: Access the feature indices (rows in `featureData`) corresponding to variables in `featureData`. Bisection search is used for fuzzy matching of m/z values.

`pixels(object, ..., coord)`: Access the pixel indices (rows in `pixelData`) corresponding to variables in `pixelData`. If specified, `coord` should be a `data.frame` where each row corresponds to the coordinates of a desired pixel.

`centroided(object), centroided(object) <- value`: Access whether the dataset consists of profile or centroided mass spectra. This is a shortcut for `centroided(processingData(object))`. A setter is also provided, and is sometimes necessary for forcing some analysis methods to accept unprocessed spectra. (This is usually a bad idea.)

`processingData(object), processingData(object) <- value`: Access and set the `processingData` slot.

Standard generic methods:

`combine(x, y, ...)`: Combine two or more `MSImageSet` objects. Unique 'sample's in `pixelData` are treated as a dimension.

`MSImageSet[i, j, ..., drop]`: Subset an `SImageSet` based on the rows (featureData components) and the columns (pixelData components). The result is a new `MSImageSet`.

See `iSet` and `SImageSet` for additional methods.

Author(s)

Kyle D. Bemis

See Also

`iSet`, `SImageSet`

Examples

```
## Create an MSImageSet object
spectra <- matrix(1:27, nrow=3)
mz <- 101:103
coord <- expand.grid(x=1:3, y=1:3)
msset <- MSImageSet(spectra=spectra, mz=mz, coord=coord)
```

```
## Access a single image corresponding to the first feature
imageData(msset)[1,,]

## Reconstruct the datacube
imageData(msset)[]

## Access the P x N matrix of column-wise mass spectra
spectra(msset)

## Subset the MSImageSet to the first 2 m/z values and first 6 mass spectra
msset2 <- msset[1:2, 1:6]
imageData(msset2)[]
msset2
```

`mz-methods` *Retrieve m/z-values from MSImageSets*

Description

This generic function accesses m/z values from `MSImageSet` objects.

Usage

```
mz(object, ...)
mz(object) <- value
```

Arguments

`object`	An `MSImageSet` object.
`value`	Value to be assigned to the corresponding object.
`...`	Additional arguments (ignored).

Value

`mz` returns a numeric vector of m/z values.

Author(s)

Kyle D. Bemis

See Also

`MSImageSet`

normalize-methods *Normalize an Imaging Dataset*

Description

Apply normalization to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet'
normalize(object, method = "tic",
...,
pixel = pixels(object),
plot = FALSE)

## TIC normalization
normalize.tic(x, tic=length(x), ...)
```

Arguments

object	An object of class MSImageSet.
method	The normalization method to use.
pixel	The pixels to normalize. If less than the extent of the dataset, this will result in a subset of the data being processed.
plot	Plot the mass spectrum for each pixel while it is being processed?
...	Additional arguments passed to the normalization method.
x	The mass spectrum to be normalized.
tic	The value to which to normalize the total ion current.

Details

Normalization is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: A numeric vector of intensities.
- ...: Additional arguments.

A user-created function should return a numeric vector of the same length.

Internally, pixelApply is used to apply the normalization. See its documentation page for more details on additional objects available to the environment installed to the normalization function.

Value

An object of class MSImageSet with the normalized spectra.

Author(s)

Kyle D. Bemis

See Also

MSImageSet, pixelApply

Examples

```
data <- generateImage(as="MSImageSet")
normalize(data, method="tic", plot=interactive())
```

OPLS-methods *Orthogonal Partial Least Squares*

Description

Performs orthogonal partial least squares (also called orthogonal projection to latent structures or O-PLS) on an imaging dataset. This will also perform discriminant analysis (O-PLS-DA) if the response is a `factor`.

Usage

```
## S4 method for signature 'SImageSet,matrix'
OPLS(x, y, ncomp = 20,
    method = "nipals",
    scale = FALSE,
    keep.Xnew = TRUE,
    iter.max = 100, ...)

## S4 method for signature 'SImageSet,numeric'
OPLS(x, y, ...)

## S4 method for signature 'SImageSet,factor'
OPLS(x, y, ...)

## S4 method for signature 'SImageSet,character'
OPLS(x, y, ...)

## S4 method for signature 'OPLS'
predict(object, newx, newy, keep.Xnew = TRUE, ...)
```

Arguments

`x`	The imaging dataset on which to perform partial least squares.
`y`	The response variable, which can be a `matrix` or a `vector` for ordinary O-PLS, or a `factor` or a `character` for O-PLS-DA.

ncomp	The number of O-PLS components to calculate.
method	The function used to calculate the projection.
scale	Shoud the data be scaled first? This is passed to scale.
keep.Xnew	Should the new data matrix be kept after filtering out the orthogonal variation?
iter.max	The number of iterations to perform for the NIPALS algorithm.
...	Passed to the next OPLS method.
object	The result of a previous call to OPLS.
newx	An imaging dataset for which to calculate their OPLS projection and predict a response from an already-calculated OPLS object.
newy	Optionally, a new response from which residuals should be calculated.

Value

An object of class OPLS, which is a ResultSet, where each component of the resultData slot contains at least the following components:

Xnew: A new data matrix that has been filtered of the orthogonal variation.

Xortho: A new data matrix that consists of *only* the orthogonal variation.

Oscores: A matrix with the orthogonal component scores for the explanatary variable.

Oloadings: A matrix objects with the orthogonal explanatory variable loadings.

Oweights: A matrix with the orthgonal explanatory variable weights.

scores: A matrix with the component scores for the explanatary variable.

loadings: A matrix with the explanatory variable loadings.

weights: A matrix with the explanatory variable weights.

Yscores: A matrix objects with the component scores for the response variable.

Yweights: A matrix objects with the response variable weights.

projection: The projection matrix.

coefficients: The matrix of the regression coefficients.

ncomp: The number of O-PLS components.

method: The method used to calculate the projection.

center: The center of the dataset. Used for calculating O-PLS scores on new data.

scale: The scaling factors for the dataset. Used for O-PLS scores on new data.

Ycenter: The centers of the response variables. Used for predicting new observations.

Yscale: The scaling factors for the response variables. Used for predicting new observation.

fitted: The fitted response.

Author(s)

Kyle D. Bemis

References

Trygg, J., & Wold, S. (2002). Orthogonal projections to latent structures (O-PLS). Journal of Chemometrics, 16(3), 119-128. doi:10.1002/cem.695

See Also

PLS, PCA, spatialShrunkenCentroids,

Examples

```
sset <- generateImage(diag(4), range=c(200, 300), step=1)

y <- factor(diag(4))

opls <- OPLS(sset, y, ncomp=1:2)
```

PCA-methods *Principal Components Analysis*

Description

Performs principal components analysis efficiently on large datasets using implicitly restarted Lanczos bi-diagonalization (IRLBA) algorithm for approximate singular value decomposition of the data matrix.

Usage

```
    ## S4 method for signature 'SImageSet'
PCA(x, ncomp = 20,
        method = c("irlba", "svd"),
        scale = FALSE, ...)

    ## S4 method for signature 'PCA'
predict(object, newx, ...)
```

Arguments

x	The imaging dataset for which to calculate the principal components.
ncomp	The number of principal components to calculate.
method	The function used to calculate the singular value decomposition.
scale	Shoud the data be scaled first? This is passed to scale.
...	Ignored.
object	The result of a previous call to PCA.
newx	An imaging dataset for which to calculate the principal components scores based on the aleady-calculated principal components loadings.

Value

An object of class PCA, which is a ResultSet, where each component of the resultData slot contains at least the following components:

scores: A matrix with the principal component scores.

loadings: A matrix with the principal component loadings.

sdev: The standard deviations of the principal components.

method: The method used to calculate the principal components.

ncomp: The number of principal components calculated.

center: The center of the dataset. Used for calculating principal components scores on new data.

scale: The scaling factors for the dataset. Used for calculating principal components scores on new data.

Author(s)

Kyle D. Bemis

See Also

OPLS, PLS, irlba, svd

Examples

```
sset <- generateImage(diag(4), range=c(200, 300), step=1)

pca <- PCA(sset, ncomp=2)
```

---

peakAlign-methods *Peak Align an Imaging Dataset*

Description

Apply peak alignment to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet,numeric'
peakAlign(object, ref, method = c("diff", "DP"),
    ...,
    pixel = pixels(object),
    plot = FALSE)

## S4 method for signature 'MSImageSet,MSImageSet'
peakAlign(object, ref, ...)

## S4 method for signature 'MSImageSet,missing'
```

```
peakAlign(object, ref, ...)

## Absolute difference alignment
peakAlign.diff(x, y, diff.max=200, units=c("ppm", "mz"), ...)

## Dynamic programming alignment
peakAlign.DP(x, y, gap=0, ...)
```

Arguments

object	An object of class MSImageSet.
ref	A reference to which to align the peaks.
method	The peak alignment method to use.
pixel	The pixels to align. If less than the extent of the dataset, this will result in a subset of the data being processed.
plot	Plot the mass spectrum for each pixel while it is being processed?
...	Additional arguments passed to the peak alignment method.
x	The vector of m/z values to be aligned.
y	The vector of reference m/z values.
diff.max	Peaks that differ less than this value will be aligned together.
units	Either parts-per-million or the raw m/z values.
gap	The gap penalty for the dynamic programming sequence alignment.

Details

If a MSImageSet object is used as the reference, then the local maxima in its mean spectrum will be calculated and used as the reference m/z values. If the reference is missing, the method will use the object itself as the reference.

Peak alignment is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: The vector of m/z values to be aligned.
- y: The vector of reference m/z values.
- ...: Additional arguments.

A user-created function should return a vector of the same length as x and y where NA values indicate no match, and non-missing values give the index of the matched peak in the reference set.

Internally, pixelApply is used to apply the peak alignment. See its documentation page for more details on additional objects available to the environment installed to the peak alignment function.

Value

An object of class MSImageSet with the peak aligned spectra.

Author(s)

Kyle D. Bemis

See Also

MSImageSet, peakPick, peakFilter, reduceDimension, pixelApply

Examples

```
data <- generateImage(diag(2), as="MSImageSet")
peaks <- peakPick(data, method="simple", plot=interactive())
peaks <- peakAlign(peaks, data, method="diff", plot=interactive())
```

---

peakFilter-methods *Peak Filter an Imaging Dataset*

Description

Apply peak filtering to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet'
peakFilter(object, method = "freq", ..., pixel, plot)

## Filter based on the frequency of a peak
peakFilter.freq(x, freq.min=length(x) / 100, ...)
```

Arguments

object	An object of class MSImageSet.
method	The peak filtering method to use.
...	Additional arguments passed to the peak filtering method.
pixel	Deprecated.
plot	Deprecated. (Never did anything anyway.)
x	The vector of ion image intensities to filter.
freq.min	Peaks that occur in the dataset fewer times than this will be removed.

Details

Unlike most other processing methods, peakFilter operates on the feature space (ion images) of the dataset.

Peak filtering is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: The vector of ion image intensities to filter.

- . . . : Additional arguments.

A user-created function should return a logical: TRUE means keep the peak, and FALSE means remove the peak.

Internally, featureApply is used to apply the filtering. See its documentation page for more details on additional objects available to the environment installed to the peak filtering function.

Value

An object of class MSImageSet with the filtered peaks.

Author(s)

Kyle D. Bemis

See Also

MSImageSet, peakPick, peakAlign, reduceDimension, featureApply

Examples

```
data <- generateImage(diag(2), as="MSImageSet")
peaks <- peakPick(data, method="simple", plot=interactive())
peaks <- peakAlign(peaks, method="diff", plot=interactive())
peaks <- peakFilter(peaks, method="freq")
```

---

peakPick-methods *Peak Pick an Imaging Dataset*

Description

Apply peak picking to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet'
peakPick(object, method = c("simple", "adaptive", "limpic"),
    ...,
    pixel = pixels(object),
    plot = FALSE)

## Local maxima and SNR with constant noise
peakPick.simple(x, SNR=6, window=5, blocks=100, ...)

## Local maxima and SNR with adaptive noise
peakPick.adaptive(x, SNR=6, window=5, blocks=100, spar=1, ...)

## LIMPIC peak detection
peakPick.limpic(x, SNR=6, window=5, blocks=100, thresh=0.75, ...)
```

Arguments

object	An object of class MSImageSet.
method	The peak picking method to use.
pixel	The pixels to peak pick. If less than the extent of the dataset, this will result in a subset of the data being processed.
plot	Plot the mass spectrum for each pixel while it is being processed?
...	Additional arguments passed to the peak picking method.
x	The mass spectrum to be peak picked.
SNR	The minimum signal-to-noise ratio to be considered a peak.
window	The window width for seeking local maxima.
blocks	The number of blocks in which to divide the mass spectrum in order to calculate the noise.
spar	Smoothing parameter for the spline smoothing applied to the spectrum in order to decide the cutoffs for throwing away false noise spikes that might occur inside peaks.
thresh	The thresholding quantile to use when comparing slopes in order to throw away peaks that are too flat.

Details

Peak picking is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: A numeric vector of intensities.
- ...: Additional arguments.

A user-created function should return a list with two vectors of the same length as x:

- peaks: A logical vector indicating peaks.
- noise: A numeric vector with the estimated noise.

Internally, pixelApply is used to apply the peak picking. See its documentation page for more details on additional objects available to the environment installed to the peak picking function.

Value

An object of class MSImageSet with the peak picking spectra.

Author(s)

Kyle D. Bemis

References

Mantini, D., Petrucci, F., Pieragostino, D., Del Boccio, P., Di Nicola, M., Di Ilio, C., et al. (2007). LIMPIC: a computational method for the separation of protein MALDI-TOF-MS signals from noise. BMC Bioinformatics, 8(101), 101. doi:10.1186/1471-2105-8-101

See Also

MSImageSet, peakAlign, peakFilter, reduceDimension, pixelApply

Examples

```
data <- generateImage(as="MSImageSet")
peakPick(data, method="simple", plot=interactive())
```

---

pixelApply-methods *Apply Functions over Imaging Datasets*

Description

Apply an existing or a user-specified function over either all of the features or all of the pixels of an SImageSet. These are provided for convenience by analogy to the 'apply' family of functions, but allowing greater control over how the functions are applied over an imaging dataset.

Usage

```
## S4 method for signature 'SImageSet'
pixelApply(.object, .fun, ...,
    .pixel,
    .feature,
    .feature.groups,
    .pixel.dependencies,
    .simplify = TRUE,
    .use.names = TRUE,
    .verbose = FALSE)

## S4 method for signature 'SImageSet'
featureApply(.object, .fun, ...,
    .feature,
    .pixel,
    .pixel.groups,
    .feature.dependencies,
    .simplify = TRUE,
    .use.names = TRUE,
    .verbose = FALSE)
```

Arguments

.object	An object of class SImageSet.
.fun	The function to be applied.
...	Additional arguments passed to .fun.
.pixel	A subset of pixels to use, given by an integer vector of numeric indices, a character vector of pixel names, or a logical vector indicating which pixels to use.

.feature	A subset of features to use, given in the same manner as pixels.
.pixel.groups	A grouping factor or a vector that can be coerced into a factor, that indicates groups of pixels over which the function should be applied. Groups pixels are treated as cells in a ragged array, by analogy to the tapply function.
.feature.groups	A grouping factor features, in the same manner as for pixels.
.pixel.dependencies	Not currently used. This may be used in the future to allow caching when applying functions to data on disk.
.feature.dependencies	Not currently used. May be used for caching in the future.
.simplify	Should the result be simplified into a matrix of higher-dimensional array rather than a list, if appropriate?
.use.names	Should the names of elements of .object (pixels, features, and grouping variables, as appropriate) be used for the names of the result?
.verbose	Used for debugging. Currently ignored.

Details

The use of .pixel and .feature can be used to apply the function over only a subset of pixels or features (or both), allowing faster computation when calculation on only a subset of data is needed.

For pixelApply, the function is applied to the feature vector belonging to each pixel. The use of .feature.groups allows codetapply-like functionality on the feature vectors, applied separately to each pixel.

For featureApply, the function is applied to the vector of intensity values (i.e., the flattened image) corresponding to each feature. The use of .feature.groups allows codetapply-like functionality on the flattened image intensity vectors, applied separately to each feature.

The fData from .object is installed into the environment of .fun for pixelApply, and the pData from .object is installed into the environment of .fun for featureApply. This allows access to the symbols from fData or pData during the execution of .fun. If .fun already has an environment, it is retained as the parent of the installed environment.

Additionally, the following objects are made available by installing them into the .fun environment:

- .Object: The passed .object. (Note the case.)
- .Index: The index of the current iteration.

It is expected that these methods will be expanded in the future for different types of imaging datasets (e.g., data read directly from disk).

Value

If .simplify = FALSE, a list. Otherwise, a matrix, or a higher-dimensional array if grouping is specified.

Author(s)

Kyle D. Bemis

See Also

MSImageSet

Examples

```
data <- matrix(1:256, nrow=4)
coord <- expand.grid(x=1:4, y=1:4, z=1:4)
sset <- SImageSet(data=data, coord=coord)

fData(sset)$flag <- rep(c(TRUE, FALSE), 2)
pixelApply(sset, max, .feature.groups=flag)

pData(sset)$flag <- rep(c(TRUE, FALSE), 32)
featureApply(sset, max, .pixel.groups=flag)
```

---

pixelData-methods *Retrieve Information on Pixels in iSet-derived Classes*

Description

This generic function accesses pixel data (experiment specific information about pixels) and pixel metadata (e.g., coordinates or experimental conditions).

Usage

```
pixelData(object)
pixelData(object) <- value
pData(object)
pData(object) <- value
```

Arguments

object	An object, possible derived from iSet.
value	Value to be assigned to the corresponding object.

Value

pixelData returns an object containing information on pixel variables and pixel metadata. pixelData returns an object containing information on pixel variables and pixel metadata. pData returns a data.frame with pixels as rows and variables as columns.

Author(s)

Kyle D. Bemis

See Also

iSet, SImageSet, MSImageSet

pixelNames-methods *Retrieve Pixel Names from iSets*

Description

This generic function accesses pixel names (typically image coordinates) stored in an object derived from iSet.

Usage

```
pixelNames(object)
pixelNames(object) <- value
```

Arguments

object	An object, possible derived from iSet.
value	Value to be assigned to the corresponding object.

Value

pixelNames returns an object containing information on pixel names.

Author(s)

Kyle D. Bemis

See Also

iSet, SImageSet, MSImageSet

pixels-methods *Retrieve Pixel or Feature Indices Based on Metadata*

Description

These are generic functions to retrieve pixel or feature row indices in an iSet-derived object's pixelData or featureData slots based on metadata variables.

Usage

```
## S4 method for signature 'iSet'
pixels(object, ...)

## S4 method for signature 'iSet'
features(object, ...)

## S4 method for signature 'MSImageSet'
pixels(object, ..., coord)

## S4 method for signature 'MSImageSet'
features(object, ..., mz)
```

Arguments

`object`	An imaging dataset object.
`...`	Variables that appear in `pixelData(object)` or `featureData(object)`.
`mz`	A vector of m/z values.
`coord`	A list or `data.frame` of named pixel coordinates.

Details

It is often more convenient to specify a pixel or feature by identifying metadata such as pixel coordinates or m/z-values than by their row indices in the `pixelData` and `featureData` slots. However, many functions expect indices rather than coordinates or m/z-values. These generic functions make it easy to retrieve indices based on such metadata.

It is important to note that when passing multiple variables via ..., the 'AND' operator is used to resolve the query. However, when vectors are passed, all combinations of the given values will be used.

For convenience, `MSImageSet` uses a special implementation for the 'mz' variable, which uses a bisection search so that exact precision is not required when searching based on m/z-values.

Value

A numeric vector of pixel or feature indices.

Author(s)

Kyle D. Bemis

See Also

`PLS`, `OPLS`, `spatialShrunkenCentroids`

Examples

```
## Create an MSImageSet object
spectra <- matrix(1:27, nrow=3)
mz <- 101:103
coord <- expand.grid(x=1:3, y=1:3)
msset <- MSImageSet(spectra=spectra, mz=mz, coord=coord)

# Find pixel indices
pixels(msset, x=2, y=2)
pixels(msset, coord=list(x=2, y=2))
pixels(msset, coord=list(x=c(2,3), y=c(2,3)))

# Find feature indices
features(msset, mz=102)
features(msset, mz=c(101,103))
features(msset, mz=c(102.2))
```

`plot-methods` *Plot the Feature-Space of an Imaging Dataset*

Description

Create and display plots in the feature space of an imaging dataset. This uses a formula interface inspired by the `lattice` graphics package.

Usage

```
## S4 method for signature 'SImageSet,missing'
plot(x, formula = ~ Feature,
    pixel,
    pixel.groups,
    groups = NULL,
    superpose = FALSE,
    strip = TRUE,
    key = FALSE,
    fun = mean,
    ...,
    xlab,
    xlim,
    ylab,
    ylim,
    layout,
    type = 'l',
    col = "black",
    subset = TRUE,
    lattice = FALSE)

## S4 method for signature 'MSImageSet,missing'
```

```
plot(x, formula = ~ mz,
    pixel = pixels(x, coord=coord),
    pixel.groups,
    coord,
    plusminus,
    ...,
    type = if (centroided(x)) 'h' else 'l')

## S4 method for signature 'ResultSet,missing'
plot(x, formula,
    model = pData(modelData(x)),
    pixel,
    pixel.groups,
    superpose = TRUE,
    strip = TRUE,
    key = superpose,
    ...,
    xlab,
    ylab,
    column,
    col = if (superpose) rainbow(nlevels(pixel.groups)) else "black",
    lattice = FALSE)

## S4 method for signature 'CrossValidated,missing'
plot(x, fold = 1:length(x), layout, ...)

## S4 method for signature 'PCA,missing'
plot(x, formula = substitute(mode ~ mz),
    mode = "loadings",
    type = 'h',
    ...)

## S4 method for signature 'PLS,missing'
plot(x, formula = substitute(mode ~ mz),
    mode = c("coefficients", "loadings",
        "weights", "projection"),
    type = 'h',
    ...)

## S4 method for signature 'OPLS,missing'
plot(x, formula = substitute(mode ~ mz),
    mode = c("coefficients", "loadings", "Oloadings",
        "weights", "Oweights", "projection"),
    type = 'h',
    ...)

## S4 method for signature 'SpatialShrunkenCentroids,missing'
plot(x, formula = substitute(mode ~ mz),
```

```
    mode = c("centers", "tstatistics"),
    type = 'h',
    ...)

## S4 method for signature 'SpatialKMeans,missing'
plot(x, formula = substitute(mode ~ mz),
    mode = c("centers", "betweenss", "withinss"),
    type = 'h',
    ...)
```

Arguments

x	An imaging dataset.
formula	A formula of the form 'y ~ x | g1 * g2 * ...' (or equivalently, 'y ~ x | g1 + g2 + ...'), indicating a LHS 'y' (on the y-axis) versus a RHS 'x' (on the x-axis) and conditioning variables 'g1, g2, ...'. Usually, the LHS is not supplied, and the formula is of the form '~ x | g1 * g2 * ...', and the y-axis is implicityl assumed to be the feature vectors corresponding to each pixel in the imaging dataset specified by the object 'x'. However, a variable evaluating to a feature vector, or a sequence of such variables, can also be supplied. The RHS is evaluated in `fData(x)` and should provide values for the x-axis. The conditioning variables are evaluated in `pData(x)`. These can be specified in the formula as 'g1 * g2 * ...'. The argument 'pixel.groups' allows an alternate way to specify a single conditioning variable. Conditioning variables specified using the formula interface will always appear on separate plots. This can be combined with 'superpose = TRUE' to both overlay plots based on a conditioning variable and use conditioning variables to create separate plots.
model	A vector or `list` specifying which fitted model to plot. If this is a vector, it should give a subset of the rows of `modelData(x)` to use for plotting. Otherwise, it should be a list giving the values of parameters in `modelData(x)`.
pixel	The pixel or vector of pixels for which to plot the feature vectors. This is an expression that evaluates to a logical or integer indexing vector.
pixel.groups	An alternative way to express a single conditioning variable. This is a variable or expression to be evaluated in `pData(x)`, expected to act as a grouping variable for the pixels specified by 'pixel', typically used to distinguish different regions of the imaging data for comparison. Feature vectors from pixels in the same pixel group will have 'fun' applied over them; 'fun' will be applied to each pixel group separately, usually for averaging. If 'superpose = FALSE' then these appear on separate plots.
groups	A variable or expression to be evaluated in `fData(x)`, expected to act as a grouping variable for the features in the feature vector(s) to be plotted, typically used to distinguish different groups of features by varying graphical parameters like color and line type. By default, if 'superpose = FALSE', these appear overlaid on the same plot.

superpose	Should feature vectors from different pixel groups specified by 'pixel.groups' be superposed on the same plot?
strip	Should strip labels indicating the plotting group be plotting along with the each panel? Passed to 'strip' in xyplot.
key	A logical, or list containing components to be used as a key for the plot. This is passed to 'key' in levelplot if 'lattice = TRUE'.
fun	A function to apply over feature vectors grouped together by 'pixel.groups'. By default, this is used for averaging over pixels.
xlab	Character or expression giving the label for the x-axis.
ylab	Character or expression giving the label for the x-axis.
xlim	A numeric vector of length 2 giving the left and right limits for the x-axis.
ylim	A numeric vector of length 2 giving the lower and upper limits for the y-axis.
layout	The layout of the plots, given by a length 2 numeric as c(ncol, nrow). This is passed to levelplot if 'lattice = TRUE'. For base graphics, this defaults to one plot per page.
col	A specification for the default plotting color(s).
type	A character indicating the type of plotting.
subset	An expression that evaluates to a logical or integer indexing vector to be evaluated in fData(x).
lattice	Should lattice graphics be used to create the plot?
...	Additional arguments passed to the underlying plot or xyplot functions.
coord	A named vector or list giving the coordinate of the pixel to plot.
plusminus	If specified, a window of pixels surrounding the one given by coord will be included in the plot with fun applied over them, and this indicates the number of pixels to include on either side.
fold	What folds of the cross-validation should be plotted.
mode	What kind of results should be plotted. This is the name of the object to plot in the ResultSet object.
column	What columns of the results should be plotted. If the results are a matrix, this corresponds to the columns to be plotted, which can be indicated either by numeric index or by name.

Author(s)

Kyle D. Bemis

See Also

image-methods

Examples

```
data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
 NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
 1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
 1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)

set.seed(1)
sset <- generateImage(data, range=c(1000,5000), centers=c(3000,4000), resolution=100)

pData(sset)$pg <- factor(data[is.finite(data)], labels=c("black", "red"))
fData(sset)$fg <- factor(rep("bg", nrow(fData(sset))), levels=c("bg", "black", "red"))
fData(sset)$fg[2950 < fData(sset)$t & fData(sset)$t < 3050] <- "black"
fData(sset)$fg[3950 < fData(sset)$t & fData(sset)$t < 4050] <- "red"

plot(sset, pixel=1)

plot(sset, ~ t, pixel=1:ncol(sset))

plot(sset, ~ t | pg, pixel=1:ncol(sset), lattice=TRUE)

plot(sset, ~ t, pixel.groups=pg, pixel=1:ncol(sset), lattice=TRUE, superpose=TRUE)

plot(sset, ~ t | pg, groups=fg, pixel=1:ncol(sset), lattice=TRUE)

set.seed(1)
msset <- generateImage(data, as="MSImageSet", resolution=50)

plot(msset, pixel=1)

plot(msset, coord=list(x=3, y=1))

plot(msset, coord=list(x=3, y=1), plusminus=1)

plot(msset, coord=list(x=5, y=5), plusminus=c(2, 1))
```

`PLS-methods` *Partial Least Squares*

Description

Performs partial least squares (also called projection to latent structures or PLS) on an imaging dataset. This will also perform discriminant analysis (PLS-DA) if the response is a `factor`.

Usage

```
    ## S4 method for signature 'SImageSet,matrix'
PLS(x, y, ncomp = 20,
        method = "nipals",
```

```
        scale = FALSE,
        iter.max = 100, ...)

    ## S4 method for signature 'SImageSet,numeric'
PLS(x, y, ...)

    ## S4 method for signature 'SImageSet,factor'
PLS(x, y, ...)

    ## S4 method for signature 'SImageSet,character'
PLS(x, y, ...)

    ## S4 method for signature 'PLS'
predict(object, newx, newy, ...)
```

Arguments

x	The imaging dataset on which to perform partial least squares.
y	The response variable, which can be a matrix or a vector for ordinary PLS, or a factor or a character for PLS-DA.
ncomp	The number of PLS components to calculate.
method	The function used to calculate the projection.
scale	Shoud the data be scaled first? This is passed to scale.
iter.max	The number of iterations to perform for the NIPALS algorithm.
...	Passed to the next PLS method.
object	The result of a previous call to PLS.
newx	An imaging dataset for which to calculate their PLS projection and predict a response from an already-calculated PLS object.
newy	Optionally, a new response from which residuals should be calcualted.

Value

An object of class PLS, which is a ResultSet, where each component of the resultData slot contains at least the following components:

scores: A matrix with the component scores for the explanatary variable.

loadings: A matrix with the explanatory variable loadings.

weights: A matrix with the explanatory variable weights.

Yscores: A matrix objects with the component scores for the response variable.

Yweights: A matrix objects with the response variable weights.

projection: The projection matrix.

coefficients: The matrix of the regression coefficients.

ncomp: The number of PLS components.

method: The method used to calculate the projection.

`center`: The center of the dataset. Used for calculating PLS scores on new data.

`scale`: The scaling factors for the dataset. Used for PLS scores on new data.

`Ycenter`: The centers of the response variables. Used for predicting new observations.

`Yscale`: The scaling factors for the response variables. Used for predicting new observation.

`fitted`: The fitted response.

Author(s)

Kyle D. Bemis

References

Trygg, J., & Wold, S. (2002). Orthogonal projections to latent structures (O-PLS). Journal of Chemometrics, 16(3), 119-128. doi:10.1002/cem.695

See Also

`OPLS`, `PCA`, `spatialShrunkenCentroids`,

Examples

```
sset <- generateImage(diag(4), range=c(200, 300), step=1)

y <- factor(diag(4))

pls <- PLS(sset, y, ncomp=1:2)
```

---

processingData-methods

Retrieve Pre-Processing Information from MSImageSets

Description

This generic function accesses pre-processing information from `MSImageSet` objects.

Usage

```
processingData(object)
processingData(object) <- value
```

Arguments

`object`	A `MSImageSet` object.
`value`	Value to be assigned to the corresponding object.

Value

`processingData` returns pre-processing information.

Author(s)

Kyle D. Bemis

See Also

MSImageProcess, MSImageSet

---

readMSIData *Read Mass Spectrometry Imaging Data Files*

Description

Read supported mass spectrometry imaging data files. Supported formats include imzML and Analyze 7.5.

Usage

```
## Read any supported MS imaging file
readMSIData(file, ...)

## Read imzML files
readImzML(name, folder=getwd(), attach.only=FALSE,
mass.accuracy=200, units.accuracy=c("ppm", "mz"), ...)

## Read Analyze 7.5 files
readAnalyze(name, folder=getwd(), attach.only=FALSE, ...)
```

Arguments

file	A description of the data file to be read. This may be either an absolute or relative path. The file extension must be included.
name	The common file name for the '.imzML' and '.ibd' files for imzML or for the '.hdr', '.t2m', and '.img' files for Analyze 7.5.
folder	The path to the folder containing the data files.
attach.only	Attach the file as a Binmat on-disk matrix for reading on-demand, rather than loading the data into memory. This feature is still experimental.
mass.accuracy	For 'processed' imzML files, the accuracy to which the m/z values will be binned after reading. This should be set to the native accuracy of the mass spectrometer, if known.
units.accuracy	The units for 'mass.accuracy'.
...	Additional arguments passed to read functions.

Details

In the current implementation, the file extensions must match exactly: '.imzML' and '.ibd' for imzML and '.hdr', '.t2m', and '.img' for Analyze 7.5.

The `readImzML` function currently supports reading and returning the 'continuous' and 'processed' formats. Note that support for the 'processed' format was added in version 1.3.0 and is still considered experimental and under development.

Value

A `MSImageSet` object.

Author(s)

Kyle D. Bemis

References

Schramm T, Hester A, Klinkert I, Both J-P, Heeren RMA, Brunelle A, Laprevote O, Desbenoit N, Robbe M-F, Stoeckli M, Spengler B, Rompp A (2012) imzML - A common data format for the flexible exchange and processing of mass spectrometry imaging data. Journal of Proteomics 75 (16):5106-5110. doi:10.1016/j.jprot.2012.07.026

See Also

`MSImageSet`

`reduceBaseline-methods`

Reduce the Baseline for an Imaging Dataset

Description

Apply baseline reduction to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet'
reduceBaseline(object, method = "median",
    ...,
    pixel = pixels(object),
    plot = FALSE)

## Median baseline reduction
reduceBaseline.median(x, blocks=500, fun=min, spar=1, ...)
```

Arguments

object	An object of class MSImageSet.
method	The baseline reduction method to use.
pixel	The pixels to baseline subtract. If less than the extent of the dataset, this will result in a subset of the data being processed.
plot	Plot the mass spectrum for each pixel while it is being processed?
...	Additional arguments passed to the baseline reduction method.
x	The mass spectrum to be baseline subtracted.
blocks	The number of intervals to break the mass spectrum into in order to choose minima or medians from which to interpolate the baseline.
fun	Function used to determine the points from which the baseline will be interpolated.
spar	Smoothing parameter for the spline smoothing applied to the spectrum in order to decide the cutoffs for throwing away baseline references that might occur inside peaks.

Details

Baseline reduction is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: A numeric vector of intensities.
- ...: Additional arguments.

A user-created function should return a numeric vector of the same length. with the baseline-subtracted intensities.

Internally, pixelApply is used to apply the baseline reduction. See its documentation page for more details on additional objects available to the environment installed to the baseline reduction function.

Value

An object of class MSImageSet with the baseline-subtracted spectra.

Author(s)

Kyle D. Bemis

See Also

MSImageSet, pixelApply

Examples

```
data <- generateImage(as="MSImageSet")
reduceBaseline(data, method="median", plot=interactive())
```

reduceDimension-methods

Reduce the Dimension of an Imaging Dataset

Description

Apply dimension reduction to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet,missing'
reduceDimension(object, method = c("bin", "resample"),
    ...,
    pixel = pixels(object),
    plot = FALSE)

## S4 method for signature 'MSImageSet,numeric'
reduceDimension(object, ref, method = "peaks", ...)

## S4 method for signature 'MSImageSet,MSImageSet'
reduceDimension(object, ref, method = "peaks", ...)

## Bin the signal
reduceDimension.bin(x, t, width=200, offset=0, units=c("ppm","mz"), fun=sum, ...)

## Resample the signal
reduceDimension.resample(x, t, step=1, offset=0, ...)

## Reduce the signal to peaks
reduceDimension.peaks(x, t, peaklist, type=c("height", "area"), ...)
```

Arguments

object	An object of class MSImageSet.
ref	A reference to use to reduce the dimension, usually a peak list of m/z values or a peak-picked and aligned MSImageSet.
method	The method to use to reduce the dimensions of the signal.
pixel	The pixels to process. If less than the extent of the dataset, this will result in a subset of the data being processed.
plot	Plot the mass spectrum for each pixel while it is being processed?
...	Additional arguments passed to the dimension reduction method.
x	The mass spectrum to be reduced.
t	The corresponding m/z values.
width	The width of a bin.

step	The step size.
offset	Offset from the nearest integer.
units	Either parts-per-million or the raw m/z values.
fun	The function to be applied to each bin.
peaklist	A numeric vector giving the m/z values of the reference peaks.
type	Should the peak height or area under the curve be taken as the intensity value?

Details

Dimension reduction is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: A numeric vector of intensities.
- t: A numeric vector of m/z values.
- tout: A numeric vector of m/z values to output.
- ...: Additional arguments.

The optional argument tout was added in version 1.3.1 to avoid cases where the output m/z values may be costly and inefficient to re-calculate for every spectrum.

A user-created function should return a list with two vectors of equal length, where the new length *must* be shorter than x and t:

- x: A numeric vector of new intensities.
- t: A numeric vector of new m/z values.

Internally, pixelApply is used to apply the dimension reduction. See its documentation page for more details on additional objects available to the environment installed to the dimension reduction function.

Value

An object of class MSImageSet with the dimension-reduced spectra.

Author(s)

Kyle D. Bemis

See Also

MSImageSet, peakPick, peakAlign, pixelApply

Examples

```
data <- generateImage(as="MSImageSet")
reduceDimension(data, method="resample", step=100, plot=interactive())
```

ResultSet-class *Class to Contain Analysis Results for Imaging Experiments*

Description

This class is used as a return value by most of the analysis methods provided by Cardinal, including PCA, PLS, OPLS, spatialKMeans, spatialShrunkenCentroids.

Slots

imageData: This slot is unused in a ResultSet.

pixelData: The pixelData from the analyzed dataset.

featureData: The featureData from the analyzed dataset.

experimentData: The experimentData from the analyzed dataset.

protocolData: The protocolData from the analyzed dataset.

resultData: A list of analysis results. Each element contains the results from a different parameter set.

modelData: An AnnotatedDataFrame containing information about the parameters of the models in resultData.

.__classVersion__: A Versions object describing the version of the class used to created the instance. Intended for developer use.

Extends

iSet, directly. VersionedBiobase, by class "iSet", distance 1. Versioned, by class "VersionedBiobase", distance 2.

Creating Objects

ResultSet is a virtual class. No instances can be created.

Methods

Class-specific methods:

resultData(object): Access and set the results of the analyses.

modelData(object): Access and set the model parameters.

Standard generic methods:

length(x): Access the number of elements of resultData.

names(x): Access the names of the components of all of the elements of resultData.

ResultSet$name: Access all of the result components with the name name.

ResultSet[[i, ...]]: Access ith element of the resultData slot.

ResultSet[i, j, ..., drop]: Subset an ResultSet based on the model parameters in modelData.

See iSet for additional methods.

Author(s)

Kyle D. Bemis

See Also

iSet, PCA, PLS, OPLS, spatialKMeans, spatialShrunkenCentroids

---

select-methods *Select Regions of an Imaging Dataset*

Description

Manually select regions-of-interest or pixels on an imaging dataset. This uses the built-in locator function. The method has the same form as the image method for plotting imaging datasets.

Usage

```
## S4 method for signature 'SImageSet'
select(x, formula = ~ x * y,
    mode = c("region", "pixel"),
    ...,
    main,
    subset = TRUE,
    lattice = FALSE)
```

Arguments

x	An imaging dataset.
formula	Passed to image.
mode	What kind of selection to perform: 'region' to select a region-of-interest, or 'pixel' to select individual pixels.
...	Addtional arguments to be passed to image.
main	Passed to image.
subset	Passed to image.
lattice	Must be false.

Value

A logical vector of length equal to the number of pixels.

Author(s)

Kyle D. Bemis

See Also

image

SImageData-class *Class Containing Sparse Image Data*

Description

A container class for holding pixel-sparse image as a virtual datacube. It is assumed there will be missing pixels, so the feature vectors are stored as a matrix for memory efficiency, and the datacube is reconstructed on-the-fly. The implementation remains efficient even for non-sparse data as long as the full datacube does not need to be reconstructed as often as single images and feature vectors. All elements of `data` must have an identical number of rows (features) and columns (pixels).

Usage

```
## Instance creation
SImageData(
    data = Hashmat(nrow=0, ncol=0),
    coord = expand.grid(
        x = seq_len(ncol(data)),
        y = seq_len(ifelse(ncol(data) > 0, 1, 0))),
    storageMode = "immutableEnvironment",
    positionArray = generatePositionArray(coord),
    dimnames = NULL,
    ...)

## Additional methods documented below
```

Arguments

`data`	A matrix-like object with number of rows equal to the number of features and number of columns equal to the number of non-missing pixels. Each column should be a feature vector. Alternatively, a multidimensional array that represents the datacube with the first dimension as the features can also be supplied. Additional dimensions could be the spatial dimensions of the image, for example.
`coord`	A `data.frame` with columns representing the spatial dimensions. Each row provides a spatial coordinate for the location of a feature vector corresponding to a column in `data`. This argument is ignored if `data` is a multidimensional array rather than a matrix.
`storageMode`	The storage mode to use for the `SImageData` object for the environment in the `data` slot. Only `"immutableEnvironment"` is allowed for `SImageData`. See documentation on the `storageMode` slot below for more details.
`positionArray`	The `positionArray` for the imaging data. This should not normally be specified the user, since it is generated automatically from the `coord` argument, unless for some reason `coord` is not specified.
`dimnames`	A `list` of length two, giving the feature names and pixel names in that order. If missing, this is taken from the 'dimnames' of the `data` argument.

... Additional Named arguments that are passed to the initialize method for instantiating the object. These must be matrices or matrix-like objects of equal dimension to data. They will be assigned into the environment in the data slot.

Slots

data: An environment which contains at least one element named "iData", which is a matrix-like object with rows equal to the number of features and columns equal to the number of non-missing pixels. Each column is a feature vector.

coord: An data.frame with rows giving the spatial coordinates of the pixels corresponding to the columns of "iData".

positionArray: An array with dimensions equal to the spatial dimensions of the image, which stores the column numbers of the feature vectors corresponding to the pixels in the "iData" element of the data slot. This allows re-construction of the imaging "datacube" on-the-fly.

dim: A length 2 integer vector analogous to the 'dim' attribute of an ordinary R matrix.

dimnames: A length 2 list analogous to the 'dimnames' attribute of an ordinary R matrix.

storageMode: A character which is one of "immutableEnvironment", "lockedEnvironment", or "environment". The values "lockedEnvironment" and "environment" behave as described in the documentation of AssayData. An "immutableEnvironment" uses a locked environment while retaining R's typical copy-on-write behavior. Whenever an object in an immutable environment is modified, a new environment is created for the data slot, and all objects copied into it. This allows usual R functional semantics while avoiding copying of large objects when other slots are modified.

.__classVersion__: A Versions object describing the version of the class used to created the instance. Intended for developer use.

Extends

Versioned

Creating Objects

SImageData instances are usually created through SImageData().

Methods

Class-specific methods:

iData(object), iData(object)<-: Return or set the matrix of image intensities. Columns should correspond to feature vectors, and rows should correspond to pixel vectors.

coord(object), coord(object)<-: Return or set the coodinates. This is a data.frame with each row corresponding to the spatial coordinates of a pixel.

positionArray(object), positionArray(object)<-: Return or set the positionArray slot. When setting, this should be an array returned by a call to generatePositionArray.

featureNames(object), featureNames(object) <- value: Access and set feature names (names of the rows of the intensity matrix).

`pixelNames(object)`, `pixelNames(object) <- value`: Access and set the pixel names (names of the columns of the intensity matrix).

`storageMode(object)`, `storageMode(object)<-`: Return or set the storage mode. See documentation on the `storageMode` slot above for more details.

Standard generic methods:

`combine(x, y, ...)`: Combine two or more `SImageData` objects. Elements must be matrix-like objects and are combined column-wise with a call to 'cbind'. The numbers of rows must match, but otherwise no checking of row or column names is performed. The pixel coordinates are checked for uniqueness.

`dim`: Return the dimensions of the (virtual) datacube. This is equal to the number of features (the number of rows in the matrix returned by `iData`) and the dimensions of the `positionArray` slot. For a standard imaging dataset, that is the number features followed by the spatial dimensions of the image.

`dims`: A matrix where each column corresponds to the dimensions of the (virtual) datacubes stored as elements in the `data` slot. See above for how the dimensions are calculated.

`SImageData[i, j, ..., drop]`: Access intensities in the (virtual) imaging datacube. The datacube is reconstructed on-the-fly. The object can be indexed like any ordinary array with number of dimensions equal to `dim(object)`. Use `drop = NA` to return a subset of the same class as the object.

Author(s)

Kyle D. Bemis

See Also

`ImageData`, `MSImageData`, `SImageSet`, `MSImageSet`

Examples

```
## Create an SImageData object
SImageData()

## Using a P x N matrix
data1 <- matrix(1:27, nrow=3)
coord <- expand.grid(x=1:3, y=1:3)
sdata1 <- SImageData(data1, coord)
sdata1[] # extract data as array

## Using a P x X x Y array
data2 <- array(1:27, dim=c(3,3,3))
sdata2 <- SImageData(data2)
sdata2[] # should be identical to above

# Missing data from some pixels
data3 <- matrix(1:9, nrow=3)
sdata3 <- SImageData(data3, coord[c(1,5,9),])
```

```
dim(sdata3) # presents as an array
iData(sdata3) # stored as matrix
sdata3[] # recontruct the datacube

iData(sdata3)[,1] <- 101:103 # assign using iData()
sdata3[] # can only assign into matrix representation

## Sparse feature vectors
data4 <- Hashmat(nrow=9, ncol=9)
sdata4 <- SImageData(data4, coord)
iData(sdata4)[] <- diag(9)
sdata4[1,,]
```

SImageSet-class *Class to Contain Pixel-Sparse Imaging Data*

Description

An iSet derived class for pixel-sparse imaging data. Data is stored to be memory efficient when there are missing pixels or when the the stored images are non-rectangular regions. The data structures remain efficient for non-sparse pixel data as long as the full datacube does not need to be reconstructed often, and single images or feature vectors are of primary interest. This class can be combined with Hashmat to be sparse in both feature space and pixel space. This is useful for datasets with sparse signals, such as processed spectra.

MSImageSet is a derived class of SImageSet for storing mass spectrometry imaging experiments.

Usage

```
## Instance creation
SImageSet(
    data = Hashmat(nrow=0, ncol=0),
    coord = expand.grid(
        x = seq_len(prod(dim(data)[-1])),
        y = seq_len(ifelse(prod(dim(data)[-1]) > 0, 1, 0))),
    imageData = SImageData(
        data=data,
        coord=coord),
    pixelData = IAnnotatedDataFrame(
        data=coord,
        varMetadata=data.frame(labelType=rep("dim", ncol(coord)))),
    featureData = AnnotatedDataFrame(
        data=data.frame(row.names=seq_len(nrow(data)))),
    protocolData = AnnotatedDataFrame(
        data=data.frame(row.names=sampleNames(pixelData))),
    experimentData = new("MIAPE-Imaging"),
    ...)

## Additional methods documented below
```

Arguments

data	A matrix-like object with number of rows equal to the number of features and number of columns equal to the number of non-missing pixels. Each column should be a feature vector. Alternatively, a multidimensional array that represents the datacube with the first dimension as the features can also be supplied. Additional dimensions could be the spatial dimensions of the image, for example.
coord	A data.frame with columns representing the spatial dimensions. Each row provides a spatial coordinate for the location of a feature vector corresponding to a column in data. This argument is ignored if data is a multidimensional array rather than a matrix.
imageData	An object of class SImageData that will contain the imaging data. Usually constructed using data and coord.
pixelData	An object of class IAnnotatedDataFrame giving the information about the pixels including coordinates of the data in imageData.
featureData	An object of class AnnotatedDataFrame giving information about the data features.
protocolData	An object of class AnnotatedDataFrame giving information about the samples. It must have one row for each of the sampleNames in pixelData.
experimentData	An object derived from class MIAxE giving information about the imaging experiment.
...	Additional arguments passed to the initializer.

Slots

imageData: An instance of SImageData, which stores one or more matrices of equal number of dimensions as elements in an 'immutableEnvironment'. This slot preserves copy-on-write behavior when it is modified specifically, but is pass-by-reference otherwise, for memory efficiency.

pixelData: Contains pixel information in an IAnnotatedDataFrame. This includes both pixel coordinates and phenotypic and sample data. Its rows correspond to the columns in imageData.

featureData: Contains variables describing features in an IAnnotatedDataFrame. Its rows correspond to the rows in imageData.

experimentData: Contains details of experimental methods. Should be an object of a derived class of MIAxE.

protocolData: Contains variables in an IAnnotatedDataFrame describing the generation of the samples in pixelData.

.__classVersion__: A Versions object describing the version of the class used to created the instance. Intended for developer use.

Extends

iSet, directly. VersionedBiobase, by class "iSet", distance 1. Versioned, by class "VersionedBiobase", distance 2.

Creating Objects

`SImageSet` instances are usually created through `SImageSet()`.

Methods

Class-specific methods:

`iData(object), iData(object) <- value`**:** Access and set the sparse image data in `imageData`. This is a matrix-like object with rows corresponding to features and columns corresponding to pixels, so that each column of the returned object is a feature vector.

`regeneratePositions`**:** Regenerates the `positionArray` in `imageData` used to reconstruct the datacube based on the coordinates in `pixelData`. Normally, this should not be called by the user. However, if the coordinates are modified manually, it can be used to re-sync the data structures.

Standard generic methods:

`combine(x, y, ...)`**:** Combine two or more `SImageSet` objects. Unique 'sample's in `pixelData` are treated as a dimension.

`SImageSet[i, j, ..., drop]`**:** Subset an `SImageSet` based on the rows (featureData components) and the columns (pixelData components). The result is a new `SImageSet`.

See `iSet` for additional methods.

Author(s)

Kyle D. Bemis

See Also

`iSet`, `SImageData`, `MSImageSet`

Examples

```
## Create an SImageSet object
data <- matrix(1:27, nrow=3)
coord <- expand.grid(x=1:3, y=1:3)
sset <- SImageSet(data=data, coord=coord)

## Access a single image corresponding to the first feature
imageData(sset)[1,,]

## Reconstruct the datacube
imageData(sset)[]

## Access the P x N matrix of column-wise feature vectors
iData(sset)

## Subset the SImageSet to the first 2 features and first 6 pixels
sset2 <- sset[1:2, 1:6]
imageData(sset2)[]
sset2
```

`smoothSignal-methods` *Smooth the Feature-Vectors of an Imaging Dataset*

Description

Apply smoothing to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImageSet'
smoothSignal(object, method = c("gaussian", "sgolay", "ma"),
    ...,
    pixel = pixels(object),
    plot = FALSE)

## Gaussian smoothing
smoothSignal.gaussian(x, sd=window/4, window=5, ...)

## Savitsky-Golay smoothing
smoothSignal.sgolay(x, order=3, window=order + 3 - order %% 2, ...)

## Moving average smoothing
smoothSignal.ma(x, coef=rep(1, window + 1 - window %% 2), window=5, ...)
```

Arguments

`object`	An object of class `MSImageSet`.
`method`	The smoothing method to use.
`pixel`	The pixels to smooth. If less than the extent of the dataset, this will result in a subset of the data being processed.
`plot`	Plot the mass spectrum for each pixel while it is being processed?
`...`	Additional arguments passed to the smoothing method.
`x`	The mass spectrum to be smoothed.
`sd`	The standard deviation for the Gaussian kernel.
`window`	The smoothing window.
`order`	The order of the smoothing filter.
`coef`	The coefficients for the moving average filter.

Details

Smoothing is usually performed using the provided functions, but a user-created function can also be passed to `method`. In this case it should take the following arguments:

- `x`: A `numeric` vector of intensities.

- . . . : Additional arguments.

A user-created function should return a `numeric` vector of the same length.

Internally, `pixelApply` is used to apply the smoooothing. See its documentation page for more details on additional objects available to the environment installed to the smoothing function.

Value

An object of class `MSImageSet` with the smoothed spectra.

Author(s)

Kyle D. Bemis

See Also

`MSImageSet`, `pixelApply`

Examples

```
data <- generateImage(as="MSImageSet")
smoothSignal(data, method="gaussian", plot=interactive())
```

---

spatialKMeans-methods *Spatially-Aware K-Means Clustering*

Description

Performs spatially-aware (SA) or spatially-aware structurally-adaptive (SASA) clustering of imaging data. The data are first projected into an embedded feature space where spatial structure is maintained using the Fastmap algorithm, and then ordinary k-means clustering is performed on the projected dataset.

Usage

```
## S4 method for signature 'SImageSet'
spatialKMeans(x, r = 1, k = 2,
    method = c("gaussian", "adaptive"),
    weights = 1, iter.max = 100, nstart = 100,
    algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
        "MacQueen"),
    ncomp = 20, ...)
```

Arguments

x	The imaging dataset to cluster.
r	The spatial neighborhood radius of nearby pixels to consider. This can be a vector of multiple radii values.
k	The number of clusters. This can be a vector to try different numbers of clusters.
method	The method to use to calculate the spatial smoothing kernels for the embedding. The 'gaussian' method refers to spatially-aware (SA) clustering, and 'adaptive' refers to spatially-aware structurally-adaptive (SASA) clustering.
weights	An optional vector of feature weights to be applied to the features during the clustering.
iter.max	The maximum number of k-means iterations.
nstart	The number of restarts for the k-means algorithm.
algorithm	The k-means algorithm to use. See kmeans for details.
ncomp	The number of fastmap components to calculate.
...	Ignored.

Value

An object of class SpatialKMeans, which is a ResultSet, where each component of the resultData slot contains at least the following components:

cluster: A vector of integers indicating the cluster for each pixel in the dataset.

centers: A matrix of cluster centers.

time: The amount of time the algorithm took to run.

r: The neighborhood spatial smoothing radius.

k: The number of clusters.

method: The method for calculating spatial distances.

weights: The feature weights (defaults to 1s).

fastmap: A list with components giving details of the Fastmap projection.

Author(s)

Kyle D. Bemis

References

Alexandrov, T., & Kobarg, J. H. (2011). Efficient spatial segmentation of large imaging mass spectrometry datasets with spatially aware clustering. Bioinformatics, 27(13), i230-i238. doi:10.1093/bioinformatics/btr246

Faloutsos, C., & Lin, D. (1995). FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. Presented at the Proceedings of the 1995 ACM SIGMOD international conference on Management of data.

See Also

spatialShrunkenCentroids

Examples

```
set.seed(1)
data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
    NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
    1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
    1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
    1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)

sset <- generateImage(data, range=c(200, 300), step=1)

clust1 <- spatialKMeans(sset, r=c(1,2), k=c(2,3), method="gaussian")

clust2 <- spatialKMeans(sset, r=c(1,2), k=c(2,3), method="adaptive")
```

---

spatialShrunkenCentroids-methods

Spatially-Aware Shrunken Centroid Clustering and Classification

---

Description

Performs spatially-aware nearest shrunken centroid clustering or classification on an imaging dataset. These methods use statistical regularization to shrink the t-statistics of the features toward 0 so that unimportant features are removed from the analysis. A Gaussian spatial kernel or an adaptive kernel based on bilateral filtering are used for spatial smoothing.

Usage

```
## S4 method for signature 'SImageSet,missing'
spatialShrunkenCentroids(x, y, r = 1, k = 2, s = 0,
    method = c("gaussian", "adaptive"),
    iter.max=10, ...)

## S4 method for signature 'SImageSet,factor'
spatialShrunkenCentroids(x, y, r = 1, s = 0,
    method = c("gaussian", "adaptive"),
    priors = table(y), ...)

## S4 method for signature 'SImageSet,character'
spatialShrunkenCentroids(x, y, ...)

## S4 method for signature 'SpatialShrunkenCentroids'
predict(object, newx, newy, ...)
```

Arguments

x	The imaging dataset to cluster.
y	A factor or character response.
r	The spatial neighborhood radius of nearby pixels to consider. This can be a vector of multiple radii values.
k	The number of clusters. This can be a vector to try different numbers of clusters.
s	The sparsity thresholding parameter by which to shrink the t-statistics.
method	The method to use to calculate the spatial smoothing kernels for the embedding. The 'gaussian' method refers to spatially-aware (SA) weights, and 'adaptive' refers to spatially-aware structurally-adaptive (SASA) weights.
iter.max	The maximum number of clustering iterations.
priors	Prior probabilities on the classes for classification. Improper priors will be normalized automatically.
...	Ignored.
object	The result of a previous call to spatialShrunkenCentroids.
newx	An imaging dataset for which to calculate the predicted response from shrunken centroids.
newy	Optionally, a new response from which residuals should be calculated.

Value

An object of class SpatialShrunkenCentroids, which is a ResultSet, where each component of the resultData slot contains at least the following components:

classes: A factor indicating the predicted class for each pixel in the dataset.

centers: A matrix of shrunken class centers.

time: The amount of time the algorithm took to run.

r: The neighborhood spatial smoothing radius.

k: The number of clusters.

s: The sparsity parameter.

method: The type of spatial kernel used.

scores: A matrix of discriminant scores.

probabilities: A matrix of class probabilities.

tstatistics: A matrix of shrunken t-statistics of the features.

sd: The pooled within-class standard deviations for each feature.

iter: The number of iterations performed.

Author(s)

Kyle D. Bemis

References

Tibshirani, R., Hastie, T., Narasimhan, B., & Chu, G. (2003). Class Prediction by Nearest Shrunken Centroids, with Applications to DNA Microarrays. Statistical Science, 18, 104-117.

Alexandrov, T., & Kobarg, J. H. (2011). Efficient spatial segmentation of large imaging mass spectrometry datasets with spatially aware clustering. Bioinformatics, 27(13), i230-i238. doi:10.1093/bioinformatics/btr246

See Also

spatialKMeans

Examples

```
set.seed(1)
data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
    NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
    1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
    1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
    1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)

sset <- generateImage(data, range=c(200, 300), step=1)

clust1 <- spatialShrunkenCentroids(sset, r=c(1,2), k=c(2,3), s=c(0,1), method="gaussian")

clust2 <- spatialShrunkenCentroids(sset, r=c(1,2), k=c(2,3), s=c(0,1), method="adaptive")

y <- factor(data[!is.na(data)], labels=c("black", "red"))

class1 <- spatialShrunkenCentroids(sset, y, r=c(1,2), s=c(0,1), method="gaussian")

class1 <- spatialShrunkenCentroids(sset, y, r=c(1,2), s=c(0,1), method="adaptive")
```

---

standardizeSamples-methods

Standardize the Samples in an Imaging Dataset

Description

Apply standardization across the samples in a mass spectrometry imaging dataset to correct for between-sample variation.

Usage

```
## S4 method for signature 'MSImageSet'
standardizeSamples(object, method = "sum", ...)

## TIC normalization
standardizeSamples.sum(x, sum=length(x), ...)
```

Arguments

object	An object of class MSImageSet.
method	The standardization method to use.
...	Additional arguments passed to the standardization method.
x	The flattened ion image to be standardized.
sum	The value to which to standardize the sum of the ion image intensity values.

Details

Standardization is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: A numeric vector of intensities.
- ...: Additional arguments.

A user-created function should return a numeric vector of the same length.

Internally, featureApply is used to apply the standardization, with .pixel.groups=sample. See its documentation page for more details on additional objects available to the environment installed to the standardization function.

Value

An object of class MSImageSet with the ion images standardized across samples.

Author(s)

Kyle D. Bemis

See Also

MSImageSet, featureApply

Examples

```
data1 <- generateImage(as="MSImageSet")
data2 <- generateImage(as="MSImageSet")
sampleNames(data2) <- "2"
data3 <- combine(data1, data2)
standardizeSamples(data3, method="sum")
```

topLabels-methods *Retrieve Top-Ranked Labels from Analysis Results*

Description

The generic function is a convenience method for retrieving top-ranked labels from the results of imaging experiment analyses. For mass spectrometry-based imaging experiments, this can be used for identifying important masses from an analysis.

Usage

```
## S4 method for signature 'ResultSet'
topLabels(object, n = 6,
    model = pData(modelData(object)),
    type = c('+', '-', 'b'),
    sort.by = fvarLabels(object),
    filter = list(),
    ...)

## S4 method for signature 'PCA'
topLabels(object, n = 6,
    sort.by = "loadings",
    ...)

## S4 method for signature 'PLS'
topLabels(object, n = 6,
    sort.by = c("coefficients", "loadings", "weights"),
    ...)

## S4 method for signature 'OPLS'
topLabels(object, n = 6,
    sort.by = c("coefficients",
            "loadings", "Oloadings",
            "weights", "Oweights"),
    ...)

## S4 method for signature 'SpatialKMeans'
topLabels(object, n = 6,
    sort.by = c("betweenss", "withinss"),
    ...)

## S4 method for signature 'SpatialShrunkenCentroids'
topLabels(object, n = 6,
    sort.by = c("tstatistics", "p.values"),
    ...)

## S4 method for signature 'CrossValidated'
```

```
topLabels(object, ...)
```

Arguments

object	A ResultSet derived object.
n	The number of top-ranked records to return.
model	If more than one model was fitted, results from which should be shown? Defaults to all models in the ResultSet. This can name the models explicitly or specify a list of parameter values.
type	How should the records be ranked? '+' shows greatest values first (decreasing order), '-' shows least values first (increasing order), and 'b' uses decreasing order based on absolute values.
sort.by	What variable should be used for sorting?
filter	A list of named variables with values to use to filter the results. For example, for testing or classification, this can be used to only show rankings for a particular condition.
...	Passed to the 'head' function when sorting the final list of results.

Value

A data.frame with the top-ranked labels from the analysis.

Author(s)

Kyle D. Bemis

See Also

ResultSet, PCA, PLS, OPLS, spatialKMeans, spatialShrunkenCentroids

Examples

```
set.seed(1)
data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
    NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
    1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
    1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
    1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)

msset <- generateImage(data, range=c(200, 300), step=1, as="MSImageSet")

clust1 <- spatialShrunkenCentroids(msset, r=c(1,2), k=c(2,3), s=c(0,1), method="gaussian")

topLabels(clust1)

topLabels(clust1, filter=list(classes=1))

topLabels(clust1, filter=list(r=1, k=2, s=1))
```

Index