Package 'mumosa'

November 12, 2026

Type Package

Version 1.20.0

Date 2026-03-05

License GPL-3

Title Multi-Modal Single-Cell Analysis Methods

Description Assorted utilities for multi-modal analyses of single-cell datasets. Includes functions to combine multiple modalities for downstream analysis, perform MNN-based batch correction across multiple modalities, and to compute correlations between assay values for different modalities.

Depends SingleCellExperiment

Imports stats, utils, methods, igraph, Matrix, BiocGenerics, BiocParallel, IRanges, S4Vectors, DelayedArray, DelayedMatrixStats, SummarizedExperiment, BiocNeighbors, BiocSingular, ScaledMatrix, beachmat, scuttle, metapod, scran, batchelor, uwot

Suggests testthat, knitr, BiocStyle, rmarkdown, scater, bluster, DropletUtils, scRNAseq

VignetteBuilder knitr

biocViews ImmunoOncology, SingleCell, RNASeq

Encoding UTF-8

RoxygenNote 7.3.2

URL http://bioconductor.org/packages/mumosa

BugReports https://support.bioconductor.org/

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

git_branch RELEASE_3_23

git_last_commit 3761d72

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Aaron Lun [aut, cre]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

computeCorrelations 2
findTopCorrelations 4
intersectClusters 7
intersectGraphs 9
multiModalMNN 10
rescaleByNeighbors 13
runMultiUMAP 15

Index **19**

`computeCorrelations` *Compute correlations between modes*

Description

Compute Spearman correlations between two sets of features, using data collected for the same cells in different modalities.

Usage

```
computeCorrelations(x, y, ...)

## S4 method for signature 'ANY'
computeCorrelations(
  x,
  y,
  subset.cols = NULL,
  block = NULL,
  equiweight = TRUE,
  use.names = TRUE,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
computeCorrelations(x, y, use.names = TRUE, ..., assay.type = "logcounts")
```

Arguments

`x, y`	Normalized expression matrices containing features in the rows and cells in the columns. Each matrix should have the same set of columns but a different set of features, usually corresponding to different modes for the same cells. Alternatively, SummarizedExperiment objects containing such a matrix. Finally, `y` may be `NULL`, in which correlations are computed between features in `x`.

...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
subset.cols	Vector indicating the columns of x (and y) to retain for computing correlations.
block	A vector or factor of length equal to the number of cells, specifying the block of origin for each cell.
equiweight	Logical scalar indicating whether each block should be given equal weight, if block is specified. If FALSE, each block is weighted by the number of cells.
use.names	Logical scalar specifying whether row names of x and/or y should be reported in the output, if available. For the SummarizedExperiment method, this may also be a string specifying the rowData column containing the names to use; or a character vector of length 2, where the first and second entries specify the rowData columns containing the names in x and y respectively. If either entry is NA, the existing row names for the corresponding object are used. Note that this only has an effect on y if it is a SummarizedExperiment.
BPPARAM	A BiocParallelParam object specifying the parallelization scheme to use.
assay.type	String or integer scalar specifying the assay containing the matrix of interest in x (and y, if a SummarizedExperiment).

Details

If block is specified, correlations are computed separately for each block of cells. For each feature pair, the reported rho is set to the average of the correlations across all blocks. If equiweight=FALSE, the average is weighted by the number of cells in each block.

Similarly, the p-value corresponding to each correlation is computed separately for each block and then combined across blocks with Stouffer's method. More specifically, combining is done using the one-sided p-values for both signs of the correlation, and the smaller p-value is taken (and multiplied by 2). This ensures that a low p-value can only be achieved if the blocks agree in the sign. If equiweight=FALSE, each per-block p-value is weighted by the number of cells.

Value

A DataFrame where each row corresponds to a pair of features in x and y. (If y=NULL, each pair corresponds to a pair of features in x.) This contains the following fields:

- feature1, the name (character) or row index (integer) of each feature in x.
- feature2, the name (character) or row index (integer) of one of the top correlated features to feature1. This is another feature in x if y=NULL, otherwise it is a feature in y.
- rho, the Spearman rank correlation for the current pair of feature1 and feature2.
- p.value, the approximate p-value associated with rho under the null hypothesis that the correlation is zero.
- FDR, the adjusted p-value.

The rows are sorted by feature1 and then p.value.

Author(s)

Aaron Lun

See Also

findTopCorrelations, to avoid computing correlations for all pairs of features when y has many rows.

Examples

```
library(scuttle)
sce1 <- mockSCE()
sce1 <- logNormCounts(sce1)

sce2 <- mockSCE(ngenes=10) # pretend this is protein data.
sce2 <- logNormCounts(sce2)

output <- computeCorrelations(sce1, sce2)
output
```

---

findTopCorrelations *Find top correlations between features*

Description

For each feature, find the subset of other features in the same or another modality that have strongest positive/negative Spearman's rank correlations in a pair of normalized expression matrices.

Usage

```
findTopCorrelations(x, number, ...)

## S4 method for signature 'ANY'
findTopCorrelations(
  x,
  number = 10,
  y = NULL,
  d = 50,
  direction = c("both", "positive", "negative"),
  subset.cols = NULL,
  block = NULL,
  equiweight = TRUE,
  use.names = TRUE,
  deferred = TRUE,
  BSPARAM = IrlbaParam(),
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam()
```

```
)

## S4 method for signature 'SummarizedExperiment'
findTopCorrelations(
  x,
  number,
  y = NULL,
  use.names = TRUE,
  ...,
  assay.type = "logcounts"
)
```

Arguments

x, y	Normalized expression matrices containing features in the rows and cells in the columns. Each matrix should have the same set of columns but a different set of features, usually corresponding to different modes for the same cells. Alternatively, SummarizedExperiment objects containing such a matrix. Finally, y may be NULL, in which correlations are computed between features in x.
number	Integer scalar specifying the number of top correlated features to report for each feature in x.
...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment method, further arguments to pass to the ANY method.
d	Integer scalar specifying the number of dimensions to use for the approximate search via PCA. If NA, no approximation of the rank values is performed prior to the search.
direction	String specifying the sign of the correlations to search for.
subset.cols	Vector indicating the columns of x (and y) to retain for computing correlations.
block	A vector or factor of length equal to the number of cells, specifying the block of origin for each cell.
equiweight	Logical scalar indicating whether each block should be given equal weight, if block is specified. If FALSE, each block is weighted by the number of cells.
use.names	Logical scalar specifying whether row names of x and/or y should be reported in the output, if available. For the SummarizedExperiment method, this may also be a string specifying the rowData column containing the names to use; or a character vector of length 2, where the first and second entries specify the rowData columns containing the names in x and y respectively. If either entry is NA, the existing row names for the corresponding object are used. Note that this only has an effect on y if it is a SummarizedExperiment.
deferred	Logical scalar indicating whether a fast deferred calculation should be used for the rank-based PCA.
BSPARAM	A BiocSingularParam object specifying the algorithm to use for the PCA.

BNPARAM	A BiocNeighborParam object specifying the algorithm to use for the neighbor search.
BPPARAM	A BiocParallelParam object specifying the parallelization scheme to use.
assay.type	String or integer scalar specifying the assay containing the matrix of interest in x (and y, if a SummarizedExperiment).

Details

In most cases, we only care about the top-correlated features, allowing us to skip a lot of unnecessary computation. This is achieved by transforming the problem of finding the largest Spearman correlation into a nearest-neighbor search in rank space. For the sake of speed, we approximate the search by performing PCA to compress the rank values for all features.

For each direction, we compute the one-sided p-value for each feature using the approximate method implemented in cor.test. The FDR correction is performed by considering all possible pairs of features, as these are implicitly tested in the neighbor search. Note that this is somewhat conservative as it does not consider strong correlations outside the reported features.

If block is specified, correlations are computed separately for each block of cells. For each feature pair, the reported rho is set to the average of the correlations across all blocks. Similarly, the p-value corresponding to each correlation is computed separately for each block and then combined across blocks with Stouffer's method. If equiweight=FALSE, the average correlation and each per-block p-value is weighted by the number of cells.

We only consider pairs of features that have computable correlations in at least one block. Blocks are ignored if one or the other feature has tied values (typically zeros) for all cells in that block. This means that a feature may not have any entries in feature1 if it forms no valid pairs, e.g., because it is not expressed. Similarly, the total number of rows may be less than the maximum if insufficient valid pairs are available.

Value

A List containing one or two DataFrames for results in each direction. These are named "positive" and "negative", and are generated according to direction; if direction="both", both DataFrames will be present.

Each DataFrame has up to nrow(x) * number rows, containing the top number correlated features for each feature in x. This contains the following fields:

- feature1, the name (character) or row index (integer) of each feature in x. Not all features may be reported here, see Details.
- feature2, the name (character) or row index (integer) of one of the top correlated features to feature1. This is another feature in x if y=NULL, otherwise it is a feature in y.
- rho, the Spearman rank correlation for the current pair of feature1 and feature2.
- p.value, the approximate p-value associated with rho under the null hypothesis that the correlation is zero.
- FDR, the adjusted p-value.

The rows are sorted by feature1 and then p.value.

Author(s)

Aaron Lun

See Also

computeCorrelations, to compute correlations for all pairs of features.

Examples

```
library(scuttle)
sce1 <- mockSCE()
sce1 <- logNormCounts(sce1)

sce2 <- mockSCE(ngenes=20) # pretend this is CITE-seq data, or something.
sce2 <- logNormCounts(sce2)

# Top 20 correlated features in 'sce2' for each feature in 'sce1':
df <- findTopCorrelations(sce1, sce2, number=20)
df
```

---

intersectClusters *Intersect pre-defined clusters*

Description

Intersect pre-defined clusters from multiple modalities, pruning out combinations that are poorly separated based on the within-cluster sum of squares (WCSS).

Usage

```
intersectClusters(clusters, coords, scale = 1, BPPARAM = SerialParam())
```

Arguments

clusters	A list of factors or vectors of the same length. Each element corresponds to one modality and contains the cluster assignments for the same set of cells.
coords	A list of matrices of length equal to clusters. Each element should have number of rows equal to the number of cells (e.g., a matrix of PC coordinates); we generally expect this to have been used to generate the corresponding entry of clusters.
scale	Numeric scalar specifying the scaling factor to apply to the limit on the WCSS for each modality.
BPPARAM	A BiocParallelParam object specifying how parallelization should be performed.

Details

We intersect clusters by only considering two cells to be in the same "output" cluster if they are also clustered together in each modality. In other words, all cells with a particular combination of identities in `clusters` are assigned to a separate output cluster.

The simplest implementation of the above idea suffers from noise in the cluster definitions that introduces combinations with very few cells. We eliminate these by greedily merging pairs of combinations, starting with the pairs that minimize the gain in the WCSS. In this process, we only consider pairs of combinations that share at least cluster across all modalities (to avoid merges across unrelated clusters).

A natural stopping point for this merging process is when the WCSS of the output clustering exceeds the WCSS of the original clustering for any modality. This aims to preserve the original clustering in each modality by preventing overly aggressive merges that would greatly increase the WCSS, while reducing the complexity of the output clustering by ensuring that the variance explained is comparable.

Users can increase the aggressiveness of the merging procedure by increasing `scale`, e.g., to 1.05 or 1. This will scale up the limit on the WCSS, allowing more merges to be performed before termination.

Value

An integer vector of length equal to the number of cells, containing the assignments to the output clusters.

Author(s)

Aaron Lun

Examples

```
mat1 <- matrix(rnorm(10000), ncol=20)
chosen <- 1:250
mat1[chosen,1] <- mat1[chosen,1] + 10
clusters1 <- kmeans(mat1, 5)$cluster
table(clusters1, chosen=mat1[,1] > 5)

# Pretending we have some other data for the same cells, e.g., ADT.
mat2 <- matrix(rnorm(10000), ncol=20)
chosen <- c(1:125, 251:375)
mat2[chosen,2] <- mat2[chosen,2] + 10
clusters2 <- kmeans(mat2, 5)$cluster
table(clusters2, mat2[,2] > 5)

# Intersecting the clusters:
clusters3 <- intersectClusters(list(clusters1, clusters2), list(mat1, mat2))
table(clusters3, mat1[,1] > 5)
table(clusters3, mat2[,2] > 5)
```

`intersectGraphs` *Intersect two graphs*

Description

Intersect two graphs by taking the product of their edge weights.

Usage

```
intersectGraphs(..., mixing = NULL, nominal = 1e-06)
```

Arguments

`...`	Any number of graph objects for the same set and order of nodes.
`mixing`	Numeric vector of length equal to the number of graphs, specifying the mixing weights for the edge weights.
`nominal`	Numeric scalar specifying the scaling factor to compute the nominal weight.

Details

The idea of taking an intersection of a weighted graph is based on the intersection of simplicial sets in the UMAP algorithm. For each edge that exists in either graph, we compute the product of the weights across all graphs and assign that value as the edge weight in the output graph. This means that edges in the output only have high weight if they are present and highly weighted in all graphs - hence, an intersection.

This approach would make for a very sparse graph if the product was taken directly. To maintain some connectivity, edges that exist in one graph but not the other are assigned nominal weights in the latter to ensure that the product is not zero. The nominal weight for each graph is defined as the product of its smallest non-zero edge weight and `nominal`. Decreasing this value will yield a more conservative intersection and a less connected graph, usually manifesting as smaller clusters after application of community detection algorithms.

By default, `mixing` is a vector of length equal to the numbeer of graphs, containing values of 1. This means that edge weights from each graph in `...` contribute equally to the product. However, it is possible to increase the contribution of some of the graphs by supplying a higher `mixing` values for those graphs.

Unweighted graphs are supported and are considered to have edge weights of 1.

Value

A graph object containing the intersection of `g1` and `g2`.

Author(s)

Aaron Lun

Examples

```
library(scran)

mat1 <- matrix(rnorm(10000), ncol=20)
chosen <- 1:250
mat1[chosen,1] <- mat1[chosen,1] + 10
g1 <- buildSNNGraph(mat1, d=NA, transposed=TRUE)
clusters1 <- igraph::cluster_walktrap(g1)$membership
table(clusters1, chosen=mat1[,1] > 5)

# Pretending we have some other data for the same cells, e.g., ADT.
mat2 <- matrix(rnorm(10000), ncol=20)
chosen <- c(1:125, 251:375)
mat2[chosen,2] <- mat2[chosen,2] + 10
g2 <- buildSNNGraph(mat2, d=NA, transposed=TRUE)
clusters2 <- igraph::cluster_walktrap(g2)$membership
table(clusters2, mat2[,2] > 5)

# Intersecting the graphs and clustering:
gcom <- intersectGraphs(g1, g2)
clustersC <- igraph::cluster_walktrap(gcom)$membership
table(clustersC, mat1[,1] > 5)
table(clustersC, mat2[,2] > 5)
```

`multiModalMNN` *Multi-modal batch correction with MNNs*

Description

Perform MNN correction on multi-modal data, based on a generalization of `fastMNN` to multiple feature sets.

Usage

```
multiModalMNN(
  ...,
  batch = NULL,
  which = NULL,
  rescale.k = 50,
  common.args = list(),
  main.args = list(),
  alt.args = list(),
  mnn.args = list(),
  BNPARAM = KmknnParam(),
  BPPARAM = SerialParam()
)
```

Arguments

...	One or more SingleCellExperiment objects, containing a shared set of alternative Experiments corresponding to different data modalities. Alternatively, one or more lists of such objects.
batch	Factor specifying the batch to which each cell belongs, when ... contains only one SingleCellExperiment object. Otherwise, each object in ... is assumed to contain cells from a single batch.
which	Character vector containing the names of the alternative Experiments to use for correction. Defaults to the names of all alternative Experiments that are present in every object of ....
rescale.k	Integer scalar specifying the number of neighbors to use in rescaleByNeighbors.
common.args	Named list of further arguments to control the PCA for all modalities.
main.args	Named list of further arguments to control the PCA for the main Experiments. Overrides any arguments of the same name in common.args.
alt.args	Named list of named lists of further arguments to control the PCA for each alternative Experiment specified in which. This should be a list where each entry is named after any alternative Experiment and contains an internal list of named arguments; these override any settings in common.args in the PCA for the corresponding modality.
mnn.args	Further arguments to pass to reducedMNN, controlling the MNN correction.
BNPARAM	A BiocNeighborParam object specifying how the nearest neighbor searches should be performed.
BPPARAM	A BiocParallelParam object specifying how parallelization should be performed.

Details

This function implements a multi-modal MNN correction for SingleCellExperiment inputs where each main and alternative Experiment corresponds to one modality. We perform a PCA within each modality with multiBatchPCA, rescale the PCs to be of a comparable scale with rescaleByNeighbors, and finally correct in low-dimensional space with reducedMNN. Corrected expression values for each modality are then recovered in the same manner as described for fastMNN.

Modality-specific arguments can be passed to the PCA step via the common.args, main.args and alt.args arguments. These mirror the corresponding arguments in applyMultiSCE - see the documentation for that function for more details. Additional arguments for the MNN step can be passed via mnn.args. Note that batch is used across all steps and must be specified as its own argument in the multiModalMNN function signature.

Most arguments in multiBatchPCA can be specified in common.args, main.args or each entry of alt.args. This includes passing d=NA to turn off the PCA or subset.row to only use a subset of features for the PCA. Additionally, the following arguments are supported:

- By default, a cosine-normalization is performed prior to the PCA for each modality. This can be disabled by passing cos.norm=FALSE to common.args, main.args or each entry of alt.args.
- Setting correct.all will reported corrected expression values for all features even when subset.row is specified. This can be used in common.args, main.args or each entry of alt.args.

Note that the function will look for `assay.type="logcounts"` by default in each entry of `...`. Users should perform log-normalization prior to calling `multiModalMNN`, most typically with `multiBatchNorm` - see Examples.

Value

A SingleCellExperiment of the same structure as that returned by `fastMNN`, i.e., with a `corrected` entry of corrected low-dimensional coordinates and a `reconstructed` assay of corrected expression values. In addition, the `altExps` entries contain corrected values for each data modality used in the correction.

Author(s)

Aaron Lun

See Also

`fastMNN`, for MNN correction within a single modality.

`multiBatchPCA`, to perform a batch-aware PCA within each modality.

`applyMultiSCE`, which inspired this interface for Experiment-specific arguments.

Examples

```
# Mocking up a gene expression + ADT dataset:
library(scater)
exprs_sce <- mockSCE()
adt_sce <- mockSCE(ngenes=20)
altExp(exprs_sce, "ADT") <- adt_sce

# Pretend we have three batches for the sake of demonstration:
batch <- sample(1:3, ncol(exprs_sce), replace=TRUE)

# Normalizing first with batchelor::multiBatchNorm:
library(batchelor)
exprs_sce <- applyMultiSCE(exprs_sce, batch=batch, FUN=multiBatchNorm)

# and perform batch correction:
corrected <- multiModalMNN(exprs_sce, batch=batch, which="ADT")

# Pass arguments to, e.g., use a subset of features for the RNA,
# turn off the PCA for the ADTs:
corrected2 <- multiModalMNN(exprs_sce, batch=batch, which="ADT",
    main.args=list(subset.row=1:500),
    alt.args=list(ADT=list(d=NA)))
```

rescaleByNeighbors *Rescale matrices for different modes*

Description

Rescale matrices for different data modalities so that their distances are more comparable, using the distances to neighbors to approximate noise.

Usage

```
rescaleByNeighbors(x, ...)

## S4 method for signature 'ANY'
rescaleByNeighbors(
  x,
  k = 50,
  weights = NULL,
  combine = TRUE,
  num.threads = 1,
  BNPARAM = NULL,
  BPPARAM = NULL
)

## S4 method for signature 'SummarizedExperiment'
rescaleByNeighbors(x, assays, extras = list(), ...)

## S4 method for signature 'SingleCellExperiment'
rescaleByNeighbors(
  x,
  assays = NULL,
  dimreds = NULL,
  altexps = NULL,
  altexp.assay = "logcounts",
  extras = list(),
  ...
)
```

Arguments

x	A list of numeric matrices where each row is a cell and each column is some dimension/variable. For gene expression data, this is usually the matrix of PC coordinates. All matrices should have the same number of rows. Alternatively, a SummarizedExperiment containing relevant matrices in its assays. Alternatively, a SingleCellExperiment containing relevant matrices in its assays, `reducedDims` or `altExps`.

...	For the generic, further arguments to pass to specific methods. For the SummarizedExperiment and SingleCellExperiment methods, further arguments to pass to the ANY method.
k	An integer scalar specifying the number of neighbors to use for the distance calculation.
weights	A numeric vector of length equal to x (if a list), specifying the weight of each mode. Defaults to equal weights for all modes. See details for how to interpret this argument when x is a SummarizedExperiment.
combine	A logical scalar specifying whether the rescaled matrices should be combined into a single matrix.
num.threads	Integer scalar specifying the number of threads to use for the neighbor search.
BNPARAM	A BiocNeighborParam object specifying the algorithm to use for the nearest-neighbor search.
BPPARAM	Deprecated, use num.threads instead.
assays	A character or integer vector of assays to extract and transpose for use in the ANY method. For the SingleCellExperiment, this argument can be missing, in which case no assays are used.
extras	A list of further matrices of similar structure to those matrices in a list-like x.
dimreds	A character or integer vector of reducedDims to extract for use in the ANY method. This argument can be missing, in which case no assays are used.
altexps	A character or integer vector of altExps to extract and transpose for use in the ANY method. This argument can be missing, in which case no alternative experiments are used.
altexp.assay	A character or integer vector specifying the assay to extract from alternative experiments, when altexp is specified. This is recycled to the same length as altexp.

Details

When dealing with multi-modal data, we may wish to combine all modes into a single matrix for downstream processing. However, a naive cbind does not account for the fact that different modes may very different scales and number of features. A mode with a larger scale or more features may dominate steps such as clustering or dimensionality reduction. This function attempts to rescale the contents for each matrix so that the modes are more comparable.

A naive approach to rescaling would be to just equalize the total variances across matrices. This is not ideal as it fails to consider the differences in biological variation captured by each mode. For example, if a biological phenomenon is only present in one mode, that matrix's total variance would naturally be higher. Scaling all matrices to the same total variance would suppress genuine variation and inflate the relative contribution of noise.

We instead use the distance to the kth nearest neighbor as an estimate of the per-mode "noise". Modes with more features or higher technical noise will have larger distances, and downscaling each matrix by the median distance will correct for differences between modes. At the same time, by only considering the nearest neighbors, we avoid capturing (and inadvertently eliminating) variance due to mode-specific population structure.

The default approach is to weight each mode equally during the rescaling process, i.e., the median distance to the kth nearest neighbor will be equal for all modes after rescaling. However, we can also set `weights` to control the fold-differences in the median distances. For example, a weight of 2 for one mode would mean that its median distance after rescaling is twice as large as that from a mode with a weight of 1. This may be useful for prioritizing modes that are more likely to be important.

The correspondence between non-`NULL` `weights` and the modes is slightly tricky whe `x` is not a list. If `x` is a SummarizedExperiment, the modes are ordered as: all entries in `assays` in the specified order, then all entries in `extras`. If `x` is a SingleCellExperiment, the modes are ordered as: all entries in `assays` in the specified order, then all entries in `dimreds`, then all entries in `altexps`, and finally all entries in `extras`.

Value

A numeric matrix with number of rows equal to the number of cells, where the columns span all variables across all modes supplied in `x`. Values are scaled so that each mode contributes the specified weight to downstream Euclidean distance calculations.

If `combine=FALSE`, a list of rescaled matrices is returned instead.

Author(s)

Aaron Lun

Examples

```
# Mocking up a gene expression + ADT dataset:
library(scater)
exprs_sce <- mockSCE()
exprs_sce <- logNormCounts(exprs_sce)
exprs_sce <- runPCA(exprs_sce)

adt_sce <- mockSCE(ngenes=20)
adt_sce <- logNormCounts(adt_sce)
altExp(exprs_sce, "ADT") <- adt_sce

combined <- rescaleByNeighbors(exprs_sce, dimreds="PCA", altexps="ADT")
dim(combined)
```

---

runMultiUMAP *Multi-modal UMAP*

Description

Perform UMAP with multiple input matrices by intersecting their simplicial sets. Typically used to combine results from multiple data modalities into a single embedding.

Usage

```
calculateMultiUMAP(x, ...)

## S4 method for signature 'ANY'
calculateMultiUMAP(x, ..., metric = "euclidean")

## S4 method for signature 'SummarizedExperiment'
calculateMultiUMAP(x, assays, extras = list(), ...)

## S4 method for signature 'SingleCellExperiment'
calculateMultiUMAP(
  x,
  assays = NULL,
  dimreds = NULL,
  altexps = NULL,
  altexp.assay = "logcounts",
  extras = list(),
  ...
)

runMultiUMAP(x, ..., name = "MultiUMAP")
```

Arguments

x	For calculateMultiUMAP, a list of numeric matrices where each row is a cell and each column is some dimension/variable. For gene expression data, this is usually the matrix of PC coordinates. All matrices should have the same number of rows. Alternatively, a SummarizedExperiment containing relevant matrices in its assays. Alternatively, a SingleCellExperiment containing relevant matrices in its assays, reducedDims or altExps. This is also the only permissible argument for runMultiUMAP.
...	For the generic, further arguments to pass to specific methods. For the ANY method, further arguments to pass to umap. For the SummarizedExperiment and SingleCellExperiment methods, and for runMultiUMAP, further arguments to pass to the ANY method.
metric	Character vector specifying the type of distance to use for each matrix in x. This is recycled to the same number of matrices supplied in x.
assays	A character or integer vector of assays to extract and transpose for use in the UMAP. For the SingleCellExperiment, this argument can be missing, in which case no assays are used.
extras	A list of further matrices of similar structure to those matrices in a list-like x.
dimreds	A character or integer vector of reducedDims to extract for use in the UMAP. This argument can be missing, in which case no assays are used.

altexps	A character or integer vector of altExps to extract and transpose for use in the UMAP. This argument can be missing, in which case no alternative experiments are used.
altexp.assay	A character or integer vector specifying the assay to extract from alternative experiments, when altexp is specified. This is recycled to the same length as altexp.
name	String specifying the name of the reducedDims in which to store the UMAP.

Details

These functions serve as convenience wrappers around umap for multi-modal analysis. The idea is that each input matrix in x corresponds to data for a different mode. A typical example would consist of the PC coordinates generated from gene expression counts, plus the log-abundance matrix for ADT counts from CITE-seq experiments; one might also include matrices of transformed intensities from indexed FACS, to name some more possibilities.

Roughly speaking, the idea is to identify nearest neighbors *within* each mode to construct the simplicial sets. Integration of multiple modes is performed by intersecting the sets to obtain a single graph, which is used in the rest of the UMAP algorithm. By performing an intersection, we focus on relationships between cells that are consistently neighboring across all the modes, thus providing greater resolution of differences at any mode. The neighbor search within each mode also avoids difficulties with quantitative comparisons of distances between modes.

The most obvious use of this function is to generate a low-dimensional embedding for visualization. However, users can also set n_components to a higher value (e.g., 10-20) to retain more information for downstream steps like clustering. Do, however, remember to set the seed appropriately.

By default, all modes use the distance metric of metric to construct the simplicial sets *within* each mode. However, it is possible to vary this by supplying a vector of metrics, e.g., "euclidean" for the first matrix, "manhattan" for the second. For the SingleCellExperiment method, matrices are extracted in the order of assays, reduced dimensions and alternative experiments, so any variation in metrics is also assumed to follow this order.

Value

For calculateMultiUMAP, a numeric matrix containing the low-dimensional UMAP embedding.

For runMultiUMAP, x is returned with a MultiUMAP field in its reducedDims.

Author(s)

Aaron Lun

See Also

runUMAP, for the more straightforward application of UMAP.

Examples

```
# Mocking up a gene expression + ADT dataset:
library(scater)
exprs_sce <- mockSCE()
```

```
exprs_sce <- logNormCounts(exprs_sce)
exprs_sce <- runPCA(exprs_sce)

adt_sce <- mockSCE(ngenes=20)
adt_sce <- logNormCounts(adt_sce)
altExp(exprs_sce, "ADT") <- adt_sce

# Running a multimodal analysis using PCs for expression
# and log-counts for the ADTs. Annoyingly, have to prefix
# this for the time being to distinguish from the scater generic.
exprs_sce <- mumosa::runMultiUMAP(exprs_sce, dimreds="PCA", altexps="ADT")
plotReducedDim(exprs_sce, "MultiUMAP")
```

Index

altExps, *12–14*, *16, 17*
applyMultiSCE, *11, 12*

BiocNeighborParam, *6*, *11*, *14*
BiocParallelParam, *3*, *6, 7*, *11*
BiocSingularParam, *5*

calculateMultiUMAP *(*runMultiUMAP*)*, 15
calculateMultiUMAP,ANY-method
        *(*runMultiUMAP*)*, 15
calculateMultiUMAP,SingleCellExperiment-method
        *(*runMultiUMAP*)*, 15
calculateMultiUMAP,SummarizedExperiment-method
        *(*runMultiUMAP*)*, 15
computeCorrelations, 2, *7*
computeCorrelations,ANY-method
        *(*computeCorrelations*)*, 2
computeCorrelations,SummarizedExperiment-method
        *(*computeCorrelations*)*, 2
cor.test, *6*

DataFrame, *6*

fastMNN, *10–12*
findTopCorrelations, *4*, 4
findTopCorrelations,ANY-method
        *(*findTopCorrelations*)*, 4
findTopCorrelations,SummarizedExperiment-method
        *(*findTopCorrelations*)*, 4

graph, *9*

intersectClusters, 7
intersectGraphs, 9

List, *6*

multiBatchNorm, *12*
multiBatchPCA, *11, 12*
multiModalMNN, 10

reducedDims, *13, 14*, *16, 17*
reducedMNN, *11*
rescaleByNeighbors, *11*, 13
rescaleByNeighbors,ANY-method
        *(*rescaleByNeighbors*)*, 13
rescaleByNeighbors,SingleCellExperiment-method
        *(*rescaleByNeighbors*)*, 13
rescaleByNeighbors,SummarizedExperiment-method
        *(*rescaleByNeighbors*)*, 13
rowData, *3*, *5*
runMultiUMAP, 15
runUMAP, *17*

SingleCellExperiment, *11–13*, *16*
SummarizedExperiment, *2*, *5*, *13*, *16*

umap, *16, 17*