

# Package ‘pRoloc’

May 25, 2024

**Type** Package

**Title** A unifying bioinformatics framework for spatial proteomics

**Version** 1.44.0

**Author** Laurent Gatto, Oliver Crook and Lisa M. Breckels with contributions from Thomas Burger and Samuel Wieczorek

**Maintainer** Laurent Gatto <laurent.gatto@uclouvain.be>

**Description** The pRoloc package implements machine learning and visualisation methods for the analysis and interrogation of quantitative mass spectrometry data to reliably infer protein sub-cellular localisation.

**Depends** R (>= 3.5), MSnbase (>= 1.19.20), MLInterfaces (>= 1.67.10), methods, Rcpp (>= 0.10.3), BiocParallel

**Imports** stats4, Biobase, mclust (>= 4.3), caret, e1071, sampling, class, kernlab, lattice, nnet, randomForest, proxy, FNN, hexbin, BiocGenerics, stats, dendextend, RColorBrewer, scales, MASS, knitr, mvtnorm, LaplacesDemon, coda, mixtools, gtools, plyr, ggplot2, biomaRt, utils, grDevices, graphics

**Suggests** testthat, rmarkdown, pRolocdata (>= 1.9.4), roxygen2, xtable, rgl, BiocStyle (>= 2.5.19), hpar (>= 1.41.0), dplyr, akima, fields, vegan, GO.db, AnnotationDbi, Rtsne (>= 0.13), nipals, reshape, magick

**LinkingTo** Rcpp, RcppArmadillo

**License** GPL-2

**VignetteBuilder** knitr

**Video** <https://www.youtube.com/playlist?list=PLvIXxpatSLA2loV5Srs2VBpJIYUIVJ4ow>

**URL** <https://github.com/lgatto/pRoloc>

**BugReports** <https://github.com/lgatto/pRoloc/issues>

**biocViews** ImmunoOncology, Proteomics, MassSpectrometry, Classification, Clustering, QualityControl

**Collate** AllGenerics.R machinelearning-framework.R  
 machinelearning-framework-theta.R  
 machinelearning-framework-map.R  
 machinelearning-framework-mcmc.R machinelearning-utils.R  
 machinelearning-functions-knn.R  
 machinelearning-functions-ksvm.R machinelearning-functions-nb.R  
 machinelearning-functions-nnet.R  
 machinelearning-functions-PerTurbo.R  
 machinelearning-functions-plsda.R  
 machinelearning-functions-rf.R machinelearning-functions-svm.R  
 machinelearning-functions-knntL.R  
 machinelearning-functions-tagm-map.R  
 machinelearning-functions-tagm-mcmc.R  
 machinelearning-functions-tagm-mcmc-helper.R RcppExports.R  
 belief.R distances.R markers.R pRolocmarkers.R chi2.R  
 MLInterfaces.R clustering-framework.R MSnSet.R  
 clustering-kmeans.R perTurbo-algorithm.R phenodisco.R  
 plotting.R plotting2.R plotting3.R plottingBayes.R  
 plotting-ellipse.R hclust.R environment.R utils.R annotation.R  
 goenv.R go.R makeGoSet.R vis.R MartInterface.R dynamics.R zzz.R  
 goannotations.R clusterdist-functions.R clusterdist-framework.R  
 qsep.R

**RoxygenNote** 7.3.1

**git\_url** <https://git.bioconductor.org/packages/pRoloc>

**git\_branch** RELEASE\_3\_19

**git\_last\_commit** a2ae8de

**git\_last\_commit\_date** 2024-04-30

**Repository** Bioconductor 3.19

**Date/Publication** 2024-05-24

## Contents

addGoAnnotations . . . . .	4
addLegend . . . . .	5
addMarkers . . . . .	6
AnnotationParams-class . . . . .	8
checkFeatureNamesOverlap . . . . .	9
checkFvarOverlap . . . . .	10
chi2-methods . . . . .	11
classWeights . . . . .	12
clustDist . . . . .	13
ClustDist-class . . . . .	14
ClustDistList-class . . . . .	16
Deprecated . . . . .	17
empPvalues . . . . .	17
fDataToUnknown . . . . .	18

filterBinMSnSet . . . . .	19
filterMaxMarkers . . . . .	20
filterMinMarkers . . . . .	21
filterZeroCols . . . . .	22
GenRegRes-class . . . . .	22
getGOFromFeatures . . . . .	24
getMarkerClasses . . . . .	25
getMarkers . . . . .	26
getNormDist . . . . .	27
getPredictions . . . . .	28
goldToTerm . . . . .	29
highlightOnPlot . . . . .	30
knnClassification . . . . .	31
knnOptimisation . . . . .	33
knntlClassification . . . . .	34
knntlOptimisation . . . . .	35
ksvmClassification . . . . .	37
ksvmOptimisation . . . . .	38
makeGoSet . . . . .	39
MAPPparams-class . . . . .	40
markerMSnSet . . . . .	43
MartInstance-class . . . . .	44
MCMCChains-class . . . . .	44
mcmc_get_outliers . . . . .	46
minMarkers . . . . .	47
mixing_posterior_check . . . . .	48
MLearn-methods . . . . .	49
move2Ds . . . . .	49
mrkConsProfiles . . . . .	51
mrkHClust . . . . .	52
mrkVecToMat . . . . .	53
nbClassification . . . . .	55
nbOptimisation . . . . .	56
nicheMeans2D . . . . .	57
ndist-methods . . . . .	58
nnetClassification . . . . .	59
nnetOptimisation . . . . .	60
orderGoAnnotations . . . . .	61
orgQuants . . . . .	63
perTurboClassification . . . . .	64
perTurboOptimisation . . . . .	65
phenoDisco . . . . .	67
plot2D . . . . .	69
plot2Ds . . . . .	73
plotConsProfiles . . . . .	75
plotDist . . . . .	76
plotEllipse . . . . .	77
plsdaClassification . . . . .	78

plsdaOptimisation . . . . .	79
pRolocmarkers . . . . .	80
QSep-class . . . . .	81
rfClassification . . . . .	84
rfOptimisation . . . . .	85
sampleMSnSet . . . . .	86
setLisacol . . . . .	87
showGOEvidenceCodes . . . . .	89
spatial2D . . . . .	89
SpatProtVis-class . . . . .	90
subsetMarkers . . . . .	92
svmClassification . . . . .	93
svmOptimisation . . . . .	94
tagmMcmcTrain . . . . .	95
testMarkers . . . . .	97
testMSnSet . . . . .	99
thetas . . . . .	100
undocumented . . . . .	101
zerosInBinMSnSet . . . . .	101

## Index 103

---

addGoAnnotations	<i>Add GO annotations</i>
------------------	---------------------------

---

### Description

Adds GO annotations to the feature data

### Usage

```
addGoAnnotations(
  object,
  params,
  evidence,
  useID = FALSE,
  fcol = "GOAnnotations",
  ...
)
```

### Arguments

object	An instance of class MSnSet.
params	An instance of class AnnotationParams. If missing, <a href="#">getAnnotationParams</a> will be used.
evidence	GO evidence filtering.
useID	Logical. Should GO term names or identifiers be used? If TRUE, identifiers will be used. If FALSE GO term names will be used.

fcol	Character. Name of the matrix of annotations to be added to the fData default is GOAnnotations
...	Other arguments passed to makeGoSet

**Value**

An updated MSnSet with new feature data column called GOAnnotations containing a matrix of GO annotations

**Author(s)**

Lisa M Breckels

**Examples**

```
library(pRolocdata)
data(dunkley2006)
par <- setAnnotationParams(inputs =
  c("Arabidopsis thaliana genes",
    "Gene stable ID"))
## add protein sets/annotation information
xx <- addGoAnnotations(dunkley2006, par)
dim(fData(xx)$GOAnnotations)

## filter sets
xx <- filterMinMarkers(xx, n = 50)
dim(fData(xx)$GOAnnotations)
xx <- filterMaxMarkers(xx, p = .25)
dim(fData(xx)$GOAnnotations)

## Subset for specific protein sets
sub <- subsetMarkers(xx, keep = c("vacuole"))

## Order protein sets
res <- orderGoAnnotations(xx, k = 1:3, p = 1/3, verbose = FALSE)
if (interactive()) {
  pRolocVis(res, fcol = "GOAnnotations")
}
```

---

addLegend

*Adds a legend*

---

**Description**

Adds a legend to a [plot2D](#) figure.

**Usage**

```

addLegend(
  object,
  fcol = "markers",
  where = c("bottomleft", "bottom", "bottomright", "left", "topleft", "top", "topright",
    "right", "center", "other"),
  col,
  bty = "n",
  ...
)

```

**Arguments**

object	An instance of class MSnSet
fcol	Feature meta-data label (fData column name) defining the groups to be differentiated using different colours. Default is markers.
where	One of "bottomleft" (default), "bottomright", "topleft", "topright" or "other" defining the location of the legend. "other" opens a new graphics device, while the other locations are passed to <a href="#">legend</a> .
col	A character defining point colours.
bty	Box type, as in legend. Default is set to "n".
...	Additional parameters passed to <a href="#">legend</a> .

**Details**

The function has been updated in version 1.3.6 to recycle the default colours when more organelle classes are provided. See [plot2D](#) for details.

**Value**

Invisibly returns NULL

**Author(s)**

Laurent Gatto

---

addMarkers	<i>Adds markers to the data</i>
------------	---------------------------------

---

**Description**

The function adds a 'markers' feature variable. These markers are read from a comma separated values (csv) spreadsheet file. This markers file is expected to have 2 columns (others are ignored) where the first is the name of the marker features and the second the group label. Alternatively, a markers named vector as provided by the [pRolocmarkers](#) function can also be used.

## Usage

```
addMarkers(object, markers, mcol = "markers", fcol, verbose = TRUE)
```

## Arguments

object	An instance of class MSnSet.
markers	A character with the name the markers' csv file or a named character of markers as provided by <a href="#">pRolocmarkers</a> .
mcol	A character of length 1 defining the feature variable label for the newly added markers. Default is "markers".
fcol	An optional feature variable to be used to match against the markers. If missing, the feature names are used.
verbose	A logical indicating if number of markers and marker table should be printed to the console.

## Details

It is essential to assure that `featureNames(object)` (or `fcol`, see below) and marker names (first column) match, i.e. the same feature identifiers and case fold are used.

## Value

A new instance of class MSnSet with an additional markers feature variable.

## Author(s)

Laurent Gatto

## See Also

See [pRolocmarkers](#) for a list of spatial markers and [markers](#) for details about markers encoding.

## Examples

```
library("pRolocdata")
data(dunkley2006)
atha <- pRolocmarkers("atha")
try(addMarkers(dunkley2006, atha)) ## markers already exists
fData(dunkley2006)$markers.org <- fData(dunkley2006)$markers
fData(dunkley2006)$markers <- NULL
marked <- addMarkers(dunkley2006, atha)
fvarLabels(marked)
## if 'makers' already exists
marked <- addMarkers(marked, atha, mcol = "markers2")
fvarLabels(marked)
stopifnot(all.equal(fData(marked)$markers, fData(marked)$markers2))
plot2D(marked)
addLegend(marked, where = "topleft", cex = .7)
```

---

 AnnotationParams-class

*Class "AnnotationParams"*


---

## Description

Class to store annotation parameters to automatically query a Biomart server, retrieve relevant annotation for a set of features of interest using, for example [getGOFromFeatures](#) and [makeGoSet](#).

## Objects from the Class

Objects can be created and set with the `setAnnotationParams` function. Object are created by calling without any arguments `setAnnotationParams()`, which will open an interactive interface. Depending on the value of `"many.graphics"` option, a graphical of a text-based menu will open (the text interface can be forced by setting the `graphics` argument to `FALSE`: `setAnnotationParams(graphics = FALSE)`). The menu will allow to select the species of interest first and the type of features (ENSEMBL gene identifier, Entrez id, ...) second.

The species that are available are those for which ENSEMBL data is available in Biomart and have a set of attributes of interest available. The compatible identifiers for downstream queries are then automatically filtered and displayed for user selection.

It is also possible to pass a parameter `inputs`, a character vector of length 2 containing a pattern uniquely matching the species of interest (in position 1) and a patterns uniquely matching the feature types (in position 2). If the matches are not unique, an error will be thrown.

A new instance of the `AnnotationParams` will be created to enable easy and automatic query of the `Mart` instance. The instance is invisibly returned and stored in a global variable in the **pRoloc** package's private environment for automatic retrieval. If a variable containing an `AnnotationParams` instance is already available, it can be set globally by passing it as argument to the `setAnnotationParams` function. Globally set `AnnotationParams` instances can be accessed with the `getAnnotationParams` function.

See the `pRoloc-theta` vignette for details.

## Slots

**mart**: Object of class `"Mart"` from the **biomaRt** package.

**martname**: Object of class `"character"` with the name of the `mart` instance.

**dataset**: Object of class `"character"` with the data set of the `mart` instance.

**filter**: Object of class `"character"` with the filter to be used when querying the `mart` instance.

**date**: Object of class `"character"` indicating when the current instance was created.

**biomaRtVersion**: Object of class `"character"` with the **biomaRt** version used to create the `AnnotationParams` instance.

**.\_\_classVersion\_\_**: Object of class `"Versions"` with the version of the `AnnotationParams` class of the current instance.



**Methods**

`show` signature(object = "AnnotationParams"): to display objects.

**Author(s)**

Laurent Gatto <lg390@cam.ac.uk>

**See Also**

[getGOFromFeatures](#), [makeGoSet](#) and the pRoloc-theta vignette.

**Examples**

```
data(andy2011params)
andy2011params
data(dunkley2006params)
dunkley2006params

try(setAnnotationParams(inputs = c("nomatch1", "nomatch2")))
setAnnotationParams(inputs = c("Human genes",
  "UniProtKB/Swiss-Prot ID"))
getAnnotationParams()
```

---

checkFeatureNamesOverlap

*Check feature names overlap*

---

**Description**

Checks the marker and unknown feature overlap of two MSnSet instances.

**Usage**

```
checkFeatureNamesOverlap(x, y, fcolx = "markers", fcoly, verbose = TRUE)
```

**Arguments**

x	An MSnSet instance.
y	An MSnSet instance.
fcolx	The feature variable to separate unknown (fData(y)\$coly == "unknown") from the marker features in the x object.
fcoly	As fcolx, for the y object. If missing, the value of fcolx is used.
verbose	If TRUE (default), the overlap is printed out on the console.

**Value**

Invisibly returns a named list of common markers, unique x markers, unique y markers in, common unknowns, unique x unknowns and unique y unknowns.

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(andy2011)
data(andy2011goCC)
checkFeatureNamesOverlap(andy2011, andy2011goCC)
featureNames(andy2011goCC)[1] <- "ABC"
res <- checkFeatureNamesOverlap(andy2011, andy2011goCC)
res$markersX
res$markersY
```

---

`checkFvarOverlap`*Compare a feature variable overlap*

---

**Description**

Extracts qualitative feature variables from two MSnSet instances and compares with a contingency table.

**Usage**

```
checkFvarOverlap(x, y, fcolx = "markers", fcoly, verbose = TRUE)
```

**Arguments**

<code>x</code>	An MSnSet instance.
<code>y</code>	An MSnSet instance.
<code>fcolx</code>	The feature variable to separate unknown ( <code>fData(y)\$coly == "unknown"</code> ) from the marker features in the <code>x</code> object.
<code>fcoly</code>	As <code>fcolx</code> , for the <code>y</code> object. If missing, the value of <code>fcolx</code> is used.
<code>verbose</code>	If TRUE (default), the contingency table of the the feature variables is printed out.

**Value**

Invisibly returns a named list with the values of the diagonal, upper and lower triangles of the contingency table.

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(dunkley2006)
res <- checkFvarOverlap(dunkley2006, dunkley2006,
                        "markers", "markers.orig")
str(res)
```

chi2-methods

*The PCP 'chi square' method***Description**

In the original protein correlation profiling (PCP), Andersen et al. use the peptide normalised profiles along gradient fractions and compared them with the reference profiles (or set of profiles) by computing  $Chi^2$  values,  $\frac{\sum(x_i - x_p)^2}{x_p}$ , where  $x_i$  is the normalised value of the peptide in fraction  $i$  and  $x_p$  is the value of the marker (from Wiese et al., 2007). The protein  $Chi^2$  is then computed as the median of the peptide  $Chi^2$  values. Peptides and proteins with similar profiles to the markers will have small  $Chi^2$  values.

The chi2 methods implement this idea and compute such  $Chi^2$  values for sets of proteins.

**Methods**

signature(x = "matrix", y = "matrix", method = "character", fun = "NULL", na.rm = "logical")  
 Compute nrow(x) times nrow(y)  $Chi^2$  values, for each x, y feature pair. Method is one of "Andersen2003" or "Wiese2007"; the former (default) computed the  $Chi^2$  as  $\sum(y-x)^2/\text{length}(x)$ , while the latter uses  $\sum((y-x)^2/x)$ . na.rm defines if missing values (NA and NaN) should be removed prior to summation. fun defines how to summarise the  $Chi^2$  values; default, NULL, does not combine the  $Chi^2$  values.

signature(x = "matrix", y = "numeric", method = "character", na.rm = "logical") Computes nrow(x)  $Chi^2$  values, for all the  $(x_i, y)$  pairs. See above for the other arguments.

signature(x = "numeric", y = "matrix", method = "character", na.rm = "logical") Computes nrow(y)  $Chi^2$  values, for all the  $(x, y_i)$  pairs. See above for the other arguments.

signature(x = "numeric", y = "numeric", method = "character", na.rm = "logical") Computes the  $Chi^2$  value for the  $(x, y)$  pairs. See above for the other arguments.

**Author(s)**

Laurent Gatto <lg390@cam.ac.uk>

**References**

Andersen, J. S., Wilkinson, C. J., Mayor, T., Mortensen, P. et al., Proteomic characterization of the human centrosome by protein correlation profiling. *Nature* 2003, 426, 570 - 574.

Wiese, S., Gronemeyer, T., Ofman, R., Kunze, M. et al., Proteomics characterization of mouse kidney peroxisomes by tandem mass spectrometry and protein correlation profiling. *Mol. Cell. Proteomics* 2007, 6, 2045 - 2057.

**See Also**[empPvalues](#)**Examples**

```
mrk <- rnorm(6)
prot <- matrix(rnorm(60), ncol = 6)
chi2(mrk, prot, method = "Andersen2003")
chi2(mrk, prot, method = "Wiese2007")
```

```
pepmark <- matrix(rnorm(18), ncol = 6)
pepprot <- matrix(rnorm(60), ncol = 6)
chi2(pepmark, pepprot)
chi2(pepmark, pepprot, fun = sum)
```

---

`classWeights`*Calculate class weights*

---

**Description**

Calculates class weights to be used for parameter optimisation and classification such as [svmOptimisation](#) or [svmClassification](#) - see the *pRoloc tutorial* vignette for an example. The weights are calculated for all non-*unknown* classes the inverse of the number of observations.

**Usage**

```
classWeights(object, fcol = "markers")
```

**Arguments**

<code>object</code>	An instance of class MSnSet
<code>fcol</code>	The name of the features to be weighted

**Value**

A table of class weights

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(hyperLOPIT2015)
classWeights(hyperLOPIT2015)
data(dunkley2006)
classWeights(dunkley2006)
```

---

clustDist *Pairwise Distance Computation for Protein Information Sets*

---

### Description

This function computes the mean (normalised) pairwise distances for pre-defined sets of proteins.

### Usage

```
clustDist(object, k = 1:5, fcol = "GOAnnotations", n = 5, verbose = TRUE, seed)
```

### Arguments

object	An instance of class " <i>MSnSet</i> ".
k	The number of clusters to try fitting to the protein set. Default is $k = 1:5$ .
fcol	The feature meta-data containing matrix of protein sets/ marker definitions. Default is <i>GOAnnotations</i> .
n	The minimum number of proteins per set. If protein sets contain less than n instances they will be ignored. Default is 5.
verbose	A logical defining whether a progress bar is displayed.
seed	An optional seed for the random number generator.

### Details

The input to the function is a *MSnSet* dataset containing a matrix appended to the feature data slot identifying the membership of protein instances to a pre-defined set(s) e.g. a specific Gene Ontology term etc.

For each protein set, the *clustDist* function (i) extracts all instances belonging to the set, (ii) using the *kmeans* algorithm fits and tests  $k = c(1:5)$  (default) cluster components to each set, (iii) calculates the mean pairwise distance for each *k* tested.

Note: currently distances are calculated in Euclidean space, but other distance metrics will be supported in the future).

The output is a list of *ClustDist* objects, one per information cluster. The *ClustDist* class summarises the algorithm information such as the number of *k*'s tested for the *kmeans*, and mean and normalised pairwise Euclidean distances per number of component clusters tested. See `?ClustDist` for more details.

### Value

An instance of "*ClustDistList*" containing a "*ClustDist*" instance for every protein set, which summarises the algorithm information such as the number of *k*'s tested for the *kmeans*, and mean and normalised pairwise Euclidean distances per number of component clusters tested.

### Author(s)

Lisa Breckels

**See Also**

For class definitions see "[ClustDistList](#)" and "[ClustDist](#)".

**Examples**

```
library(pRolocdata)
data(dunkley2006)
par <- setAnnotationParams(inputs =
                           c("Arabidopsis thaliana genes",
                              "Gene stable ID"))
## add protein sets/annotation information
xx <- addGoAnnotations(dunkley2006, par)
## filter
xx <- filterMinMarkers(xx, n = 50)
xx <- filterMaxMarkers(xx, p = .25)
## get distances for protein sets
dd <- clustDist(xx)
## plot clusters for first 'ClustDist' object
## in the 'ClustDistList'
plot(dd[[1]], xx)
## plot distances for all protein sets
plot(dd)
## Extract normalised distances
## Normalise by n1/3
minDist <- getNormDist(dd, p = 1/3)
## Get new order according to lowest distance
o <- order(minDist)
## Re-order GOAnnotations
fData(xx)$GOAnnotations <- fData(xx)$GOAnnotations[, o]
if (interactive()) {
  pRolocVis(xx, fcol = "GOAnnotations")
}
```

---

ClustDist-class

*Class "ClustDist"*

---

**Description**

The ClustDist summaries algorithm information, from running the clustDist function, such as the number of k's tested for the kmeans, and mean and normalised pairwise (Euclidean) distances per number of component clusters tested.

**Objects from the Class**

Object of this class are created with the clustDist function.

**Slots**

**k:** Object of class "numeric" storing the number of k clusters tested.

**dist:** Object of class "list" storing the list of distance matrices.

**term:** Object of class "character" describing GO term name.

**id:** Object of class "character" describing the GO term ID.

**nrow:** Object of class "numeric" showing the number of instances in the set

**clustsz:** Object of class "list" describing the number of instances for each cluster for each k tested

**components:** Object of class "vector" storing the class membership of each protein for each k tested.

**fcol:** Object of class "character" showing the feature column name in the corresponding MSnSet where the protein set information is stored.

**Methods**

**plot** Plots the kmeans clustering results.

**show** Shows the object.

**Author(s)**

Lisa M Breckels <lms79@cam.ac.uk>

**Examples**

```
showClass("ClustDist")

library('pRolocdata')
data(dunkley2006)
par <- setAnnotationParams(inputs =
  c("Arabidopsis thaliana genes",
    "Gene stable ID"))

## add protein set/annotation information
xx <- addGoAnnotations(dunkley2006, par)

## filter
xx <- filterMinMarkers(xx, n = 50)
xx <- filterMaxMarkers(xx, p = .25)

## get distances for protein sets
dd <- clustDist(xx)

## plot clusters for first 'ClustDist' object
## in the 'ClustDistList'
plot(dd[[1]], xx)

## plot distances for all protein sets
plot(dd)
```

---

ClustDistList-class    *Storing multiple ClustDist instances*

---

### Description

A class for storing lists of [ClustDist](#) instances.

### Objects from the Class

Object of this class are created with the `clustDist` function.

### Slots

**x:** Object of class `list` containing valid `ClustDist` instances.

**log:** Object of class `list` containing an object creation log, containing among other elements the call that generated the object.

**.\_\_classVersion\_\_:** The version of the instance. For development purposes only.

### Methods

`"["` Extracts a single `ClustDist` at position.

`"["` Extracts one of more `ClustDists` as `ClustDistList`.

`length` Returns the number of `ClustDists`.

`names` Returns the names of `ClustDists`, if available. The replacement method is also available.

`show` Display the object by printing a short summary.

`lapply(x, FUN, ...)` Apply function `FUN` to each element of the input `x`. If the application of `FUN` returns and `ClustDist`, then the return value is an `ClustDistList`, otherwise a `list`.

`plot` Plots a boxplot of the distance results per protein set.

### Author(s)

Lisa M Breckels <lms79@cam.ac.uk>

### Examples

```
library('pRolocdata')
data(dunkley2006)
par <- setAnnotationParams(inputs =
  c("Arabidopsis thaliana genes",
    "Gene stable ID"))

## add protein set/annotation information
xx <- addGoAnnotations(dunkley2006, par)

## filter
xx <- filterMinMarkers(xx, n = 50)
```



```

xx <- filterMaxMarkers(xx, p = .25)

## get distances for protein sets
dd <- clustDist(xx)

## plot distances for all protein sets
plot(dd)

names(dd)

## Extract first 4 ClustDist objects of the ClustDistList
dd[1:4]

## Extract 1st ClustDist object
dd[[1]]

```

---

Deprecated *pRoloc* *Deprecated and Defunct*

---

### Description

The function, class, or data object you have asked for has been deprecated or made defunct.

Deprecated: `minClassScore`; use the replacement [getPredictions](#)

Defunct:

Deprecated functions are provided for compatibility with older versions of the `pRoloc` package only, and will be defunct at the next release.

---

`empPvalues` *Estimate empirical p-values for  $Chi^2$  protein correlations.*

---

### Description

Andersen et al. (2003) used a fixed  $Chi^2$  threshold of 0.05 to identify organelle-specific candidates. This function computes empirical p-values by permutation the markers relative intensities and computed null  $Chi^2$  values.

### Usage

```
empPvalues(marker, corMatrix, n = 100, ...)
```

### Arguments

<code>marker</code>	A numerics with markers relative intensities.
<code>corMatrix</code>	A matrix of <code>nrow(corMatrix)</code> protein relative intensities to be compares against the marker.
<code>n</code>	The number of iterations.
<code>...</code>	Additional parameters to be passed to <code>chi2</code> .

**Value**

A numeric of length `nrow(corMatrix)`.

**Author(s)**

Laurent Gatto <lg390@cam.ac.uk>

**References**

Andersen, J. S., Wilkinson, C. J., Mayor, T., Mortensen, P. et al., Proteomic characterization of the human centrosome by protein correlation profiling. *Nature* 2003, 426, 570 - 574.

**See Also**

[chi2](#) for  $Chi^2$  calculation.

**Examples**

```
set.seed(1)
mrk <- rnorm(6, 5, 1)
prot <- rbind(matrix(rnorm(120, 5, 1), ncol = 6),
               mrk + rnorm(6))
mrk <- mrk/sum(mrk)
prot <- prot/rowSums(prot)
empPvalues(mrk, prot)
```

---

fDataToUnknown

*Update a feature variable*

---

**Description**

This function replaces a string or regular expression in a feature variable using the [sub](#) function.

**Usage**

```
fDataToUnknown(object, fcol = "markers", from = "^$", to = "unknown", ...)
```

**Arguments**

<code>object</code>	An instance of class <code>MSnSet</code> .
<code>fcol</code>	Feature variable to be modified. Default is "markers". If <code>NULL</code> , all feature variables will be updated.
<code>from</code>	A character defining the string or regular expression of the pattern to be replaced. Default is the empty string, i.e. the regular expression <code>"^\$"</code> . See <a href="#">sub</a> for details. If <code>NA</code> , then <code>NA</code> values are replaced by <code>to</code> .
<code>to</code>	A replacement for matched pattern. Default is "unknown". See <a href="#">sub</a> for details.
<code>...</code>	Additional arguments passed to <a href="#">sub</a> .

**Value**

An updated MSnSet.

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(dunkley2006)
getMarkers(dunkley2006, "markers")
dunkley2006 <- fDataToUnknown(dunkley2006,
                             from = "unknown", to = "unassigned")
getMarkers(dunkley2006, "markers")
```

---

filterBinMSnSet	<i>Filter a binary MSnSet</i>
-----------------	-------------------------------

---

**Description**

Removes columns or rows that have a certain proportion or absolute number of 0 values.

**Usage**

```
filterBinMSnSet(object, MARGIN = 2, t, q, verbose = TRUE)
```

**Arguments**

object	An MSnSet
MARGIN	1 or 2. Default is 2.
t	Rows/columns that have t or less 1s, it will be filtered out. When t and q are missing, default is to use t = 1.
q	If a row has a higher quantile than defined by q, it will be filtered out.
verbose	A logical defining of a message is to be printed. Default is TRUE.

**Value**

A filtered MSnSet.

**Author(s)**

Laurent Gatto

**See Also**

[zerosInBinMSnSet](#), [filterZeroCols](#), [filterZeroRows](#).

**Examples**

```

set.seed(1)
m <- matrix(sample(0:1, 25, replace=TRUE), 5)
m[1, ] <- 0
m[, 1] <- 0
rownames(m) <- colnames(m) <- letters[1:5]
fd <- data.frame(row.names = letters[1:5])
x <- MSnSet(exprs = m, fData = fd, pData = fd)
exprs(x)
## Remove columns with no 1s
exprs(filterBinMSnSet(x, MARGIN = 2, t = 0))
## Remove columns with one 1 or less
exprs(filterBinMSnSet(x, MARGIN = 2, t = 1))
## Remove columns with two 1s or less
exprs(filterBinMSnSet(x, MARGIN = 2, t = 2))
## Remove columns with three 1s
exprs(filterBinMSnSet(x, MARGIN = 2, t = 3))
## Remove columns that have half or less of 1s
exprs(filterBinMSnSet(x, MARGIN = 2, q = 0.5))

```

---

filterMaxMarkers	<i>Removes class/annotation information from a matrix of candidate markers that appear in the fData.</i>
------------------	--

---

**Description**

Removes annotation information that contain more that a certain number/percentage of proteins

**Usage**

```
filterMaxMarkers(object, n, p = 0.2, fcol = "GOAnnotations", verbose = TRUE)
```

**Arguments**

object	An instance of class MSnSet.
n	Maximum number of proteins allowed per class/information term.
p	Maximum percentage of proteins per column. Default is 0.2 i.e. remove columns that have information for greater than 20 of the total number of proteins in the dataset (note: this is useful for example, if information is GO terms, for removing very general and uninformative terms).
fcol	The name of the matrix of marker information. Default is GOAnnotations.
verbose	Number of marker candidates retained after filtering.

**Value**

An updated MSnSet

**See Also**

addGoAnnotations and example therein.

---

filterMinMarkers	<i>Removes class/annotation information from a matrix of candidate markers that appear in the fData.</i>
------------------	--

---

**Description**

Removes annotation information that contain less than a certain number/percentage of proteins

**Usage**

```
filterMinMarkers(object, n = 10, p, fcol = "GOAnnotations", verbose = TRUE)
```

**Arguments**

object	An instance of class MSnSet.
n	Minimum number of proteins allowed per column. Default is 10.
p	Minimum percentage of proteins per column.
fcol	The name of the matrix of marker information. Default is GOAnnotations.
verbose	Number of marker candidates retained after filtering.

**Value**

An updated MSnSet.

**Author(s)**

Lisa M Breckels

**See Also**

addGoAnnotations and example therein.

---

filterZeroCols	<i>Remove 0 columns/rows</i>
----------------	------------------------------

---

**Description**

Removes all assay data columns/rows that are composed of only 0, i.e. have a colSum/rowSum of 0.

**Usage**

```
filterZeroCols(object, verbose = TRUE)
```

```
filterZeroRows(object, verbose = TRUE)
```

**Arguments**

object	A MSnSet object.
verbose	Print a message with the number of filtered out columns/row (if any).

**Value**

An MSnSet.

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(andy2011goCC)
any(colSums(exprs(andy2011goCC)) == 0)
exprs(andy2011goCC)[, 1:5] <- 0
ncol(andy2011goCC)
ncol(filterZeroCols(andy2011goCC))
```

---

GenRegRes-class	<i>Class "GenRegRes" and "ThetaRegRes"</i>
-----------------	--

---

**Description**

Regularisation framework containers.

**Objects from the Class**

Object of this class are created with the respective regularisation function: [knnOptimisation](#), [svmOptimisation](#), [plsdaOptimisation](#), [knnt1Optimisation](#), ...

**Slots**

**algorithm:** Object of class "character" storing the machine learning algorithm name.

**hyperparameters:** Object of class "list" with the respective algorithm hyper-parameters tested.

**design:** Object of class "numeric" describing the cross-validation design, the test data size and the number of replications.

**log:** Object of class "list" with warnings thrown during the hyper-parameters regularisation.

**seed:** Object of class "integer" with the random number generation seed.

**results:** Object of class "matrix" of dimensions times (see design) by number of hyperparameters + 1 storing the macro F1 values for the respective best hyper-parameters for each replication.

**f1Matrices:** Object of class "list" with respective times cross-validation F1 matrices.

**cmMatrices:** Object of class "list" with respective times contingency matrices.

**testPartitions:** Object of class "list" with respective times test partitions.

**datasize:** Object of class "list" with details about the respective inner and outer training and testing data sizes.

Only in ThetaRegRes:

**predictions:** A list of predictions for the optimisation iterations.

**otherWeights:** Alternative best theta weights: a vector per iterations, NULL if no other best weights were found.

**Methods**

**getF1Scores** Returns a matrix of F1 scores for the optimisation parameters.

**f1Count** signature(object = "GenRegRes", t = "numeric") and signature(object = "ThetaRegRes", t = "numeric"): Constructs a table of all possible parameter combination and count how many have an F1 scores greater or equal than t. When t is missing (default), the best F1 score is used. This method is useful in conjunctin with plot.

**getParams** Returns the *best* parameters. It is however strongly recommended to inspect the optimisation results. For a ThetaRegRes optimisation result, the method to chose the best parameters can be "median" (default) or "mean" (the median or mean of the best weights is chosen), "max" (the first weights with the highest macro-F1 score, considering that multiple max scoring combinations are possible) or "count" (the observed weight that get the maximum number of observations, see f1Count). The favourP argument can be used to prioritise weights that favour the primary data (i.e. heigh weights). See favourPrimary below.

**getSeed** Returns the seed used for the optimisation run.

**getWarnings** signature(object = "GenRegRes"): Returns a vector of recorded warnings.

**levelPlot** signature(object = "GenRegRes"): Plots a heatmap of of the optimisation results. Only for "GenRegRes" instances.

**plot** Plots the optisation results.

**show** Shows the object.

**Other functions**

Only for ThetaRegRes:

`combineThetaRegRes(object)` Takes a list of ThetaRegRes instances to be combined and returns a new ThetaRegRes instance.

`favourPrimary(primary, auxiliary, object, verbose = TRUE)` Takes the primary and auxiliary data sources (two [MSnSet](#) instances) and a ThetaRegRes object and returns an updated ThetaRegRes instance containing best parameters/weights (see the `getParams` function) favouring the primary data when multiple best theta weights are available.

**Author(s)**

Laurent Gatto <lg390@cam.ac.uk>

**Examples**

```
showClass("GenRegRes")
showClass("ThetaRegRes")
```

---

`getGOFromFeatures`      *Retrieve GO terms for feature names*

---

**Description**

The function pulls the gene ontology (GO) terms for a set of feature names.

**Usage**

```
getGOFromFeatures(
  id,
  namespace = "cellular_component",
  evidence = NULL,
  params = NULL,
  verbose = FALSE,
  nmax = 500
)
```

**Arguments**

<code>id</code>	An character with feature names to be pulled from biomart. If an <code>MSnSet</code> is provided, then <code>featureNames(id)</code> is used.
<code>namespace</code>	The GO namespace. One of <code>biological_process</code> , <code>cellular_component</code> (default) or <code>molecular_function</code> .
<code>evidence</code>	The GO evidence code. See <code>showGOEvidenceCodes</code> for details. If <code>NULL</code> (default), no filtering based on the evidence code is performed.
<code>params</code>	An instance of class " <a href="#">AnnotationParams</a> ".



`verbose` A logical defining verbosity of the function. Default is FALSE.

`nmax` As described in <https://support.bioconductor.org/p/86358/>, the Biomart result can be unreliable for large queries. This argument splits the input in chunks of length `nmax` (default is 500). If set to NULL, the query is performed in full.

### Value

A data.frame with relevant GO terms.

### Author(s)

Laurent Gatto

### Examples

```
library(pRolocdata)
data(dunkley2006)
data(dunkley2006params)
dunkley2006params
fn <- featureNames(dunkley2006)[1:5]
getGOFromFeatures(fn, params = dunkley2006params)
```

---

`getMarkerClasses`      *Returns the organelle classes in an 'MSnSet'*

---

### Description

Convenience accessor to the organelle classes in an 'MSnSet'. This function returns the organelle classes of an MSnSet instance. As a side effect, it prints out the classes.

### Usage

```
getMarkerClasses(object, fcol = "markers", ...)
```

### Arguments

`object` An instance of class "MSnSet".

`fcol` The name of the markers column in the featureData slot. Default is markers.

`...` Additional parameters passed to sort from the base package.

### Value

A character vector of the organelle classes in the data.

### Author(s)

Lisa Breckels and Laurent Gatto

**See Also**

[getMarkers](#) to extract the marker proteins. See [markers](#) for details about spatial markers storage and encoding.

**Examples**

```
library("pRolocdata")
data(dunkley2006)
organelles <- getMarkerClasses(dunkley2006)
## same if markers encoded as a matrix
dunkley2006 <- mrkVecToMat(dunkley2006, mfc01 = "Markers")
organelles2 <- getMarkerClasses(dunkley2006, fcol = "Markers")
stopifnot(all.equal(organelles, organelles2))
```

---

getMarkers

*Get the organelle markers in an MSnSet*


---

**Description**

Convenience accessor to the organelle markers in an MSnSet. This function returns the organelle markers of an MSnSet instance. As a side effect, it print out a marker table.

**Usage**

```
getMarkers(object, fcol = "markers", names = TRUE, verbose = TRUE)
```

**Arguments**

object	An instance of class " <a href="#">MSnSet</a> ".
fcol	The name of the markers column in the featureData slot. Default is "markers".
names	A logical indicating if the markers vector should be named. Ignored if markers are encoded as a matrix.
verbose	If TRUE, a marker table is printed and the markers are returned invisibly. If FALSE, the markers are returned.

**Value**

A character (matrix) of length (ncol) ncol(object), depending on the vector or matrix encoding of the markers.

**Author(s)**

Laurent Gatto

**See Also**

See [getMarkerClasses](#) to get the classes only. See [markers](#) for details about spatial markers storage and encoding.

## Examples

```
library("pRolocdata")
data(dunkley2006)
## marker vectors
myVmarkers <- getMarkers(dunkley2006)
head(myVmarkers)
## marker matrix
dunkley2006 <- mrkVecToMat(dunkley2006, mfc01 = "Markers")
myMmarkers <- getMarkers(dunkley2006, fcol = "Markers")
head(myMmarkers)
```

---

getNormDist	<i>Extract Distances from a "ClustDistList" object</i>
-------------	--

---

## Description

This function computes and outputs normalised distances from a `"ClustDistList"` object.

## Usage

```
getNormDist(object, p = 1/3)
```

## Arguments

object	An instance of class <code>"ClustDistList"</code> .
p	The normalisation factor. Default is 1/3.

## Value

An numeric of normalised distances, one per protein set in the `ClustDistList`.

## Author(s)

Lisa Breckels

## See Also

`"ClustDistList"`, `"ClustDist"`, and examples in `clustDist`.

---

getPredictions	Returns the predictions in an 'MSnSet'
----------------	--

---

### Description

Convenience accessor to the predicted feature localisation in an 'MSnSet'. This function returns the predictions of an MSnSet instance. As a side effect, it prints out a prediction table.

### Usage

```
getPredictions(object, fcol, scol, mcol = "markers", t = 0, verbose = TRUE)
```

### Arguments

object	An instance of class "MSnSet".
fcol	The name of the prediction column in the featureData slot.
scol	The name of the prediction score column in the featureData slot. If missing, created by pasting '.scores' after fcol.
mcol	The feature meta data column containing the labelled training data.
t	The score threshold. Predictions with score < t are set to 'unknown'. Default is 0. It is also possible to define thresholds for each prediction class, in which case, t is a named numeric with names exactly matching the unique prediction class names.
verbose	If TRUE, a prediction table is printed and the predictions are returned invisibly. If FALSE, the predictions are returned.

### Value

An instance of class "MSnSet" with fcol.pred feature variable storing the prediction results according to the chosen threshold.

### Author(s)

Laurent Gatto and Lisa Breckels

### See Also

[orgQuants](#) for calculating organelle-specific thresholds.

### Examples

```
library("pRolocdata")
data(dunkley2006)
res <- svmClassification(dunkley2006, fcol = "pd.markers",
                        sigma = 0.1, cost = 0.5)
fData(res)$svm[500:510]
fData(res)$svm.scores[500:510]
```

```

getPredictions(res, fcol = "svm", t = 0) ## all predictions
getPredictions(res, fcol = "svm", t = .9) ## single threshold
## 50% top predictions per class
ts <- orgQuants(res, fcol = "svm", t = .5)
getPredictions(res, fcol = "svm", t = ts)

```

---

goIdToTerm

*Convert GO ids to/from terms*


---

### Description

Converts GO identifiers to/from GO terms, either explicitly or by checking if (any items in) the input contains "GO:".

### Usage

```

goIdToTerm(x, names = TRUE, keepNA = TRUE)

goTermToId(x, names = TRUE, keepNA = TRUE)

flipGoTermId(x, names = TRUE, keepNA = TRUE)

prettyGoTermId(x)

```

### Arguments

x	A character of GO ids or terms.
names	Should a named character be returned? Default is TRUE.
keepNA	Should any GO term/id names that are missing or obsolete be replaced with a NA? Default is TRUE. If FALSE then the GO term/id names is kept.

### Value

A character of GO terms (ids) if x were ids (terms).

### Author(s)

Laurent Gatto

### Examples

```

goIdToTerm("GO:0000001")
goIdToTerm("GO:0000001", names = FALSE)
goIdToTerm(c("GO:0000001", "novalid"))
goIdToTerm(c("GO:0000001", "GO:0000002", "notvalid"))
goTermToId("mitochondrion inheritance")
goTermToId("mitochondrion inheritance", name = FALSE)
goTermToId(c("mitochondrion inheritance", "notvalid"))

```

```
prettyGoTermId("mitochondrion inheritance")
prettyGoTermId("GO:0000001")
flipGoTermId("mitochondrion inheritance")
flipGoTermId("GO:0000001")
flipGoTermId("GO:0000001", names = FALSE)
```

---

highlightOnPlot      *Highlight features of interest on a spatial proteomics plot*

---

### Description

Highlights a set of features of interest given as a `FeaturesOfInterest` instance on a PCA plot produced by `plot2D` or `plot3D`. If none of the features of interest are found in the `MSnset`'s `featureNames`, an warning is thrown.

### Usage

```
highlightOnPlot(object, foi, labels, args = list(), ...)
```

```
highlightOnPlot3D(object, foi, labels, args = list(), radius = 0.1 * 3, ...)
```

### Arguments

<code>object</code>	The main dataset described as an <code>MSnSet</code> or a <code>matrix</code> with the coordinates of the features on the PCA plot produced (and invisibly returned) by <code>plot2D</code> .
<code>foi</code>	An instance of <code>FeaturesOfInterest</code> , or, alternatively, a character of feature names.
<code>labels</code>	A character of length 1 with a feature variable name to be used to label the features of interest. This is only valid if <code>object</code> is an <code>MSnSet</code> . Alternatively, if <code>TRUE</code> , then <code>featureNames(object)</code> (or <code>rownames(object)</code> , if <code>object</code> is a <code>matrix</code> ) are used. Default is missing, which does not add any labels.
<code>args</code>	A named list of arguments to be passed to <code>plot2D</code> if the PCA coordinates are to be calculated. Ignored if the PCA coordinates are passed directly, i.e. <code>object</code> is a <code>matrix</code> .
<code>...</code>	Additional parameters passed to <code>points</code> or <code>text</code> (when <code>labels</code> is <code>TRUE</code> ) when adding to <code>plot2D</code> , or <code>spheres3d</code> or <code>text3d</code> when adding the <code>plot3D</code> .
<code>radius</code>	Radius of the spheres to be added to the visualisation produced by <code>plot3D</code> . Default is 0.3 (i.e. <code>plot3D</code> 's <code>radius1 * 3</code> ), to emphasise the features with regard to unknown ( <code>radius1 = 0.1</code> ) and marker ( <code>radius1 * 2</code> ) features.

### Value

NULL; used for its side effects.

### Author(s)

Laurent Gatto

**Examples**

```

library("pRolocdata")
data("tan2009r1")
x <- FeaturesOfInterest(description = "A test set of features of interest",
                        fnames = featureNames(tan2009r1)[1:10],
                        object = tan2009r1)

## using FeaturesOfInterest or feature names
par(mfrow = c(2, 1))
plot2D(tan2009r1)
highlightOnPlot(tan2009r1, x)
plot2D(tan2009r1)
highlightOnPlot(tan2009r1, featureNames(tan2009r1)[1:10])

.pca <- plot2D(tan2009r1)
head(.pca)
highlightOnPlot(.pca, x, col = "red")
highlightOnPlot(tan2009r1, x, col = "red", cex = 1.5)
highlightOnPlot(tan2009r1, x, labels = TRUE)

.pca <- plot2D(tan2009r1, dims = c(1, 3))
highlightOnPlot(.pca, x, pch = "+", dims = c(1, 3))
highlightOnPlot(tan2009r1, x, args = list(dims = c(1, 3)))

.pca2 <- plot2D(tan2009r1, mirrorX = TRUE, dims = c(1, 3))
## previous pca matrix, need to mirror X axis
highlightOnPlot(.pca, x, pch = "+", args = list(mirrorX = TRUE))
## new pca matrix, with X mirrors (and 1st and 3rd PCs)
highlightOnPlot(.pca2, x, col = "red")

plot2D(tan2009r1)
highlightOnPlot(tan2009r1, x)
highlightOnPlot(tan2009r1, x, labels = TRUE, pos = 3)
highlightOnPlot(tan2009r1, x, labels = "Flybase.Symbol", pos = 1)

## in 3 dimensions
if (interactive()) {
  plot3D(tan2009r1, radius1 = 0.05)
  highlightOnPlot3D(tan2009r1, x, labels = TRUE)
  highlightOnPlot3D(tan2009r1, x)
}

```

---

knnClassification      *knn classification*

---

**Description**

Classification using for the k-nearest neighbours algorithm.

**Usage**

```
knnClassification(
  object,
  assessRes,
  scores = c("prediction", "all", "none"),
  k,
  fcol = "markers",
  ...
)
```

**Arguments**

object	An instance of class <code>"MSnSet"</code> .
assessRes	An instance of class <code>"GenRegRes"</code> , as generated by <code>knnOptimisation</code> .
scores	One of <code>"prediction"</code> , <code>"all"</code> or <code>"none"</code> to report the score for the predicted class only, for all classes or none.
k	If <code>assessRes</code> is missing, a <code>k</code> must be provided.
fcol	The feature meta-data containing marker definitions. Default is <code>markers</code> .
...	Additional parameters passed to <code>knn</code> from package <code>class</code> .

**Value**

An instance of class `"MSnSet"` with `knn` and `knn.scores` feature variables storing the classification results and scores respectively.

**Author(s)**

Laurent Gatto

**Examples**

```
library(pRolocdata)
data(dunkley2006)
## reducing parameter search space and iterations
params <- knnOptimisation(dunkley2006, k = c(3, 10), times = 3)
params
plot(params)
f1Count(params)
levelPlot(params)
getParams(params)
res <- knnClassification(dunkley2006, params)
getPredictions(res, fcol = "knn")
getPredictions(res, fcol = "knn", t = 0.75)
plot2D(res, fcol = "knn")
```



---

knnOptimisation	<i>knn parameter optimisation</i>
-----------------	-----------------------------------

---

**Description**

Classification parameter optimisation for the k-nearest neighbours algorithm.

**Usage**

```
knnOptimisation(
  object,
  fcol = "markers",
  k = seq(3, 15, 2),
  times = 100,
  test.size = 0.2,
  xval = 5,
  fun = mean,
  seed,
  verbose = TRUE,
  ...
)
```

**Arguments**

object	An instance of class <code>"MSnSet"</code> .
fcol	The feature meta-data containing marker definitions. Default is markers.
k	The hyper-parameter. Default values are seq(3, 15, 2).
times	The number of times internal cross-validation is performed. Default is 100.
test.size	The size of test data. Default is 0.2 (20 percent).
xval	The n-cross validation. Default is 5.
fun	The function used to summarise the xval macro F1 matrices.
seed	The optional random number generator seed.
verbose	A logical defining whether a progress bar is displayed.
...	Additional parameters passed to <a href="#">knn</a> from package class.

**Details**

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e. mean(F1)). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.

**Value**

An instance of class `"GenRegRes"`.

**Author(s)**

Laurent Gatto

**See Also**[knnClassification](#) and example therein.

---

knnt1Classification    *knn transfer learning classification*


---

**Description**

Classification using a variation of the KNN implementation of Wu and Dietterich's transfer learning schema

**Usage**

```
knnt1Classification(
  primary,
  auxiliary,
  fcol = "markers",
  bestTheta,
  k,
  scores = c("prediction", "all", "none"),
  seed
)
```

**Arguments**

primary	An instance of class " <a href="#">MSnSet</a> ".
auxiliary	An instance of class " <a href="#">MSnSet</a> ".
fcol	The feature meta-data containing marker definitions. Default is markers.
bestTheta	Best theta vector as output from <a href="#">knnt1Optimisation</a> , see <a href="#">knnt1Optimisation</a> for details
k	Numeric vector of length 2, containing the best k parameters to use for the primary and auxiliary datasets. If k k is not specified it will be calculated internally.
scores	One of "prediction", "all" or "none" to report the score for the predicted class only, for all classes or none.
seed	The optional random number generator seed.

**Value**

A character vector of the classifications for the unknowns

**Author(s)**

Lisa Breckels

**See Also**[knntlOptimisation](#)**Examples**

```

library(pRolocdata)
data(andy2011)
data(andy2011goCC)
## reducing calculation time of k by pre-running knnOptimisation
x <- c(andy2011, andy2011goCC)
k <- lapply(x, function(z)
            knnOptimisation(z, times=5,
                           fcol = "markers.orig",
                           verbose = FALSE))
k <- sapply(k, function(z) getParams(z))
k
## reducing parameter search with theta = 1,
## weights of only 1 or 0 will be considered
opt <- knntlOptimisation(andy2011, andy2011goCC,
                        fcol = "markers.orig",
                        times = 2,
                        by = 1, k = k)

opt
th <- getParams(opt)
plot(opt)
res <- knntlClassification(andy2011, andy2011goCC,
                          fcol = "markers.orig", th, k)

res

```

---

knntlOptimisation      *theta parameter optimisation*

---

**Description**

Classification parameter optimisation for the KNN implementation of Wu and Dietterich's transfer learning schema

**Usage**

```

knntlOptimisation(
  primary,
  auxiliary,
  fcol = "markers",
  k,

```

```

times = 50,
test.size = 0.2,
xval = 5,
by = 0.5,
length.out,
th,
xfolds,
BPPARAM = BiocParallel::bpparam(),
method = "Breckels",
log = FALSE,
seed
)

```

### Arguments

primary	An instance of class <code>"MSnSet"</code> .
auxiliary	An instance of class <code>"MSnSet"</code> .
fcol	The feature meta-data containing marker definitions. Default is markers.
k	Numeric vector of length 2, containing the best k parameters to use for the primary (k[1]) and auxiliary (k[2]) datasets. See <code>knnOptimisation</code> for generating best k.
times	The number of times cross-validation is performed. Default is 50.
test.size	The size of test (validation) data. Default is 0.2 (20 percent).
xval	The number of rounds of cross-validation to perform.
by	The increment for theta, must be one of <code>c(1, 0.5, 0.25, 0.2, 0.15, 0.1, 0.05)</code>
length.out	Alternative to using by parameter. Specifies the desired length of the sequence of theta to test.
th	A matrix of theta values to test for each class as generated from the function <code>thetas</code> , the number of columns should be equal to the number of classes contained in fcol. Note: columns will be ordered according to <code>getMarkerClasses(primary, fcol)</code> . This argument is only valid if the default method 'Breckels' is used.
xfolds	Option to pass specific folds for the cross validation.
BPPARAM	Required for parallelisation. If not specified selects a default <code>BiocParallelParam</code> , from global options or, if that fails, the most recently registered() back-end.
method	The k-NN transfer learning method to use. The default is 'Breckels' as described in the Breckels et al (2016). If 'Wu' is specified then the original method implemented Wu and Dietterich (2004) is implemented.
log	A logical defining whether logging should be enabled. Default is FALSE. Note that logging produces considerably bigger objects.
seed	The optional random number generator seed.

### Details

`knnt1Optimisation` implements a variation of Wu and Dietterich's transfer learning schema: P. Wu and T. G. Dietterich. Improving SVM accuracy by training on auxiliary data sources. In Proceedings of the Twenty-First International Conference on Machine Learning, pages 871 - 878. Morgan Kaufmann, 2004. A grid search for the best theta is performed.

**Value**

A list of containing the theta combinations tested, associated macro F1 score and accuracy for each combination over each round (specified by times).

**Author(s)**

Lisa Breckels

**References**

Breckels LM, Holden S, Wonjar D, Mulvey CM, Christoforou A, Groen AJ, Kohlbacher O, Lilley KS, Gatto L. Learning from heterogeneous data sources: an application in spatial proteomics. bioRxiv. doi: <http://dx.doi.org/10.1101/022152>

Wu P, Dietterich TG. Improving SVM Accuracy by Training on Auxiliary Data Sources. Proceedings of the 21st International Conference on Machine Learning (ICML); 2004.

**See Also**

[knnt1Classification](#) and example therein.

---

ksvmClassification      *ksvm classification*

---

**Description**

Classification using the support vector machine algorithm.

**Usage**

```
ksvmClassification(  
  object,  
  assessRes,  
  scores = c("prediction", "all", "none"),  
  cost,  
  fcol = "markers",  
  ...  
)
```

**Arguments**

object	An instance of class " <a href="#">MSnSet</a> ".
assessRes	An instance of class " <a href="#">GenRegRes</a> ", as generated by <a href="#">ksvmOptimisation</a> .
scores	One of "prediction", "all" or "none" to report the score for the predicted class only, for all classes or none.
cost	If assessRes is missing, a cost must be provided.
fcol	The feature meta-data containing marker definitions. Default is markers.
...	Additional parameters passed to <a href="#">ksvm</a> from package kernlab.

**Value**

An instance of class "*MSnSet*" with *ksvm* and *ksvm.scores* feature variables storing the classification results and scores respectively.

**Author(s)**

Laurent Gatto

**Examples**

```
library(pRocdata)
data(dunkley2006)
## reducing parameter search space and iterations
params <- ksvmOptimisation(dunkley2006, cost = 2^seq(-1,4,5), times = 3)
params
plot(params)
f1Count(params)
levelPlot(params)
getParams(params)
res <- ksvmClassification(dunkley2006, params)
getPredictions(res, fcol = "ksvm")
getPredictions(res, fcol = "ksvm", t = 0.75)
plot2D(res, fcol = "ksvm")
```

---

ksvmOptimisation

*ksvm parameter optimisation*

---

**Description**

Classification parameter optimisation for the support vector machine algorithm.

**Usage**

```
ksvmOptimisation(
  object,
  fcol = "markers",
  cost = 2^(-4:4),
  times = 100,
  test.size = 0.2,
  xval = 5,
  fun = mean,
  seed,
  verbose = TRUE,
  ...
)
```

## Arguments

object	An instance of class " <a href="#">MSnSet</a> ".
fcol	The feature meta-data containing marker definitions. Default is markers.
cost	The hyper-parameter. Default values are $2^{-4}$ :4.
times	The number of times internal cross-validation is performed. Default is 100.
test.size	The size of test data. Default is 0.2 (20 percent).
xval	The n-cross validation. Default is 5.
fun	The function used to summarise the xval macro F1 matrices.
seed	The optional random number generator seed.
verbose	A logical defining whether a progress bar is displayed.
...	Additional parameters passed to <a href="#">ksvm</a> from package kernlab.

## Details

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e.  $\text{mean}(F1)$ ). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.

## Value

An instance of class "[GenRegRes](#)".

## Author(s)

Laurent Gatto

## See Also

[ksvmClassification](#) and example therein.

---

makeGoSet

*Creates a GO feature MSnSet*

---

## Description

Creates a new "[MSnSet](#)" instance populated with a GO term binary matrix based on an original object.

## Usage

```
makeGoSet(object, params, namespace = "cellular_component", evidence = NULL)
```

**Arguments**

object	An instance of class "MSnSet" or a character of feature names.
params	An instance of class "AnnotationParams", compatible with featureNames(object)'s format.
namespace	The ontology name space. One or several of "biological_process", "cellular_component" or "molecular_function".
evidence	GO evidence filtering.

**Value**

A new "MSnSet" with the GO terms for the respective features in the original object.

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(dunkley2006)
data(dunkley2006params)
goset <- makeGoSet(dunkley2006[1:10, ],
                  dunkley2006params)

goset
exprs(goset)[1:10, 1:5]
image(goset)
```

---

MAPPParams-class	<i>The 'logPosteriors' function can be used to extract the log-posteriors at each iteration of the EM algorithm to check for convergence.</i>
------------------	---

---

**Description**

These functions implement the T augmented Gaussian mixture (TAGM) model for mass spectrometry-based spatial proteomics datasets using the maximum a posteriori (MAP) optimisation routine.

**Usage**

```
## S4 method for signature 'MAPPParams'
show(object)

logPosteriors(x)

tagmMapTrain(
  object,
  fcol = "markers",
  method = "MAP",
```



```

    numIter = 100,
    mu0 = NULL,
    lambda0 = 0.01,
    nu0 = NULL,
    S0 = NULL,
    beta0 = NULL,
    u = 2,
    v = 10,
    seed = NULL
)

tagmMapPredict(
  object,
  params,
  fcol = "markers",
  probJoint = FALSE,
  probOutlier = TRUE
)

```

### Arguments

object	An <a href="#">MSnbase::MSnSet</a> containing the spatial proteomics data to be passed to <code>tagmMapTrain</code> and <code>tagmPredict</code> .
x	An object of class 'MAPPParams'.
fcol	The feature meta-data containing marker definitions. Default is markers.
method	A character() describing the inference method for the TAGM algorithm. Default is "MAP".
numIter	The number of iterations of the expectation-maximisation algorithm. Default is 100.
mu0	The prior mean. Default is colMeans of the expression data.
lambda0	The prior shrinkage. Default is 0.01.
nu0	The prior degree of freedom. Default is <code>ncol(exprs(object)) + 2</code>
S0	The prior inverse-wishary scale matrix. Empirical prior used by default.
beta0	The prior Dirichlet distribution concentration. Default is 1 for each class.
u	The prior shape parameter for Beta(u, v). Default is 2
v	The prior shape parameter for Beta(u, v). Default is 10.
seed	The optional random number generator seed.
params	An instance of class <a href="#">MAPPParams</a> , as generated by <code>tagmMapTrain()</code> .
probJoint	A logical(1) indicating whether to return the joint probability matrix, i.e. the probability for all classes as a new <code>tagm.map.joint</code> feature variable.
probOutlier	A logical(1) indicating whether to return the probability of being an outlier as a new <code>tagm.map.outlier</code> feature variable. A high value indicates that the protein is unlikely to belong to any annotated class (and is hence considered an outlier).

## Details

The `tagmMapTrain` function generates the MAP parameters (object or class `MAPParams`) based on an annotated quantitative spatial proteomics dataset (object of class `MSnbase:MSnSet`). Both are then passed to the `tagmPredict` function to predict the sub-cellular localisation of protein of unknown localisation. See the *pRoloc-bayesian* vignette for details and examples. In this implementation, if numerical instability is detected in the covariance matrix of the data a small multiple of the identity is added. A message is printed if this conditioning step is performed.

## Value

`tagmMapTrain` returns an instance of class `MAPParams()`.

`tagmPredict` returns an instance of class `MSnbase:MSnSet` containing the localisation predictions as a new `tagm.map.allocation` feature variable.

## Slots

`method` A character() storing the TAGM method name.

`priors` A list() with the priors for the parameters

`seed` An integer() with the random number generation seed.

`posteriors` A list() with the updated posterior parameters and log-posterior of the model.

`datasize` A list() with details about size of data

## Author(s)

Laurent Gatto

Oliver M. Crook

## References

*A Bayesian Mixture Modelling Approach For Spatial Proteomics* Oliver M Crook, Claire M Mulvey, Paul D. W. Kirk, Kathryn S Lilley, Laurent Gatto bioRxiv 282269; doi: <https://doi.org/10.1101/282269>

## See Also

The `plotEllipse()` function can be used to visualise TAGM models on PCA plots with ellipses. The `tagmMapTrain()` function to use the TAGM MAP method.

---

markerMSnSet	<i>Extract marker/unknown subsets</i>
--------------	---------------------------------------

---

### Description

These function extract the marker or unknown proteins into a new MSnSet.

### Usage

```
markerMSnSet(object, fcol = "markers")
```

```
unknownMSnSet(object, fcol = "markers")
```

### Arguments

object	An instance of class MSnSet
fcol	The name of the feature data column, that will be used to separate the markers from the proteins of unknown localisation. When the markers are encoded as vectors, features of unknown localisation are defined as <code>fData(object)[, fcol] == "unknown"</code> . For matrix-encoded markers, unlabelled proteins are defined as <code>rowSums(fData(object)[, fcol]) == 0</code> . Default is "markers".

### Value

An new MSnSet with marker/unknown proteins only.

### Author(s)

Laurent Gatto

### See Also

[sampleMSnSet](#) [testMSnSet](#) and [markers](#) for markers encoding.

### Examples

```
library("pRolocdata")
data(dunkley2006)
mrk <- markerMSnSet(dunkley2006)
unk <- unknownMSnSet(dunkley2006)
dim(dunkley2006)
dim(mrk)
dim(unk)
table(fData(dunkley2006)$markers)
table(fData(mrk)$markers)
table(fData(unk)$markers)
## matrix-encoded markers
dunkley2006 <- mrkVecToMat(dunkley2006)
dim(markerMSnSet(dunkley2006, "Markers"))
```

```
stopifnot(all.equal(featureNames(markerMSnSet(dunkley2006, "Markers")),
                    featureNames(markerMSnSet(dunkley2006, "markers"))))
dim(unknownMSnSet(dunkley2006, "Markers"))
stopifnot(all.equal(featureNames(unknownMSnSet(dunkley2006, "Markers")),
                    featureNames(unknownMSnSet(dunkley2006, "markers"))))
```

---

MartInstance-class      *Class "MartInstance"*

---

### Description

Internal infrastructure to query/handle several individual mart instance. See `MartInterface.R` for details.

### Author(s)

Laurent Gatto <lg390@cam.ac.uk>

---

MCMCChains-class      *Infrastructure to store and process MCMC results*

---

### Description

The MCMCParams infrastructure is used to store and process Markov chain Monte Carlo results for the T-Augmented Gaussian Mixture model (TAGM) from Crook et al. (2018).

### Usage

```
chains(object)

## S4 method for signature 'MCMCParams'
show(object)

## S4 method for signature 'ComponentParam'
show(object)

## S4 method for signature 'MCMCChain'
show(object)

## S4 method for signature 'MCMCChains'
length(x)

## S4 method for signature 'MCMCParams'
length(x)

## S4 method for signature 'MCMCChains,ANY,ANY'
```

```

x[[i, j = "missing", drop = "missing"]]

## S4 method for signature 'MCMCParams,ANY,ANY'
x[[i, j = "missing", drop = "missing"]]

## S4 method for signature 'MCMCChains,ANY,ANY,ANY'
x[i, j = "missing", drop = "missing"]

## S4 method for signature 'MCMCParams,ANY,ANY,ANY'
x[i, j = "missing", drop = "missing"]

## S4 method for signature 'MCMCChains'
show(object)

```

### Arguments

object	An instance of appropriate class.
x	Object to be subset.
i	An integer(). Should be of length 1 for [[].
j	Missing.
drop	Missing.

### Details

Objects of the MCMCParams class are created with the `tagmMcmcTrain()` function. These objects store the *priors* of the generative TAGM model and the results of the MCMC chains, which themselves are stored as an instance of class MCMCChains and can be accessed with the `chains()` function. A summary of the MCMC chains (or class MCMCSummary) can be further computed with the `tagmMcmcProcess()` function.

See the *pRoloc-bayesian* vignette for examples.

### Slots

`chains` `list()` containing the individual full MCMC chain results in an MCMCChains instance. Each element must be a valid MCMCChain instance.

`posteriorEstimates` A data.frame documenting the posterior priors in an MCMCSummary instance. It contains N rows and columns `tagm.allocation`, `tagm.probability`, `tagm.outlier`, `tagm.probability.lowerquantile`, `tagm.probability.upperquantile` and `tagm.mean.shannon`.

`diagnostics` A matrix of dimensions 1 by 2 containing the MCMCSummary diagnostics.

`tagm.joint` A matrix of dimensions N by K storing the joint probability in an MCMCSummary instance.

`method` character(1) describing the method in the MCMCParams object.

`chains` Object of class MCMCChains containing the full MCMC chain results stored in the MCMCParams object.

`priors` `list()`

summary Object of class MCMCSummary the summarised MCMC results available in the MCMCParams instance.

n integer(1) indicating the number of MCMC interactions. Stored in an MCMCChain instance.

K integer(1) indicating the number of components. Stored in an MCMCChain instance.

N integer(1) indicating the number of proteins. Stored in an MCMCChain instance.

Component matrix(N, n) component allocation results of an MCMCChain instance.

ComponentProb matrix(N, n, K) component allocation probabilities of an MCMCChain instance.

Outlier matrix(N, n) outlier allocation results.

OutlierProb matrix(N, n, 2) outlier allocation probabilities of an MCMCChain instance.

### See Also

The function tagmMcmcTrain() to construct object of this class.

---

mcmc_get_outliers	<i>Number of outlier at each iteration of MCMC</i>
-------------------	--

---

### Description

Helper function to get the number of outlier at each MCMC iteration.

Helper function to get mean component allocation at each MCMC iteration.

Helper function to get mean probability of belonging to outlier at each iteration.

Wrapper for the geweke diagnostics from coda package also return p-values.

Helper function to pool chains together after processing

Helper function to burn n iterations from the front of the chains

Helper function to subsample the chains, known informally as thinning.

Produces a violin plot with the protein posterior probabilities distributions for all organelles.

### Usage

```
mcmc_get_outliers(x)
```

```
mcmc_get_meanComponent(x)
```

```
mcmc_get_meanoutliersProb(x)
```

```
geweke_test(k)
```

```
mcmc_pool_chains(param)
```

```
mcmc_burn_chains(x, n = 50)
```

```
mcmc_thin_chains(x, freq = 5)
```

```
## S4 method for signature 'MCMCParams,character'
```

```
plot(x, y, ...)
```

**Arguments**

x	Object of class MCMCParams
k	A list of <code>coda::mcmc</code> objects, as returned by <code>mcmc_get_outliers</code> , <code>mcmc_get_meanComponent</code> and <code>mcmc_get_meanoutliersProb</code> .
param	An object of class MCMCParams.
n	<code>integer(1)</code> defining number of iterations to burn. The default is 50
freq	Thinning frequency. The function retains every 'freq'th iteration and is an 'integer(1)'. The default thinning frequency is '5'.
y	A 'character(1)' with a protein name.
...	Currently ignored.

**Value**

A list of length `length(x)`.  
 A list of length `length(x)`.  
 A list of length `length(x)`.  
 A matrix with the test z- and p-values for each chain.  
 A pooled MCMCParams object.  
 An updated MCMCParams object.  
 A thinned 'MCMCParams' object.  
 A `ggplot2` object.

**Author(s)**

Laurent Gatto

---

minMarkers

*Creates a reduced marker variable*

---

**Description**

This function updates an `MSnSet` instances and sets markers class to unknown if there are less than `n` instances.

**Usage**

```
minMarkers(object, n = 10, fcol = "markers")
```

**Arguments**

object	An instance of class " <code>MSnSet</code> ".
n	Minimum of marker instances per class.
fcol	The name of the markers column in the <code>featureData</code> slot. Default is <code>markers</code> .

**Value**

An instance of class "MSnSet" with a new feature variables, named after the original fcol variable and the n value.

**Author(s)**

Laurent Gatto

**See Also**

[getPredictions](#) to filter based on classification scores.

**Examples**

```
library(pRocdata)
data(dunkley2006)
d2 <- minMarkers(dunkley2006, 20)
getMarkers(dunkley2006)
getMarkers(d2, fcol = "markers20")
```

---

mixing\_posterior\_check

*Model calibration plots*

---

**Description**

Model calibration model with posterior z-scores and posterior shrinkage

**Usage**

```
mixing_posterior_check(object, params, priors, fcol = "markers")
```

**Arguments**

object	A valid object of class MSnset
params	A valid object of class MCMCParams that has been processed and checked for convergence
priors	The prior that were used in the model
fcol	The columns of the feature data which contain the marker data.

**Value**

Used for side effect of producing plot. Invisibly returns an ggplot object that can be further manipulated

**Author(s)**

Oliver M. Crook <omc25@cam.ac.uk>



**Examples**

```
## Not run:
library("pRoloc")
data("tan2009r1")

tanres <- tagmMcmcTrain(object = tan2009r1)
tanres <- tagmMcmcProcess(tanres)
tan2009r1 <- tagmMcmcPredict(object = tan2009r1, params = tanres, probJoint = TRUE)
myparams <- chains(e14Tagm_converged_pooled)[[1]]
myparams2 <- chains(mcmc_pool_chains(tanres))[[1]]
priors <- tanres@priors
pRoloc::mixing_posterior_check(object = tan2009r1, params = myparams2, priors = priors)

## End(Not run)
```

---

MLearn-methods

*The MLearn interface for machine learning*


---

**Description**

This method implements MLInterfaces' MLearn method for instances of the class "[MSnSet](#)".

**Methods**

```
signature(formula = "formula", data = "MSnSet", .method = "learnerSchema", trainInd = "numeric")
  The learning problem is stated with the formula and applies the .method schema on the
  MSnSet data input using the trainInd numeric indices as train data.

signature(formula = "formula", data = "MSnSet", .method = "learnerSchema", trainInd = "xvalSpec")
  In this case, an instance of xvalSpec is used for cross-validation.

signature(formula = "formula", data = "MSnSet", .method = "clusteringSchema", trainInd = "missing")
  Hierarchical (hclustI), k-means (kmeansI) and partitioning around medoids (pamI) cluster-
  ing algorithms using MLInterface's MLearn interface.
```

**See Also**

The MLInterfaces package documentation, in particular [MLearn](#).

---

move2Ds

*Displays a spatial proteomics animation*


---

**Description**

Given two MSnSet instances of one MSnSetList with at least two items, this function produces an animation that shows the transition from the first data to the second.

**Usage**

```
move2Ds(object, pcol, fcol = "markers", n = 25, h1)
```

**Arguments**

object	An <code>linkS4class{MSnSet}</code> or a <code>MSnSetList</code> . In the latter case, only the two first elements of the list will be used for plotting and the others will be silently ignored.
pcol	If object is an <code>MSnSet</code> , a factor or the name of a phenotype variable ( <code>phenoData</code> slot) defining how to split the single <code>MSnSet</code> into two or more data sets. Ignored if object is a <code>MSnSetList</code> .
fcol	Feature meta-data label ( <code>fData</code> column name) defining the groups to be differentiated using different colours. Default is <code>markers</code> . Use <code>NULL</code> to suppress any colouring.
n	Number of frames, Default is 25.
h1	An optional instance of class <code>linkS4class{FeaturesOfInterest}</code> to track features of interest.

**Value**

Used for its side effect of producing a short animation.

**Author(s)**

Laurent Gatto

**See Also**

[plot2Ds](#) to a single figure with the two datasets.

**Examples**

```
library("pRolocdata")
data(dunkley2006)

## Create a relevant MSnSetList using the dunkley2006 data
xx <- split(dunkley2006, "replicate")
xx1 <- xx[[1]]
xx2 <- xx[[2]]
fData(xx1)$markers[374] <- "Golgi"
fData(xx2)$markers[412] <- "unknown"
xx@x[[1]] <- xx1
xx@x[[2]] <- xx2

## The features we want to track
foi <- FeaturesOfInterest(description = "test",
                          fnames = featureNames(xx[[1]])[c(374, 412)])

## (1) visualise each experiment separately
```

```

par(mfrow = c(2, 1))
plot2D(xx[[1]], main = "condition A")
highlightOnPlot(xx[[1]], foi)
plot2D(xx[[2]], mirrorY = TRUE, main = "condition B")
highlightOnPlot(xx[[2]], foi, args = list(mirrorY = TRUE))

## (2) plot both data on the same plot
par(mfrow = c(1, 1))
tmp <- plot2Ds(xx)
highlightOnPlot(data1(tmp), foi, lwd = 2)
highlightOnPlot(data2(tmp), foi, pch = 5, lwd = 2)

## (3) create an animation
move2Ds(xx, pcol = "replicate")
move2Ds(xx, pcol = "replicate", hl = foi)

```

---

mrkConsProfiles

*Marker consensus profiles*


---

## Description

A function to calculate average marker profiles.

## Usage

```
mrkConsProfiles(object, fcol = "markers", method = mean)
```

## Arguments

object	An instance of class MSnSet.
fcol	Feature meta-data label (fData column name) defining the groups to be differentiated using different colours. Default is markers.
method	A function to average marker profiles. Default is mean.

## Value

A matrix of dimensions *number of clusters* (excluding unknowns) by *number of fractions*.

## Author(s)

Laurent Gatto and Lisa M. Breckels

## See Also

The [mrkHClust](#) function to produce a hierarchical cluster.

**Examples**

```

library("pRolocdata")
data(dunkley2006)
mrkConsProfiles(dunkley2006)
mrkConsProfiles(dunkley2006, method = median)
mm <- mrkConsProfiles(dunkley2006)
## Reorder fractions
o <- order(dunkley2006$fraction)
## Plot mean organelle profiles using the
## default pRoloc colour palette.
matplot(t(mm[, o]), type = "l",
        xlab = "Fractions", ylab = "Relative intensity",
        main = "Mean organelle profiles",
        col = getStockcol(), lwd = 2, lty = 1)
## Add a legend
addLegend(markerMSnSet(dunkley2006), where = "topleft")

```

---

mrkHClust

*Draw a dendrogram of subcellular clusters*


---

**Description**

This functions calculates an average protein profile for each marker class (proteins of unknown localisation are ignored) and then generates a dendrogram representing the relation between marker classes. The colours used for the dendrogram labels are taken from the default colours (see [getStockcol](#)) so as to match the colours with other spatial proteomics visualisations such as [plot2D](#).

**Usage**

```

mrkHClust(
  object,
  fcol = "markers",
  distargs,
  hclustargs,
  method = mean,
  plot = TRUE,
  ...
)

```

**Arguments**

object	An instance of class MSnSet.
fcol	Feature meta-data label (fData column name) defining the groups to be differentiated using different colours. Default is markers.
distargs	A list of arguments to be passed to the <a href="#">dist</a> function.
hclustargs	A list of arguments to be passed to the <a href="#">hclust</a> function.
method	A function to average marker profiles. Default is mean.

plot            A logical defining whether the dendrogram should be plotted. Default is TRUE.  
 ...            Additional parameters passed when plotting the [dendrogram](#).

**Value**

Invisibly returns a dendrogram object, containing the hierarchical cluster as computed by hclust.

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(dunkley2006)
mrkHClust(dunkley2006)
```

---

mrkVecToMat	<i>Create a marker vector or matrix.</i>
-------------	--

---

**Description**

Functions producing a new vector (matrix) marker vector set from an existing matrix (vector) marker set.

**Usage**

```
mrkVecToMat(object, vfcol = "markers", mfcol = "Markers")
mrkMatToVec(object, mfcol = "Markers", vfcol = "markers")
mrkMatAndVec(object, vfcol = "markers", mfcol = "Markers")
showMrkMat(object, mfcol = "Markers")
isMrkMat(object, fcol = "Markers")
isMrkVec(object, fcol = "markers")
mrkEncoding(object, fcol = "markers")
```

**Arguments**

object	An MSnSet object
vfcol	The name of the <i>vector</i> marker feature variable. Default is "markers".
mfcol	The name of the <i>matrix</i> marker feature variable. Default is "Markers".
fcol	A marker feature variable name.

## Details

Sub-cellular markers can be encoded in two different ways. Sets of spatial markers can be represented as character *vectors* (character or factor, to be accurate), stored as feature metadata, and proteins of unknown or uncertain localisation (unlabelled, to be classified) are marked with the "unknown" character. While very handy, this encoding suffers from some drawbacks, in particular the difficulty to label proteins that reside in multiple (possible or actual) localisations. The markers vector feature data is typically named markers. A new *matrix* encoding is also supported. Each spatial compartment is defined in a column in a binary markers matrix and the resident proteins are encoded with 1s. The markers matrix feature data is typically named Markers. If proteins are assigned unique localisations only (i.e. no multi-localisation) or their localisation is unknown (unlabelled), then both encodings are equivalent. When the markers are encoded as vectors, features of unknown localisation are defined as `fData(object)[, fcol] == "unknown"`. For matrix-encoded markers, unlabelled proteins are defined as `rowSums(fData(object)[, fcol]) == 0`.

The `mrkMatToVec` and `mrkVecToMat` functions enable the conversion from matrix (vector) to vector (matrix). The `mrkMatAndVec` function generates the missing encoding from the existing one. If the destination encoding already exists, or, more accurately, if the feature variable of the destination encoding exists, an error is thrown. During the conversion from matrix to vector, if multiple possible label exists, they are dropped, i.e. they are converted to "unknown". Function `isMrkVec` and `isMrkMat` can be used to test if a marker set is encoded as a vector or a matrix. `mrkEncoding` returns either "vector" or "matrix" depending on the nature of the markers.

## Value

An updated `MSnSet` with a new vector (matrix) marker set.

## Author(s)

Laurent Gatto and Lisa Breckels

## See Also

Other functions that operate on markers are [getMarkers](#), [getMarkerClasses](#) and [markerMSnSet](#). To add markers to an existing `MSnSet`, see the [addMarkers](#) function and [pRolocmarkers](#), for a list of suggested markers.

## Examples

```
library("pRolocdata")
data(dunkley2006)
dunk <- mrkVecToMat(dunkley2006)
head(fData(dunk)$Markers)
fData(dunk)$markers <- NULL
dunk <- mrkMatToVec(dunk)
stopifnot(all.equal(fData(dunkley2006)$markers,
                    fData(dunk)$markers))
```

---

nbClassification	<i>nb classification</i>
------------------	--------------------------

---

**Description**

Classification using the naive Bayes algorithm.

**Usage**

```
nbClassification(  
  object,  
  assessRes,  
  scores = c("prediction", "all", "none"),  
  laplace,  
  fcol = "markers",  
  ...  
)
```

**Arguments**

object	An instance of class <code>"MSnSet"</code> .
assessRes	An instance of class <code>"GenRegRes"</code> , as generated by <code>nbOptimisation</code> .
scores	One of "prediction", "all" or "none" to report the score for the predicted class only, for all classes or none.
laplace	If assessRes is missing, a laplace must be provided.
fcol	The feature meta-data containing marker definitions. Default is markers.
...	Additional parameters passed to <code>naiveBayes</code> from package <code>e1071</code> .

**Value**

An instance of class `"MSnSet"` with `nb` and `nb.scores` feature variables storing the classification results and scores respectively.

**Author(s)**

Laurent Gatto

**Examples**

```
library(pRocdata)  
data(dunkley2006)  
## reducing parameter search space and iterations  
params <- nbOptimisation(dunkley2006, laplace = c(0, 5), times = 3)  
params  
plot(params)  
f1Count(params)  
levelPlot(params)
```

```

getParams(params)
res <- nbClassification(dunkley2006, params)
getPredictions(res, fcol = "naiveBayes")
getPredictions(res, fcol = "naiveBayes", t = 1)
plot2D(res, fcol = "naiveBayes")

```

---

nbOptimisation

*nb paramter optimisation*


---

## Description

Classification algorithm parameter for the naive Bayes algorithm.

## Usage

```

nbOptimisation(
  object,
  fcol = "markers",
  laplace = seq(0, 5, 0.5),
  times = 100,
  test.size = 0.2,
  xval = 5,
  fun = mean,
  seed,
  verbose = TRUE,
  ...
)

```

## Arguments

object	An instance of class " <a href="#">MSnSet</a> ".
fcol	The feature meta-data containing marker definitions. Default is markers.
laplace	The hyper-parameter. Default values are seq(0, 5, 0.5).
times	The number of times internal cross-validation is performed. Default is 100.
test.size	The size of test data. Default is 0.2 (20 percent).
xval	The n-cross validation. Default is 5.
fun	The function used to summarise the xval macro F1 matrices.
seed	The optional random number generator seed.
verbose	A logical defining whether a progress bar is displayed.
...	Additional parameters passed to <a href="#">naiveBayes</a> from package <a href="#">e1071</a> .

## Details

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e. mean(F1)). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.



**Value**

An instance of class "[GenRegRes](#)".

**Author(s)**

Laurent Gatto

**See Also**

[nbClassification](#) and example therein.

---

nicheMeans2D

*Uncertainty plot organelle means*

---

**Description**

Produces a pca plot with uncertainty in organelle means projected onto the PCA plot with contours.

**Usage**

```
nicheMeans2D(
  object,
  params,
  priors,
  dims = c(1, 2),
  fcol = "markers",
  aspect = 0.5
)
```

**Arguments**

object	A valid object of class MSnset
params	A valid object of class MCMCParams that has been processed and checked for convergence
priors	The prior that were used in the model
dims	The PCA dimension in which to project he data, default is c(1, 2)
fcol	The columns of the feature data which contain the marker data.
aspect	A argument to change the plotting aspect of the PCA

**Value**

Used for side effect of producing plot. Invisibly returns an ggplot object that can be further manipulated

**Author(s)**

Oliver M. Crook <omc25@cam.ac.uk>

**Examples**

```
## Not run:
library("pRolocdata")
data("tan2009r1")

tanres <- tagmMcmcTrain(object = tan2009r1)
tanres <- tagmMcmcProcess(tanres)
tan2009r1 <- tagmMcmcPredict(object = tan2009r1, params = tanres, probJoint = TRUE)
myparams <- chains(e14Tagm_converged_pooled)[[1]]
myparams2 <- chains(mcmc_pool_chains(tanres))[[1]]
priors <- tanres@priors
pRoloc::nicheMeans2D(object = tan2009r1, params = myparams2, priors = priors)

## End(Not run)
```

---

nndist-methods

*Nearest neighbour distances*


---

**Description**

Methods computing the nearest neighbour indices and distances for matrix and MSnSet instances.

**Methods**

signature(object = "matrix", k = "numeric", dist = "character", ...) Calculates indices and distances to the k (default is 3) nearest neighbours of each feature (row) in the input matrix object. The distance dist can be either of "euclidean" or "mahalanobis". Additional parameters can be passed to the internal function `FNN::get.knn`. Output is a matrix with  $2 * k$  columns and `nrow(object)` rows.

signature(object = "MSnSet", k = "numeric", dist = "character", ...) As above, but for an MSnSet input. The indices and distances to the k nearest neighbours are added to the object's feature metadata.

signature(object = "matrix", query = "matrix", k = "numeric", ...) If two matrix instances are provided as input, the k (default is 3) indices and distances of the nearest neighbours of query in object are returned as a matrix of dimensions  $2 * k$  by `nrow(query)`. Additional parameters are passed to `FNN::get.knx`. Only euclidean distance is available.

**Examples**

```
library("pRolocdata")
data(dunkley2006)

## Using a matrix as input
m <- exprs(dunkley2006)
m[1:4, 1:3]
head(nndist(m, k = 5))
tail(nndist(m[1:100, ], k = 2, dist = "mahalanobis"))
```

```
## Same as above for MSnSet
d <- ndist(dunkley2006, k = 5)
head(fData(d))

d <- ndist(dunkley2006[1:100, ], k = 2, dist = "mahalanobis")
tail(fData(d))

## Using a query
ndist(m[1:100, ], m[101:110, ], k = 2)
```

---

nnetClassification      *nnet classification*

---

## Description

Classification using the artificial neural network algorithm.

## Usage

```
nnetClassification(
  object,
  assessRes,
  scores = c("prediction", "all", "none"),
  decay,
  size,
  fcol = "markers",
  ...
)
```

## Arguments

object	An instance of class <code>"MSnSet"</code> .
assessRes	An instance of class <code>"GenRegRes"</code> , as generated by <code>nnetOptimisation</code> .
scores	One of <code>"prediction"</code> , <code>"all"</code> or <code>"none"</code> to report the score for the predicted class only, for all classes or none.
decay	If <code>assessRes</code> is missing, a decay must be provided.
size	If <code>assessRes</code> is missing, a size must be provided.
fcol	The feature meta-data containing marker definitions. Default is <code>markers</code> .
...	Additional parameters passed to <code>nnet</code> from package <code>nnet</code> .

## Value

An instance of class `"MSnSet"` with `nnet` and `nnet.scores` feature variables storing the classification results and scores respectively.

**Author(s)**

Laurent Gatto

**Examples**

```

library(pRolocdata)
data(dunkley2006)
## reducing parameter search space and iterations
params <- nnetOptimisation(dunkley2006, decay = 10^(c(-1, -5)), size = c(5, 10), times = 3)
params
plot(params)
f1Count(params)
levelPlot(params)
getParams(params)
res <- nnetClassification(dunkley2006, params)
getPredictions(res, fcol = "nnet")
getPredictions(res, fcol = "nnet", t = 0.75)
plot2D(res, fcol = "nnet")

```

---

nnetOptimisation

*nnet parameter optimisation*


---

**Description**

Classification parameter optimisation for artificial neural network algorithm.

**Usage**

```

nnetOptimisation(
  object,
  fcol = "markers",
  decay = c(0, 10^(-1:-5)),
  size = seq(1, 10, 2),
  times = 100,
  test.size = 0.2,
  xval = 5,
  fun = mean,
  seed,
  verbose = TRUE,
  ...
)

```

**Arguments**

object	An instance of class "MSnSet".
fcol	The feature meta-data containing marker definitions. Default is markers.
decay	The hyper-parameter. Default values are $c(0, 10^{(-1:-5)})$ .

size	The hyper-parameter. Default values are seq(1, 10, 2).
times	The number of times internal cross-validation is performed. Default is 100.
test.size	The size of test data. Default is 0.2 (20 percent).
xval	The n-cross validation. Default is 5.
fun	The function used to summarise the xval macro F1 matrices.
seed	The optional random number generator seed.
verbose	A logical defining whether a progress bar is displayed.
...	Additional parameters passed to <a href="#">nnet</a> from package <code>nnet</code> .

### Details

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e. mean(F1)). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.

### Value

An instance of class "[GenRegRes](#)".

### Author(s)

Laurent Gatto

### See Also

[nnetClassification](#) and example therein.

---

orderGoAnnotations      *Orders annotation information*

---

### Description

For a given matrix of annotation information, this function returns the information ordered according to the best fit with the data.

### Usage

```
orderGoAnnotations(  
  object,  
  fcol = "GOAnnotations",  
  k = 1:5,  
  n = 5,  
  p = 1/3,  
  verbose = TRUE,  
  seed  
)
```

**Arguments**

object	An instance of class MSnSet.
fcol	The name of the annotations matrix. Default is GOAnnotations.
k	The number of clusters to test. Default is k = 1 : 5
n	The minimum number of proteins per component cluster.
p	The normalisation factor, per k tested
verbose	A logical indicating if a progress bar should be displayed. Default is TRUE.
seed	An optional random number generation seed.

**Details**

As there are typically many protein/annotation sets that may fit the data we order protein sets by best fit i.e. cluster tightness, by computing the mean normalised Euclidean distance for all instances per protein set.

For each protein set i.e. proteins that have been labelled with a specified term/information criteria, we find the best k cluster components for the set (the default is to test k = 1 : 5) according to the minimum mean normalised pairwise Euclidean distance over all component clusters. (Note: when testing k if any components are found to have less than n proteins these components are not included and k is reduced by 1).

Each component cluster is normalised by  $N^p$  (where N is the total number of proteins per component, and p is the power). Hueristically,  $p = 1/3$  and normalising by  $N^{1/3}$  has been found the optimum normalisation factor.

Candidates in the matrix are ordered according to lowest mean normalised pairwise Euclidean distance as we expect high density, tight clusters to have the smallest mean normalised distance.

This function is a wrapper for running `clustDist`, `getNormDist`, see the "Annotating spatial proteomics data" vignette for more details.

**Value**

An updated MSnSet containing the newly ordered fcol matrix.

**Author(s)**

Lisa M Breckels

**See Also**

`addGoAnnotations` and example therein.

---

`orgQuants`*Returns organelle-specific quantile scores*

---

**Description**

This function produces organelle-specific quantiles corresponding to the given classification scores.

**Usage**

```
orgQuants(object, fcol, scol, mcol = "markers", t, verbose = TRUE)
```

**Arguments**

<code>object</code>	An instance of class "MSnSet".
<code>fcol</code>	The name of the prediction column in the featureData slot.
<code>scol</code>	The name of the prediction score column in the featureData slot. If missing, created by pasting '.scores' after fcol.
<code>mcol</code>	The name of the column containing the training data in the featureData slot. Default is markers.
<code>t</code>	The quantile threshold.
<code>verbose</code>	If TRUE, the calculated thresholds are printed.

**Value**

A named vector of organelle thresholds.

**Author(s)**

Lisa Breckels

**See Also**

[getPredictions](#) to get organelle predictions based on calculated thresholds.

**Examples**

```
library("pRolocdata")
data(dunkley2006)
res <- svmClassification(dunkley2006, fcol = "pd.markers",
                        sigma = 0.1, cost = 0.5)
## 50% top predictions per class
ts <- orgQuants(res, fcol = "svm", t = .5)
getPredictions(res, fcol = "svm", t = ts)
```

---

```
perTurboClassification
    perTurbo classification
```

---

### Description

Classification using the PerTurbo algorithm.

### Usage

```
perTurboClassification(
  object,
  assessRes,
  scores = c("prediction", "all", "none"),
  pRegul,
  sigma,
  inv,
  reg,
  fcol = "markers"
)
```

### Arguments

object	An instance of class <code>"MSnSet"</code> .
assessRes	An instance of class <code>"GenRegRes"</code> , as generated by <a href="#">svmRegularisation</a> .
scores	One of "prediction", "all" or "none" to report the score for the predicted class only, for all classes or none.
pRegul	If assessRes is missing, a pRegul must be provided. See <a href="#">perTurboOptimisation</a> for details.
sigma	If assessRes is missing, a sigma must be provided. See <a href="#">perTurboOptimisation</a> for details.
inv	The type of algorithm used to invert the matrix. Values are : "Inversion Cholesky" ( <a href="#">chol2inv</a> ), "Moore Penrose" ( <a href="#">ginv</a> ), "solve" ( <a href="#">solve</a> ), "svd" ( <a href="#">svd</a> ). Default value is "Inversion Cholesky".
reg	The type of regularisation of matrix. Values are "none", "trunc" or "tikhonov". Default value is "tikhonov".
fcol	The feature meta-data containing marker definitions. Default is markers.

### Value

An instance of class `"MSnSet"` with `perTurbo` and `perTurbo.scores` feature variables storing the classification results and scores respectively.

### Author(s)

Thomas Burger and Samuel Wieczorek



## References

N. Courty, T. Burger, J. Laurent. "PerTurbo: a new classification algorithm based on the spectrum perturbations of the Laplace-Beltrami operator", The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2011), D. Gunopulos et al. (Eds.): ECML PKDD 2011, Part I, LNAI 6911, pp. 359 - 374, Athens, Greece, September 2011.

## Examples

```
library(pRolocdata)
data(dunkley2006)
## reducing parameter search space
params <- perTurboOptimisation(dunkley2006,
                              pRegul = 2^seq(-2,2,2),
                              sigma = 10^seq(-1, 1, 1),
                              inv = "Inversion Cholesky",
                              reg ="tikhonov",
                              times = 3)

params
plot(params)
f1Count(params)
levelPlot(params)
getParams(params)
res <- perTurboClassification(dunkley2006, params)
getPredictions(res, fcol = "perTurbo")
getPredictions(res, fcol = "perTurbo", t = 0.75)
plot2D(res, fcol = "perTurbo")
```

---

perTurboOptimisation    *PerTurbo parameter optimisation*

---

## Description

Classification parameter optimisation for the PerTurbo algorithm

## Usage

```
perTurboOptimisation(
  object,
  fcol = "markers",
  pRegul = 10^(seq(from = -1, to = 0, by = 0.2)),
  sigma = 10^(seq(from = -1, to = 1, by = 0.5)),
  inv = c("Inversion Cholesky", "Moore Penrose", "solve", "svd"),
  reg = c("tikhonov", "none", "trunc"),
  times = 1,
  test.size = 0.2,
  xval = 5,
  fun = mean,
```

```

    seed,
    verbose = TRUE
)

```

### Arguments

object	An instance of class " <a href="#">MSnSet</a> ".
fc01	The feature meta-data containing marker definitions. Default is markers.
pRegul	The hyper-parameter for the regularisation (values are in ]0,1]). If reg=="trunc", pRegul is for the percentage of eigen values in matrix. If reg=="tikhonov", then 'pRegul' is the parameter for the tikhonov regularisation. Available configurations are : "Inversion Cholesky" - ("tikhonov" / "none"), "Moore Penrose" - ("tikhonov" / "none"), "solve" - ("tikhonov" / "none"), "svd" - ("tikhonov" / "none" / "trunc").
sigma	The hyper-parameter.
inv	The type of algorithm used to invert the matrix. Values are : "Inversion Cholesky" ( <a href="#">chol2inv</a> ), "Moore Penrose" ( <a href="#">ginv</a> ), "solve" ( <a href="#">solve</a> ), "svd" ( <a href="#">svd</a> ). Default value is "Inversion Cholesky".
reg	The type of regularisation of matrix. Values are "none", "trunc" or "tikhonov". Default value is "tikhonov".
times	The number of times internal cross-validation is performed. Default is 100.
test.size	The size of test data. Default is 0.2 (20 percent).
xval	The n-cross validation. Default is 5.
fun	The function used to summarise the times macro F1 matrices.
seed	The optional random number generator seed.
verbose	A logical defining whether a progress bar is displayed.

### Details

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e. mean(F1)). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.

### Value

An instance of class "[GenRegRes](#)".

### Author(s)

Thomas Burger and Samuel Wiczorek

### See Also

[perTurboClassification](#) and example therein.

---

phenoDisco	<i>Runs the phenoDisco algorithm.</i>
------------	---------------------------------------

---

### Description

phenoDisco is a semi-supervised iterative approach to detect new protein clusters.

### Usage

```
phenoDisco(
  object,
  fcol = "markers",
  times = 100,
  GS = 10,
  allIter = FALSE,
  p = 0.05,
  ndims = 2,
  modelNames = mclust.options("emModelNames"),
  G = 1:9,
  BPPARAM,
  tmpfile,
  seed,
  verbose = TRUE,
  dimred = c("PCA", "t-SNE"),
  ...
)
```

### Arguments

object	An instance of class MSnSet.
fcol	A character indicating the organellar markers column name in feature meta-data. Default is markers.
times	Number of runs of tracking. Default is 100.
GS	Group size, i.e how many proteins make a group. Default is 10 (the minimum group size is 4).
allIter	logical, defining if predictions for all iterations should be saved. Default is FALSE.
p	Significance level for outlier detection. Default is 0.05.
ndims	Number of principal components to use as input for the discovery analysis. Default is 2. Added in version 1.3.9.
modelNames	A vector of characters indicating the models to be fitted in the EM phase of clustering using Mclust. The help file for <code>mclust::mclustModelNames</code> describes the available models. Default model names are <code>c("EII", "VII", "EEI", "VEI", "EVI", "VVI", "EEE", "EEV", "VEV", "VVV")</code> , as returned by <code>mclust.options("emModelNames")</code> . Note that using all these possible models substantially increases the running

	time. Legacy models are c("EEE", "EEV", "VEV", "VVV"), i.e. only ellipsoidal models.
G	An integer vector specifying the numbers of mixture components (clusters) for which the BIC is to be calculated. The default is G=1:9 (as in Mclust).
BPPARAM	Support for parallel processing using the BiocParallel infrastructure. When missing (default), the default registered BiocParallelParam parameters are used. Alternatively, one can pass a valid BiocParallelParam parameter instance: SnowParam, MulticoreParam, DoparParam, ... see the BiocParallel package for details. To revert to the original serial implementation, use NULL.
tmpfile	An optional character to save a temporary MSnSet after each iteration. Ignored if missing. This is useful for long runs to track phenotypes and possibly kill the run when convergence is observed. If the run completes, the temporary file is deleted before returning the final result.
seed	An optional numeric of length 1 specifying the random number generator seed to be used. Only relevant when executed in serialised mode with BPPARAM = NULL. See BPPARAM for details.
verbose	Logical, indicating if messages are to be printed out during execution of the algorithm.
dimred	A character defining which of Principal Component Analysis ("PCA") or t-Distributed Stochastic Neighbour Embedding ("t-SNE") should be used to reduce dimensions prior to running phenoDisco novelty detection.
...	Additional arguments passed to the dimensionality reduction method. For both PCA and t-SNE, the data is scaled and centred by default, and these parameters (scale and centre for PCA, and pca_scale and pca_center for t-SNE can't be set). When using t-SNE however, it is important to tune the perplexity and max iterations parameters. See the <i>Dimensionality reduction</i> section in the pRoloc vignette for details.

## Details

The algorithm performs a phenotype discovery analysis as described in Breckels et al. Using this approach one can identify putative subcellular groupings in organelle proteomics experiments for more comprehensive validation in an unbiased fashion. The method is based on the work of Yin et al. and used iterated rounds of Gaussian Mixture Modelling using the Expectation Maximisation algorithm combined with a non-parametric outlier detection test to identify new phenotype clusters.

One requires 2 or more classes to be labelled in the data and at a very minimum of 6 markers per class to run the algorithm. The function will check and remove features with missing values using the `filterNA` method.

A parallel implementation, relying on the BiocParallel package, has been added in version 1.3.9. See the BPPARAM argument for details.

Important: Prior to version 1.1.2 the row order in the output was different from the row order in the input. This has now been fixed and row ordering is now the same in both input and output objects.

## Value

An instance of class MSnSet containing the phenoDisco predictions.

**Author(s)**

Lisa M. Breckels <lms79@cam.ac.uk>

**References**

Yin Z, Zhou X, Bakal C, Li F, Sun Y, Perrimon N, Wong ST. Using iterative cluster merging with improved gap statistics to perform online phenotype discovery in the context of high-throughput RNAi screens. *BMC Bioinformatics*. 2008 Jun 5;9:264. PubMed PMID: 18534020.

Breckels LM, Gatto L, Christoforou A, Groen AJ, Lilley KS and Trotter MWB. The Effect of Organelle Discovery upon Sub-Cellular Protein Localisation. *J Proteomics*. 2013 Aug 2;88:129-40. doi: 10.1016/j.jprot.2013.02.019. Epub 2013 Mar 21. PubMed PMID: 23523639.

**Examples**

```
## Not run:
library(pRolocdata)
data(tan2009r1)
pdres <- phenoDisco(tan2009r1, fcol = "PLSDA")
getPredictions(pdres, fcol = "pd", scol = NULL)
plot2D(pdres, fcol = "pd")

## to pre-process the data with t-SNE instead of PCA
pdres <- phenoDisco(tan2009r1, fcol = "PLSDA", dimred = "t-SNE")

## End(Not run)
```

---

plot2D

*Plot organelle assignment data and results.*

---

**Description**

Generate 2 or 3 dimensional feature distribution plots to illustrate localisation clusters. Rows/features containing NA values are removed prior to dimension reduction except for the "nipals" method. For this method, it is advised to set the method argument 'ncomp' to a low number of dimensions to avoid computing all components when analysing large datasets.

**Usage**

```
plot2D(
  object,
  fcol = "markers",
  fpch,
  unknown = "unknown",
  dims = 1:2,
  score = 1,
  method = "PCA",
  methargs,
```

```

    axsSwitch = FALSE,
    mirrorX = FALSE,
    mirrorY = FALSE,
    col,
    pch,
    cex,
    index = FALSE,
    idx.cex = 0.75,
    addLegend,
    identify = FALSE,
    plot = TRUE,
    grid = TRUE,
    ...
)

## S4 method for signature 'MSnSet'
plot3D(
  object,
  fcol = "markers",
  dims = c(1, 2, 3),
  radius1 = 0.1,
  radius2 = radius1 * 2,
  plot = TRUE,
  ...
)

```

### Arguments

object	An instance of class MSnSet.
fcol	Feature meta-data label (fData column name) defining the groups to be differentiated using different colours. Default is markers. Use NULL to suppress any colouring.
fpch	Feature meta-data label (fData column name) defining the groups to be differentiated using different point symbols.
unknown	A character (default is "unknown") defining how proteins of unknown/unlabelled localisation are labelled.
dims	A numeric of length 2 (or 3 for plot3D) defining the dimensions to be plotted. Defaults are c(1, 2) and c(1, 2, 3). Always 1:2 for MDS.
score	A numeric specifying the minimum organelle assignment score to consider features to be assigned an organelle. (not yet implemented).
method	A character describe how to transform the data or what to plot. One of "PCA" (default), "MDS", "kpca", "nipals", "t-SNE" or "lda", defining what dimensionality reduction is applied: principal component analysis (see <a href="#">prcomp</a> ), classical multidimensional scaling (see <a href="#">cmdscale</a> ), kernel PCA (see <a href="#">kpca</a> ), nipals (principal component analysis by NIPALS, non-linear iterative partial least squares which support missing values; see <a href="#">nipals</a> ) t-SNE (see <a href="#">Rtsne</a> ) or linear

discriminant analysis (see [lda](#)). The last method uses `fc01` to defined the sub-cellular clusters so that the ration between within ad between cluster variance is maximised. All the other methods are unsupervised and make use `fc01` only to annotate the plot. Prior to t-SNE, duplicated features are removed and a message informs the user if such filtering is needed.

"scree" can also be used to produce a scree plot. "hexbin" applies PCA to the data and uses bivariate binning into hexagonal cells from [hexbin](#) to emphasise cluster density.

If none is used, the data is plotted as is, i.e. without any transformation. In this case, object can either be an `MSnSet` or a `matrix` (as invisibly returned by `plot2D`). This enables to re-generate the figure without computing the dimensionality reduction over and over again, which can be time consuming for certain methods. If object is a `matrix`, an `MSnSet` containing the feature metadata must be provided in `methargs` (see below for details).

Available methods are listed in `plot2Dmethods`.

<code>methargs</code>	A list of arguments to be passed when method is called. If missing, the data will be scaled and centred prior to PCA and t-SNE (i.e. <code>Rtsne</code> 's arguments <code>pca_center</code> and <code>pca_scale</code> are set to <code>TRUE</code> ). If <code>method = "none"</code> and object is a <code>matrix</code> , then the first and only argument of <code>methargs</code> must be an <code>MSnSet</code> with matching features with object.
<code>axsSwitch</code>	A logical indicating whether the axes should be switched.
<code>mirrorX</code>	A logical indicating whether the x axis should be mirrored?
<code>mirrorY</code>	A logical indicating whether the y axis should be mirrored?
<code>col</code>	A character of appropriate length defining colours.
<code>pch</code>	A character of appropriate length defining point character.
<code>cex</code>	Character expansion.
<code>index</code>	A logical (default is <code>FALSE</code> , indicating of the feature indices should be plotted on top of the symbols).
<code>idx.cex</code>	A numeric specifying the character expansion (default is 0.75) for the feature indices. Only relevant when <code>index</code> is <code>TRUE</code> .
<code>addLegend</code>	A character indicating where to add the legend. See <a href="#">addLegend</a> for details. If missing (default), no legend is added.
<code>identify</code>	A logical (default is <code>TRUE</code> ) defining if user interaction will be expected to identify individual data points on the plot. See also <a href="#">identify</a> .
<code>plot</code>	A logical defining if the figure should be plotted. Useful when retrieving data only. Default is <code>TRUE</code> .
<code>grid</code>	A logical indicating whether a grid should be plotted. Default is <code>TRUE</code> .
<code>...</code>	Additional parameters passed to <code>plot</code> and <code>points</code> .
<code>radius1</code>	A numeric specifying the radius of feature of unknown localisation. Default is 0.1, which is specidied on the data scale. See <a href="#">plot3d</a> for details.
<code>radius2</code>	A numeric specifying the radius of marker feature. Default is <code>radius * 2</code> .

## Details

plot3D relies on the `##'` `rgl` package, that will be loaded automatically.

- Note that plot2D has been update in version 1.3.6 to support more organelle classes than colours defined in `getStockcol`. In such cases, the default colours are recycled using the default plotting characters defined in `getStockpch`. See the example for an illustration. The `alpha` argument is also depreciated in version 1.3.6. Use `setStockcol` to set colours with transparency instead. See example below.
- Version 1.11.3: to plot data as is, i.e. without any transformation, method can be set to "none" (as opposed to passing pre-computed values to method as a matrix, in previous versions). If object is an `MSnSet`, the untransformed values in the assay data will be plotted. If object is a matrix with coordinates, then a matching `MSnSet` must be passed to `methargs`.

## Value

Used for its side effects of generating a plot. Invisibly returns the 2 or 3 dimensions that are plotted.

## Author(s)

Laurent Gatto <lg390@cam.ac.uk>

## See Also

`addLegend` to add a legend to plot2D figures (the legend is added by default on plot3D) and `plotDist` for alternative graphical representation of quantitative organelle proteomics data. `plot2Ds` to overlay 2 data sets on the same PCA plot. The `plotEllipse` function can be used to visualise TAGM models on PCA plots with ellipses.

## Examples

```
library("pRolocdata")
data(dunkley2006)
plot2D(dunkley2006, fcol = NULL)
plot2D(dunkley2006, fcol = NULL, col = "black")
plot2D(dunkley2006, fcol = "markers")
addLegend(dunkley2006,
          fcol = "markers",
          where = "topright",
          cex = 0.5, bty = "n", ncol = 3)
title(main = "plot2D example")
## available methods
plot2Dmethods
plot2D(dunkley2006, fcol = NULL, method = "kpca", col = "black")
plot2D(dunkley2006, fcol = NULL, method = "kpca", col = "black",
      methargs = list(kpar = list(sigma = 1)))
plot2D(dunkley2006, method = "lda")
plot2D(dunkley2006, method = "hexbin")
## Using transparent colours
setStockcol(paste0(getStockcol(), "80"))
plot2D(dunkley2006, fcol = "markers")
```



```

## New behaviour in 1.3.6 when not enough colours
setStockcol(c("blue", "red", "green"))
getStockcol() ## only 3 colours to be recycled
getMarkers(dunkley2006)
plot2D(dunkley2006)
## reset colours
setStockcol(NULL)
plot2D(dunkley2006, method = "none") ## plotting along 2 first fractions
plot2D(dunkley2006, dims = c(3, 5), method = "none") ## plotting along fractions 3 and 5
## pre-calculate PC1 and PC2 coordinates
pca <- plot2D(dunkley2006, plot=FALSE)
head(pca)
plot2D(pca, method = "none", methargs = list(dunkley2006))

## plotting in 3 dimensions
plot3D(dunkley2006)
plot3D(dunkley2006, radius2 = 0.3)
plot3D(dunkley2006, dims = c(2, 4, 6))

```

---

plot2Ds

*Draw 2 data sets on one PCA plot*


---

### Description

Takes 2 `linkS4class{MSnSet}` instances as input to plot the two data sets on the same PCA plot. The second data points are projected on the PC1 and PC2 dimensions calculated for the first data set.

### Usage

```

plot2Ds(
  object,
  pcol,
  fcol = "markers",
  cex.x = 1,
  cex.y = 1,
  pch.x = 21,
  pch.y = 23,
  col,
  mirrorX = FALSE,
  mirrorY = FALSE,
  plot = TRUE,
  ...
)

```

### Arguments

`object` An `MSnSet` or a `MSnSetList`. In the latter case, only the two first elements of the list will be used for plotting and the others will be silently ignored.

pcol	If object is an MSnSet, a factor or the name of a phenotype variable (phenoData slot) defining how to split the single MSnSet into two or more data sets. Ignored if object is a <a href="#">MSnSetList</a> .
fcol	Feature meta-data label (fData column name) defining the groups to be differentiated using different colours. Default is markers. Use NULL to suppress any colouring.
cex.x	Character expansion for the first data set. Default is 1.
cex.y	Character expansion for the second data set. Default is 1.
pch.x	Plotting character for the first data set. Default is 21.
pch.y	Plotting character for the second data set. Default is 23.
col	A vector of colours to highlight the different classes defined by fcol. If missing (default), default colours are used (see <a href="#">getStockcol</a> ).
mirrorX	A logical indicating whether the x axis should be mirrored?
mirrorY	A logical indicating whether the y axis should be mirrored?
plot	If TRUE (default), a plot is produced.
...	Additional parameters passed to plot and points.

### Value

Used for its side effects of producing a plot. Invisibly returns an object of class plot2Ds, which is a list with the PCA analyses results (see [prcomp](#)) of the first data set and the new coordinates of the second data sets, as used to produce the plot and the respective point colours. Each of these elements can be accessed with data1, data2, col1 and code2 respectively.

### Author(s)

Laurent Gatto

### See Also

See [plot2D](#) to plot a single data set and [move2Ds](#) for a animation.

### Examples

```
library("pRolocdata")
data(tan2009r1)
data(tan2009r2)
msnl <- MSnSetList(list(tan2009r1, tan2009r2))
plot2Ds(msnl)
## tweaking the parameters
plot2Ds(list(tan2009r1, tan2009r2),
        fcol = NULL, cex.x = 1.5)
## input is 1 MSnSet containing 2 data sets
data(dunkley2006)
plot2Ds(dunkley2006, pcol = "replicate")
## no plot, just the data
res <- plot2Ds(dunkley2006, pcol = "replicate",
```

```
      plot = FALSE)
res
head(data1(res))
head(col1(res))
```

---

plotConsProfiles      *Plot marker consensus profiles.*

---

## Description

The function plots marker consensus profiles obtained from `mrkConsProfile`

## Usage

```
plotConsProfiles(object, order = NULL, plot = TRUE)
```

## Arguments

`object`      A matrix containing marker consensus profiles as output from `mrkConsProfiles()`.  
`order`      Order for markers (optional).  
`plot`      A logical(1) defining whether the heatmap should be plotted. Default is TRUE.

## Value

Invisibly returns ggplot2 object.

## Author(s)

Tom Smith

## Examples

```
library("pRolocdata")
data(E14TG2aS1)
hc <- mrkHClust(E14TG2aS1, plot = FALSE)
mm <- getMarkerClasses(E14TG2aS1)
ord <- levels(factor(mm))[order.dendrogram(hc)]
fmat <- mrkConsProfiles(E14TG2aS1)
plotConsProfiles(fmat, order = ord)
```

plotDist

*Plots the distribution of features across fractions***Description**

Produces a line plot showing the feature abundances across the fractions.

**Usage**

```
plotDist(
  object,
  markers,
  fcol = NULL,
  mcol = "steelblue",
  pcol = getUnknowncol(),
  alpha = 0.3,
  type = "b",
  lty = 1,
  fractions = sampleNames(object),
  ylab = "Intensity",
  xlab = "Fractions",
  ylim,
  ...
)
```

**Arguments**

object	An instance of class MSnSet.
markers	A character, numeric or logical of appropriate length and or content used to subset object and define the organelle markers.
fcol	Feature meta-data label (fData column name) defining the groups to be differentiated using different colours. If NULL (default) ignored and mcol and pcol are used.
mcol	A character define the colour of the marker features. Default is "steelblue".
pcol	A character define the colour of the non-markers features. Default is the colour used for features of unknown localisation, as returned by <a href="#">getUnknowncol</a> .
alpha	A numeric defining the alpha channel (transparency) of the points, where $0 \leq \alpha \leq 1$ , 0 and 1 being completely transparent and opaque.
type	Character string defining the type of lines. For example "p" for points, "l" for lines, "b" for both. See <a href="#">plot</a> for all possible types.
lty	Vector of line types for the marker profiles. Default is 1 (solid). See <a href="#">par</a> for details.
fractions	A character defining the phenoData variable to be used to label the fraction along the x axis. Default is to use <code>sampleNames(object)</code> .

ylab	y-axis label. Default is "Intensity".
xlab	x-axis label. Default is "Fractions".
ylim	A numeric vector of length 2, giving the y coordinates range.
...	Additional parameters passed to <code>plot</code> .

**Value**

Used for its side effect of producing a feature distribution plot. Invisibly returns the data matrix.

**Author(s)**

Laurent Gatto

**Examples**

```
library("pRolocdata")
data(tan2009r1)
j <- which(fData(tan2009r1)$markers == "mitochondrion")
i <- which(fData(tan2009r1)$PLSDA == "mitochondrion")
plotDist(tan2009r1[i, ], markers = featureNames(tan2009r1)[j])
plotDist(tan2009r1[i, ], markers = featureNames(tan2009r1)[j],
         fractions = "Fractions")
## plot and colour all marker profiles
tanmrk <- markerMSnSet(tan2009r1)
plotDist(tanmrk, fcol = "markers")
```

---

plotEllipse	<i>A function to plot probability ellipses on marker PCA plots to visualise and assess TAGM models.</i>
-------------	---

---

**Description**

Note that when running PCA, this function does not scale the data (centring is performed), as opposed to `[plot2D()]`. Only marker proteins are displayed; the protein of unknown location, that are not used to estimate the MAP parameters, are filtered out.

**Usage**

```
plotEllipse(object, params, dims = c(1, 2), method = "MAP", ...)
```

**Arguments**

object	An [ <code>'MSnbase::MSnset'</code> ] containing quantitative spatial proteomics data.
params	An [ <code>'MAPPparams'</code> ] with the TAGM-MAP parameters, as generated by <code>'tagmMapTrain'</code> .
dims	A <code>'numeric(2)'</code> with the principal components along which to project the data. Default is <code>'c(1, 2)'</code> .
method	The method used. Currently <code>'"MAP"'</code> only.
...	Additional parameters passed to <code>[plot2D()]</code> .

**Value**

A PCA plot of the marker data with probability ellipses. The outer ellipse contains 99 probability whilst the middle and inner ellipses contain 95 and 90 clusters are represented by black circumpunct (circled dot).

**See Also**

[plot2D()] to visualise spatial proteomics data using various dimensionality reduction methods. For details about TAGM models, see [tagmPredict()] and the \*pRoloc-bayesian\* vignette.

---

plsdaClassification    *plsda classification*

---

**Description**

Classification using the partial least square discriminant analysis algorithm.

**Usage**

```
plsdaClassification(
  object,
  assessRes,
  scores = c("prediction", "all", "none"),
  ncomp,
  fcol = "markers",
  ...
)
```

**Arguments**

object	An instance of class "MSnSet".
assessRes	An instance of class "GenRegRes", as generated by <a href="#">plsdaOptimisation</a> .
scores	One of "prediction", "all" or "none" to report the score for the predicted class only, for all classes or none.
ncomp	If assessRes is missing, a ncomp must be provided.
fcol	The feature meta-data containing marker definitions. Default is markers.
...	Additional parameters passed to <a href="#">plsda</a> from package caret.

**Value**

An instance of class "MSnSet" with plsda and plsda.scores feature variables storing the classification results and scores respectively.

**Author(s)**

Laurent Gatto

## Examples

```
## not running this one for time considerations
library(pRolocdata)
data(dunkley2006)
## reducing parameter search space and iterations
params <- plsdaOptimisation(dunkley2006, ncomp = c(3, 10), times = 2)
params
plot(params)
f1Count(params)
levelPlot(params)
getParams(params)
res <- plsdaClassification(dunkley2006, params)
getPredictions(res, fcol = "plsda")
getPredictions(res, fcol = "plsda", t = 0.9)
plot2D(res, fcol = "plsda")
```

---

plsdaOptimisation      *plsda parameter optimisation*

---

## Description

Classification parameter optimisation for the partial least square discriminant analysis algorithm.

## Usage

```
plsdaOptimisation(
  object,
  fcol = "markers",
  ncomp = 2:6,
  times = 100,
  test.size = 0.2,
  xval = 5,
  fun = mean,
  seed,
  verbose = TRUE,
  ...
)
```

## Arguments

object	An instance of class " <a href="#">MSnSet</a> ".
fcol	The feature meta-data containing marker definitions. Default is markers.
ncomp	The hyper-parameter. Default values are 2:6.
times	The number of times internal cross-validation is performed. Default is 100.
test.size	The size of test data. Default is 0.2 (20 percent).

xval	The n-cross validation. Default is 5.
fun	The function used to summarise the xval macro F1 matrices.
seed	The optional random number generator seed.
verbose	A logical defining whether a progress bar is displayed.
...	Additional parameters passed to <a href="#">plsda</a> from package caret.

### Details

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e. mean(F1)). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.

### Value

An instance of class "[GenRegRes](#)".

### Author(s)

Laurent Gatto

### See Also

[plsdaClassification](#) and example therein.

---

pRolocmarkers

*Organelle markers*

---

### Description

This function retrieves a list of organelle markers or, if no species is provided, prints a description of available marker sets. The markers can be added to an MSnSet using the [addMarkers](#) function.

### Usage

```
pRolocmarkers(species)
```

### Arguments

species      The species of interest.



**Details**

The markers have been contributed by various members of the Cambridge Centre for Proteomics, in particular Dr Dan Nightingale for yeast, Dr Andy Christoforou and Dr Claire Mulvey for human, Dr Arnoud Groen for Arabodopsis and Dr Claire Mulvey for mouse. In addition, original (curated) markers from the pRolocdata datasets have been extracted (see pRolocdata for details and references). Curation involved verification of publicly available subcellular localisation annotation based on the curators knowledge of the organelles/proteins considered and tracing the original statement in the literature.

These markers are provided as a starting point to generate reliable sets of organelle markers but still need to be verified against any new data in the light of the quantitative data and the study conditions.

**Value**

Prints a description of the available marker lists if species is missing or a named character with organelle markers.

**Author(s)**

Laurent Gatto

**See Also**

[addMarkers](#) to add markers to an MSnSet and [markers](#) for more information about marker encoding.

**Examples**

```
pRolocmarkers()
table(pRolocmarkers("atha"))
table(pRolocmarkers("hsap"))
```

---

QSep-class

*Quantify resolution of a spatial proteomics experiment*

---

**Description**

The QSep infrastructure provide a way to quantify the resolution of a spatial proteomics experiment, i.e. to quantify how well annotated sub-cellular clusters are separated from each other.

The QSep function calculates all between and within cluster average distances. These distances are then divided column-wise by the respective within cluster average distance. For example, for a dataset with only 2 spatial clusters, we would obtain

$$\begin{array}{ccc}
 & c_1 & c_2 \\
 c_1 & d_{11} & d_{12} \\
 c_2 & d_{21} & d_{22}
 \end{array}$$

Normalised distance represent the ratio of between to within average distances, i.e. how much bigger the average distance between cluster  $c_i$  and  $c_j$  is compared to the average distance within cluster  $c_i$ .

$$\begin{array}{cc} & \begin{array}{c} c_1 \\ c_2 \end{array} \\ \begin{array}{c} c_1 \\ c_2 \end{array} & \begin{array}{cc} & \begin{array}{c} c_1 \\ c_2 \end{array} \\ 1 & \frac{d_{12}}{d_1^2} \\ \frac{d_{21}}{d_1^1} & 1 \end{array} \end{array}$$

Note that the normalised distance matrix is not symmetric anymore and the normalised distance ratios are proportional to the tightness of the reference cluster (along the columns).

Missing values only affect the fractions containing the NA when the distance is computed (see the example below) and further used when calculating mean distances. Few missing values are expected to have negligible effect, but data with a high proportion of missing data will produce skewed distances. In QSep, we take a conservative approach, using the data as provided by the user, and expect that the data missingness is handled before proceeding with this or any other analysis.

### Objects from the Class

Objects can be created by calls using the constructor QSep (see below).

### Slots

**x**: Object of class "matrix" containing the pairwise distance matrix, accessible with `qseq(. , norm = FALSE)`.

**xnorm**: Object of class "matrix" containing the normalised pairwise distance matrix, accessible with `qsep(. , norm = TRUE)` or `qsep(.)`.

**object**: Object of class "character" with the variable name of `MSnSet` object that was used to generate the QSep object.

**.\_\_classVersion\_\_**: Object of class "Versions" storing the class version of the object.

### Extends

Class "Versioned", directly.

### Methods and functions

**QSeq** signature(object = "MSnSet", fcol = "character"): constructor for QSep objects. The fcol argument defines the name of the feature variable that annotates the sub-cellular clusters. Non-marker proteins, that are marked as "unknown" are automatically removed prior to distance calculation.

**qsep** signature(object = "QSep", norm = "logical"): accessor for the normalised (when norm is TRUE, which is default) and raw (when norm is FALSE) pairwise distance matrices.

**names** signature(object = "QSep"): method to retrieve the names of the sub-cellular clusters originally defined in QSep's fcol argument. A replacement method `names(.) <-` is also available.

**summary** signature(object = "QSep", ..., verbose = "logical"): Invisible return all between cluster average distances and prints (when verbose is TRUE, default) a summary of those.

**levelPlot** signature(object = "QSep", norm = "logical", ...): plots an annotated heatmap of all normalised pairwise distances. norm (default is TRUE) defines whether normalised distances should be plotted. Additional arguments ... are passed to the `levelplot`.

**plot** signature(object = "QSep", norm = "logical", ...): produces a boxplot of all normalised pairwise distances. The red points represent the within average distance and black points between average distances. norm (default is TRUE) defines whether normalised distances should be plotted.

### Author(s)

Laurent Gatto <lg390@cam.ac.uk>

### References

Assessing sub-cellular resolution in spatial proteomics experiments Laurent Gatto, Lisa M Breckels, Kathryn S Lilley bioRxiv 377630; doi: <https://doi.org/10.1101/377630>

### Examples

```
## Test data from Christoforou et al. 2016
library("pRolocdata")
data(hyperLOPIT2015)

## Create the object and get a summary
hlq <- QSep(hyperLOPIT2015)
hlq
summary(hlq)

## mean distance matrix
qsep(hlq, norm = FALSE)

## normalised average distance matrix
qsep(hlq)

## Update the organelle cluster names for better
## rendering on the plots
names(hlq) <- sub("/", "\n", names(hlq))
names(hlq) <- sub(" - ", "\n", names(hlq))
names(hlq)

## Heatmap of the normalised intensities
levelPlot(hlq)

## Boxplot of the normalised intensities
par(mar = c(3, 10, 2, 1))
plot(hlq)

## Boxplot of all between cluster average distances
x <- summary(hlq, verbose = FALSE)
boxplot(x)

## Missing data example, for 4 proteins and 3 fractions
```

```

x <- rbind(c(1.1, 1.2, 1.3), rep(1, 3), c(NA, 1, 1), c(1, 1, NA))
rownames(x) <- paste0("P", 1:4)
colnames(x) <- paste0("F", 1:3)

## P1 is the reference, against which we will calculate distances. P2
## has a complete profile, producing the *real* distance. P3 and P4 have
## missing values in the first and last fraction respectively.
x

## If we drop F1 in P3, which represents a small difference of 0.1, the
## distance only considers F2 and F3, and increases. If we drop F3 in
## P4, which represents a large distance of 0.3, the distance only
## considers F1 and F2, and decreases. dist(x)

```

---

rfClassification	<i>rf classification</i>
------------------	--------------------------

---

## Description

Classification using the random forest algorithm.

## Usage

```

rfClassification(
  object,
  assessRes,
  scores = c("prediction", "all", "none"),
  mtry,
  fcol = "markers",
  ...
)

```

## Arguments

object	An instance of class " <a href="#">MSnSet</a> ".
assessRes	An instance of class " <a href="#">GenRegRes</a> ", as generated by <a href="#">rfOptimisation</a> .
scores	One of "prediction", "all" or "none" to report the score for the predicted class only, for all classes or none.
mtry	If assessRes is missing, a mtry must be provided.
fcol	The feature meta-data containing marker definitions. Default is markers.
...	Additional parameters passed to <a href="#">randomForest</a> from package randomForest.

## Value

An instance of class "[MSnSet](#)" with rf and rf.scores feature variables storing the classification results and scores respectively.

**Author(s)**

Laurent Gatto

**Examples**

```
library(pRolocdata)
data(dunkley2006)
## reducing parameter search space and iterations
params <- rfOptimisation(dunkley2006, mtry = c(2, 5, 10), times = 3)
params
plot(params)
f1Count(params)
levelPlot(params)
getParams(params)
res <- rfClassification(dunkley2006, params)
getPredictions(res, fcol = "rf")
getPredictions(res, fcol = "rf", t = 0.75)
plot2D(res, fcol = "rf")
```

---

rfOptimisation

*svm parameter optimisation*

---

**Description**

Classification parameter optimisation for the random forest algorithm.

**Usage**

```
rfOptimisation(
  object,
  fcol = "markers",
  mtry = NULL,
  times = 100,
  test.size = 0.2,
  xval = 5,
  fun = mean,
  seed,
  verbose = TRUE,
  ...
)
```

**Arguments**

object	An instance of class <code>"MSnSet"</code> .
fcol	The feature meta-data containing marker definitions. Default is markers.
mtry	The hyper-parameter. Default value is NULL.
times	The number of times internal cross-validation is performed. Default is 100.

test.size	The size of test data. Default is 0.2 (20 percent).
xval	The n-cross validation. Default is 5.
fun	The function used to summarise the xval macro F1 matrices.
seed	The optional random number generator seed.
verbose	A logical defining whether a progress bar is displayed.
...	Additional parameters passed to <a href="#">randomForest</a> from package randomForest.

### Details

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e. mean(F1)). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.

### Value

An instance of class "[GenRegRes](#)".

### Author(s)

Laurent Gatto

### See Also

[rfClassification](#) and example therein.

---

sampleMSnSet	<i>Extract a stratified sample of an MSnSet</i>
--------------	---

---

### Description

This function extracts a stratified sample of an MSnSet.

### Usage

```
sampleMSnSet(object, fcol = "markers", size = 0.2, seed)
```

### Arguments

object	An instance of class <a href="#">MSnSet</a>
fcol	The feature meta-data column name containing the marker (vector or matrix) definitions on which the MSnSet will be stratified. Default is markers.
size	The size of the stratified sample to be extracted. Default is 0.2 (20 percent).
seed	The optional random number generator seed.

**Value**

A stratified sample (according to the defined fcol) which is an instance of class "MSnSet".

**Author(s)**

Lisa Breckels

**See Also**

[testMSnSet](#) [unknownMSnSet](#) [markerMSnSet](#). See [markers](#) for details about markers encoding.

**Examples**

```
library(pRocdata)
data(tan2009r1)
dim(tan2009r1)
smp <- sampleMSnSet(tan2009r1, fcol = "markers")
dim(smp)
getMarkers(tan2009r1)
getMarkers(smp)
```

---

setLisacol

*Manage default colours and point characters*

---

**Description**

These functions allow to get/set the colours and point character that are used when plotting organelle clusters and unknown features. These values are parametrised at the session level. Two palettes are available: the default palette (previously *Lisa's colours*) containing 30 colours and the old (original) palette, containing 13 colours.

**Usage**

```
setLisacol()
getLisacol()
getOldcol()
setOldcol()
getStockcol()
setStockcol(cols)
getStockpch()
setStockpch(pchs)
```

```
getUnknowncol()
```

```
setUnknowncol(col)
```

```
getUnknownpch()
```

```
setUnknownpch(pch)
```

### Arguments

cols	A vector of colour characters or NULL, which sets the colours to the default values.
pchs	A vector of numeric or NULL, which sets the point characters to the default values.
col	A colour character or NULL, which sets the colour to #E7E7E7 (grey91), the default colour for unknown features.
pch	A numeric vector of length 1 or NULL, which sets the point character to 21, the default.

### Value

The set functions set (and invisibly returns) colours. The get functions returns a character vector of colours. For the pch functions, numerics rather than characters.

### Author(s)

Laurent Gatto

### Examples

```
## defaults for clusters
getStockcol()
getStockpch()
## unknown features
getUnknownpch()
getUnknowncol()
## an example
library(pRolocdata)
data(dunkley2006)
par(mfrow = c(2, 1))
plot2D(dunkley2006, fcol = "markers", main = 'Default colours')
setUnknowncol("black")
plot2D(dunkley2006, fcol = "markers", main = 'setUnknowncol("black")')
getUnknowncol()
setUnknowncol(NULL)
getUnknowncol()
getStockcol()
getOldcol()
```



---

showGOEvidenceCodes    *GO Evidence Codes*

---

**Description**

This function prints a textual description of the Gene Ontology evidence codes.

**Usage**

```
showGOEvidenceCodes()
```

```
getGOEvidenceCodes()
```

**Value**

These functions are used for their side effects of printing evidence codes and their description.

**Author(s)**

Laurent Gatto

**Examples**

```
showGOEvidenceCodes()  
getGOEvidenceCodes()
```

---

spatial2D                    *Uncertainty plot in localisation probabilities*

---

**Description**

Produces a pca plot with spatial variation in localisation probabilities

**Usage**

```
spatial2D(  
  object,  
  dims = c(1, 2),  
  cov.function = fields::wendland.cov,  
  theta = 1,  
  derivative = 2,  
  k = 1,  
  breaks = c(0.99, 0.95, 0.9, 0.85, 0.8, 0.75, 0.7),  
  aspect = 0.5  
)
```

**Arguments**

object	A valid object of class MSnset with mcmc prediction results from tagmMCMCPredict
dims	The PCA dimension in which to project the data, default is c(1, 2)
cov.function	The covariance function used default is wendland.cov. See fields package.
theta	A hyperparameter to the covariance function. See fields package. Default is 1.
derivative	The number of derivative of the wendland kernel. See fields package. Default is 2.
k	A hyperparameter to the covariance function. See fields package. Default is 1.
breaks	Probability values at which to draw the contour bands. Default is c(0.99, 0.95, 0.9, 0.85, 0.8, 0.75, 0.7)
aspect	A argument to change the plotting aspect of the PCA

**Value**

Used for side effect of producing plot. Invisibly returns an ggplot object that can be further manipulated

**Author(s)**

Oliver M. Crook <omc25@cam.ac.uk>

**Examples**

```
## Not run:
library("pRolocdata")
data("tan2009r1")

tanres <- tagmMcmcTrain(object = tan2009r1)
tanres <- tagmMcmcProcess(tanres)
tan2009r1 <- tagmMcmcPredict(object = tan2009r1, params = tanres, probJoint = TRUE)
spatial2D(object = tan2009r1)

## End(Not run)
```

---

SpatProtVis-class

*Class* SpatProtVis

---

**Description**

A class for spatial proteomics visualisation, that upon instantiation, pre-computes all defined visualisations. Objects can be created with the SpatProtVis constructor and visualised with the plot method.

The class is essentially a wrapper around several calls to `plot2D` that stores the dimensionality reduction outputs, and is likely to be updated in the future.

## Usage

```
SpatProtVis(x, methods, dims, methargs, ...)
```

## Arguments

x	An instance of class <a href="#">MSnSet</a> to visualise.
methods	Dimensionality reduction methods to be used to visualise the data. Must be contained in <a href="#">plot2Dmethods</a> (except "scree"). See <a href="#">plot2D</a> for details.
dims	A list of numerics defining dimensions used for plotting. Default are 1 and 2. If provided, the length of this list must be identical to the length of methods.
methargs	A list of additional arguments to be passed for each visualisation method. If provided, the length of this list must be identical to the length of methods.
...	Additional arguments. Currently ignored.

## Slots

**vismats:** A "list" of matrices containing the feature projections in 2 dimensions.

**data:** The original spatial proteomics data stored as an "MSnSet".

**methargs:** A "list" of additional plotting arguments.

**objname:** A "character" defining how to name the dataset. By default, this is set using the variable name used at object creation.

## Methods

**plot:** Generates the figures for the respective methods and additional arguments defined in the constructor. If used in an interactive session, the user is prompted to press 'Return' before new figures are displayed.

**show:** A simple textual summary of the object.

## Author(s)

Laurent Gatto <lg390@cam.ac.uk>

## See Also

The data for the individual visualisations is created by [plot2D](#).

## Examples

```
library("pRolocdata")
data(dunkley2006)
## Default parameters for a set of methods
## (in the interest of time, don't use t-SNE)
m <- c("PCA", "MDS", "kpca")
vis <- SpatProtVis(dunkley2006, methods = m)
vis
plot(vis)
```

```

plot(vis, legend = "topleft")

## Setting method arguments
margs <- c(list(kpar = list(sigma = 0.1)),
           list(kpar = list(sigma = 1.0)),
           list(kpar = list(sigma = 10)),
           list(kpar = list(sigma = 100)))
vis <- SpatProtVis(dunkley2006,
                  methods = rep("kpca", 4),
                  methargs = margs)
par(mfrow = c(2, 2))
plot(vis)

## Multiple PCA plots but different PCs
dims <- list(c(1, 2), c(3, 4))
vis <- SpatProtVis(dunkley2006, methods = c("PCA", "PCA"), dims = dims)
plot(vis)

```

---

subsetMarkers

*Subsets markers*


---

### Description

Subsets a matrix of markers by specific terms

### Usage

```
subsetMarkers(object, fcol = "GOAnnotations", keep)
```

### Arguments

object	An instance of class MSnSet.
fcol	The name of the markers matrix. Default is GOAnnotations.
keep	Integer or character vector specifying the columns to keep in the markers matrix, as defined by fcol.

### Value

An updated MSnSet

### Author(s)

Lisa M Breckels

### See Also

addGoAnnotations and example therein.

---

svmClassification      *svm classification*

---

## Description

Classification using the support vector machine algorithm.

## Usage

```
svmClassification(  
  object,  
  assessRes,  
  scores = c("prediction", "all", "none"),  
  cost,  
  sigma,  
  fcol = "markers",  
  ...  
)
```

## Arguments

object	An instance of class "MSnSet".
assessRes	An instance of class "GenRegRes", as generated by <a href="#">svmOptimisation</a> .
scores	One of "prediction", "all" or "none" to report the score for the predicted class only, for all classes or none.
cost	If assessRes is missing, a cost must be provided.
sigma	If assessRes is missing, a sigma must be provided.
fcol	The feature meta-data containing marker definitions. Default is markers.
...	Additional parameters passed to <a href="#">svm</a> from package e1071.

## Value

An instance of class "MSnSet" with svm and svm.scores feature variables storing the classification results and scores respectively.

## Author(s)

Laurent Gatto

## Examples

```
library(pRolocdata)  
data(dunkley2006)  
## reducing parameter search space and iterations  
params <- svmOptimisation(dunkley2006, cost = 2^seq(-2,2,2), sigma = 10^seq(-1, 1, 1), times = 3)  
params
```

```

plot(params)
f1Count(params)
levelPlot(params)
getParams(params)
res <- svmClassification(dunkley2006, params)
getPredictions(res, fcol = "svm")
getPredictions(res, fcol = "svm", t = 0.75)
plot2D(res, fcol = "svm")

```

---

svmOptimisation

*svm parameter optimisation*


---

## Description

Classification parameter optimisation for the support vector machine algorithm.

## Usage

```

svmOptimisation(
  object,
  fcol = "markers",
  cost = 2^(-4:4),
  sigma = 10^(-3:2),
  times = 100,
  test.size = 0.2,
  xval = 5,
  fun = mean,
  seed,
  verbose = TRUE,
  ...
)

```

## Arguments

<code>object</code>	An instance of class <code>"MSnSet"</code> .
<code>fcol</code>	The feature meta-data containing marker definitions. Default is <code>markers</code> .
<code>cost</code>	The hyper-parameter. Default values are $2^{-4:4}$ .
<code>sigma</code>	The hyper-parameter. Default values are $10^{-2:3}$ .
<code>times</code>	The number of times internal cross-validation is performed. Default is 100.
<code>test.size</code>	The size of test data. Default is 0.2 (20 percent).
<code>xval</code>	The n-cross validation. Default is 5.
<code>fun</code>	The function used to summarise the <code>xval</code> macro F1 matrices.
<code>seed</code>	The optional random number generator seed.
<code>verbose</code>	A logical defining whether a progress bar is displayed.
<code>...</code>	Additional parameters passed to <code>svm</code> from package <code>e1071</code> .

**Details**

Note that when performance scores precision, recall and (macro) F1 are calculated, any NA values are replaced by 0. This decision is motivated by the fact that any class that would have either a NA precision or recall would result in an NA F1 score and, eventually, a NA macro F1 (i.e. mean(F1)). Replacing NAs by 0s leads to F1 values of 0 and a reduced yet defined final macro F1 score.

**Value**

An instance of class "[GenRegRes](#)".

**Author(s)**

Laurent Gatto

**See Also**

[svmClassification](#) and example therein.

---

tagmMcmcTrain

*Localisation of proteins using the TAGM MCMC method*

---

**Description**

These functions implement the T augmented Gaussian mixture (TAGM) model for mass spectrometry-based spatial proteomics datasets using Markov-chain Monte-Carlo (MCMC) for inference.

**Usage**

```
tagmMcmcTrain(  
  object,  
  fcol = "markers",  
  method = "MCMC",  
  numIter = 1000L,  
  burnin = 100L,  
  thin = 5L,  
  mu0 = NULL,  
  lambda0 = 0.01,  
  nu0 = NULL,  
  S0 = NULL,  
  beta0 = NULL,  
  u = 2,  
  v = 10,  
  numChains = 4L,  
  BPPARAM = BiocParallel::bpparam()  
)  
  
tagmMcmcPredict(  
  object,  
  fcol = "markers",  
  method = "MCMC",  
  numIter = 1000L,  
  burnin = 100L,  
  thin = 5L,  
  mu0 = NULL,  
  lambda0 = 0.01,  
  nu0 = NULL,  
  S0 = NULL,  
  beta0 = NULL,  
  u = 2,  
  v = 10,  
  numChains = 4L,  
  BPPARAM = BiocParallel::bpparam()  
)
```

```

    object,
    params,
    fcol = "markers",
    probJoint = FALSE,
    probOutlier = TRUE
)

tagmPredict(
  object,
  params,
  fcol = "markers",
  probJoint = FALSE,
  probOutlier = TRUE
)

tagmMcmcProcess(params)

```

### Arguments

object	An <code>MSnbase::MSnSet</code> containing the spatial proteomics data to be passed to <code>tagmMcmcTrain</code> and <code>tagmPredict</code> .
fcol	The feature meta-data containing marker definitions. Default is <code>markers</code> .
method	A <code>character()</code> describing the inference method for the TAGM algorithm. Default is <code>"MCMC"</code> .
numIter	The number of iterations of the MCMC algorithm. Default is 1000.
burnin	The number of samples to be discarded from the beginning of the chain. Default is 100.
thin	The thinning frequency to be applied to the MCMC chain. Default is 5.
mu0	The prior mean. Default is <code>colMeans</code> of the expression data.
lambda0	The prior shrinkage. Default is 0.01.
nu0	The prior degree of freedom. Default is <code>ncol(exprs(object)) + 2</code>
S0	The prior inverse-wishart scale matrix. Empirical prior used by default.
beta0	The prior Dirichlet distribution concentration. Default is 1 for each class.
u	The prior shape parameter for <code>Beta(u, v)</code> . Default is 2
v	The prior shape parameter for <code>Beta(u, v)</code> . Default is 10.
numChains	The number of parallel chains to be run. Default is 4.
BPPARAM	Support for parallel processing using the <code>BiocParallel</code> infrastructure. When missing (default), the default registered <code>BiocParallelParam</code> parameters are used. Alternatively, one can pass a valid <code>BiocParallelParam</code> parameter instance: <code>SnowParam</code> , <code>MulticoreParam</code> , <code>DoparParam</code> , ... see the <code>BiocParallel</code> package for details.
params	An instance of class <code>MCMCParams</code> , as generated by <code>tagmMcmcTrain()</code> .
probJoint	A <code>logical(1)</code> indicating whether to return the joint probability matrix, i.e. the probability for all classes as a new <code>tagm.mcmc.joint</code> feature variable.



probOutlier A logical(1) indicating whether to return the probability of being an outlier as a new tagm.mcmc.outlier feature variable. A high value indicates that the protein is unlikely to belong to any annotated class (and is hence considered an outlier).

### Details

The tagmMcmcTrain function generates the samples from the posterior distributions (object or class MCMCParams) based on an annotated quantitative spatial proteomics dataset (object of class MSnbase:MSnSet). Both are then passed to the tagmPredict function to predict the sub-cellular localisation of protein of unknown localisation. See the *pRoloc-bayesian* vignette for details and examples. In this implementation, if numerical instability is detected in the covariance matrix of the data a small multiple of the identity is added. A message is printed if this conditioning step is performed.

### Value

tagmMcmcTrain returns an instance of class MCMCParams.

tagmMcmcPredict returns an instance of class MSnbase:MSnSet containing the localisation predictions as a new tagm.mcmc.allocation feature variable. The allocation probability is encoded as tagm.mcmc.probability (corresponding to the mean of the distribution probability). In addition the upper and lower quantiles of the allocation probability distribution are available as tagm.mcmc.probability.lowerquantile and tagm.mcmc.probability.upperquantile feature variables. The Shannon entropy is available in the tagm.mcmc.mean.shannon feature variable, measuring the uncertainty in the allocations (a high value representing high uncertainty; the highest value is the natural logarithm of the number of classes).

tagmMcmcProcess returns an instance of class MCMCParams with its summary slot populated.

### References

*A Bayesian Mixture Modelling Approach For Spatial Proteomics* Oliver M Crook, Claire M Mulvey, Paul D. W. Kirk, Kathryn S Lilley, Laurent Gatto bioRxiv 282269; doi: <https://doi.org/10.1101/282269>

### See Also

The `plotEllipse()` function can be used to visualise TAGM models on PCA plots with ellipses.

---

testMarkers

*Tests marker class sizes*

---

### Description

Tests if the marker class sizes are large enough for the parameter optimisation scheme, i.e. the size is greater than  $xval + n$ , where the default `xval` is 5 and `n` is 2. If the test is unsuccessful, a warning is thrown.

**Usage**

```
testMarkers(object, xval = 5, n = 2, fcol = "markers", error = FALSE)
```

**Arguments**

object	An instance of class " <a href="#">MSnSet</a> ".
xval	The number cross-validation partitions. See the xval argument in the parameter optimisation function(s). Default is 5.
n	Number of additional examples.
fcol	The name of the prediction column in the featureData slot. Default is "markers".
error	A logical specifying if an error should be thrown, instead of a warning.

**Details**

In case the test indicates that a class contains too few examples, it is advised to either add some or, if not possible, to remove the class altogether (see [minMarkers](#)) as the parameter optimisation is likely to fail or, at least, produce unreliable results for that class.

**Value**

If successful, the test invisibly returns NULL. Else, it invisibly returns the names of the classes that have too few examples.

**Author(s)**

Laurent Gatto

**See Also**

[getMarkers](#) and [minMarkers](#)

**Examples**

```
library("pRolocdata")
data(dunkley2006)
getMarkers(dunkley2006)
testMarkers(dunkley2006)
toosmall <- testMarkers(dunkley2006, xval = 15)
toosmall
try(testMarkers(dunkley2006, xval = 15, error = TRUE))
```

---

testMSnSet	<i>Create a stratified 'test' MSnSet</i>
------------	--

---

### Description

This function creates a stratified 'test' MSnSet which can be used for algorithmic development. A "MSnSet" containing only the marker proteins, as defined in `fcol`, is returned with a new feature data column appended called `test` in which a stratified subset of these markers has been relabelled as 'unknowns'.

### Usage

```
testMSnSet(object, fcol = "markers", size = 0.2, seed)
```

### Arguments

<code>object</code>	An instance of class "MSnSet"
<code>fcol</code>	The feature meta-data column name containing the marker definitions on which the data will be stratified. Default is <code>markers</code> .
<code>size</code>	The size of the data set to be extracted. Default is 0.2 (20 percent).
<code>seed</code>	The optional random number generator seed.

### Value

An instance of class "MSnSet" which contains only the proteins that have a labelled localisation i.e. the marker proteins, as defined in `fcol` and a new column in the feature data slot called `test` which has part of the labels relabelled as "unknown" class (the number of proteins renamed as "unknown" is according to the parameter `size`).

### Author(s)

Lisa Breckels

### See Also

[sampleMSnSet](#) [unknownMSnSet](#) [markerMSnSet](#)

### Examples

```
library(pRolocdata)
data(tan2009r1)
sample <- testMSnSet(tan2009r1)
getMarkers(sample, "test")
all(dim(sample) == dim(markerMSnSet(tan2009r1)))
```

---

thetas

*Draw matrix of thetas to test*

---

### Description

The possible weights to be considered is a sequence from 0 (favour auxiliary data) to 1 (favour primary data). Each possible combination of weights for nclass classes must be tested. The thetas function produces a weight matrix for nclass columns (one for each class) with all possible weight combinations (number of rows).

### Usage

```
thetas(nclass, by = 0.5, length.out, verbose = TRUE)
```

### Arguments

nclass	Number of marker classes
by	The increment of the weights. One of 1, 0.5, 0.25, 2, 0.1 or 0.05.
length.out	The desired length of the weight sequence.
verbose	A logical indicating if the weight sequences should be printed out. Default is TRUE.

### Value

A matrix with all possible theta weight combinations.

### Author(s)

Lisa Breckels

### Examples

```
dim(thetas(4, by = 0.5))
dim(thetas(4, by = 0.2))
dim(thetas(5, by = 0.2))
dim(thetas(5, length.out = 5))
dim(thetas(6, by = 0.2))
```

---

undocumented	<i>Undocumented/unexported entries</i>
--------------	--

---

**Description**

This is just a dummy entry for methods from unexported classes that generate warnings during package checking.

**Author(s)**

Laurent Gatto <lg390@cam.ac.uk>

---

zerosInBinMSnSet	<i>Compute the number of non-zero values in each marker classes</i>
------------------	---

---

**Description**

The function assumes that its input is a binary MSnSet and computes, for each marker class, the number of non-zero expression profiles. The function is meant to be used to produce heatmaps (see the example) and visualise binary (such as GO) MSnSet objects and assess their utility: all zero features/classes will not be informative at all (and can be filtered out with [filterBinMSnSet](#)) while features/classes with many annotations (GO terms) are likely not be be informative either.

**Usage**

```
zerosInBinMSnSet(object, fcol = "markers", as.matrix = TRUE, percent = TRUE)
```

**Arguments**

object	An instance of class MSnSet with binary data.
fcol	A character defining the feature data variable to be used as markers. Default is "markers".
as.matrix	If TRUE (default) the data is formatted and returned as a matrix. Otherwise, a list is returned.
percent	If TRUE, percentages are returned. Otherwise, absolute values.

**Value**

A matrix or a list indicating the number of non-zero value per marker class.

**Author(s)**

Laurent Gatto

**See Also**[filterBinMSnSet](#)**Examples**

```
library(pRolocdata)
data(hyperLOPIT2015goCC)
zerosInBinMSnSet(hyperLOPIT2015goCC)
zerosInBinMSnSet(hyperLOPIT2015goCC, percent = FALSE)
pal <- colorRampPalette(c("white", "blue"))
library(lattice)
levelplot(zerosInBinMSnSet(hyperLOPIT2015goCC),
          xlab = "Number of non-0s",
          ylab = "Marker class",
          col.regions = pal(140))
```

# Index

- \* **classes**
  - AnnotationParams-class, 8
  - ClustDist-class, 14
  - ClustDistList-class, 16
  - GenRegRes-class, 22
  - QSep-class, 81
  - SpatProtVis-class, 90
- \* **internal**
  - Deprecated, 17
- \* **methods**
  - chi2-methods, 11
  - MLearn-methods, 49
  - nddist-methods, 58
  - .MCMCChain (MCMCChains-class), 44
  - .MCMCChains (MCMCChains-class), 44
  - .MCMCParams (MCMCChains-class), 44
  - .MCMCSummary (MCMCChains-class), 44
  - [, ClustDistList, ANY, ANY, ANY-method (ClustDistList-class), 16
  - [, ClustDistList, ANY, missing, missing-method (ClustDistList-class), 16
  - [, MCMCChains, ANY, ANY, ANY-method (MCMCChains-class), 44
  - [, MCMCParams, ANY, ANY, ANY-method (MCMCChains-class), 44
  - [, MartInstanceList, ANY, ANY, ANY-method (MartInstance-class), 44
  - [, MartInstanceList, ANY, ANY-method (MartInstance-class), 44
  - [, MartInstanceList-method (MartInstance-class), 44
  - [[, ClustDistList, ANY, ANY-method (ClustDistList-class), 16
  - [[, ClustDistList, ANY, missing-method (ClustDistList-class), 16
  - [[, MCMCChains, ANY, ANY-method (MCMCChains-class), 44
  - [[, MCMCParams, ANY, ANY-method (MCMCChains-class), 44
  - [[, MartInstanceList, ANY, ANY, ANY-method (MartInstance-class), 44
  - [[, MartInstanceList-method (MartInstance-class), 44
  - addGoAnnotations, 4
  - addLegend, 5, 71, 72
  - addMarkers, 6, 54, 80, 81
  - andy2011params (AnnotationParams-class), 8
  - AnnotationParams, 24, 40
  - AnnotationParams (AnnotationParams-class), 8
  - AnnotationParams-class, 8
  - as.data.frame.MartInstance (MartInstance-class), 44
  - as.data.frame.MartInstanceList (MartInstance-class), 44
  - chains (MCMCChains-class), 44
  - checkFeatureNamesOverlap, 9
  - checkFvarOverlap, 10
  - chi2, 18
  - chi2 (chi2-methods), 11
  - chi2, matrix, matrix-method (chi2-methods), 11
  - chi2, matrix, numeric-method (chi2-methods), 11
  - chi2, numeric, matrix-method (chi2-methods), 11
  - chi2, numeric, numeric-method (chi2-methods), 11
  - chi2-methods, 11
  - chol2inv, 64, 66
  - class::QSep (QSep-class), 81
  - class:AnnotationParams (AnnotationParams-class), 8
  - class:ClustDist (ClustDist-class), 14
  - class:ClustDistList (ClustDistList-class), 16

- class:GenRegRes (GenRegRes-class), 22
- class:MAPParams (MAPParams-class), 40
- class:MCMCChain (MCMCChains-class), 44
- class:MCMCChains (MCMCChains-class), 44
- class:MCMCParams (MCMCChains-class), 44
- class:MCMCSummary (MCMCChains-class), 44
- class:SpatProtVis (SpatProtVis-class), 90
- class:ThetaRegRes (GenRegRes-class), 22
- classWeights, 12
- ClustDist, 13, 14, 16, 27
- ClustDist (ClustDist-class), 14
- clustDist, 13
- ClustDist-class, 14
- ClustDistList, 13, 14, 27
- ClustDistList (ClustDistList-class), 16
- ClustDistList-class, 16
- cmdscale, 70
- coda: :mcmc, 47
- col1 (plot2Ds), 73
- col2 (plot2Ds), 73
- combineThetaRegRes (GenRegRes-class), 22
- data1 (plot2Ds), 73
- data2 (plot2Ds), 73
- dendrogram, 53
- Deprecated, 17
- dist, 52
- dunkley2006params (AnnotationParams-class), 8
- empPvalues, 12, 17
- f1Count (GenRegRes-class), 22
- f1Count, GenRegRes-method (GenRegRes-class), 22
- f1Count, ThetaRegRes-method (GenRegRes-class), 22
- favourPrimary (GenRegRes-class), 22
- fDataToUnknown, 18
- FeaturesOfInterest, 30
- filterAttrs (MartInstance-class), 44
- filterBinMSnSet, 19, 101, 102
- filterMaxMarkers, 20
- filterMinMarkers, 21
- filterNA, 68
- filterZeroCols, 19, 22
- filterZeroRows, 19
- filterZeroRows (filterZeroCols), 22
- flipGoTermId (goIdToTerm), 29
- GenRegRes, 32, 33, 37, 39, 55, 57, 59, 61, 64, 66, 78, 80, 84, 86, 93, 95
- GenRegRes (GenRegRes-class), 22
- GenRegRes-class, 22
- getAnnotationParams, 4
- getAnnotationParams (AnnotationParams-class), 8
- getF1Scores (GenRegRes-class), 22
- getF1Scores, GenRegRes-method (GenRegRes-class), 22
- getF1Scores, ThetaRegRes-method (GenRegRes-class), 22
- getFilterList (MartInstance-class), 44
- getGOEvidenceCodes (showGOEvidenceCodes), 89
- getGOFromFeatures, 8, 9, 24
- getLisacol (setLisacol), 87
- getMarkerClasses, 25, 26, 54
- getMarkers, 26, 26, 54, 98
- getMartInstanceList (MartInstance-class), 44
- getMartTab (MartInstance-class), 44
- getNormDist, 27
- getOldcol (setLisacol), 87
- getParams (GenRegRes-class), 22
- getParams, ClustRegRes-method (undocumented), 101
- getParams, GenRegRes-method (GenRegRes-class), 22
- getParams, ThetaRegRes-method (GenRegRes-class), 22
- getPredictions, 17, 28, 48, 63
- getRegularisedParams (GenRegRes-class), 22
- getRegularisedParams, GenRegRes-method (GenRegRes-class), 22
- getRegularizedParams (GenRegRes-class), 22
- getRegularizedParams, GenRegRes-method (GenRegRes-class), 22
- getSeed (GenRegRes-class), 22
- getSeed, GenRegRes-method (GenRegRes-class), 22
- getStockcol, 52, 72, 74
- getStockcol (setLisacol), 87
- getStockpch, 72
- getStockpch (setLisacol), 87



- getUnknowncol, [76](#)
- getUnknowncol (setLisacol), [87](#)
- getUnknownpch (setLisacol), [87](#)
- getWarnings (GenRegRes-class), [22](#)
- getWarnings, GenRegRes-method (GenRegRes-class), [22](#)
- geweke\_test (mcmc\_get\_outliers), [46](#)
- ginv, [64](#), [66](#)
- goIdToTerm, [29](#)
- goTermToId (goIdToTerm), [29](#)
  
- hclust, [52](#)
- hexbin, [71](#)
- highlightOnPlot, [30](#)
- highlightOnPlot3D (highlightOnPlot), [30](#)
  
- identify, [71](#)
- isMrkMat (mrkVecToMat), [53](#)
- isMrkVec (mrkVecToMat), [53](#)
  
- knn, [32](#), [33](#)
- knnClassification, [31](#), [34](#)
- knnOptimisation, [22](#), [32](#), [33](#)
- knnOptimization (knnOptimisation), [33](#)
- knnPrediction (knnClassification), [31](#)
- knnRegularisation (knnOptimisation), [33](#)
- knntlClassification, [34](#), [37](#)
- knntlOptimisation, [22](#), [35](#), [35](#)
- kpca, [70](#)
- ksvm, [37](#), [39](#)
- ksvmClassification, [37](#), [39](#)
- ksvmOptimisation, [37](#), [38](#)
- ksvmOptimization (ksvmOptimisation), [38](#)
- ksvmPrediction (ksvmClassification), [37](#)
- ksvmRegularisation (ksvmOptimisation), [38](#)
  
- lapply, ClustDistList-method (ClustDistList-class), [16](#)
- lapply, MartInstanceList, ANY-method (MartInstance-class), [44](#)
- lapply, MartInstanceList-method (MartInstance-class), [44](#)
- lda, [71](#)
- legend, [6](#)
- length, ClustDistList-method (ClustDistList-class), [16](#)
- length, MCMCChains-method (MCMCChains-class), [44](#)
- length, MCMCParams-method (MCMCChains-class), [44](#)
- levelPlot (GenRegRes-class), [22](#)
- levelplot, [83](#)
- levelPlot, ClustRegRes-method (undocumented), [101](#)
- levelPlot, GenRegRes-method (GenRegRes-class), [22](#)
- levelPlot, QSep-method (QSep-class), [81](#)
- logPosteriors (MAPPParams-class), [40](#)
  
- makeGoSet, [8](#), [9](#), [39](#)
- MAPPParams, [41](#)
- MAPPParams (MAPPParams-class), [40](#)
- MAPPParams(), [42](#)
- MAPPParams-class, [40](#)
- markerMSnSet, [43](#), [54](#), [87](#), [99](#)
- markers, [7](#), [26](#), [43](#), [81](#), [87](#)
- markers (mrkVecToMat), [53](#)
- MartInstance (MartInstance-class), [44](#)
- MartInstance-class, [44](#)
- MartInstanceList (MartInstance-class), [44](#)
- MartInstanceList-class (MartInstance-class), [44](#)
- mcmc\_burn\_chains (mcmc\_get\_outliers), [46](#)
- mcmc\_get\_meanComponent (mcmc\_get\_outliers), [46](#)
- mcmc\_get\_meanoutliersProb (mcmc\_get\_outliers), [46](#)
- mcmc\_get\_outliers, [46](#)
- mcmc\_pool\_chains (mcmc\_get\_outliers), [46](#)
- mcmc\_thin\_chains (mcmc\_get\_outliers), [46](#)
- MCMCChain (MCMCChains-class), [44](#)
- MCMCChain-class (MCMCChains-class), [44](#)
- MCMCChains (MCMCChains-class), [44](#)
- MCMCChains-class, [44](#)
- MCMCParams-class (MCMCChains-class), [44](#)
- MCMCSummary (MCMCChains-class), [44](#)
- MCMCSummary-class (MCMCChains-class), [44](#)
- minClassScore (Deprecated), [17](#)
- minMarkers, [47](#), [98](#)
- mixing\_posterior\_check, [48](#)
- MLearn, [49](#)
- MLearn, formula, MSnSet, clusteringSchema, missing-method (MLearn-methods), [49](#)
- MLearn, formula, MSnSet, learnerSchema, numeric-method (MLearn-methods), [49](#)

- MLearn, formula, MSnSet, learnerSchema, xvalSpec-methods, (MLearn-methods), 49
- MLearn-methods, 49
- MLearnMSnSet (MLearn-methods), 49
- move2Ds, 49, 74
- mrkConsProfiles, 51
- mrkConsProfiles(), 75
- mrkEncoding (mrkVecToMat), 53
- mrkHClust, 51, 52
- mrkMatAndVec (mrkVecToMat), 53
- mrkMatToVec (mrkVecToMat), 53
- mrkVecToMat, 53
- MSnbase::MSnSet, 41, 42, 96, 97
- MSnSet, 13, 24–26, 28, 32–34, 36–40, 47–49, 55, 56, 59, 60, 63, 64, 66, 73, 78, 79, 82, 84–87, 91, 93, 94, 98, 99
- MSnSetList, 50, 74
- MSnSetMLean (MLearn-methods), 49
  
- naiveBayes, 55, 56
- names, ClustDistList-method (ClustDistList-class), 16
- names, QSep-method (QSep-class), 81
- names<-, ClustDistList, ANY-method (ClustDistList-class), 16
- names<-, QSep, character-method (QSep-class), 81
- nbClassification, 55, 57
- nbOptimisation, 55, 56
- nbOptimization (nbOptimisation), 56
- nbPrediction (nbClassification), 55
- nbRegularisation (nbOptimisation), 56
- nDatasets (MartInstance-class), 44
- nicheMeans2D, 57
- nipals, 70
- nndist (nndist-methods), 58
- nndist, matrix, matrix-method (nndist-methods), 58
- nndist, matrix, missing-method (nndist-methods), 58
- nndist, MSnSet, missing-method (nndist-methods), 58
- nndist-methods, 58
- nnet, 59, 61
- nnetClassification, 59, 61
- nnetOptimisation, 59, 60
- nnetOptimization (nnetOptimisation), 60
- nnetPrediction (nnetClassification), 59
  
- nbRegularisation (nnetOptimisation), 60
- orderGoAnnotations, 61
- orgQuants, 28, 63
  
- par, 76
- perTurboClassification, 64, 66
- perTurboOptimisation, 64, 65
- perTurboOptimization (perTurboOptimisation), 65
- phenoDisco, 67
- plot, 77
- plot, ClustDist, MSnSet-method (ClustDist-class), 14
- plot, ClustDistList, missing-method (ClustDistList-class), 16
- plot, ClustRegRes, missing-method (undocumented), 101
- plot, GenRegRes, missing-method (GenRegRes-class), 22
- plot, MCMCParams, character-method (mcmc\_get\_outliers), 46
- plot, QSep, missing-method (QSep-class), 81
- plot, QSep-method (QSep-class), 81
- plot, SpatProtVis, missing-method (SpatProtVis-class), 90
- plot, ThetaRegRes, missing-method (GenRegRes-class), 22
- plot2D, 5, 6, 52, 69, 74, 90, 91
- plot2Dmethods, 91
- plot2Dmethods (plot2D), 69
- plot2Ds, 50, 72, 73
- plot3d, 71
- plot3D, MSnSet-method (plot2D), 69
- plotConsProfiles, 75
- plotDist, 72, 76
- plotEllipse, 72, 77
- plotEllipse(), 42, 97
- plsda, 78, 80
- plsdaClassification, 78, 80
- plsdaOptimisation, 22, 78, 79
- plsdaOptimization (plsdaOptimisation), 79
- plsdaPrediction (plsdaClassification), 78
- plsdaRegularisation (plsdaOptimisation), 79

- prcomp, [70, 74](#)
- prettyGoTermId (goIdToTerm), [29](#)
- pRoloc-defunct (Deprecated), [17](#)
- pRoloc-deprecated (Deprecated), [17](#)
- pRolocmarkers, [6, 7, 54, 80](#)
- QSep (QSep-class), [81](#)
- qsep (QSep-class), [81](#)
- QSep-class, [81](#)
- randomForest, [84, 86](#)
- rfClassification, [84, 86](#)
- rfOptimisation, [84, 85](#)
- rfOptimization (rfOptimisation), [85](#)
- rfPrediction (rfClassification), [84](#)
- rfRegularisation (rfOptimisation), [85](#)
- Rtsne, [70](#)
- sampleMSnSet, [43, 86, 99](#)
- sapply, ClustDistList-method  
(ClustDistList-class), [16](#)
- sapply, MartInstanceList, ANY-method  
(MartInstance-class), [44](#)
- sapply, MartInstanceList-method  
(MartInstance-class), [44](#)
- setAnnotationParams  
(AnnotationParams-class), [8](#)
- setLisacol, [87](#)
- setOldcol (setLisacol), [87](#)
- setStockcol (setLisacol), [87](#)
- setStockpch (setLisacol), [87](#)
- setUnknowncol (setLisacol), [87](#)
- setUnknownpch (setLisacol), [87](#)
- show, AnnotationParams-method  
(AnnotationParams-class), [8](#)
- show, ClustDist-method  
(ClustDist-class), [14](#)
- show, ClustDistList-method  
(ClustDistList-class), [16](#)
- show, ClustRegRes-method (undocumented),  
[101](#)
- show, ComponentParam-method  
(MCMCChains-class), [44](#)
- show, GenRegRes-method  
(GenRegRes-class), [22](#)
- show, MAPPParams-method  
(MAPPParams-class), [40](#)
- show, MartInstance-method  
(MartInstance-class), [44](#)
- show, MCMCChain-method  
(MCMCChains-class), [44](#)
- show, MCMCChains-method  
(MCMCChains-class), [44](#)
- show, MCMCParams-method  
(MCMCChains-class), [44](#)
- show, QSep-method (QSep-class), [81](#)
- show, SpatProtVis-method  
(SpatProtVis-class), [90](#)
- show, ThetaRegRes-method  
(GenRegRes-class), [22](#)
- showGOEvidenceCodes, [89](#)
- showMrkMat (mrkVecToMat), [53](#)
- solve, [64, 66](#)
- spatial2D, [89](#)
- SpatProtVis (SpatProtVis-class), [90](#)
- SpatProtVis-class, [90](#)
- sub, [18](#)
- subsetMarkers, [92](#)
- summary, QSep-method (QSep-class), [81](#)
- svd, [64, 66](#)
- svm, [93, 94](#)
- svmClassification, [12, 93, 95](#)
- svmOptimisation, [12, 22, 93, 94](#)
- svmOptimization (svmOptimisation), [94](#)
- svmPrediction (svmClassification), [93](#)
- svmRegularisation, [64](#)
- svmRegularisation (svmOptimisation), [94](#)
- tagmMapPredict (MAPPParams-class), [40](#)
- tagmMapTrain (MAPPParams-class), [40](#)
- tagmMapTrain(), [41, 42](#)
- tagmMcmcPredict (tagmMcmcTrain), [95](#)
- tagmMcmcProcess (tagmMcmcTrain), [95](#)
- tagmMcmcTrain, [95](#)
- tagmMcmcTrain(), [96](#)
- tagmPredict (tagmMcmcTrain), [95](#)
- testMarkers, [97](#)
- testMSnSet, [43, 87, 99](#)
- ThetaRegRes (GenRegRes-class), [22](#)
- ThetaRegRes-class (GenRegRes-class), [22](#)
- thetas, [36, 100](#)
- undocumented, [101](#)
- unknownMSnSet, [87, 99](#)
- unknownMSnSet (markerMSnSet), [43](#)
- Versioned, [82](#)
- xvalSpec, [49](#)

zerosInBinMSnSet, [19](#), [101](#)