Package 'Cardinal'

November 5, 2025

Type Package

Title A mass spectrometry imaging toolbox for statistical analysis

Version 3.13.0

Date 2015-1-12

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 | file LICENSE

Depends R (>= 4.4), BiocParallel, BiocGenerics, ProtGenerics, S4Vectors, methods, stats, stats4

Imports CardinalIO, Biobase, EBImage, graphics, grDevices, irlba, Matrix, matter (>= 2.7.10), nlme, parallel, utils

Suggests BiocStyle, testthat, knitr, rmarkdown, emmeans, lme4, lmerTest

VignetteBuilder knitr

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

URL http://www.cardinalmsi.org

BugReports https://github.com/kuwisdelu/Cardinal/issues

git_url https://git.bioconductor.org/packages/Cardinal

git_branch devel

git_last_commit 2d83be5

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-05

Author Kylie Ariel Bemis [aut, cre]

Maintainer Kylie Ariel Bemis <k.bemis@northeastern.edu>

Contents

`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 `browseVignettes("Cardinal")` to view a user's guide and vignettes of common workflows.

Options

The following Cardinal-specific options are available:

- `getCardinalParallel()`, `setCardinalParallel(workers=snowWorkers())`: Set up a default parallelization backend (if passed `TRUE`, a number of workers, or a vector of node names, or turn off parallelization (if `FALSE` or `NULL`.
- `getCardinalBPPARAM()`, `setCardinalBPPARAM(BPPARAM=NULL)`: The default backend to use for parallel processing. By default, this is initially set to `NULL` (no parallelization). Otherwise, it must be a `BiocParallelParam` instance. See documentation for `bplapply`.
- `getCardinalVerbose()`, `setCardinalVerbose(verbose=interactive())`: Should progress messages be printed?

The following Cardinal-controlled `matter` chunk options are available:

- `getCardinalNChunks()`, `setCardinalNChunks(nchunks=20L)`: The default number of data chunks used when iterating over large datasets. Used by many methods internally.
- `getCardinalChunksize()`, `setCardinalChunksize(chunksize=NA, units=names(chunksize))`: The approximate size of data chunks used when iterating over large datasets. Can be used as an alternative to setting the number of chunks. The default (`NA`) means to ignore this parameter and use the `getCardinalNChunks()`.

- `getCardinalSerialize()`, `setCardinalSerialize(serialize=NA)`: Whether data chunks should be loaded on the manager and serialized to the workers (`TRUE`), or loaded on the workers (`FALSE`). The default (`NA`) means to choose automatically based on the type of data and the type of cluster.

The following Cardinal-controlled `matter` logging options are available:

- `getCardinalLogger()`, `setCardinalLogger(logger=matter_logger())`: The logger used by Cardinal for messages, warnings, and errors. The logger must be of class `simple_logger`.
- `saveCardinalLog(file="Cardinal.log")`): Save the log to a file. Note that Cardinal will continue to log to the specified file until the end of the R session or until saved to a new location.

Additionally, visualization parameters are available:

- `vizi_style()`: Set the default plotting style and color palettes.
- `vizi_engine()`: Set the default plotting engine.
- `vizi_par()`: Set default graphical parameters.

Author(s)

Kylie A. Bemis

Maintainer: Kylie A. Bemis <k.bemis@northeastern.edu>

`bin` *Bin spectra*

Description

Apply on-the-fly binning to spectra.

Usage

```
## S4 method for signature 'MSImagingExperiment'
bin(x, ref,
    spectra = "intensity", index = "mz",
    method = c("sum", "mean", "max", "min",
        "linear", "cubic", "gaussian", "lanczos"),
    resolution = NA, tolerance = NA, units = c("ppm", "mz"),
    mass.range = NULL, ...)

## S4 method for signature 'MSImagingArrays'
bin(x, ref,
    spectra = "intensity", index = "mz",
    method = c("sum", "mean", "max", "min",
        "linear", "cubic", "gaussian", "lanczos"),
    resolution = NA, tolerance = NA, units = c("ppm", "mz"),
```

```
    mass.range = NULL, ...)

## S4 method for signature 'SpectralImagingExperiment'
bin(x, ref,
    spectra = "intensity", index = NULL,
    method = c("sum", "mean", "max", "min",
        "linear", "cubic", "gaussian", "lanczos"),
    resolution = NA, tolerance = NA, units = c("relative", "absolute"),
    verbose = getCardinalVerbose(), ...)

## S4 method for signature 'SpectralImagingArrays'
bin(x, ref,
    spectra = "intensity", index = NULL,
    method = c("sum", "mean", "max", "min",
        "linear", "cubic", "gaussian", "lanczos"),
    resolution = NA, tolerance = NA, units = c("relative", "absolute"),
    verbose = getCardinalVerbose(), ...)
```

Arguments

x	A spectral imaging dataset.
ref	Optional. The bin centers, or their range if resolution is specified. Created from resolution if not provided.
spectra	The name of the array in spectraData() to bin.
index	The name of the array in spectraData() (for SpectralImagingArrays) or column in featureData() (for SpectralImagingExperiment) to use for the bins.
method	The peak picking method to use. See approx1 for details.
resolution	The spacing between bin centers. If tolerance is not provided, then this is also used to calculate the bin width.
tolerance	The half-bin width.
units	The units for the above resolution.
mass.range	An alternative way of specifying the mass range (replaces the value of ref).
verbose	Should progress messages be printed?
...	Ignored.

Details

The binning is applied but not processed immediately. It is performed on-the-fly whenever the spectra are accessed.

Value

A new object derived from SpectralImagingExperiment with the binned spectra.

Author(s)

Kylie A. Bemis

See Also

approx1, estimateDomain, estimateReferenceMz

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3))

# bin to unit resolution
mse2 <- bin(mse, resolution=1, units="mz")

# bin to a specific range and resolution
mse3 <- bin(mse, ref=c(800, 1000), resolution=1, units="mz")
```

---

colocalized *Colocalized features*

Description

Find colocalized features in an imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment'
colocalized(object, mz, ...)

## S4 method for signature 'SpectralImagingExperiment'
colocalized(object, i, ref,
    threshold = median, n = Inf,
    sort.by = c("cor", "MOC", "M1", "M2", "Dice", "none"),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialDGMM'
colocalized(object, ref,
    threshold = median, n = Inf,
    sort.by = c("MOC", "M1", "M2", "Dice", "none"),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

object	An imaging experiment.
mz	An m/z value of a feature in object to use as a reference.
i	The index of a feature in object to use as a reference.
ref	Either a flattened image (i.e., a numeric vector) or a logical mask of a region-of-interest to use as a reference.
threshold	Either a function that returns the cutoff to use for creating logical masks of numeric references, or a numeric threshold to use.
n	The number of top-ranked colocalized features to return.
sort.by	The colocalization measure used to rank colocalized features. Possible options include Pearson's correlation ("cor"), Manders overlap coefficient ("MOC"), Manders colocalization coefficients ("M1" and "M2"), and Dice similarity coefficient ("Dice".
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Options passed to chunkApply.

Value

A data frame with the colocalized features, or a list of data frames if multiple references are given.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- simulateImage(preset=1, dim=c(10,10), centroided=TRUE)

# find features colocalized with first feature
colocalized(x, i=1)
```

---

deprecated *Deprecated and defunct objects in Cardinal*

Description

These functions are provided for compatibility with older versions of Cardinal, and will be removed in the future.

`estimateDomain` *Estimate shared domain*

Description

For unaligned spectral data, it is often necessary to estimate a suitable shared domain in order to calculate statistical summaries like the mean spectrum.

Usage

```
estimateDomain(xlist,
    width = c("median", "min", "max", "mean"),
    units = c("relative", "absolute"),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

estimateReferenceMz(object,
    width = c("median", "min", "max", "mean"),
    units = c("ppm", "mz"),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

estimateReferencePeaks(object, SNR = 2,
    method = c("diff", "sd", "mad", "quantile", "filter", "cwt"),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

`xlist`	A list of the domain values (e.g., m/z values) for each spectrum.
`object`	A mass spectral imaging dataset.
`units`	Should the spacing between domain values use absolute or relative units?
`width`	How the domain spacing should be estimated from the distribution of resolutions across all spectra.
`method`	The peak picking method to use. See `findpeaks` for details.
`SNR`	The signal-to-noise threshold to use to determine a peak.
`verbose`	Should progress messages be printed?
`chunkopts`	Chunk processing options. See `chunkApply` for details.
`BPPARAM`	An optional instance of `BiocParallelParam`. See documentation for `bplapply`.
`...`	Options passed to `chunkLapply`.

Details

For `estimateDomain`, the domain is estimated by first finding the resolution of each spectrum's individual domain values (e.g., the spacing between m/z values), and then creating a sequence of domain values using (by default) the median resolution of all spectra.

The `estimateReferenceMz` function simply calls `estimateDomain` on the appropriate components of a mass spectral imaging dataset to estimate profile m/z bins.

The `estimateReferencePeaks` function calculates the mean spectrum (or looks for a "mean" column in `featureData()`) and performs peak picking on the mean spectrum. It can be used to create a set of reference peaks if all relevant peaks appear in the mean spectrum.

Value

A vector of domain values, m/z values, or peaks.

Author(s)

Kylie A. Bemis

See Also

`summarizeFeatures`, `recalibrate`, `peakAlign`, `peakProcess`

`features` *Find feature indices*

Description

Search for the row indices of a spectral imaging dataset that correspond to specificor features, based on a set of conditions.

Usage

```
## S4 method for signature 'MSImagingExperiment'
features(object, ..., mz, tolerance = NA, units = c("ppm", "mz"),
    env = NULL)

## S4 method for signature 'SpectralImagingExperiment'
features(object, ..., env = NULL)
```

Arguments

`object`	A spectral imaging dataset.
`...`	Expressions that evaluate to logical vectors in the environment of `featureData()`.
`mz`	The m/z values of features to include.
`tolerance`	The tolerance for matching features to m/z values.
`units`	The units for the above tolerance.
`env`	The enclosing environment for evaluating `...`.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

features(mse, mz > 800, mz < 1800)
features(mse, mz=metadata(mse)$design$featureData$mz)
```

---

findNeighbors *Find spatial neighbors*

Description

Find the indices of spatial neighbors for all observations in a dataset.

Usage

```
## S4 method for signature 'ANY'
findNeighbors(x, r = 1, groups = NULL,
    metric = "maximum", p = 2, matrix = FALSE, ...)

## S4 method for signature 'SpectralImagingData'
findNeighbors(x, r = 1, groups = run(x), ...)

## S4 method for signature 'PositionDataFrame'
findNeighbors(x, r = 1, groups = run(x), ...)
```

Arguments

x	An imaging dataset or data frame with spatial dimensions.
r	The spatial maximum distance for an observation to be considered a neighbor.
groups	A vector coercible to a factor giving which observations should be treated as spatially-independent. Observations in the same group are assumed to have a spatial relationship.
metric	Distance metric to use when finding the nearest neighbors. Supported metrics include "euclidean", "maximum", "manhattan", and "minkowski".
p	The power for the Minkowski distance.
matrix	Should the neighbors be returned as a sparse adjacency matrix instead of a list?
...	Additional arguments passed to the next call.

Value

Either a list of indices of neighbors or a sparse adjacency matrix (sparse_mat).

Author(s)

Kylie A. Bemis

See Also

spatialWeights

Examples

```
pdata <- PositionDataFrame(coord=expand.grid(x=1:8, y=1:8))

# find spatial neighbors
findNeighbors(pdata, r=1)
```

MassDataFrame-class *MassDataFrame: Extended data frame with key columns*

Description

A data frame for mass spectrometry feature metadata with a required column for m/z values.

Usage

```
MassDataFrame(mz, ..., row.names = FALSE)
```

Arguments

mz	A sorted vector of m/z values.
...	Arguments passed to the DataFrame().
row.names	Either a vector of row names or a logical value indicating whether row names should be generated automatically (from the m/z values).

Methods

mz(object, ...), mz(object, ...) <- value: Get or set the m/z values.

Author(s)

Kylie A. Bemis

See Also

XDataFrame, PositionDataFrame

Examples

```
## Create an MassDataFrame object
MassDataFrame(mz=sort(500 * runif(10)), label=LETTERS[1:10])
```

MeansTest *Linear model-based testing for summarized imaging experiments*

Description

Performs hypothesis testing for imaging experiments by fitting linear mixed models to summarizations or segmentations.

Usage

```
## S4 method for signature 'ANY'
meansTest(x, data, fixed, random, samples,
    response = "y", reduced = ~ 1, byrow = FALSE,
    use_lmer = FALSE,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
meansTest(x, fixed, random, samples = run(x),
    response = "intensity", ...)

## S4 method for signature 'SpatialDGMM'
meansTest(x, fixed, random, class = 1L,
    response = "intensity", reduced = ~ 1,
    use_lmer = FALSE,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

segmentationTest(x, fixed, random, samples = run(x),
    class = 1L, response = "intensity", reduced = ~ 1, ...)

## S4 method for signature 'MeansTest'
topFeatures(object, n = Inf, sort.by = "statistic", ...)

## S4 method for signature 'MeansTest,missing'
plot(x, i = 1L, type = "boxplot", show.obs = TRUE,
        fill = FALSE, layout = NULL, ...)

contrastTest(object, specs, method = "pairwise", emm_adjust = "none",
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

x	A dataset in P x N matrix format or a set of spatially segmented images.
data	A data frame of additional variables parallel to x.

fixed	A one-sided formula giving the fixed effects of the model on the RHS. The response will added to the LHS, and the formula will be passed to the underlying modeling function.
random	A one-sided formula giving the random effects of the model on the RHS. See lme for the allowed specifications.
samples	A vector coercible to a factor giving the observational unit (i.e., the samples and replicates).
class	For SpatialDGMM, the class (segment) from the Gaussian mixture models that should be used for the comparison. By default, compare the classes (segments) with the highest means in each sample.
response	The name of to assign the response variable in the fitted models.
reduced	A one-sided formula specifying the fixed effects in the reduced model for the null hypothesis. The default is an intercept-only model. Random effects are retained.
use_lmer	Logical. If TRUE, use lmerTest::lmer instead of nlme::lme for fitting mixed effects models. This provides Satterthwaite degrees of freedom and is typically faster. When TRUE, likelihood ratio tests are not performed; use contrastTest() for post-hoc tests. Requires the lmerTest package.
byrow	For the default method, are the rows or the columns the x .
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	For meansTest, passed to internal linear modeling methods (lm, lme, or lmerTest::lmer). For contrastTest, passed to emmeans::emmeans() or emmeans::contrast().
specs	For contrastTest, specification for estimated marginal means passed to emmeans::emmeans(). Can be a character vector of factor names (e.g., "condition"), a formula (e.g., ~ condition), or more complex specifications. See ?emmeans::emmeans for details.
method	For contrastTest, the contrast method. Common options include "pairwise" (all pairwise comparisons), "trt.vs.ctrl" (treatment vs control), "poly" (polynomial contrasts), or a named list of custom contrasts. See ?emmeans::contrast for all options.
emm_adjust	For contrastTest, the p-value adjustment method passed to emmeans::contrast(). Options include "none", "bonferroni", "tukey", "fdr", etc. See ?emmeans::summary.emmGrid for all options. Note: adjustment only affects results when testing multiple contrasts (e.g., 3-level factors produce 3 pairwise contrasts).
object	A fitted model object to summarize.
n, sort.by	For topFeatures, the number of top features to return and how to sort them.
i	The index of the model(s)/feature(s) to plot.
type	The type of plot to display.
show.obs	Should individual observations (i.e., the summarized mean for each sample) be plotted too?
fill	Should the boxplots be filled?
layout	A vector of the form c(nrow, ncol) specifying the number of rows and columns in the facet grid.

Details

Likelihood ratio tests are used for hypothesis testing when `use_lmer = FALSE` (the default). When `use_lmer = TRUE`, models are fit with `lmerTest::lmer` using REML, but no hypothesis tests are performed. Instead, use the `contrastTest()` function for post-hoc comparisons.

The `contrastTest()` function provides flexible post-hoc testing for models fit with `use_lmer = TRUE`. It uses `emmeans::emmeans()` to compute estimated marginal means and `emmeans::contrast()` to perform comparisons. The function operates on all m/z features in parallel and returns a `ContrastTest` object with contrast statistics.

When using `contrastTest()`, p-value adjustments (via the `adjust` parameter) only show differences from unadjusted values when testing multiple contrasts. For example, a 2-level factor produces only 1 contrast, so Bonferroni adjustment yields $p * 1 = p$ (no change). A 3-level factor produces 3 pairwise contrasts, demonstrating visible adjustment effects.

Value

For `meansTest`: An object of class `MeansTest` derived from `ResultsList`, where each element contains a linear model (`lm`, `lme`, or `lmerMod` object).

For `contrastTest`: A `ContrastTest` object (extends `ResultsList`) where each element contains an `emmeans` contrast object. The `mcols` metadata includes contrast estimates and p-values with columns named as `"[contrast_name].estimate"` and `"[contrast_name].pvalue"`.

Author(s)

Dan Guo, Kylie A. Bemis, and Ethan Rogers

See Also

`lm`, `lme`, `lmerTest::lmer`, `emmeans::emmeans`, `emmeans::contrast`, `spatialDGMM`

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- simulateImage(preset=4, nrun=3, npeaks=10,
    dim=c(10,10), peakheight=5, peakdiff=2,
    centroided=TRUE)

samples <- replace(run(x), !(x$circleA | x$circleB), NA)

# Fit with nlme::lme (performs likelihood ratio tests)
fit <- meansTest(x, ~condition, samples=samples)
print(fit)

# Fit with lmerTest::lmer (no tests, use contrastTest() instead)
## Not run:
fit_lmer <- meansTest(x, ~condition, samples=samples, use_lmer=TRUE)

# Post-hoc contrasts with p-value adjustment
contr <- contrastTest(fit_lmer, specs="condition",
                      method="pairwise", emm_adjust="bonferroni")
```

```
print(contr[[1]])  # First m/z feature
mcols(contr)  # All contrast statistics

## End(Not run)
```

MSImagingArrays-class *MSImagingArrays: MS imaging data with arbitrary m/z values*

Description

The MSImagingArrays class provides a list-like container for high-throughput mass spectrometry imaging data where every mass spectrum may have its own m/z values. It is designed for easy access to raw mass spectra for the purposes of pre-processing.

It can be converted to a MSImagingExperiment object for easier image slicing and for applying statistical models and machine learning methods.

Usage

```
## Instance creation
MSImagingArrays(spectraData = SimpleList(),
    pixelData = PositionDataFrame(), experimentData = NULL,
    centroided = NA, continuous = NA, metadata = list())

## Additional methods documented below
```

Arguments

spectraData	Either a list-like object with lists of individual spectra and lists of their domain values, or a SpectraArrays instance.
pixelData	A PositionDataFrame with pixel metadata, with a row for each spectrum.
experimentData	Either NULL or a ImzMeta object with MS-specific experiment-level metadata.
centroided	A logical value indicated whether the spectra have been centroided.
continuous	A logical value indicated whether the spectra all have the same m/z values.
metadata	A list of arbitrary metadata.

Slots

spectraData**:** A SpectraArrays object storing one or more array-like data elements with conformable dimensions.

elementMetadata**:** A PositionDataFrame containing spectrum-level metadata, including each spectrum's pixel coordinates and experimental run information.

processing**:** A list containing unexecuted ProcessingStep objects.

experimentData**:** Either NULL or an ImzMeta object containing experiment-level metadata (necessary for writing the data to imzML).

centroided**:** A logical value indicated whether the spectra have been centroided (if known).

continuous**:** A logical value indicated whether the spectra all have the same m/z values (if known).

Methods

All methods for SpectralImagingData and SpectralImagingArrays also work on MSImagingArrays objects. Additional methods are documented below:

`mz(object, i = NULL, ...)`, `mz(object, i = NULL, ...) <- value`: Get or set the m/z arrays in the spectraData slot.

`intensity(object, i = NULL, ...)`, `intensity(object, i = NULL, ...) <- value`: Get or set the intensity arrays in the spectraData slot.

`centroided(object, ...)`, `centroided(object, ...) <- value`: Get or set the centroided slot.

`isCentroided(object, ...)`: Equivalent to `isTRUE(centroided(object))`.

`experimentData(object)`, `experimentData(object) <- value`: Get or set the experimentData slot.

Author(s)

Kylie A. Bemis

See Also

SpectralImagingArrays, MSImagingExperiment

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- replicate(9, rlnorm(10), simplify=FALSE)
mz <- replicate(9, 500 * sort(runif(10)), simplify=FALSE)
coord <- expand.grid(x=1:3, y=1:3)

msa <- MSImagingArrays(
    spectraData=list(intensity=x, mz=mz),
    pixelData=PositionDataFrame(coord))

print(msa)
```

MSImagingExperiment-class

MSImagingExperiment: MS imaging data with shared m/z values

Description

The MSImagingExperiment class provides a matrix-like container for high-throughput mass spectrometry imaging data where every mass spectrum shares the same m/z values. It is designed to provide easy access to both the spectra (as columns) and sliced images (as rows).

It can be converted from a MSImagingArrays object which is designed for representing raw mass spectra.

Usage

```
## Instance creation
MSImagingExperiment(spectraData = SimpleList(),
    featureData = MassDataFrame(), pixelData = PositionDataFrame(),
    experimentData = NULL, centroided = NA, metadata = list())

## Additional methods documented below
```

Arguments

spectraData	Either a matrix-like object with number of rows equal to the number of features and number of columns equal to the number of pixels, a list of such objects, or a SpectraArrays instance.
featureData	A MassDataFrame with feature metadata, with a row for each feature.
pixelData	A PositionDataFrame with pixel metadata, with a row for each spectrum.
experimentData	Either NULL or a ImzMeta object with MS-specific experiment-level metadata.
centroided	A logical value indicated whether the spectra have been centroided.
metadata	A list of arbitrary metadata.

Slots

spectraData: A SpectraArrays object storing one or more array-like data elements with conformable dimensions.

featureData: A MassDataFrame containing feature-level metadata.

elementMetadata: A PositionDataFrame containing spectrum-level metadata, including each spectrum's pixel coordinates and experimental run information.

processing: A list containing unexecuted ProcessingStep objects.

experimentData: Either NULL or an ImzMeta object containing experiment-level metadata (necessary for writing the data to imzML).

centroided: A logical value indicated whether the spectra have been centroided (if known).

Methods

All methods for SpectralImagingData and SpectralImagingExperiment also work on MSImagingExperiment objects. Additional methods are documented below:

mz(object, ...), mz(object, ...) <- value: Get or set the m/z column in the featureData slot.

intensity(object, ...), intensity(object, ...) <- value: Get or set the intensity matrix in the spectraData slot.

centroided(object, ...), centroided(object, ...) <- value: Get or set the centroided slot.

isCentroided(object, ...): Equivalent to isTRUE(centroided(object)).

experimentData(object), experimentData(object) <- value: Get or set the experimentData slot.

Author(s)

Kylie A. Bemis

See Also

SpectralImagingExperiment, MSImagingArrays

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- matrix(rlnorm(81), nrow=9, ncol=9)
mz <- sort(runif(9))
coord <- expand.grid(x=1:3, y=1:3)

mse <- MSImagingExperiment(
    spectraData=x,
    featureData=MassDataFrame(mz=mz),
    pixelData=PositionDataFrame(coord))

print(mse)
```

---

normalize *Normalize spectra*

Description

Apply deferred normalization to spectra.

Usage

```
## S4 method for signature 'MSImagingExperiment_OR_Arrays'
normalize(object,
    method = c("tic", "rms", "reference"),
    scale = NA, ref = NULL, ...)

## S4 method for signature 'SpectralImagingData'
normalize(object,
    method = c("tic", "rms", "reference"), ...)
```

Arguments

object	A spectral imaging dataset.
method	The normalization method to use. See rescale for details.
scale	The scaling value to use for the normalized spectra.
ref	The reference peaks to use for normalization.
...	Additional arguments passed to the normalization function.

Details

The supported normalization methods are:

- "tic": Total ion current normalization using `rescale_sum`.
- "rms": Root-mean-squared normalization using `rescale_rms`.
- "reference": Normalization according to a reference feature using `rescale_ref`.

Value

An object of the same class with the processing step queued.

Note

The normalization is deferred until `process()` is called.

Author(s)

Kylie A. Bemis

See Also

`smooth`, `recalibrate`, `reduceBaseline`, `peakPick`, `process`

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3))

# queue normalization
mse2 <- normalize(mse, method="tic")

# apply normalization
mse2 <- process(mse2)
```

---

peakAlign *Align peaks across spectra*

Description

Align peaks across spectra in a spectral imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment'
peakAlign(object, ref,
    spectra = "intensity", index = "mz",
    binfun = "min", binratio = 2,
    tolerance = NA, units = c("ppm", "mz"), ...)

## S4 method for signature 'MSImagingArrays'
peakAlign(object, ref,
    spectra = "intensity", index = "mz",
    binfun = "min", binratio = 2,
    tolerance = NA, units = c("ppm", "mz"), ...)

## S4 method for signature 'SpectralImagingExperiment'
peakAlign(object, ref,
    spectra = "intensity", index = NULL,
    binfun = "min", binratio = 2,
    tolerance = NA, units = c("relative", "absolute"),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingArrays'
peakAlign(object, ref,
    spectra = "intensity", index = NULL,
    binfun = "min", binratio = 2,
    tolerance = NA, units = c("relative", "absolute"),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

`object`	A spectral imaging dataset.
`ref`	The locations of reference peaks to use for the alignment.
`spectra`	The name of the array in `spectraData()` to use for the peak intensities.
`index`	The name of the array in `spectraData()` (for `SpectralImagingArrays`) or column in `featureData()` (for `SpectralImagingExperiment`) to use for the peak locations.
`binfun`	The function used to summarize the minimum distance between same-spectrum peaks across all spectra. This is passed to `estimateDomain` as `width` (see "Details").
`binratio`	The ratio between the alignment tolerance and the peak bin widths. If `tolerance` is `NA`, then this is also used to estimate the tolerance from the shared domain (see "Details").
`tolerance`	The alignment tolerance for matching a detected peak to a reference. If `NA`, then the tolerance is automatically determined from `binratio` times the minimum distance between same-spectrum peaks (see "Details").

units	The units for the above tolerance.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Options passed to process().

Details

Before peak alignment, process() is called to apply any queued pre-processing steps. It is assumed that peakPick() has either been queued or already applied to the data.

If ref is provided, then the aligned peaks are returned immediately without additional processing. (Peaks are binned on-the-fly to the reference peak locations.)

If ref is not provided, then the shared peaks must be determined automatically. This starts with creation of a shared domain giving a list of possible peak locations. The shared domain is estimated by estimateDomain().

Next, binpeaks is used to bin the observed peaks to the shared domain. Then, mergepeaks is used to merge peaks that are separated by a distance less than the given tolerance.

The averaged locations of the merged peaks in each bin are used as the shared peaks for the full dataset, and the aligned peaks are returned. (Peaks are binned on-the-fly to the shared peak locations.)

Value

A new object derived from SpectralImagingExperiment with the aligned peaks.

Author(s)

Kylie A. Bemis

See Also

process peakPick, peakProcess

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3))

# queue peak picking
mse2 <- peakPick(mse, method="diff", SNR=6)

# align peaks
mse2 <- peakAlign(mse2)
plot(mse2, i=4)
```

peakPick *Peak pick spectra*

Description

Apply deferred peak picking to spectra.

Usage

```
## S4 method for signature 'MSImagingExperiment'
peakPick(object, ref,
    method = c("diff", "sd", "mad", "quantile", "filter", "cwt"),
    SNR = 2, type = c("height", "area"),
    tolerance = NA, units = c("ppm", "mz"), ...)

## S4 method for signature 'MSImagingArrays'
peakPick(object, ref,
    method = c("diff", "sd", "mad", "quantile", "filter", "cwt"),
    SNR = 2, type = c("height", "area"),
    tolerance = NA, units = c("ppm", "mz"), ...)

## S4 method for signature 'SpectralImagingData'
peakPick(object, ref,
    method = c("diff", "sd", "mad", "quantile", "filter", "cwt"),
    SNR = 2, type = c("height", "area"),
    tolerance = NA, units = c("relative", "absolute"), ...)
```

Arguments

object	A spectral imaging dataset.
ref	Optional vector giving locations of reference peaks to extract from the dataset.
method	The peak picking method to use. See findpeaks for details.
SNR	The signal-to-noise threshold to use to determine a peak.
type	The type of value to use to summarize the peak.
tolerance	If ref is specified, the tolerance to use when deciding if a local peak in a spectrum matches a reference peak. If NA, then the tolerance is automatically determined as half the minimum distance between peaks in the reference.
units	The units for the above tolerance.
...	Additional arguments passed to the peak picking function.

Details

Unless otherwise specified, peaks are detected as local maxima which are then compared to the estimated noise level to determine a signal-to-noise ratio for each peak. Most of the peak detection methods below are differentiated by how they estimate the noise in the specturm.

The supported peak picking methods are:

- "diff": Estimate noise based on the derivative of the signal using estnoise_diff.
- "sd": Estimate noise from standard deviation using estnoise_sd.
- "mad": Estimate noise from mean absolute deviation using estnoise_mad.
- "quantile": Estimate noise from a rolling quantile of the difference between the raw signal and a smoothed signal using estnoise_quant.
- "filter": Estimate noise using dynamic filtering of the local peaks using estnoise_filt.
- "cwt": Detect peaks based on continuous wavelet transform (CWT) using findpeaks_cwt.

If ref is provided, then the signal-to-noise ratio is not determined, and any detected local maxima are summarized as long as they match to a reference peak.

Value

An object of the same class with the processing step queued.

Note

The peak picking is deferred until process() is called.

Author(s)

Kylie A. Bemis

See Also

process, peakAlign, peakProcess, estimateReferencePeaks

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3))

# queue peak picking
mse2 <- peakPick(mse, method="diff", SNR=6)
plot(mse2, i=4)

# apply peak picking
mse2 <- process(mse2)
```

peakProcess *Process peaks in mass spectra*

Description

Apply peak picking and alignment to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment_OR_Arrays'
peakProcess(object, ref,
    spectra = "intensity", index = "mz",
    method = c("diff", "sd", "mad", "quantile", "filter", "cwt"),
    SNR = 2, type = c("height", "area"),
    tolerance = NA, units = c("ppm", "mz"),
    sampleSize = NA, filterFreq = TRUE, outfile = NULL,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

object	A spectral imaging dataset.
ref	The locations of reference peaks to use for the alignment.
spectra	The name of the array in spectraData() to use for the peak intensities.
index	The name of the array in spectraData() (for MSImagingArrays) or column in featureData() (for MSImagingExperiment) to use for the peak locations.
method	The peak picking method to use. See findpeaks for details.
SNR	The signal-to-noise threshold to use to determine a peak.
type	The type of value to use to summarize the peak.
tolerance	The tolerance for matching a detected peak to the reference peaks or the shared m/z values. Passed to peakPick and peakAlign.
units	The units for the above tolerance.
sampleSize	The count or proportion giving a subset of spectra to use to determine reference peaks.
filterFreq	Either a logical value indicating whether singleton peaks should be removed, or a count or frequency used as a threshold to filter the peaks.
outfile	Optional. The name of a file to write the resulting dataset as imzML.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Options passed to process().

Details

This method provides a combined interface for peakPick and peakAlign for the most common approaches to peak processing.

If peakPick() has been queued already, then it will be applied. Otherwise, it will be called internally with the provided arguments.

There are two main paths depending on whether (1) peaks should be extracted based on a reference or (2) peak picking should be performed on the full dataset and then aligned.

If either ref is provided or sampleSize is finite, then (1) is chosen and peaks are extracted based on the reference. If the reference is not provided, then peak picking and alignment performed on a subset of spectra (according to sampleSize) to create the reference peaks. The reference peaks are then used to extract peaks from the full dataset.

Otherwise, (2) is chosen and peaks are picked and aligned across all spectra.

The advantage of (1) is that all reference peaks will be summarized even they would not have a high enough signal-to-noise ratio to be detected in some spectra. The disadvantage is that rare peaks that do not appear in the sampled subset of spectra will not be included in the process peaks.

The advantage of (2) is that rare peaks will be included because peak detection is performed on all spectra. The disadvantage is that some peaks may be missing from some spectra despite having nonzero intensities, because they did not have a high enough signal-to-noise ratio to be detected as peaks.

Setting sampleSize to 1 will balance these advantages and disadvantages because the reference will be based on all spectra. However, this means the full dataset must be processed at least twice (possibly more if intermediate calculations are necessary), so it will be more time-consuming.

Value

A new object derived from MSImagingExperiment with the processed peaks.

Author(s)

Kylie A. Bemis

See Also

process peakPick, peakAlign

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3))

# process peaks
mse2 <- peakProcess(mse, method="diff", SNR=3)
plot(mse2, i=4)
```

pixels *Find pixel indices*

Description

Search for the column indices of a spectral imaging dataset that correspond to specific pixels, based on a set of conditions.

Usage

```
## S4 method for signature 'SpectralImagingExperiment'
pixels(object, ..., coord, run, tolerance = NA,
    env = NULL)

## S4 method for signature 'SpectralImagingArrays'
pixels(object, ..., coord, run, tolerance = NA,
    env = NULL)

## S4 method for signature 'SpectralImagingData'
pixels(object, ..., env = NULL)
```

Arguments

object	A spectral imaging dataset.
...	Expressions that evaluate to logical vectors in the environment of pixelData().
coord	The coordinates of the pixels to include.
run	The run of the pixels to include.
tolerance	The tolerance for matching pixels to coordinates.
env	The enclosing environment for evaluating ....

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

pixels(mse, x > 6, y > 6)
pixels(mse, coord=expand.grid(x=1:3, y=1:3))
```

plot-image *Plot images from a spectral imaging dataset*

Description

Create and display sliced images from the spectra or pixel data of a spectral imaging dataset using a formula interface.

Usage

```
## S4 method for signature 'MSImagingExperiment'
image(x,
    formula = intensity ~ x * y,
    i = features(x, mz=mz),
    mz = NULL,
    tolerance = NA,
    units = c("ppm", "mz"),
    ...,
    xlab, ylab)

## S4 method for signature 'SpectralImagingExperiment'
image(x,
    formula,
    i = 1L,
    run = NULL,
    groups = NULL,
    superpose = FALSE,
    key = TRUE,
    ...,
    enhance = NULL,
    smooth = NULL,
    scale = NULL,
    subset = TRUE)

## S4 method for signature 'PositionDataFrame'
image(x,
    formula,
    run = NULL,
    superpose = FALSE,
    key = TRUE,
    ...,
    enhance = NULL,
    smooth = NULL,
    scale = NULL,
    subset = TRUE)
```

Arguments

x	A spectral imaging dataset.
formula	A formula of the form `vals ~ x * y` giving the image values and the pixel coordinates. The LHS is taken to be the name of an array in `spectraData()` and the RHS is taken to be columns of `pixelData()`. Alternatively, if `formula` is a string or if `i` is NULL, then the LHS is interpreted as the name of a column of `pixelData()` as well.
i	The index of the feature(s) to plot for the image(s).
mz	The m/z value(s) to plot for the image(s).
tolerance	If specified, the tolerance to consider a feature as being equal to the given `mz` value.
units	The units for the above tolerance.
...	Additional arguments passed to `plot_image`.
xlab, ylab	Plotting labels.
run	The names of experimental runs to include, or the index of the levels of the runs to include.
groups	A vector coercible to a factor indicating which of the specified features should be plotted with the same color.
superpose	If multiple images are plotted, should they be superposed on top of each other, or plotted seperately?
key	Should a legend or colorkey be plotted?
enhance	The name of a contrast enhancement method, such as `"hist"` or `"adapt"` for `enhance_hist()` and `enhance_adapt()`, etc. See `enhance` for details.
smooth	The name of a smoothing method, such as `"gauss"` or `"bi"` for `filt2_gauss()` and `filt2_bi()`, etc. See `filt2` for details.
scale	If multiple images are plotted, should they be scaled to the same intensity scale?
subset	A logical vector indicating which pixels to include in the image.

Author(s)

Kylie A. Bemis

See Also

`image`, `plot_image`, `selectROI`

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- simulateImage(preset=2, npeaks=10, dim=c(16,16))
peaks <- mz(metadata(x)$design$featureData)

image(x, mz=peaks[1L], tolerance=0.5, units="mz")
image(x, mz=peaks[1L], smooth="gaussian")
```

```
image(x, mz=peaks[1:9], smooth="adaptive")

x <- summarizePixels(x, stat=c(TIC="mean"))
image(x, "TIC")
```

---

plot-spectra *Plot spectra from a spectral imaging dataset*

Description

Create and display plots from the spectra or feature data of a spectral imaging dataset using a formula interface.

Usage

```
## S4 method for signature 'MSImagingExperiment,missing'
plot(x,
    formula = intensity ~ mz,
    i = pixels(x, coord=coord, run=run),
    coord = NULL,
    run = NULL,
    ...,
    xlab, ylab,
    isPeaks = isCentroided(x))

## S4 method for signature 'MSImagingArrays,missing'
plot(x,
    formula = intensity ~ mz,
    i = pixels(x, coord=coord, run=run),
    coord = NULL,
    run = NULL,
    ...,
    xlab, ylab,
    isPeaks = isCentroided(x))

## S4 method for signature 'SpectralImagingExperiment,missing'
plot(x,
    formula,
    i = 1L,
    groups = NULL,
    superpose = FALSE,
    key = TRUE,
    ...,
    n = Inf,
    downsampler = "lttb",
    isPeaks = FALSE,
    annPeaks = 0)
```

```
## S4 method for signature 'SpectralImagingArrays,missing'
plot(x,
    formula,
    i = 1L,
    groups = NULL,
    superpose = FALSE,
    key = TRUE,
    ...,
    n = Inf,
    downsampler = "lttb",
    isPeaks = FALSE,
    annPeaks = 0)

## S4 method for signature 'XDataFrame,missing'
plot(x,
    formula,
    superpose = FALSE,
    key = TRUE,
    ...,
    n = Inf,
    downsampler = "lttb",
    isPeaks = FALSE,
    annPeaks = 0)
```

Arguments

x	A spectral imaging dataset.
formula	A formula of the form `vals ~ t` giving the spectra values and their domain locations. The LHS is taken to be the name of an array in `spectraData()` and the RHS is either an array in `spectraData()` for `SpectralImagingArrays`-derived classes or a column of `featureData()` for `SpectralImagingExperiment`-derived classes. Alternatively, if `formula` is a string or if `i` is NULL, then the LHS is interpreted as the name of a column of `featureData()` for `SpectralImagingExperiment` as well.
i	The index of the spectrum to plot.
coord	The coordinates of the spectrum to plot.
run	The run of the spectrum to plot.
...	Additional arguments passed to `plot_signal`.
xlab, ylab	Plotting labels.
isPeaks	Should the spectrum be plotted as peaks or as a continuous signal?
annPeaks	If `isPeaks` is `TRUE`, either an integer giving the number of peaks to annotate (i.e., label with their location), or a plotting symbol (e.g., "circle", "cross", etc.) to indicate the peak locations.
groups	A vector coercible to a factor indicating which of the specified spectra should be plotted with the same color.

superpose	If multiple spectra are plotted, should they be superposed on top of each other, or plotted seperately?
key	Should a legend or colorkey be plotted?
n, downsampler	A spectrum can contain far more data points than are needed to visualize it, potentially making the plotting unnecessarily slow. Downsampling can be performed to improve plotting speed while maintaining the visual integrity of the spectrum. See downsample for details.

Author(s)

Kylie A. Bemis

See Also

plot, plot_signal

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- simulateImage(preset=1, npeaks=10, dim=c(3,3))

plot(x, i=4)
plot(x, coord=c(x=1, y=2))
plot(x, log2(intensity + 1) ~ mz, i=4,
    xlab=expression(italic("m/z")),
    ylab=expression(italic("Log Intensity")))
```

PositionDataFrame-class

PositionDataFrame: Extended data frame with key columns

Description

A data frame for metadata with spatial coordinates and multiple experimental runs.

Usage

```
PositionDataFrame(coord, run, ..., row.names = FALSE)
```

Arguments

coord	A data frame or matrix of coordinates.
run	A factor giving the experimental runs.
...	Arguments passed to the DataFrame().
row.names	Either a vector of row names or a logical value indicating whether row names should be generated automatically (from the m/z values).

Methods

`coord(object)`, `coord(object) <- value`: Get or set the coordinate columns.

`coordNames(object)`, `coordNames(object) <- value`: Get or set the names of the coordinate columns.

`run(object)`, `run(object) <- value`: Get or set the experimental run column.

`runNames(object)`, `runNames(object) <- value`: Get or set the experimental run levels.

`nrun(object)`: Get the number of experimental runs.

`is3D(object)`: Check if the number of spatial dimensions is greater than 2.

Author(s)

Kylie A. Bemis

See Also

XDataFrame, MassDataFrame

Examples

```
## Create an PositionDataFrame object
coord <- expand.grid(x=1:3, y=1:3)
PositionDataFrame(coord=coord, label=LETTERS[1:9])
```

---

process *Apply queued processing to spectra*

Description

Queue pre-processing steps on an imaging dataset and apply them, possibly writing out the processed data to a file.

Usage

```
## S4 method for signature 'MSImagingExperiment'
process(object, spectra = "intensity", index = "mz",
    domain = NULL, outfile = NULL, ...)

## S4 method for signature 'MSImagingArrays'
process(object, spectra = "intensity", index = "mz",
    domain = NULL, outfile = NULL, ...)

## S4 method for signature 'SpectralImagingExperiment'
process(object, spectra = "intensity", index = NULL,
    domain = NULL, outfile = NULL,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)
```

```
## S4 method for signature 'SpectralImagingArrays'
process(object, spectra = "intensity", index = NULL,
    domain = NULL, outfile = NULL,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingData'
addProcessing(object, FUN, label, metadata = list(),
    verbose = getCardinalVerbose(), ...)

reset(object, ...)
```

Arguments

object	A spectral imaging dataset.
spectra	The name of the array in spectraData() to use for the peak intensities.
index	The name of the array in spectraData() (for MSImagingArrays) or column in featureData() (for MSImagingExperiment) to use for the peak locations.
domain	Optional. The name of the array in spectraData() (for MSImagingArrays) or column in featureData() (for MSImagingExperiment) to use for output domain (if known).
outfile	Optional. The name of a file to write the resulting dataset. Creates an imzML file for MSImagingExperiment or MSImagingArrays. The "continuous" format will be written if domain is specified; otherwise the "processed" format will be used in most cases.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	For process, options passed to chunk_mapply or chunk_colapply. For addProcessing, arguments to FUN.
FUN	A user-specified processing function.
label	The name of the processing step.
metadata	A list of processing metadata to be added to the object's metadata after processing has been applied. Concatenated with any arguments passed to FUN via dots.

Details

This method allows queueing of delayed processing to an imaging dataset. All of the queued processing steps will be applied in sequence whenever process() is called next. Use reset() to remove all queued processing steps.

Typically, processing steps are queued using methods like normalize, smooth, peakPick, etc.

However, a processing step can be queued manually with addProcessing.

In this case, the user-specified function *must* accept (1) a first argument giving the spectral intensities as a numeric vector and (2) a second argument giving the intensity locations (e.g., m/z values) as a numeric vector.

The value returned by a user-specified function must return either (1) a numeric vector of the same length as the input intensities or (2) a 2-column matrix where the first column is the new locations (e.g., m/z values of peaks) and the second column is the new intensities.

Value

An object of the same class as the original object, with all processing steps applied.

Author(s)

Kylie A. Bemis

See Also

normalize, smooth, recalibrate, reduceBaseline, peakPick

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, dim=c(3,3), baseline=1)

mse2 <- smooth(mse, width=11)
mse2 <- reduceBaseline(mse2)
plot(mse2, i=4)

mse2 <- process(mse2)
```

---

readMSIData *Read mass spectrometry imaging data files*

Description

Read supported mass spectrometry imaging data files, including imzML and Analyze 7.5.

Usage

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

## Read imzML file
readImzML(file, memory = FALSE, check = FALSE,
 mass.range = NULL, resolution = NA, units = c("ppm", "mz"),
 guess.max = 1000L, as = "auto", parse.only=FALSE,
 verbose = getCardinalVerbose(), chunkopts = list(),
 BPPARAM = getCardinalBPPARAM(), ...)
```

```
## Read Analyze 7.5 file
readAnalyze(file, memory = FALSE, as = "auto",
 verbose = getCardinalVerbose(), chunkopts = list(),
 BPPARAM = getCardinalBPPARAM(), ...)

## Convert from MSImagingExperiment to MSImagingArrays
convertMSImagingExperiment2Arrays(object)

## Convert from MSImagingArrays to MSImagingExperiment
convertMSImagingArrays2Experiment(object, mz = NULL,
 mass.range = NULL, resolution = NA, units = c("ppm", "mz"),
 guess.max = 1000L, tolerance = 0.5 * resolution,
 verbose = getCardinalVerbose(), chunkopts = list(),
 BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

file	The absolute or relative file path. The file extension must be included for readMSIData.
memory	Should the spectra be loaded into memory? If TRUE, the spectra are loaded into in-memory R objects. If FALSE, the spectra are attached as file-backed matter objects. If memory="shared", the spectra are attached as shared memory-backed matter objects.
check	Should the UUID and checksum of the binary data file be checked against the corresponding imzML tags?
mass.range	The mass range to use when converting the data to an MSImagingExperiment.
resolution	The mass resolution to use when converting the data to an MSImagingExperiment. This is the *inverse* of the instrument resolution, if known. It is the width of the m/z bins when converting the data to an MSImagingExperiment.
tolerance	If the spectra have been centroided but the peaks are unaligned, then this is passed to peakAlign.
units	The units for the above resolution.
guess.max	The number of spectra to use when guessing the mass range and resolution, if they are not provided.
as	After reading in the data, what class of object should be returned? The data is initially loaded as an MSImagingArrays object. It may be converted to an MSImagingExperiment object. Setting to "auto" means to determine whichever is more appropriate depending on whether the spectra appear to have been processed and centroided.
parse.only	If TRUE, return only the parsed imzML metadata without creating a new MSImagingArrays or MSImagingExperiment object.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to parseImzML or parseAnalyze.

object	A mass spectrometry imaging dataset to convert from one class to another.
mz	A vector of shared m/z values for converting to MSImagingExperiment, if not to be determined automatically.

Details

The spectra are initially loaded into a MSImagingArrays object before conversion to MSImagingExperiment (if applicable).

This conversion can be sped up by specifying the mass.range and resolution so they do not have to be determined from the spectra directly. Using a larger value of guess.max can improve the accuracy of the m/z binning for downstream analysis at the expense of a longer conversion time.

If greater control is desired, spectra should be imported as MSImagingArrays, and processing to MSImagingExperiment can be performed manually.

If problems are encountered while trying to import imzML files, the files should be verified and fixed with imzMLValidator.

A Java version of imzML validator can be found at: https://gitlab.com/imzML/imzMLValidator.

A web-based version of imzML validator can be found at: https://imzml.github.io.

Value

A MSImagingExperiment or MSImagingArrays object.

Author(s)

Kylie A. 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

parseImzML, parseAnalyze writeMSIData

---

recalibrate *Recalibrate spectra*

Description

Apply deferred recalibration to spectra.

Usage

```
## S4 method for signature 'MSImagingExperiment_OR_Arrays'
recalibrate(object, ref,
    method = c("locmax", "dtw", "cow"),
    tolerance = NA, units = c("ppm", "mz"), ...)

## S4 method for signature 'SpectralImagingData'
recalibrate(object, ref,
    method = c("locmax", "dtw", "cow"),
    tolerance = NA, units = c("relative", "absolute"), ...)
```

Arguments

object	A spectral imaging dataset.
ref	The domain (m/z) values or indices of reference peaks to use for the recalibration.
method	The recalibration method to use. See warp1 for details.
tolerance	The tolerance for how much a peak can be shifted in either direction.
units	The units for the above tolerance.
...	Additional arguments passed to the recalibration function.

Details

The supported recalibration methods are:

- "locmax": Align to local maxima using warp1_loc.
- "dtw": Dynamic time warping using warp1_dtw.
- "cow": Correlation optimized warping using warp1_cow.

Value

An object of the same class with the processing step queued.

Note

The recalibration is deferred until process() is called.

Author(s)

Kylie A. Bemis

See Also

normalize, smooth, recalibrate, peakPick, process

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3), sdmz=250)
plot(mse, i=c(2,4,5), superpose=TRUE, xlim=c(1260,1320))

# queue recalibration
peaks <- estimateReferencePeaks(mse)
mse2 <- recalibrate(mse, ref=peaks, method="locmax", tolerance=500)

# apply recalibration
mse2 <- process(mse2)
plot(mse2, i=c(2,4,5), superpose=TRUE, xlim=c(1260,1320))
```

---

reduceBaseline *Reduce baselines in spectra*

Description

Apply deferred baseline reduction to spectra.

Usage

```
## S4 method for signature 'SpectralImagingData'
reduceBaseline(object,
    method = c("locmin", "hull", "snip", "median"), ...)
```

Arguments

object	A spectral imaging dataset.
method	The baseline estimation method to use. See estbase for details.
...	Additional arguments passed to the baseline estimation function.

Details

The supported baseline estimation methods are:

- "locmin": Interpolate from local minima using estbase_loc.
- "hull": Convex hull estimation using estbase_hull.
- "snip": Sensitive nonlinear iterative peak (SNIP) clipping using estbase_snip.
- "median": Running medians using estbase_med.

Value

An object of the same class with the processing step queued.

Note

The baseline reduction is deferred until process() is called.

Author(s)

Kylie A. Bemis

See Also

normalize, smooth, reduceBaseline, peakPick, process

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3), baseline=1)

# queue baseline reduction
mse2 <- reduceBaseline(mse, method="locmin")
plot(mse2, i=4)

# apply baseline reduction
mse2 <- process(mse2)
```

---

reexports *Re-exported objects from Cardinal*

Description

These functions are re-exported from Cardinal for user convenience. Please see the documentation in their original packages.

ResultsList-class *ResultsList: List of modeling results*

Description

The `ResultsList` class provides a container for modeling results with spatial metadata. Specialized subclasses include `MeansTest` for linear model testing, `SegmentationTest` for segmentation-based testing, and `ContrastTest` for post-hoc contrast analysis.

Usage

```
## Instance creation
ResultsList(..., mcols = NULL)

## Additional methods documented below
```

Arguments

`...`	The modeling results.
`mcols`	The metadata columns.

Methods

All methods for SimpleList also work on ResultsList objects. Additional methods are documented below:

fitted(object, ...): Extract fitted values from each modeling results object in the list.

predict(object, ...): Predict on each modeling results object in the list.

topFeatures(object, ...): Rank top features for each modeling results object in the list.

plot(x, i = 1L, ...): Plot the ith modeling results.

image(x, i = 1L, ...): Display images for the ith modeling results.

Author(s)

Kylie A. Bemis

See Also

SpatialResults, meansTest, segmentationTest

---

selectROI *Select regions-of-interest in an image*

Description

Manually select regions-of-interest or pixels on an imaging dataset. The selectROI method uses the built-in locator function. It can be used with an existing image plot, or a new image will be plotted if image arguments are passed via ....

The regions of interest are returned as logical vectors indicating which pixels have been selected. These logical vectors can be combined into factors using the makeFactor function.

Usage

```
## S4 method for signature 'SpectralImagingExperiment'
selectROI(object, ..., mode = c("region", "pixels"))

makeFactor(..., ordered = FALSE)
```

Arguments

object	A spectral imaging dataset.
mode	The mode of selection: "region" to select a region-of-interest as a polygon, or "pixels" to select individual pixels.
...	Additional arguments to be passed to image for selectROI, or name-value pairs of logical vectors to be combined by makeFactor.
ordered	Should the resulting factor be ordered or not?

Value

A `logical` vector of length equal to the number of pixels for `selectROI`.

A factor of the same length as the logical vectors for `makeFactor`.

Author(s)

Kylie A. Bemis

See Also

`image`

`simulateSpectra` *Simulate a mass spectrum or MS imaging experiment*

Description

Simulate mass spectra or complete MS imaging experiments, including a possible baseline, spatial and spectral noise, mass drift, mass resolution, and multiplicative variation, etc.

A number of preset imaging designs are available for quick-and-dirty simulation of images.

These functions are designed for small proof-of-concept examples and testing, and may not scale well to simulating larger datasets.

Usage

```
simulateSpectra(n = 1L, npeaks = 50L,
    mz = rlnorm(npeaks, 7, 0.3), intensity = rlnorm(npeaks, 1, 0.9),
    from = 0.9 * min(mz), to = 1.1 * max(mz), by = 400,
    sdpeaks = sdpeakmult * log1p(intensity), sdpeakmult = 0.2,
    sdnoise = 0.1, sdmz = 10, resolution = 1000, fmax = 0.5,
    baseline = 0, decay = 10, units=c("ppm", "mz"),
    centroided = FALSE, ...)

simulateImage(pixelData, featureData, preset,
    from = 0.9 * min(mz), to = 1.1 * max(mz), by = 400,
    sdrun = 1, sdpixel = 1, spcorr = 0.3, SAR = TRUE,
    resolution = 1000, fmax = 0.5, units=c("ppm", "mz"),
    centroided = FALSE, continuous = TRUE,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

addShape(pixelData, center, size, shape=c("circle", "square"), name=shape)

presetImageDef(preset = 1L, nrun = 1, npeaks = 30L,
    dim = c(20L, 20L), peakheight = exp(1), peakdiff = exp(1),
    sdsample = 0.2, jitter = TRUE, ...)
```

Arguments

n	The number of spectra to simulate.
npeaks	The number of peaks to simulate. Not used if mz and intensity are provided.
mz	The theoretical m/z values of the simulated peaks.
intensity	The mean intensities of the simulated peaks.
from	The minimum m/z value used for the mass range.
to	The maximum m/z value used for the mass range.
by	The step-size used for the observed m/z-values of the profile spectrum.
sdpeaks	The standard deviation(s) for the distributions of observed peak intensities on the log scale.
sdpeakmult	A multiplier used to calculate sdpeaks based on the mean intensities of peaks; used to simulate multiplicative variance. Not used if sdpeaks is provided.
sdnoise	The standard deviation of the random noise in the spectrum on the log scale.
sdmz	The standard deviation of the mass error in the observed m/z values of peaks, in units indicated by units.
resolution	The mass resolution as defined by m / dm, where m is the observed mass and dm is the width of the peak at a proportion of its maximum height defined by fmax (defaults to full-width-at-half-maximum – FWHM – definition). Note that this is NOT the same as the definition of resolution in the readImzML function.
fmax	The fraction of the maximum peak height to use when defining the mass resolution.
baseline	The maximum intensity of the baseline. Note that baseline=0 means there is no baseline.
decay	A constant used to calculate the exponential decay of the baseline. Larger values mean the baseline decays more sharply at the lower mass range of the spectrum.
units	The units for by and sdmz. Either parts-per-million or absolute m/z units.
centroided	Should the simulated spectrum representation be centroided (TRUE) or profile (FALSE)?
continuous	Should the simulated spectrum storage type be continuous (TRUE) or processed (FALSE), where "continuous" means a dense representation and "processed" means a sparse representation?
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
pixelData	A PositionDataFrame giving the pixel design of the experiment. The names of the columns should match the names of columns in featureData. Each column should be a logical vector corresponding to a morphological substructure, indicate which pixels belong to that substructure.
featureData	A MassDataFrame giving the feature design of the experiment. Each row should correspond to an expected peak. The names of the columns should match the names of columns in pixelData. Each column should be a numeric vector corresponding to a morphological substructure, giving the mean intensity of that peak for that substructure.

preset	A number indicating a preset image definition to use.
nrun	The number of runs to simulate for each condition.
sdrun	A standard deviation giving the run-to-run variance.
sdpixel	A standard deviation giving the pixel-to-pixel variance.
spcorr	The spatial autocorrelation. Must be between 0 and 1, where spcorr=0 indicates no spatial autocorrelation.
SAR	Should a spatial autoregressive (SAR) model be used for simulating spatially-correlated noise (TRUE) versus a simpler model that uses spatially-smoothed Gaussian noise (FALSE)? The calculation of the SAR matrix for large images can be very time-consuming, so if the simpler model is adequate, then setting this to FALSE can result in significantly faster simulation.
...	Additional arguments to pass to simulateSpectra or presetImageDef.
dim	The dimensions of the preset image.
peakheight	Reference intensities used for peak heights by the preset.
peakdiff	A reference intensity difference used for the mean peak height difference between conditions, for presets that simulate multiple conditions.
sdsample	A standard deviation giving the amount of variation from the true peak heights for this simulated sample.
jitter	Should random noise be added to the location and size of the shapes?
center	The center of the shape.
size	The size of the shape (from the center).
shape	What type of shape to add.
name	The name of the added column.

Details

The simulateSpectra() and simulateImage() functions are used to simulate mass spectra and MS imaging experiments. They provide a great deal of control over the parameters of the simulation, including all sources of variation.

For simulateImage(), the user should provide the design of the simulated experiment via matching columns in pixelData and featureData, where each column corresponds to different morphological substructures or differing conditions. These design data frames are returned in the metadata() of the returned object for later reference.

A number of presets are defined by presetImageDef(), which returns only the pixelData and featureData necessary to define the experiment for simulateImage(). These can be referenced for help in understanding how to define experiments for simulateImage().

The preset images are:

- 1: a centered circle
- 2: a topleft circle and a bottomright square
- 3: two corner squares and a centered circle
- 4: a centered circle with conditions A and B in different runs

- 5: a topleft circle and a bottomright square with conditions A and B in different runs
- 6: two corner squares and a centered circle; the circle has conditions A and B in different runs
- 7: matched pairs of circles with conditions A and B within the same runs; includes reference peaks
- 8: matched pairs of circles inside squares with conditions A and B within the same runs; includes reference peaks
- 9: a small sphere inside a larger sphere (3D)

The `addShape()` function is provided for convenience when generating the `pixelData` for `simulateImage()`, as a simple way of adding morphological substructures using basic shapes such as squares and circles.

Value

For `simulateSpectra`, a `MassDataFrame` with elements:

- `mz`: a numeric vector of the observed m/z values
- `intensity`: a numeric vector or matrix of the intensities

For `simulateImage`, a `MSImagingExperiment` object.

For `addShape`, a new `PositionDataFrame` with a logical column added for the corresponding shape.

For `presetImageDef`, a list with two elements: the `pixelData` and `featureData` to be used as input to `simulateImage()`.

Author(s)

Kylie A. Bemis

See Also

`simspec`

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")

# generate a spectrum
s <- simulateSpectra(1)
plot(s$intensity ~ s$mz, type="l")

# generate a noisy low-resolution spectrum with a baseline
s <- simulateSpectra(1, baseline=2, sdnoise=0.3, resolution=100)
plot(s$intensity ~ s$mz, type="l")

# generate a high-resolution spectrum
s <- simulateSpectra(1, npeaks=100, resolution=10000)
plot(s$intensity ~ s$mz, type="l")
```

```
# generate an image
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))
peaks <- mz(metadata(mse)$design$featureData)

image(mse, mz=peaks[c(1,4,5,6)])
plot(mse, coord=c(x=3,y=3))
```

`sliceImage` *Slice an image*

Description

Slice a spectral imaging dataset as a "data cube".

Usage

```
sliceImage(x, i = features(x, ...), ..., run = NULL,
    simplify = TRUE, drop = TRUE)
```

Arguments

`x`	A spectral imaging dataset.
`i`	The indices of features to slice for the images.
`...`	Conditions describing features to slice, passed to `features()`.
`run`	The names of experimental runs to include, or the index of the levels of the runs to include.
`simplify`	The image slices be returned as a list, or simplified to an array?
`drop`	Should redundant array dimensions be dropped? If TRUE, dimensions with only one level are dropped using `drop`.

Value

A list or array of the sliced image(s). If multiple images are sliced and `simplify=TRUE`, then the *last* dimension will be the features.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10), centroided=TRUE)
peaks <- mz(metadata(mse)$design$featureData)

# slice image for first feature
sliceImage(mse, 1)
```

```
# slice by m/z-value
sliceImage(mse, mz=peaks[1])

# slice multiple
sliceImage(mse, mz=peaks[1:3])
```

`smooth` *Smooth spectra*

Description

Apply deferred smoothing to spectra.

Usage

```
## S4 method for signature 'SpectralImagingData'
smooth(x,
    method = c("gaussian", "bilateral", "adaptive",
        "diff", "guide", "pag", "sgolay", "ma"), ...)
```

Arguments

`x`	A spectral imaging dataset.
`method`	The smoothing method to use. See `filt1` for details.
`...`	Additional arguments passed to the smoothing function.

Details

The supported smoothing methods are:

- "gaussian": Gaussian smoothing using `filt1_gauss`.
- "bilateral": Bilateral filter using `filt1_bi`.
- "adaptive": Adaptive bilateral filter using `filt1_adapt`.
- "diff": Nonlinear diffusion smoothing using `filt1_diff`.
- "guide": Guided filter using `filt1_guide`.
- "pag": Peak-aware guided filter using `filt1_pag`.
- "sgolay": Savitzky-Golar filter using `filt1_sg`.
- "ma": Moving average filter using `filt1_ma`.

Value

An object of the same class with the processing step queued.

Note

The smoothing is deferred until `process()` is called.

Author(s)

Kylie A. Bemis

See Also

`normalize`, `recalibrate`, `reduceBaseline`, `peakPick`, `process`

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(3,3))

# queue smoothing
mse2 <- smooth(mse, method="gaussian", width=11)
plot(mse2, i=4)

# apply smoothing
mse2 <- process(mse2)
```

SpatialCV *Cross-validation for spectral imaging data*

Description

Apply cross-validation with an existing or a user-specified modeling function over folds of a spectral imaging dataset.

Usage

```
crossValidate(fit., x, y, folds = run(x), ...,
    predict. = predict, keep.models = FALSE,
    trainProcess = peakProcess, trainArgs = list(),
    testProcess = peakProcess, testArgs = list(),
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM())

## S4 method for signature 'SpatialCV'
fitted(object, type = c("response", "class"), ...)

## S4 method for signature 'SpatialCV'
image(x, i = 1L, type = c("response", "class"),
        layout = NULL, free = "", ...)
```

Arguments

`fit.`	The function used to fit the model.
`x, y`	The data and response variable, where `x` is assumed to be an P x N dataset such as a `SpectralImagingExperiment`
`folds`	A vector coercible to a factor giving the fold for each row or column of `x`.
`...`	Additional arguments passed to `fit.` and `predict.`.
`predict.`	The function used to predict on new data from the fitted model. The fitted model is passed as the 1st argument and the test data is passed as the 2nd argument.
`keep.models`	Should the models be kept and returned?
`trainProcess, trainArgs`	A function and arguments used for processing the training sets. The training set is passed as the 1st argument to `trainProcess`.
`testProcess, testArgs`	A function and arguments used for processing the test sets. The test set is passed as the 1st argument to `trainProcess`, and the processed training set is passed as the 2nd argument.
`verbose`	Should progress be printed for each iteration?
`chunkopts`	Chunk processing options. See `chunkApply` for details.
`BPPARAM`	An optional instance of `BiocParallelParam`. See documentation for `bplapply`. *Passed* to `fit.`, `predict.`, `trainProcess` and `testProcess`.
`object`	An object inheriting from `SpatialCV`.
`type`	The type of prediction, where `"response"` means the fitted response matrix and `"class"` will be the vector of class predictions (only for classification).
`i`	If predictions are made for multiple sets of parameters, which set of parameters (i.e., which element of the `fitted.values` list) should be plotted?
`layout`	A vector of the form `c(nrow, ncol)` specifying the number of rows and columns in the facet grid.
`free`	A string specifying the free spatial dimensions during faceting. E.g., `""`, `"x"`, `"y"`, `"xy"`, `"yx"`.

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 conform to certain expectations. Internally, `cv_do` from the `matter` package is used to perform the cross-validation. See `?cv_do` for details.

Value

An object of class `SpatialCV` derived from `SpatialResults` and containing accuracies for each fold, the predictions for each fold, and (optionally) the fitted models.

Author(s)

Kylie A. Bemis

See Also

cv_do, spatialShrunkenCentroids, PLS, OPLS

SpatialDGMM *Spatially-aware Dirichlet Gaussian mixture model*

Description

Fit a spatially-aware Gaussian mixture models to each feature. The model uses Dirichlet prior is used to achieve spatial smoothing. The means and standard deviations of the Gaussian components are estimated using gradient descent. Simulated annealing is used to avoid local optimia and achieve better parameter estimates.

Usage

```
## S4 method for signature 'ANY'
spatialDGMM(x, coord, i, r = 1, k = 2, groups = NULL,
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(coord, r=r, groups=groups),
    annealing = TRUE, compress = TRUE, byrow = FALSE,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
spatialDGMM(x, i, r = 1, k = 2, groups = run(x),
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(coord(x), r=r, groups=groups), ...)

## S4 method for signature 'SpatialDGMM'
logLik(object, ...)

## S4 method for signature 'SpatialDGMM,missing'
plot(x, i = 1L, type = "density",
    layout = NULL, free = "", ...)

## S4 method for signature 'SpatialDGMM'
image(x, i = 1L, type = "class",
    layout = NULL, free = "", ...)
```

Arguments

x	A spatial dataset in P x N matrix format.
i	The rows/columns of x to segment (if not all of them).
coord	The spatial coordinates of the rows/columns of x. Ignored if neighbors is provided.

r	The spatial maximum distance for an observation to be considered a neighbor. Ignored if neighbors is provided.
k	The number of Gaussian components.
groups	Observations belonging to the different groups will be segmented independently. This should be set to the samples if statistic testing (via meansTest is to be performed.)
weights	The type of spatial weights to use for the smoothing. Gaussian weights are weighted only by distance, while adaptive weights also consider the dissimilarity between neighboring observations.
neighbors	A factor giving which observations should be treated as spatially-independent. Observations in the same group are assumed to have a spatial relationship.
annealing	Should simulated annealing be used?
compress	Should the results be compressed? The results can be larger than the original dataset, so compressing them is useful. If this option is used, then the class probabilities are not returned, and the class assignments are compressed using drle.
byrow	Should the rows or columns of x be segmented?
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to the next method.
object	A SpatialDGMM object.
type	The type of plot to display.
layout	A vector of the form c(nrow, ncol) specifying the number of rows and columns in the facet grid.
free	A string specifying the free spatial dimensions during faceting. E.g., "", "x", "y", "xy", "yx".

Value

An object of class SpatialDGMM derived from SpatialResults, containing the fitted sgmixn object and the spatial metadata.

Author(s)

Dan Guo and Kylie A. Bemis

References

Guo, D., Bemis, K., Rawlins, C., Agar, J., and Vitek, O. (2019.) Unsupervised segmentation of mass spectrometric ion images characterizes morphology of tissues. Proceedings of ISMB/ECCB, Basel, Switzerland, 2019.

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=3, dim=c(10,10), npeaks=9,
    peakheight=c(3,6,9), centroided=TRUE)

gmm <- spatialDGMM(mse, r=1, k=4, weights="adaptive")

image(gmm, i=1:9)
```

---

spatialDists *Calculate spatially-smoothed distances*

Description

Calculate distances between observations with smoothing based on their spatial structure.

Usage

```
## S4 method for signature 'ANY'
spatialDists(x, y, coord, r = 1, byrow = TRUE,
        metric = "euclidean", p = 2, weights = NULL,
        neighbors = findNeighbors(coord, r=r),
        neighbors.weights = spatialWeights(coord, r=r),
        verbose = getCardinalVerbose(), chunkopts = list(),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
spatialDists(x, y, r = 1,
        neighbors = findNeighbors(x, r=r),
        neighbors.weights = spatialWeights(x, r=r), ...)

## S4 method for signature 'PositionDataFrame'
spatialDists(x, y, r = 1,
        neighbors = findNeighbors(x, r=r),
        neighbors.weights = spatialWeights(x, r=r), ...)
```

Arguments

x	A data matrix with rows or columns located at the coordinates given by coord.
y	A data matrix from which to calculate distances with the observations in x.
coord	The spatial coordinates of the rows/columns of x. Ignored if neighbors is provided.
r	The spatial maximum distance for an observation to be considered a neighbor. Ignored if neighbors is provided.
byrow	Are the distances calculated based on the dissimilarity between the rows (TRUE) or the columns (FALSE) of x and y.

metric	Distance metric to use when finding the nearest neighbors. Supported metrics include "euclidean", "maximum", "manhattan", and "minkowski".
p	The power for the Minkowski distance.
weights	A numeric vector of *feature* weights for the distance components if calculating weighted distances. For example, the weighted Euclidean distance is `sqrt(sum(w * (x - y)^2))`.
neighbors	A list of numeric vectors giving the row or column indices of the spatial neighbors for the rows or columns of `x`.
neighbors.weights	A list of numeric vectors giving the spatial weights corresponding to `neighbors`.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See `chunkApply` for details.
BPPARAM	An optional instance of `BiocParallelParam`. See documentation for `bplapply`.
...	Additional arguments passed to the next method.

Value

A matrix of distances with rows equal to the number of observations in `x` and columns equal to the number of observations in `y`.

Author(s)

Kylie A. Bemis

See Also

`findNeighbors`, `spatialWeights`

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, dim=c(10,10))

# calculate spatially-aware distances from first 5 spectra
spatialDists(mse, spectra(mse)[,1:5], r=1)
```

SpatialFastmap *Spatially-aware FastMap projection*

Description

Compute spatially-aware FastMap projection.

Usage

```
## S4 method for signature 'ANY'
spatialFastmap(x, coord, r = 1, ncomp = 3,
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(coord, r=r),
    transpose = TRUE, niter = 10L,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
spatialFastmap(x, r = 1, ncomp = 3,
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(x, r=r), ...)

## S4 method for signature 'SpatialFastmap'
predict(object, newdata,
    weights = object$weights, r = object$r,
    neighbors = findNeighbors(newdata, r=r),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialFastmap,missing'
plot(x, type = c("scree", "x"), ..., xlab, ylab)

## S4 method for signature 'SpatialFastmap'
image(x, type = "x", ...)
```

Arguments

x	A spatial dataset in P x N matrix format.
coord	The spatial coordinates of the rows/columns of x. Ignored if neighbors is provided.
r	The spatial maximum distance for an observation to be considered a neighbor. Ignored if neighbors is provided.
ncomp	The number of FastMap components.
weights	The type of spatial weights to use for the smoothing. Gaussian weights are weighted only by distance, while adaptive weights also consider the dissimilarity between neighboring observations.
neighbors	A factor giving which observations should be treated as spatially-independent. Observations in the same group are assumed to have a spatial relationship.
transpose	Should x be considered P x N?
niter	The number of iterations used to calculate the pivots for each FastMap component.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

...	Additional arguments passed to the next method.
object	A SpatialFastmap object.
newdata	A new SpectralImagingExperiment for which to calculate the scores.
type	The type of plot to display.
xlab, ylab	Plotting labels.

Value

An object of class SpatialFastmap derived from SpatialResults, containing the fitted fastmap object and the spatial metadata.

Author(s)

Kylie A. 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

PCA, NMF, spatialKMeans

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=2, npeaks=20, dim=c(10,10),
    centroided=TRUE)

# project to FastMap components
fm <- spatialFastmap(mse, r=1, ncomp=2, weights="adaptive")

# visualize first 2 components
image(fm)
```

SpatialKMeans *Spatially-aware K-means clustering*

Description

Perform spatially-aware k-means clustering. First the data is projected to a reduced dimension space using spatialFastmap. Then ordinary k-means clustering is applied to the projected data.

Usage

```
## S4 method for signature 'ANY'
spatialKMeans(x, coord, r = 1, k = 2, ncomp = max(k),
        weights = c("gaussian", "adaptive"),
        neighbors = findNeighbors(coord, r=r),
        transpose = TRUE, niter = 10L,
        centers = TRUE, correlation = TRUE,
        verbose = getCardinalVerbose(), chunkopts = list(),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
spatialKMeans(x, r = 1, k = 2, ncomp = max(k),
        weights = c("gaussian", "adaptive"),
        neighbors = findNeighbors(x, r=r), ...)

## S4 method for signature 'SpatialKMeans'
topFeatures(object, n = Inf, sort.by = "correlation", ...)

## S4 method for signature 'SpatialKMeans,missing'
plot(x, type = c("correlation", "centers"), ..., xlab, ylab)

## S4 method for signature 'SpatialKMeans'
image(x, type = "cluster", ...)
```

Arguments

x	A spatial dataset in P x N matrix format.
coord	The spatial coordinates of the rows/columns of x. Ignored if neighbors is provided.
r	The spatial maximum distance for an observation to be considered a neighbor. Ignored if neighbors is provided.
k	The number of clusters.
ncomp	The number of FastMap components.
weights	The type of spatial weights to use for the smoothing. Gaussian weights are weighted only by distance, while adaptive weights also consider the dissimilarity between neighboring observations.
neighbors	A factor giving which observations should be treated as spatially-independent. Observations in the same group are assumed to have a spatial relationship.
transpose	Should x be considered P x N?
niter	The number of iterations used to calculate the pivots for each FastMap component.
centers	Should the cluster centers be re-calculated on the original data?
correlation	Should the correlations between features and the clusters be calculated?
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.

BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to the next method.
object	A SpatialKMeans object.
n, sort.by	For topFeatures, the number of top features to return and how to sort them.
type	The type of plot to display.
xlab, ylab	Plotting labels.

Value

An object of class SpatialKMeans derived from SpatialResults, containing the fitted kmeans object and the spatial metadata.

Author(s)

Kylie A. 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

spatialKMeans spatialShrunkenCentroids

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=3, dim=c(10,10), npeaks=20,
    peakheight=c(3,6,9), centroided=TRUE)

# fit spatial k-means
skm <- spatialKMeans(mse, r=1, k=4, weights="adaptive")

# visualize clusters
image(skm)
```

SpatialNMF *Non-negative matrix factorization*

Description

Compute nonnegative matrix factorization using alternating least squares or multiplicative updates.

Usage

```
## S4 method for signature 'ANY'
NMF(x, ncomp = 3, method = c("als", "mult"),
    verbose = getCardinalVerbose(), ...)

## S4 method for signature 'SpectralImagingExperiment'
NMF(x, ncomp = 3, method = c("als", "mult"), ...)

## S4 method for signature 'SpatialNMF'
predict(object, newdata, ...)

## S4 method for signature 'SpatialNMF,missing'
plot(x, type = c("activation", "x"), ..., xlab, ylab)

## S4 method for signature 'SpatialNMF'
image(x, type = "x", ...)
```

Arguments

x	A dataset in P x N matrix format.
ncomp	The number of components to calculate.
method	The method to use. Alternating least squares ("als") tends to be faster and potentially more accurate, but can be numerically unstable for data with high correlated features. Multiplicative updates ("mult") can be slower, but is more numerically stable.
verbose	Should progress messages be printed?
...	Options passed to irlba.
object	A SpatialNMF object.
newdata	A new SpectralImagingExperiment for which to calculate the scores.
type	The type of plot to display.
xlab, ylab	Plotting labels.

Value

An object of class SpatialNMF derived from SpatialResults, containing the fitted nnmf object and the spatial metadata.

Author(s)

Kylie A. Bemis

See Also

nnmf_als, nnmf_mult, PCA, spatialFastmap

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=2, npeaks=20, dim=c(10,10),
    centroided=TRUE)

# project to principal components
mf <- NMF(mse, ncomp=2)

# visualize first 2 components
image(mf, superpose=FALSE, scale=TRUE)
```

SpatialPCA *Principal components analysis*

Description

Compute principal components efficiently using implicitly restarted Lanczos bi-diagonalization (IRLBA) algorithm for approximate singular value decomposition.

Usage

```
## S4 method for signature 'ANY'
PCA(x, ncomp = 3,
    center = TRUE, scale = FALSE,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
PCA(x, ncomp = 3,
    center = TRUE, scale = FALSE, ...)

## S4 method for signature 'SpatialPCA'
predict(object, newdata,
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialPCA,missing'
plot(x, type = c("rotation", "scree", "x"), ..., xlab, ylab)

## S4 method for signature 'SpatialPCA'
image(x, type = "x", ...)
```

Arguments

x	A dataset in P x N matrix format.
ncomp	The number of principal components to calculate.
center	Should the data be centered?
scale	Shoud the data be scaled?
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Options passed to irlba.
object	A SpatialPCA object.
newdata	A new SpectralImagingExperiment for which to calculate the scores.
type	The type of plot to display.
xlab, ylab	Plotting labels.

Value

An object of class SpatialPCA derived from SpatialResults, containing the fitted prcomp_lanczos object and the spatial metadata.

Author(s)

Kylie A. Bemis

See Also

prcomp_lanczos, NMF, spatialFastmap, irlba, svd

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=2, npeaks=20, dim=c(10,10),
    centroided=TRUE)

# project to principal components
pc <- PCA(mse, ncomp=2)

# visualize first 2 components
image(pc, superpose=FALSE, scale=TRUE)
```

SpatialPLS *Partial least squares (projection to latent structures)*

Description

Compute partial least squares (also called projection to latent structures or PLS). This will also perform discriminant analysis (PLS-DA) if the response is a factor. Orthogonal partial least squares options (O-PLS and O-PLS-DA) is also supported; in this case, O-PLS step is a pre-processing step to remove noise orthogonal to the response, before fitting a PLS model with a single component.

Usage

```
## S4 method for signature 'ANY'
PLS(x, y, ncomp = 3,
    method = c("nipals", "simpls", "kernel1", "kernel2"),
    center = TRUE, scale = FALSE, bags = NULL,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
PLS(x, y, ncomp = 3,
    method = c("nipals", "simpls", "kernel1", "kernel2"),
    center = TRUE, scale = FALSE, ...)

## S4 method for signature 'SpatialPLS'
fitted(object, type = c("response", "class"), ...)

## S4 method for signature 'SpatialPLS'
predict(object, newdata, ncomp,
        type = c("response", "class"), simplify = TRUE, ...)

## S4 method for signature 'SpatialPLS'
topFeatures(object, n = Inf, sort.by = c("vip", "coefficients"), ...)

## S4 method for signature 'SpatialPLS,missing'
plot(x, type = c("coefficients", "vip", "scores"), ..., xlab, ylab)

## S4 method for signature 'SpatialPLS'
image(x, type = c("response", "class"), ...)

## S4 method for signature 'ANY'
OPLS(x, y, ncomp = 3, retx = TRUE,
    center = TRUE, scale = FALSE, bags = NULL,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
```

```
OPLS(x, y, ncomp = 3, retx = FALSE,
    center = TRUE, scale = FALSE, ...)

## S4 method for signature 'SpatialOPLS'
coef(object, ...)

## S4 method for signature 'SpatialOPLS'
residuals(object, ...)

## S4 method for signature 'SpatialOPLS'
fitted(object, type = c("response", "class"), ...)

## S4 method for signature 'SpatialOPLS'
predict(object, newdata, ncomp,
        type = c("response", "class"), simplify = TRUE, ...)

## S4 method for signature 'SpatialOPLS'
topFeatures(object, n = Inf, sort.by = c("vip", "coefficients"), ...)

## S4 method for signature 'SpatialOPLS,missing'
plot(x, type = c("coefficients", "vip", "scores"), ..., xlab, ylab)

## S4 method for signature 'SpatialOPLS'
image(x, type = c("response", "class"), ...)
```

Arguments

x	A dataset in P x N matrix format.
y	The response variable.
ncomp	The number of principal components to calculate.
method	The method used for calculating the principal components. See pls for details.
center	Should the data be centered?
scale	Shoud the data be scaled?
bags	Bags for multiple instance learning. If provided, then it is assumed all observations within a bag have the same label, and if a single observation is "positive" then all observations in the bag are "positive". Multiple instance learning is performed using mi_learn.
retx	Should the (potentially large) processed data matrix be included in the result?
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Options passed to irlba.
object	A SpatialPLS or SpatialOPLS object.
newdata	A new SpectralImagingExperiment for which to make predictions.
type	The type of fitted values to extract or the type of predictions to make.

simplify	If predictions are made using multiple numbers of components, should they be returned as a list, or simplified to an array?
n, sort.by	For topFeatures, the number of top features to return and how to sort them.
xlab, ylab	Plotting labels.

Value

An object of class SpatialPLS or SpatialOPLS derived from SpatialResults, containing the fitted pls or opls model and the spatial metadata.

Author(s)

Kylie A. 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

PCA, spatialShrunkenCentroids,

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=2, npeaks=20, dim=c(10,10), centroided=TRUE)
cls <- makeFactor(circle=pData(mse)$circle, square=pData(mse)$square)

# fit a PLS model with 3 components
pls <- PLS(mse, cls, ncomp=1:3)
plot(pls, type="coefficients", annPeaks="circle")

# visualize predictions
image(pls)
```

SpatialResults-class *SpatialResults: Modeling results with spatial metadata*

Description

The SpatialResults class provides a container for modeling results with spatial metadata. Most modeling functions applied to a SpectralImagingExperiment will return a SpatialResults-derived model object.

Usage

```
## Instance creation
SpatialResults(model, data,
    featureData = if (!missing(data)) fData(data) else NULL,
    pixelData = if (!missing(data)) pData(data) else NULL)

## S4 method for signature 'SpatialResults,ANY'
plot(x, y, ...,
    select = NULL, groups = NULL,
    superpose = TRUE, reducedDims = FALSE)

## S4 method for signature 'SpatialResults'
image(x, y, ...,
    select = NULL, subset = TRUE,
    superpose = TRUE)

## Additional methods documented below
```

Arguments

model	The model object.
data	An object (typically the original dataset) with featureData and pixelData components.
featureData	A DataFrame with feature metadata, with a row for each feature.
pixelData	A PositionDataFrame with pixel metadata, with a row for each spectrum.
x, y	The model object and results to plot. (Not typically called directly.)
...	Additional options passed to plotting methods.
select	Select elements of the results to plot. For example, this selects a subset of matrix columns or a subset of factor levels to plot.
subset	A logical vector indicating which pixels to include in the image.
groups	A vector coercible to a factor indicating which of the specified spectra should be plotted with the same color.
superpose	If multiple results are plotted, should they be superposed on top of each other, or plotted seperately?
reducedDims	Does this results component represent reduced dimensions (e.g., from PCA)?

Slots

model: The model.

featureData: A DataFrame containing feature-level metadata (e.g., a color channel, a molecular analyte, or a mass-to-charge ratio).

pixelData: A PositionDataFrame containing spatial metadata, including each observations's pixel coordinates and experimental run information.

Methods

`modelData(object)`, `modelData(object) <- value`: Get or set the `model` slot.

`featureData(object)`, `featureData(object) <- value`: Get or set the `featureData` slot.

`fData(object)`, `fData(object) <- value`: Get or set the `featureData` slot.

`featureNames(object)`, `featureNames(object) <- value`: Get or set the feature names (i.e., the row names of the `featureData` slot).

`pixelData(object)`, `pixelData(object) <- value`: Get or set the `elementMetadata` slot.

`pData(object)`, `pData(object) <- value`: Get or set the `elementMetadata` slot.

`pixelNames(object)`, `pixelNames(object) <- value`: Get or set the pixel names (i.e., the row names of the `elementMetadata` slot).

`coord(object)`, `coord(object) <- value`: Get or set the pixel coordinate columns in `pixelData`.

`coordNames(object)`, `coordNames(object) <- value`: Get or set the names of the pixel coordinate columns in `pixelData`.

`run(object)`, `run(object) <- value`: Get or set the experimental run column from `pixelData`.

`runNames(object)`, `runNames(object) <- value`: Get or set the experimental run levels from `pixelData`.

`nrun(object)`: Get the number of experimental runs.

Author(s)

Kylie A. Bemis

See Also

`ResultsList`

`SpatialShrunkenCentroids` *Spatially-aware shrunken centroid clustering and classification*

Description

Perform spatially-aware nearest shrunken centroid clustering or classification. These methods use statistical regularization to shrink the t-statistics of the features toward 0 so that unimportant features are removed from the model. The dissimilarity to class centroids are spatially smoothed.

Usage

```
## S4 method for signature 'ANY,ANY'
spatialShrunkenCentroids(x, y, coord, r = 1, s = 0,
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(coord, r=r), bags = NULL,
    priors = table(y), center = NULL, transpose = TRUE,
```

```
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment,ANY'
spatialShrunkenCentroids(x, y, r = 1, s = 0,
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(x, r=r), ...)

## S4 method for signature 'ANY,missing'
spatialShrunkenCentroids(x, coord, r = 1, k = 2, s = 0,
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(coord, r=r),
    init = NULL, threshold = 0.01, niter = 10L,
    center = NULL, transpose = FALSE,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment,missing'
spatialShrunkenCentroids(x, r = 1, k = 2, s = 0,
    weights = c("gaussian", "adaptive"),
    neighbors = findNeighbors(x, r=r), ...)

## S4 method for signature 'SpatialShrunkenCentroids'
fitted(object, type = c("response", "class"), ...)

## S4 method for signature 'SpatialShrunkenCentroids'
predict(object, newdata,
        type = c("response", "class"),
        weights = object$weights, r = object$r,
        neighbors = findNeighbors(newdata, r=r),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialShrunkenCentroids'
logLik(object, ...)

## S4 method for signature 'SpatialShrunkenCentroids'
topFeatures(object, n = Inf, sort.by = c("statistic", "centers"), ...)

## S4 method for signature 'SpatialShrunkenCentroids,missing'
plot(x, type = c("statistic", "centers"), ..., xlab, ylab)

## S4 method for signature 'SpatialShrunkenCentroids'
image(x, type = c("probability", "class"), ...)
```

Arguments

x	A spatial dataset in P x N matrix format.
y	The response variable.

coord	The spatial coordinates of the rows/columns of x. Ignored if neighbors is provided.
r	The spatial maximum distance for an observation to be considered a neighbor. Ignored if neighbors is provided.
k	The number of classes for clustering.
s	The sparsity parameter.
weights	The type of spatial weights to use for the smoothing. Gaussian weights are weighted only by distance, while adaptive weights also consider the dissimilarity between neighboring observations.
neighbors	A factor giving which observations should be treated as spatially-independent. Observations in the same group are assumed to have a spatial relationship.
bags	Bags for multiple instance learning. If provided, then it is assumed all observations within a bag have the same label, and if a single observation is "positive" then all observations in the bag are "positive". Multiple instance learning is performed using mi_learn.
priors	The (unnormalized) prior probabilities for each class.
center	The global centroid (if known).
transpose	Should x be considered P x N?
init	A list of initial cluster configurations. (Should resemble the output of kmeans.)
threshold	Stop iteration when the proportion of cluster assignment updates is less than this threshold.
niter	The maximum number of iterations.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to the next method.
object	A SpatialShrunkenCentroids object.
newdata	A new SpectralImagingExperiment for which to make predictions.
type	The type of fitted values to extract or the type of predictions to make.
n, sort.by	For topFeatures, the number of top features to return and how to sort them.
xlab, ylab	Plotting labels.

Value

An object of class SpatialShrunkenCentroids derived from SpatialResults, containing the fitted nscentroids object and the spatial metadata.

Author(s)

Kylie A. Bemis

References

Bemis, K., Harry, A., Eberlin, L. S., Ferreira, C., van de Ven, S. M., Mallick, P., Stolowitz, M., and Vitek, O. (2016.) Probabilistic segmentation of mass spectrometry images helps select important ions and characterize confidence in the resulting segments. Molecular & Cellular Proteomics. doi:10.1074/mcp.O115.053918

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, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=3, dim=c(10,10), npeaks=20,
    peakheight=c(3,6,9), centroided=TRUE)

# fit spatial shrunken centroids
ssc <- spatialShrunkenCentroids(mse, r=1, k=4, s=c(0,3,6,9), weights="adaptive")

# visualize classes
image(ssc, i=1:4)

# visualize t-statistics
plot(ssc, i=1:4)
```

---

spatialWeights *Calculate spatial weights*

Description

Calculate weights for neighboring observations based on either the spatial distance between the neighbors or the dissimilarity between the observations.

Usage

```
## S4 method for signature 'ANY'
spatialWeights(x, coord = x, r = 1, byrow = TRUE,
    neighbors = findNeighbors(coord, r=r),
    weights = c("gaussian", "adaptive"),
    sd = ((2 * r) + 1) / 4, matrix = FALSE,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)
```

```
## S4 method for signature 'SpectralImagingExperiment'
spatialWeights(x, r = 1,
        neighbors = findNeighbors(x, r=r),
        weights = c("gaussian", "adaptive"), ...)

## S4 method for signature 'PositionDataFrame'
spatialWeights(x, r = 1,
        neighbors = findNeighbors(x, r=r),
        weights = c("gaussian", "adaptive"), ...)
```

Arguments

x	Either a matrix or data frame of spatial coordinates, or a data matrix with rows or columns located at the coordinates given by coord.
coord	The spatial coordinates of the rows/columns of x. Ignored if neighbors is provided.
r	The spatial maximum distance for an observation to be considered a neighbor. Ignored if neighbors is provided.
byrow	If x is a data matrix, then are the weights calculated based on the dissimilarity between the rows (TRUE) or the columns (FALSE).
neighbors	A list of numeric vectors giving the row or column indices of the spatial neighbors for the rows or columns of x.
weights	The type of weights to calculate. Either Gaussian weights with a constant standard deviation, or adaptive weights with a standard deviation based on the dissimilarity between the neighboring observations.
sd	The standard deviation for the Gaussian weights. Ignored with weights="adaptive".
matrix	Should the weights be returned as a sparse adjacency matrix instead of a list?
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to the next method.

Value

Either a list of weights of neighbors or a sparse adjacency matrix (sparse_mat).

Author(s)

Kylie A. Bemis

See Also

findNeighbors

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, dim=c(10,10))

# calculate weights based on distance
spatialWeights(pixelData(mse), r=1)

# calculate weights based on spectral dissimilarity
spatialWeights(mse, r=1)
```

SpectraArrays-class *SpectraArrays: List of spectra arrays*

Description

The SpectraArrays class provides a list-like container for spectra arrays of conformable dimensions.

Usage

```
## Instance creation
SpectraArrays(arrays = SimpleList())

## Additional methods documented below
```

Arguments

arrays	A list of arrays.

Details

The SpectraArrays class is intended to be flexible and the arrays do not need to be "array-like" (i.e., have non-NULL dim().) One dimensional arrays and lists are allowd. Every array must have the same NROW() and NCOL().

It supports lossless coercion to and from SimpleList.

Methods

length(object): Get the number of spectra in the object.

names(object), names(object) <- value: Get or set the names of spectra arrays in the object.

object[[i]], object[[i]] <- value: Get or set an array in the object.

object[i, j, ..., drop]: Subset as a list or array, depending on the number of dimensions of the stored spectra arrays. The result is the same class as the original object.

rbind(...), cbind(...): Combine SpectraArrays objects by row or column.

c(...): Combine SpectraArrays objects as lists.

fetch(object, ...): Pull data arrays into shared memory.

flash(object, ...): Push data arrays to a temporary file.

Author(s)

Kylie A. Bemis

See Also

SpectralImagingData, MSImagingArrays

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- matrix(rlnorm(128), nrow=16, ncol=8)
y <- matrix(rlnorm(128), nrow=16, ncol=8)

s <- SpectraArrays(list(x=x, y=y))

print(s)
```

SpectralImagingArrays-class

SpectralImagingArrays: Spectral imaging data with arbitrary domain

---

Description

The SpectralImagingArrays class provides a list-like container for high-dimensional spectral imaging data where every spectrum may have its own domain values. It is designed to provide easy access to raw individual spectra, but images cannot be easily reconstructed.

The MSImagingArrays class extends SpectralImagingArrays for mass spectrometry-based imaging experiments with unaligned mass features.

Usage

```
## Instance creation
SpectralImagingArrays(spectraData = SimpleList(),
    pixelData = PositionDataFrame(), metadata = list())

## Additional methods documented below
```

Arguments

spectraData	Either a list-like object with lists of individual spectra and lists of their domain values, or a SpectraArrays instance.
pixelData	A PositionDataFrame with pixel metadata, with a row for each spectrum.
metadata	A list with experimental-level metadata.

Slots

`spectraData`: A `SpectraArrays` object storing one or more array-like data elements with conformable dimensions.

`elementMetadata`: A `PositionDataFrame` containing spectrum-level metadata, including each spectrum's pixel coordinates and experimental run information.

`processing`: A list containing unexecuted `ProcessingStep` objects.

Methods

All methods for `SpectralImagingData` also work on `SpectralImagingArrays` objects. Additional methods are documented below:

`length(object)`: Get the number of spectra in the object.

`object[i, ..., drop]`: Subset as a list based on the spectra. The result is the same class as the original object.

`rbind(...)`, `cbind(...)`: Combine `SpectralImagingArrays` objects by row or column.

Author(s)

Kylie A. Bemis

See Also

`SpectralImagingData`, `MSImagingArrays`

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- replicate(9, rlnorm(10), simplify=FALSE)
t <- replicate(9, sort(runif(10)), simplify=FALSE)
coord <- expand.grid(x=1:3, y=1:3)

sa <- SpectralImagingArrays(
    spectraData=list(intensity=x, wavelength=t),
    pixelData=PositionDataFrame(coord))

print(sa)
```

`SpectralImagingData-class`

SpectralImagingData: Abstract class for spectral imaging data

---

Description

The SpectralImagingData class is an abstract container for high-dimensional spectral imaging data. Every spectrum is associated with spatial coordinates so that an image can be constructed from the spectral intensities.

The SpectralImagingArrays and SpectralImagingExperiment classes directly extend this class, where SpectralImagingArrays is primarily intended for unprocessed spectra with unaligned features, and SpectralImagingExperiment is intended for processed spectra with aligned features.

The MSImagingArrays and MSImagingExperiment classes further extend these classes for mass spectrometry imaging data.

Slots

spectraData: A SpectraArrays object storing one or more array-like data elements with conformable dimensions.

elementMetadata: A PositionDataFrame containing spectrum-level metadata, including each spectrum's pixel coordinates and experimental run information.

processing: A list containing unexecuted ProcessingStep objects.

Methods

spectraData(object, ...), spectraData(object, ...) <- value: Get or set the spectraData slot.

spectraNames(object, ...), spectraNames(object, ...) <- value: Get or set the names of the spectra in the spectraData slot.

spectra(object, i = 1L, ...), spectra(object, i = 1L, ...) <- value: Get or set a specific spectra array in the spectraData slot.

pixelData(object), pixelData(object) <- value: Get or set the elementMetadata slot.

pData(object), pData(object) <- value: Get or set the elementMetadata slot.

pixelNames(object), pixelNames(object) <- value: Get or set the pixel names (i.e., the row names of the elementMetadata slot).

spectraVariables(object, ...): Get the names of the spectrum-level variables (i.e., the columns of the elementMetadata slot).

coord(object), coord(object) <- value: Get or set the pixel coordinate columns in pixelData.

coordNames(object), coordNames(object) <- value: Get or set the names of the pixel coordinate columns in pixelData.

run(object), run(object) <- value: Get or set the experimental run column from pixelData.

runNames(object), runNames(object) <- value: Get or set the experimental run levels from pixelData.

nrun(object): Get the number of experimental runs.

is3D(object): Check if the number of spatial dimensions is greater than 2.

processingData(object, ...), processingData(object, ...) <- value: Get or set the processing slot.

fetch(object, ...): Pull spectraData into shared memory.

flash(object, ...): Push spectraData to a temporary file.

Author(s)

Kylie A. Bemis

See Also

SpectralImagingExperiment, SpectralImagingArrays, MSImagingExperiment, MSImagingArrays

---

SpectralImagingExperiment-class
SpectralImagingExperiment: Spectral imaging data with shared domain

---

Description

The SpectralImagingExperiment class provides a matrix-like container for high-dimensional spectral imaging data where every spectrum shares the same domain values. It is designed to provide easy access to both the spectra (as columns) and sliced images (as rows).

The MSImagingExperiment class extends SpectralImagingExperiment for mass spectrometry-based imaging experiments with aligned mass features.

Usage

```
## Instance creation
SpectralImagingExperiment(spectraData = SimpleList(),
    featureData = DataFrame(), pixelData = PositionDataFrame(),
    metadata = list())

## Additional methods documented below
```

Arguments

spectraData	Either a matrix-like object with number of rows equal to the number of features and number of columns equal to the number of pixels, a list of such objects, or a SpectraArrays instance.
featureData	A DataFrame with feature metadata, with a row for each feature.
pixelData	A PositionDataFrame with pixel metadata, with a row for each spectrum.
metadata	A list with experimental-level metadata.

Slots

spectraData: A SpectraArrays object storing one or more array-like data elements with conformable dimensions.

featureData: A DataFrame containing feature-level metadata (e.g., a color channel, a molecular analyte, or a mass-to-charge ratio).

elementMetadata: A PositionDataFrame containing spectrum-level metadata, including each spectrum's pixel coordinates and experimental run information.

processing: A list containing unexecuted ProcessingStep objects.

Methods

All methods for SpectralImagingData also work on SpectralImagingExperiment objects. Additional methods are documented below:

`featureData(object)`, `featureData(object) <- value`: Get or set the `featureData` slot.

`fData(object)`, `fData(object) <- value`: Get or set the `featureData` slot.

`featureNames(object)`, `featureNames(object) <- value`: Get or set the feature names (i.e., the row names of the `featureData` slot).

`length(object)`: Get the number of spectra in the object.

`nrow(object)`, `ncol(object)`: Get the number of rows (features) or the number of columns (pixels) in the object.

`object[i, j, ..., drop]`: Subset based on the rows (`featureData`) and the columns (`pixelData`). The result is the same class as the original object.

`rbind(...)`, `cbind(...)`: Combine `SpectralImagingExperiment` objects by row or column.

Author(s)

Kylie A. Bemis

See Also

SpectralImagingData, MSImagingExperiment

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
x <- matrix(rlnorm(81), nrow=9, ncol=9)
index <- 1:9
coord <- expand.grid(x=1:3, y=1:3)

se <- SpectralImagingExperiment(
    spectraData=x,
    featureData=DataFrame(index=1:9),
    pixelData=PositionDataFrame(coord))

print(se)
```

---

spectrapply *Apply functions over spectra*

Description

Apply a user-specified function over all spectra in a spectral imaging dataset.

Usage

```
## S4 method for signature 'SpectralImagingExperiment'
spectrapply(object, FUN, ...,
        spectra = "intensity", index = NULL,
        simplify = TRUE, outpath = NULL,
        verbose = getCardinalVerbose(), chunkopts = list(),
        BPPARAM = getCardinalBPPARAM())

## S4 method for signature 'SpectralImagingArrays'
spectrapply(object, FUN, ...,
        spectra = "intensity", index = NULL,
        simplify = TRUE, outpath = NULL,
        verbose = getCardinalVerbose(), chunkopts = list(),
        BPPARAM = getCardinalBPPARAM())
```

Arguments

object	A spectral imaging dataset.
FUN	A function to be applied. The first argument will be the spectra elements. Additional arguments are passed for each index component.
...	Options passed to chunkMapply or chunkApply.
spectra	The name of the array in spectraData() to use for the spectral intensities.
index	The name of the array in spectraData() (for SpectralImagingArrays) or column in featureData() (for SpectralImagingExperiment) to use for the spectral locations.
simplify	Should the result be simplified to an array if possible?
outpath	Optional. The name of a file to write the resulting data.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Value

A list if simplify=FALSE. Otherwise, a vector or matrix, or a higher-dimensional array if the attempted simplification is successful.

Author(s)

Kylie A. Bemis

See Also

summarizeFeatures, summarizePixels

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

# find m/z locations of peaks in each spectrum
peaks <- spectrapply(mse, index="mz",
    function(x, mz) mz[matter::findpeaks(x)])

head(peaks[[1L]])
head(peaks[[2L]])
```

`subsetFeatures` *Subset a spectral imaging dataset*

Description

Returns a subset of the dataset that meets the conditions.

Usage

```
## S4 method for signature 'SpectralImagingArrays'
subset(x, subset, ...)

## S4 method for signature 'SpectralImagingExperiment'
subset(x, select, subset, ...)

subsetFeatures(x, ...)

subsetPixels(x, ...)
```

Arguments

`x`	A spectral imaging dataset.
`select`	Logical expression to be evaluated in the object's `featureData()` indicating which rows (features) to keep.
`subset`	Logical expression to be evaluated in the object's `pixelData()` indicating which columns (pixels) to keep.
`...`	Conditions describing rows (features) or columns (pixels) to be retained. Passed to `features()` and `pixels()` methods to obtain the subset indices.

Value

An object of the same class as `x` with the appropriate subsetting applied to it.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

# subset features to mass range 1000 - 1500
subsetFeatures(mse, 1000 < mz, mz < 1500)

# select pixels to coordinates x = 1..3, y = 1..3
subsetPixels(mse, x <= 3, y <= 3)

# subset both features + pixels
subset(mse, 1000 < mz & mz < 1500, x <= 3 & y <= 3)
```

---

summarizeFeatures *Summarize a spectral imaging dataset*

Description

Summarizes over the rows or columns of the dataset.

Usage

```
summarizeFeatures(x, stat = "mean", groups = NULL,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

summarizePixels(x, stat = c(tic="sum"), groups = NULL,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
rowStats(x, stat, ...,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM())

## S4 method for signature 'SpectralImagingExperiment'
colStats(x, stat, ...,
    verbose = getCardinalVerbose(), chunkopts = list(),
    BPPARAM = getCardinalBPPARAM())

## S4 method for signature 'SpectralImagingExperiment'
rowSums(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'SpectralImagingExperiment'
colSums(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'SpectralImagingExperiment'
```

```
rowMeans(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'SpectralImagingExperiment'
colMeans(x, na.rm = FALSE, dims = 1, ...)
```

Arguments

x	A spectral imaging dataset.
stat	The name of summary statistics to compute over the rows or columns of a matrix. Allowable values include: "min", "max", "prod", "sum", "mean", "var", "sd", "any", "all", and "nnzero".
groups	A vector coercible to a factor giving groups to summarize.
na.rm	If TRUE, remove NA values before summarizing.
dims	Ignored.
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to rowStats or colStats, such as the number of chunks.

Value

For summarizeFeatures and summarizePixels, an object of the same class as x with the statistical summaries added as columns in the featureData() or pixelData(), respectively.

For rowStats, colStats, etc., a vector, matrix, or array with the summary statistics.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1, kind="L'Ecuyer-CMRG")
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

# summarize mean spectrum
mse <- summarizeFeatures(mse, stat="mean")
plot(mse, "mean")

# summarize total ion current
mse <- summarizePixels(mse, stat=c(TIC="sum"))
image(mse, "TIC")
```

writeMSIData *Write mass spectrometry imaging data files*

Description

Write supported mass spectrometry imaging data files, including imzML and Analyze 7.5.

Usage

```
writeMSIData(object, file, ...)

## S4 method for signature 'MSImagingExperiment_OR_Arrays'
writeImzML(object, file, bundle = TRUE,
 verbose = getCardinalVerbose(), chunkopts = list(),
 BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'MSImagingExperiment'
writeAnalyze(object, file,
 verbose = getCardinalVerbose(), chunkopts = list(),
 BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpectralImagingExperiment'
writeAnalyze(object, file,
 verbose = getCardinalVerbose(), chunkopts = list(),
 BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

object	A spectral imaging dataset.
file	The absolute or relative file path. The file extension must be included for writeMSIData.
bundle	Should the ".imzML" and ".ibd" files be bundled into a new directory of the same name?
verbose	Should progress messages be printed?
chunkopts	Chunk processing options. See chunkApply for details.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to writeImzML or writeAnalyze.

Details

The writeImzML function supports writing both the "continuous" and "processed" formats.

Exporting the experimental metadata to cvParam tags is lossy, and not all metadata will be preserved. If exporting an object that was originally imported from an imzML file, only metadata that appears in experimentData() will be preserved when writing.

Datasets with multiple experimental runs will be merged into a single file. The object's `pixelData()` and `featureData()` will also be written to tab-delimted files if appropriate. These will be read back in by `readImzML()`.

The imzML files can be modified after writing (such as to add additional experimental metadata) using the Java-based imzMLValidator application: https://gitlab.com/imzML/imzMLValidator/.

Value

`TRUE` if the file was written successfully, with the output file paths and data objects attached as attributes.

Author(s)

Kylie A. 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

`readMSIData`

XDataFrame-class *XDataFrame: Extended data frame with key columns*

Description

The `XDataFrame` extends the `DataFrame` class from the `S4Vectors` package with support for columns (or sets of columns) designated as keys.

Usage

```
XDataFrame(..., keys = list())
```

Arguments

`...`	Arguments passed to the `DataFrame()`.
`keys`	A named list of character vectors giving the names of key columns. The names of the list become the names of the keys (which may be different from the columns). The character vectors specify the names of columns that compose that key.

Details

For the most part, `XDataFrame` behaves identically to `DataFrame`, and key columns can be get or set as usual.

The `XDataFrame` class is primarily intended as a way to enforce additional requirements or constraints on specific sets of columns in a structured way. It provides an abstracted way of manipulating sets of columns that are expected to follow certain rules. The keys remain consistent and accessible even if the columns of the data frame are renamed.

The base class currently has only minimal requirements for keys (i.e., that they are valid columns in the data frame). Additionally, keys are checked for compatibility when combining data frames. Uniqueness is *not* checked.

Subclasses can enforce additional constraints on key columns. For example, the `PositionDataFrame` and `MassDataFrame` classes.

Methods

`keys(object, i = NULL, ..., drop = TRUE)`, `keys(object, i = NULL, ...) <- value`: Get or set the key columns. By default, this gets or sets the `keys` slot. Provide `i` to get or set specific keys.

`dropkeys(object, ...)`: Return a `DataFrame` copy of the object without the key columns.

Author(s)

Kylie A. Bemis

See Also

`DataFrame`, `MassDataFrame`, `PositionDataFrame`

Examples

```
## Create an XDataFrame object
XDataFrame(id=1:10, letter=LETTERS[1:10], keys=list(index="id"))
```

Index