Package: Coralysis (via r-universe)

May 30, 2026

Type Package

Title Coralysis sensitive identification of imbalanced cell types and states in single-cell data via multi-level integration

Version 1.3.0

Description Coralysis is an R package featuring a multi-level integration algorithm for sensitive integration, reference-mapping, and cell-state identification in single-cell data. The multi-level integration algorithm is inspired by the process of assembling a puzzle - where one begins by grouping pieces based on low-to high-level features, such as color and shading, before looking into shape and patterns. This approach progressively blends the batch effects and separates cell types across multiple rounds of divisive clustering.

License GPL-3

Imports Matrix, aricode, LiblineaR, SparseM, ggplot2, umap, Rtsne, pheatmap, reshape2, dplyr, SingleCellExperiment, SummarizedExperiment, S4Vectors, methods, stats, utils, RANN, sparseMatrixStats, irlba, flexclust, scran, class, matrixStats, tidyr, cowplot, uwot, scatterpie, RColorBrewer, ggrastr, ggrepel, RSpectra, BiocParallel, withr

Depends R (>= 4.2.0)

Suggests knitr, rmarkdown, bluster, ComplexHeatmap, circlize, scater, viridis, scRNAseq, SingleR, MouseGastrulationData, testthat (>= 3.0.0), BiocStyle, scrapper

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

biocViews SingleCell, RNASeq, Proteomics, Transcriptomics, GeneExpression, BatchEffect, Clustering, Annotation, Classification, DifferentialExpression, DimensionReduction, Software

NeedsCompilation no

URL https://github.com/elolab/Coralysis,
https://elolab.github.io/Coralysis/

BugReports https://github.com/elolab/Coralysis/issues

Config/testthat/edition 3

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev libglpk-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev python3 zlib1g-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-04-28 13:05:04 UTC

RemoteUrl https://github.com/bioc/Coralysis

RemoteRef HEAD

RemoteSha de2d3abc3491254c65937ad682f39031ff498695

Contents

AggregateDataByBatch *Aggregates feature expression by cell clusters, per batch if provided.*

Description

The function aggregates feature expression by cell clusters, per batch if provided.

Usage

```
AggregateDataByBatch.SingleCellExperiment(object, batch.label, nhvg, p, ...)

## S4 method for signature 'SingleCellExperiment'
AggregateDataByBatch(object, batch.label, nhvg = 2000L, p = 30L, ...)
```

Arguments

object	An object of SingleCellExperiment class.
batch.label	Cluster identities vector corresponding to the cells in mtx.
nhvg	Integer of the number of highly variable features to select. By default 2000.
p	Integer. By default 30.
...	Parameters to be passed to ClusterCells() function.

Value

A SingleCellExperiment object with feature expression aggregated by clusters.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Import data from Zenodo
data.url <- "https://zenodo.org/records/14871436/files/pbmc_10Xassays.rds?download=1"
sce <- readRDS(file = url(data.url))

# Run with a batch
set.seed(1204)
sce <- AggregateDataByBatch(object = sce, batch.label = "batch")
logcounts(sce)[1:10, 1:10]
head(metadata(sce)$clusters)

# Run without a batch
set.seed(1204)
sce <- AggregateDataByBatch(object = sce, batch.label = NULL)
logcounts(sce)[1:10, 1:10]
head(metadata(sce)$clusters)
```

`BinCellClusterProbability` *Bin cell cluster probability*

Description

Bin cell cluster probability by a given cell label.

Usage

```
BinCellClusterProbability.SingleCellExperiment(
  object,
  label,
  icp.run,
  icp.round,
  funs,
  bins,
  aggregate.bins.by,
  use.assay
)

## S4 method for signature 'SingleCellExperiment'
BinCellClusterProbability(
  object,
  label,
  icp.run = NULL,
  icp.round = NULL,
  funs = "mean",
  bins = 20,
  aggregate.bins.by = "mean",
  use.assay = "logcounts"
)
```

Arguments

`object`	An object of `SingleCellExperiment` class with ICP cell cluster probability tables saved in `metadata(object)$coralysis$joint.probability`. After running one of `RunParallelICP` or `RunParallelDivisiveICP`.
`label`	Label of interest available in `colData(object)` to group by the bins of cell cluster probability.
`icp.run`	ICP run(s) to retrieve from `metadata(object)$coralysis$joint.probability`. By default `NULL`, i.e., all are retrieved. Specify a numeric vector to retrieve a specific set of tables.
`icp.round`	ICP round(s) to retrieve from `metadata(object)$coralysis$joint.probability`. By default `NULL`, i.e., all are retrieved. Only relevant if probabilities were obtained with the function `RunParallelDivisiveICP`, i.e., divisive ICP was per-

formed. Otherwise it is ignored and internally assumed as `icp.round = 1`, i.e., only one round.

`funs` One function to summarise ICP cell cluster probability. One of `"mean"` or `"median"`. By default `"mean"`.

`bins` Number of bins to bin cell cluster probability by cell `label` given. By default `20`.

`aggregate.bins.by` One function to aggregate One of `"mean"` or `"median"`. By default `"mean"`.

`use.assay` Name of the assay that should be used to obtain the average expression of features across cell `label` probability bins.

Value

A `SingleCellExperiment` class object with feature average expression by cell `label` probability bins.

Examples

```
# Packages
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Import data from Zenodo
data.url <- "https://zenodo.org/records/14845751/files/pbmc_10Xassays.rds?download=1"
sce <- readRDS(file = url(data.url))

# Prepare data
sce <- PrepareData(object = sce)

# Multi-level integration - 'L = 4' just for highlighting purposes
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "batch", L = 4,
    threads = 2
)

# Cell states SCE object for a given cell type annotation or clustering
cellstate.sce <- BinCellClusterProbability(
    object = sce, label = "cell_type",
    icp.round = 4, bins = 20
)
cellstate.sce
```

`CellBinsFeatureCorrelation`

Cell bins feature correlation

Description

Correlation between cell bins for the given labels and features.

Usage

```
CellBinsFeatureCorrelation.SingleCellExperiment(object, labels, method)

## S4 method for signature 'SingleCellExperiment'
CellBinsFeatureCorrelation(object, labels = NULL, method = "pearson")
```

Arguments

object	An object of SingleCellExperiment class obtained with the function BinCellClusterProbability().
labels	Character of label(s) from the label provided to the function BinCellClusterProbability(). By default NULL, i.e., all labels are used.
method	Character specifying the correlation method to use. One of "pearson", "kendall" or "spearman". By default "pearson" is used.

Value

A data frame with the correlation coefficient for each feature (rows) across labels (columns).

Examples

```
# Packages
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Import data from Zenodo
data.url <- "https://zenodo.org/records/14845751/files/pbmc_10Xassays.rds?download=1"
sce <- readRDS(file = url(data.url))

# Prepare data
sce <- PrepareData(object = sce)

# Multi-level integration - 'L = 4' just for highlighting purposes
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "batch", L = 4,
    threads = 2
)

# Cell states SCE object for a given cell type annotation or clustering
cellstate.sce <- BinCellClusterProbability(
    object = sce, label = "cell_type",
    icp.round = 4, bins = 20
)
cellstate.sce

# Pearson correlated features with "Monocyte"
cor.features.mono <- CellBinsFeatureCorrelation(
```

```
    object = cellstate.sce,
    labels = "Monocyte"
)
```

CellClusterProbabilityDistribution
Cell cluster probability distribution

Description

Plot cell cluster probability distribution per label by group.

Usage

```
CellClusterProbabilityDistribution.SingleCellExperiment(
  object,
  label,
  group,
  probability
)

## S4 method for signature 'SingleCellExperiment'
CellClusterProbabilityDistribution(
  object,
  label,
  group,
  probability = "scaled_mean_probs"
)
```

Arguments

object	An object of SingleCellExperiment class with aggregated cell cluster probability available in colData(object), which can be obtained after running SummariseCellClusterProbability().
label	Character specifying the colData variable to use as cell type/cluster label.
group	Character specifying the colData variable to use as categorical group variable.
probability	Character specifying the aggregated cell cluster probability variable available in colData, used to plot its distribution. One of "mean_probs", "scaled_mean_probs", "median_probs", "scaled_median_probs". The availability of these variables in colData depends on the parameters given to the function SummariseCellClusterProbability() beforehand. By default assumes that "scaled_mean_probs" is available in colData, which is only true if SummariseCellClusterProbability() function was run with funs = "mean" and scale.funs = TRUE.

Value

A plot of class ggplot.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 4, L = 25, C = 1, d = 0.5,
    train.with.bnn = FALSE,
    use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)

# Summarise cell cluster probability
sce <- SummariseCellClusterProbability(object = sce, icp.round = 2) # saved in 'colData'

# Search for differences in probabilities across group(s)
# give an interesting variable to the "group" parameter
prob.dist <- CellClusterProbabilityDistribution(
    object = sce, label = "Species",
    group = "Batch",
    probability = "scaled_mean_probs"
)
prob.dist # print plot
```

FindAllClusterMarkers *Identification of feature markers for all clusters*

Description

FindAllClusterMarkers enables identifying feature markers for all clusters at once. This is done by differential expresission analysis where cells from one cluster are compared against the cells from the rest of the clusters. Feature and cell filters can be applied to accelerate the analysis, but this might lead to missing weak signals.

Usage

```
FindAllClusterMarkers.SingleCellExperiment(
  object,
  clustering.label,
  test,
  log2fc.threshold,
  min.pct,
  min.diff.pct,
  min.cells.group,
  max.cells.per.cluster,
  return.thresh,
  only.pos
)

## S4 method for signature 'SingleCellExperiment'
FindAllClusterMarkers(
  object,
  clustering.label,
  test = "wilcox",
  log2fc.threshold = 0.25,
  min.pct = 0.1,
  min.diff.pct = NULL,
  min.cells.group = 3,
  max.cells.per.cluster = NULL,
  return.thresh = 0.01,
  only.pos = FALSE
)
```

Arguments

object	A SingleCellExperiment object.
clustering.label	A variable name (of class character) available in the cell metadata colData(object) with the clustering labels (character or factor) to use.
test	Which test to use. Only "wilcox" (the Wilcoxon rank-sum test, AKA Mann-Whitney U test) is supported at the moment.
log2fc.threshold	Filters out features that have log2 fold-change of the averaged feature expression values below this threshold. Default is 0.25.
min.pct	Filters out features that have dropout rate (fraction of cells expressing a feature) below this threshold in both comparison groups. Default is 0.1.

`min.diff.pct` Filters out features that do not have this minimum difference in the dropout rates (fraction of cells expressing a feature) between the two comparison groups. Default is `NULL`.

`min.cells.group` The minimum number of cells in the two comparison groups to perform the DE analysis. If the number of cells is below the threshold, then the DE analysis of this cluster is skipped. Default is `3`.

`max.cells.per.cluster` The maximum number of cells per cluster if downsampling is performed to speed up the DE analysis. Default is `NULL`, i.e., no downsampling.

`return.thresh` If `only.pos=TRUE`, then return only features that have the adjusted p-value (adjusted by the Bonferroni method) below or equal to this threshold. Default is `0.01`.

`only.pos` Whether to return only features that have an adjusted p-value (adjusted by the Bonferroni method) below or equal to the threshold. Default is `FALSE`.

Value

A data frame of the results if positive results were found, else `NULL`.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Markers
dge <- FindAllClusterMarkers(sce, clustering.label = "Species")
dge
```

`FindClusterMarkers` *Differential expression between cell clusters*

Description

FindClusterMarkers enables identifying feature markers for one cluster or two arbitrary combinations of clusters, e.g. 1_2 vs. 3_4_5. Feature and cell filters can be applied to accelerate the analysis, but this might lead to missing weak signals.

Usage

```
FindClusterMarkers.SingleCellExperiment(
  object,
  clustering.label,
  clusters.1,
  clusters.2,
  test,
  log2fc.threshold,
  min.pct,
  min.diff.pct,
  min.cells.group,
  max.cells.per.cluster,
  return.thresh,
  only.pos
)

## S4 method for signature 'SingleCellExperiment'
FindClusterMarkers(
  object,
  clustering.label,
  clusters.1 = NULL,
  clusters.2 = NULL,
  test = "wilcox",
  log2fc.threshold = 0.25,
  min.pct = 0.1,
  min.diff.pct = NULL,
  min.cells.group = 3,
  max.cells.per.cluster = NULL,
  return.thresh = 0.01,
  only.pos = FALSE
)
```

Arguments

object	A SingleCellExperiment object.
clustering.label	A variable name (of class character) available in the cell metadata colData(object) with the clustering labels (character or factor) to use.
clusters.1	a character or numeric vector denoting which clusters to use in the first group (named group.1 in the results)
clusters.2	a character or numeric vector denoting which clusters to use in the second group (named group.2 in the results)

`test` Which test to use. Only "wilcoxon" (the Wilcoxon rank-sum test, AKA Mann-Whitney U test) is supported at the moment.

`log2fc.threshold` Filters out features that have log2 fold-change of the averaged feature expression values below this threshold. Default is `0.25`.

`min.pct` Filters out features that have dropout rate (fraction of cells expressing a feature) below this threshold in both comparison groups Default is `0.1`.

`min.diff.pct` Filters out features that do not have this minimum difference in the dropout rates (fraction of cells expressing a feature) between the two comparison groups. Default is `NULL`.

`min.cells.group` The minimum number of cells in the two comparison groups to perform the DE analysis. If the number of cells is below the threshold, then the DE analysis is not performed. Default is `3`.

`max.cells.per.cluster` The maximun number of cells per cluster if downsampling is performed to speed up the DE analysis. Default is `NULL`, i.e. no downsampling.

`return.thresh` If only.pos=TRUE, then return only features that have the adjusted p-value (adjusted by the Bonferroni method) below or equal to this threshold. Default is `0.01`.

`only.pos` Whether to return only features that have an adjusted p-value (adjusted by the Bonferroni method) below or equal to the threshold. Default is `FALSE`.

Value

a data frame of the results if positive results were found, else NULL

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Markers between versicolor vs virginica
dge <- FindClusterMarkers(sce,
    clustering.label = "Species",
    clusters.1 = "versicolor",
```

```
    clusters.2 = "virginica"
)
dge
```

`GetCellClusterProbability` *Get ICP cell cluster probability*

Description

Get ICP cell cluster probability table(s)

Usage

```
GetCellClusterProbability.SingleCellExperiment(
  object,
  icp.run,
  icp.round,
  concatenate
)

## S4 method for signature 'SingleCellExperiment'
GetCellClusterProbability(
  object,
  icp.run = NULL,
  icp.round = NULL,
  concatenate = TRUE
)
```

Arguments

`object`	An object of `SingleCellExperiment` class with ICP cell cluster probability tables saved in `metadata(object)$coralysis$joint.probability`. After running `RunParallelDivisiveICP`.
`icp.run`	ICP run(s) to retrieve from `metadata(object)$coralysis$joint.probability`. By default `NULL`, i.e., all are retrieved. Specify a numeric vector to retrieve a specific set of tables.
`icp.round`	ICP round(s) to retrieve from `metadata(object)$coralysis$joint.probability`. By default `NULL`, i.e., all are retrieved.
`concatenate`	Concatenate list of ICP cell cluster probability tables retrieved. By default `TRUE`, i.e., the list of ICP cell cluster probability tables is concatenated.

Value

A list with ICP cell cluster probability tables or a matrix with concatenated tables.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 2, L = 25, C = 1, train.k.nn = 10,
    train.k.nn.prop = NULL, use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)

# Get cluster probability for all ICP runs
probs <- GetCellClusterProbability(object = sce, icp.round = 1, concatenate = TRUE)
probs[1:10, 1:5]
```

`GetFeatureCoefficients` *Get feature coefficients*

Description

Get feature coefficients from ICP models.

Usage

```
GetFeatureCoefficients.SingleCellExperiment(
  object,
  icp.run = NULL,
  icp.round = NULL
```

```
)

## S4 method for signature 'SingleCellExperiment'
GetFeatureCoefficients(object, icp.run = NULL, icp.round = NULL)
```

Arguments

object	An object of SingleCellExperiment class with ICP cell cluster probability tables saved in metadata(object)$coralysis$joint.probability. After running RunParallelDivisiveICP.
icp.run	ICP run(s) to retrieve from metadata(object)$coralysis$joint.probability. By default NULL, i.e., all are retrieved. Specify a numeric vector to retrieve a specific set of tables.
icp.round	ICP round(s) to retrieve from metadata(object)$coralysis$joint.probability. By default NULL, i.e., all are retrieved.

Value

A list of feature coefficient weights per cluster per ICP run/round.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 4, L = 25, C = 1, d = 0.5,
    train.with.bnn = FALSE,
    use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)
```

```
# GetFeatureCoefficients
gene_coefficients_icp_7_1 <- GetFeatureCoefficients(object = sce, icp.run = 7, icp.round = 1)
head(gene_coefficients_icp_7_1$icp_13)
```

`HeatmapFeatures` *Heatmap visualization of the expression of features by clusters*

Description

The `HeatmapFeatures` function draws a heatmap of features by cluster identity.

Usage

```
HeatmapFeatures.SingleCellExperiment(
  object,
  clustering.label,
  features,
  use.color,
  seed.color,
  ...
)

## S4 method for signature 'SingleCellExperiment'
HeatmapFeatures(
  object,
  clustering.label,
  features,
  use.color = NULL,
  seed.color = 123,
  ...
)
```

Arguments

`object`	of `SingleCellExperiment` class
`clustering.label`	A variable name (of class `character`) available in the cell metadata `colData(object)` with the clustering labels (`character` or `factor`) to use.
`features`	Feature names to plot by cluster (`character`) matching `row.names(object)`.
`use.color`	Character specifying the colors for the clusters. By default `NULL`, i.e., colors are randomly chosen based on the seed given at `seed.color`. It is overwritten in case the argument `annotation_colors` is provided.
`seed.color`	Seed to randomly select colors for the clusters. By default `123`. It is overwritten in case the argument `annotation_colors` is provided.
`...`	Parameters to pass to `pheatmap::pheatmap` function.

Value

nothing

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Plot features by clustering, i.e., grouping variable
# without scaling rows (using 'logcounts' expression):
HeatmapFeatures(
    object = sce, clustering.label = "Species",
    features = row.names(sce)[1:4]
)

# scaling rows:
HeatmapFeatures(
    object = sce, clustering.label = "Species",
    features = row.names(sce)[1:4], scale = "row"
) # scale
```

MajorityVotingFeatures

Majority voting features by label

Description

Get ICP feature coefficients for a label of interest by majority voting label across ICP clusters.

Usage

```
MajorityVotingFeatures.SingleCellExperiment(object, label)

## S4 method for signature 'SingleCellExperiment'
MajorityVotingFeatures(object, label)
```

Arguments

object	An object of SingleCellExperiment class with ICP cell cluster probability tables saved in metadata(object)$coralysis$joint.probability. After running RunParallelDivisiveICP.
label	Label of interest available in colData(object).

Value

A list of with a list of data frames with feature weights per label and a data frame with a summary by label.

Examples

```
## Not run:
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Import data from Zenodo
data.url <- "https://zenodo.org/records/14845751/files/pbmc_10Xassays.rds?download=1"
sce <- readRDS(file = url(data.url))

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "batch",
    k = 4, L = 10, C = 1, d = 0.5,
    train.with.bnn = FALSE, use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2
)

# Get coefficients by majority voting for a given categorical variable
coeff <- MajorityVotingFeatures(object = sce, label = "cell_type")
gene_coeff$summary
order.rows <- order(coeff$feature_coeff$Monocyte$coeff_clt2,
    decreasing = TRUE
)
head(coeff$feature_coeff$Monocyte[order.rows, ], n = 10)

## End(Not run)
```

PCAElbowPlot *Elbow plot of the standard deviations of the principal components*

Description

Draw an elbow plot of the standard deviations of the principal components to deduce an appropriate value for p.

Usage

```
PCAElbowPlot.SingleCellExperiment(object, dimred.name, return.plot)

## S4 method for signature 'SingleCellExperiment'
PCAElbowPlot(object, dimred.name = "PCA", return.plot = FALSE)
```

Arguments

object	A SingleCellExperiment object obtained after running RunParallelDivisiveICP.
dimred.name	Dimensional reduction name of the PCA to select from reducedDimNames(object). By default "PCA".
return.plot	logical indicating if the ggplot2 object should be returned. By default FALSE.

Value

A ggplot2 object, if return.plot=TRUE.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 2, L = 25, C = 1, train.k.nn = 10,
    train.k.nn.prop = NULL, use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)

# Integrated PCA
set.seed(125) # to ensure reproducibility for the default 'irlba' method
sce <- RunPCA(object = sce, assay.name = "joint.probability", p = 10)
```

```
# Plot result
cowplot::plot_grid(
    PlotDimRed(
        object = sce, color.by = "Batch",
        legend.nrow = 1
    ),
    PlotDimRed(
        object = sce, color.by = "Species",
        legend.nrow = 1
    ),
    ncol = 2
)

# Plot Elbow
PCAElbowPlot(sce)
```

PlotClusterTree *Plot cluster tree*

Description

Plot cluster tree by or cluster probability or categorical variable.

Usage

```
PlotClusterTree.SingleCellExperiment(
  object,
  icp.run,
  color.by,
  use.color,
  seed.color,
  legend.title,
  return.data
)

## S4 method for signature 'SingleCellExperiment'
PlotClusterTree(
  object,
  icp.run,
  color.by = NULL,
  use.color = NULL,
  seed.color = 123,
  legend.title = color.by,
  return.data = FALSE
)
```

Arguments

object	An object of SingleCellExperiment class.
icp.run	ICP run(s) to retrieve from metadata(object)$coralysis$joint.probability. By default NULL, i.e., all are retrieved. Specify a numeric vector to retrieve a specific set of tables.
color.by	Categorical variable available in colData(object) to plot. If NULL the cluster probability is represented instead. By default NULL.
use.color	Character specifying the colors. By default NULL, i.e., colors are randomly chosen based on the seed given at seed.color.
seed.color	Seed to randomly select colors. By default 123.
legend.title	Legend title. By default the same as given at color.by. Ignored if color.by is NULL.
return.data	Return data frame used to plot. Logical. By default FALSE, i.e., only the plot is returned.

Value

A plot of class ggplot or a list with a plot of class ggplot and a data frame.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch", k = 4,
    L = 25, C = 1, d = 0.5, train.with.bnn = FALSE,
    use.cluster.seed = FALSE, build.train.set = FALSE,
    ari.cutoff = 0.1, threads = 2, RNGseed = 1024
)
```

```
# Plot probability
PlotClusterTree(object = sce, icp.run = 2)

# Plot batch label distribution
PlotClusterTree(object = sce, icp.run = 2, color.by = "Batch")

# Plot species label distribution
PlotClusterTree(object = sce, icp.run = 2, color.by = "Species")
```

PlotDimRed *Plot dimensional reduction categorical variables*

Description

Plot categorical variables in dimensional reduction.

Usage

```
PlotDimRed.SingleCellExperiment(
  object,
  color.by,
  dimred,
  dims,
  use.color,
  point.size,
  point.stroke,
  legend.nrow,
  seed.color,
  label,
  plot.theme,
  rasterise,
  rasterise.dpi,
  legend.justification,
  legend.size,
  legend.title
)

## S4 method for signature 'SingleCellExperiment'
PlotDimRed(
  object,
  color.by,
  dimred = tail(reducedDimNames(object), n = 1),
  dims = 1:2,
  use.color = NULL,
  point.size = 1,
  point.stroke = 1,
  legend.nrow = 2,
```

```
  seed.color = 123,
  label = FALSE,
  plot.theme = theme_classic(),
  rasterise = (ncol(object) <= 30000),
  rasterise.dpi = 300,
  legend.justification = "center",
  legend.size = 10,
  legend.title = color.by
)
```

Arguments

object	An object of SingleCellExperiment class.
color.by	Categorical variable available in colData(object) to plot.
dimred	Dimensional reduction available in ReducedDimNames(object) to plot. By default the last dimensional reduction in the object is used.
dims	Dimensions from the dimensional reduction embedding to plot.
use.color	Character specifying the colors. By default NULL, i.e., colors are randomly chosen based on the seed given at seed.color.
point.size	Size of points. By default 1.
point.stroke	Size of stroke. By default 1.
legend.nrow	Display legend items by this number of rows. By default 2.
seed.color	Seed to randomly select colors. By default 123.
label	Logical to add or not categorical labels to the centroid categories. By default FALSE, i.e., labels are not added.
plot.theme	Plot theme available in ggplot2. By default theme_classic().
rasterise	Logical specifying if points should be rasterised or not. By default TRUE, if more than 3e4 cells, otherwise FALSE.
rasterise.dpi	In case rasterise = TRUE, DPI to use. By default 300.
legend.justification	Legend justification. By default "center".
legend.size	Legend size. By default 10
legend.title	Legend title. By default the same as given at color.by.

Value

A plot of class ggplot.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
```

```
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Compute dimensional reduction
sce <- RunPCA(
    object = sce, assay.name = "logcounts", p = 4,
    pca.method = "stats"
)

# Plot batch
PlotDimRed(object = sce, color.by = "Batch", dimred = "PCA", legend.nrow = 1)

# Plot cell type annotations
PlotDimRed(
    object = sce, color.by = "Species", legend.nrow = 1,
    dimred = "PCA", label = TRUE
)
```

PlotExpression *Plot dimensional reduction feature expression*

Description

Plot feature expression in dimensional reduction.

Usage

```
PlotExpression.SingleCellExperiment(
  object,
  color.by,
  dimred,
  scale.values,
  color.scale,
  plot.theme,
  legend.title,
  point.size,
  point.stroke
)

## S4 method for signature 'SingleCellExperiment'
```

```
PlotExpression(
  object,
  color.by,
  dimred = tail(reducedDimNames(object), n = 1),
  scale.values = FALSE,
  color.scale = "inferno",
  plot.theme = theme_classic(),
  legend.title = color.by,
  point.size = 1,
  point.stroke = 1
)
```

Arguments

object	An object of SingleCellExperiment class.
color.by	Categorical variable available in colData(object) to plot.
dimred	Dimensional reduction available in ReducedDimNames(object) to plot. By default the last dimensional reduction in the object is used.
scale.values	Logical specifying if values should be scaled. By default FALSE, i.e., values are not scaled.
color.scale	Character of color scale palette to be passed to ggplot2::scale_color_viridis_c. By default inferno. Other palettes are also available such as viridis.
plot.theme	Plot theme available in ggplot2. By default theme_classic().
legend.title	Legend title. By default the same as given at color.by.
point.size	Size of points. By default 1.
point.stroke	Size of stroke. By default 1.

Value

A plot of class ggplot.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))
```

```
# Compute dimensional reduction
sce <- RunPCA(
    object = sce, assay.name = "logcounts", p = 4,
    pca.method = "stats"
)

# Plot expression level of one or more features
## one
PlotExpression(object = sce, color.by = "Petal.Width")

## more than one
features <- row.names(sce)[1:4]
exp.plots <- lapply(X = features, FUN = function(x) {
    PlotExpression(object = sce, color.by = x, scale.values = TRUE)
})
cowplot::plot_grid(plotlist = exp.plots, ncol = 2, align = "vh")
```

PrepareData *Prepare* SingleCellExperiment *object for analysis*

Description

This function prepares the SingleCellExperiment object for analysis. The only required input is an object of class SingleCellExperiment with at least data in the logcounts slot.

Usage

```
PrepareData.SingleCellExperiment(object)

## S4 method for signature 'SingleCellExperiment'
PrepareData(object)
```

Arguments

object	An object of SingleCellExperiment class.

Value

An object of SingleCellExperiment class.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
```

```
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))
sce <- PrepareData(sce)
```

ReferenceMapping *Reference mapping*

Description

This function allows to project new query data sets onto a reference built with Coralysis as well as transfer cell labels from the reference to queries.

Usage

```
ReferenceMapping.SingleCellExperiment(
  ref,
  query,
  ref.label,
  label.prune.cutoff,
  scale.query.by,
  project.umap,
  select.icp.models,
  k.nn,
  dimred.name.prefix
)

## S4 method for signature 'SingleCellExperiment,SingleCellExperiment'
ReferenceMapping(
  ref,
  query,
  ref.label,
  label.prune.cutoff = 0.5,
  scale.query.by = NULL,
  project.umap = FALSE,
  select.icp.models = metadata(ref)$coralysis$pca.params$select.icp.tables,
  k.nn = 10,
  dimred.name.prefix = ""
)
```

Arguments

ref	An object of SingleCellExperiment class trained with Coralysis and after running RunPCA(..., return.model = TRUE) function.
query	An object of SingleCellExperiment class to project onto ref.
ref.label	A character cell metadata column name from the ref object to transfer to the queries.
label.prune.cutoff	A numeric cutoff value used to prune low-confidence predicted cell labels, based on the confidence probability scores stored in the coral_probability column of colData. By default is 0.5, i.e., cell labels with confidence scores less than or equal to 0.5 are considered unclassified and set to NA. The resulting pruned cell labels are stored in pruned_coral_labels. Set to 0 to ignore it.
scale.query.by	Should the query data be scaled by cell or by feature. By default is NULL, i.e., is not scaled. Scale it if reference was scaled.
project.umap	Project query data onto reference UMAP (logical). By default FALSE. If TRUE, the ref object needs to have a UMAP embedding obtained with RunUMAP(..., return.model = TRUE) function.
select.icp.models	Select the reference ICP models to use for query cluster probability prediction. By default metadata(ref)$coralysis$pca.params$select.icp.tables, i.e., the models selected to compute the reference PCA are selected. If NULL all are used. Otherwise a numeric vector should be given to select the ICP models of interest.
k.nn	The number of k nearest neighbors to use in the classification KNN algorithm used to transfer labels from the reference to queries (integer). By default 10.
dimred.name.prefix	Dimensional reduction name prefix to add to the computed PCA and UMAP. By default nothing is added, i.e., dimred.name.prefix = "".

Value

An object of SingleCellExperiment class.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
```

```
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Create reference & query SCE objects
ref <- sce[, sce$Batch == "b1"]
query <- sce[, sce$Batch == "b2"]

# 1) Train the reference
set.seed(123)
ref <- RunParallelDivisiveICP(
    object = ref, k = 2, L = 25, C = 1,
    train.k.nn = 10, train.k.nn.prop = NULL,
    use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)
# 2) Compute reference PCA & UMAP
ref <- RunPCA(ref, p = 5, return.model = TRUE, pca.method = "stats")
set.seed(123)
ref <- RunUMAP(ref, return.model = TRUE)

# Plot
PlotDimRed(object = ref, color.by = "Species", legend.nrow = 1)

# 3) Project & predict query cell labels
map <- ReferenceMapping(
    ref = ref, query = query, ref.label = "Species",
    project.umap = TRUE
)

# Confusion matrix: predictions (rows) x ground-truth (cols)
preds_x_truth <- table(map$coral_labels, map$Species)
print(preds_x_truth)

# Accuracy score
acc <- sum(diag(preds_x_truth)) / sum(preds_x_truth) * 100
print(paste0("Coralysis accuracy score: ", round(acc), "%"))

# Visualize: ground-truth, prediction, confidence scores
cowplot::plot_grid(
    PlotDimRed(
        object = map, color.by = "Species",
        legend.nrow = 1
    ),
    PlotDimRed(
        object = map, color.by = "coral_labels",
        legend.nrow = 1
    ),
    PlotExpression(
        object = map, color.by = "coral_probability",
        color.scale = "viridis"
    ),
```

```
    ncol = 2, align = "vh"
)
```

RunParallelDivisiveICP *Multi-level integration*

Description

Run divisive ICP clustering in parallel in order to perform multi-level integration.

Usage

```
RunParallelDivisiveICP.SingleCellExperiment(
  object,
  batch.label,
  k,
  d,
  L,
  r,
  C,
  reg.type,
  max.iter,
  threads,
  icp.batch.size,
  train.with.bnn,
  train.k.nn,
  train.k.nn.prop,
  build.train.set,
  build.train.params,
  scale.by,
  use.cluster.seed,
  divisive.method,
  allow.free.k,
  ari.cutoff,
  verbose,
  RNGseed,
  BPPARAM
)

## S4 method for signature 'SingleCellExperiment'
RunParallelDivisiveICP(
  object,
  batch.label = NULL,
  k = 16,
  d = 0.3,
```

```
  L = 50,
  r = 5,
  C = 0.3,
  reg.type = "L1",
  max.iter = 200,
  threads = 0,
  icp.batch.size = Inf,
  train.with.bnn = TRUE,
  train.k.nn = 10,
  train.k.nn.prop = 0.3,
  build.train.set = TRUE,
  build.train.params = list(),
  scale.by = NULL,
  use.cluster.seed = TRUE,
  divisive.method = "cluster.batch",
  allow.free.k = TRUE,
  ari.cutoff = 0.3,
  verbose = FALSE,
  RNGseed = 123,
  BPPARAM = NULL
)
```

Arguments

object	An object of SingleCellExperiment class.
batch.label	A variable name (of class character) available in the cell metadata colData(object) with the batch labels (character or factor) to use. The variable provided must not contain NAs. By default NULL, i.e., cells are sampled evenly regardless their batch.
k	A positive integer power of two, i.e., 2**n, where n>0, specifying the number of clusters in the last Iterative Clustering Projection (ICP) round. Decreasing k leads to smaller cell populations diversity and vice versa. Default is 16, i.e., the divisive clustering 2 -> 4 -> 8 -> 16 is performed.
d	A numeric greater than 0 and smaller than 1 that determines how many cells n are down- or oversampled from each cluster into the training data (n=N/k*d), where N is the total number of cells, k is the number of clusters in ICP. Increasing above 0.3 leads greadually to smaller cell populations diversity. Default is 0.3.
L	A positive integer greater than 1 denoting the number of the ICP runs to run. Default is 50.
r	A positive integer that denotes the number of reiterations performed until the ICP algorithm stops. Increasing recommended with a significantly larger sample size (tens of thousands of cells). Default is 5.
C	A positive real number denoting the cost of constraints violation in the L1-regularized logistic regression model from the LIBLINEAR library. Decreasing leads to more stringent feature selection, i.e. less features are selected that are used to build the projection classifier. Decreasing to a very low value (~ 0.01) can lead to failure to identify central cell populations. Default 0.3.

reg.type "L1" or "L2". L2-regularization was not investigated in the manuscript, but it leads to a more conventional outcome (less subpopulations). Default is "L1".

max.iter A positive integer that denotes the maximum number of iterations performed until ICP stops. This parameter is only useful in situations where ICP converges extremely slowly, preventing the algorithm to run too long. In most cases, reaching the number of reiterations (r=5) terminates the algorithm. Default is 200.

threads A positive integer that specifies how many logical processors (threads) to use in parallel computation. Set 1 to disable parallelism altogether or 0 to use all available threads except one. Default is 0. This argument is ignored if BPPARAM is provided as threads should be given directly to the BiocParallelParam object.

icp.batch.size A positive integer that specifies how many cells to randomly select. It behaves differently depending on build.train.set. If build.train.set=FALSE, it randomly samples cells for each ICP run from the complete dataset. If build.train.set=TRUE, it randomly samples cells once, before building the training set with the sampled cells (per batch if batch.label different than NULL). Default is Inf, which means using all cells.

train.with.bnn Train data with batch nearest neighbors. Default is TRUE. Only used if batch.label is given.

train.k.nn Train data with batch nearest neighbors using k nearest neighbors. Default is 10. Only used if train.with.bnn is TRUE and train.k.nn.prop is NULL.

train.k.nn.prop

A numeric (higher than 0 and lower than 1) corresponding to the fraction of cells per cluster to use as train.k.nn nearest neighbors. If NULL the number of train.k.nn nearest neighbors is equal to train.k.nn. If given, train.k.nn parameter is ignored and train.k.nn is calculated based on train.k.nn.prop. By default 0.3 meaning that 30 proportions for the different divisive clustering rounds can be given, otherwise the same value is given for all.

build.train.set

Logical specifying if a training set should be built from the data or the whole data should be used for training. By default TRUE.

build.train.params

A list of parameters to be passed to the function AggregateDataByBatch(). Only provided if build.train.set is TRUE.

scale.by A character specifying if the data should be scaled by cell or by feature before training. Default is NULL, i.e., the data is not scaled before training.

use.cluster.seed

Should the same starting clustering result be provided to ensure more reproducible results (logical). If FALSE, each ICP run starts with a total random clustering and, thus, independent clustering. By default TRUE, i.e., the same clustering result is provided based on PCA density sampling. If batch.label different than NULL, the PCA density sampling is performed in a batch wise manner.

divisive.method

Divisive method (character). One of "random" (randomly sample two clusters out of every cluster previously found), "cluster" or "cluster.batch" (sample two clusters out of every cluster previously found based on the cluster probability distribution across batches or per batch). By default "cluster.batch". If

batch.label is NULL, it is automatically set to cluster. It can be set to random if explicitly provided.

allow.free.k — Allow free k (logical). Allow ICP algorithm to decrease the k given in case it does not find k target clusters. By default TRUE.

ari.cutoff — Include ICP models and probability tables with an Adjusted Rand Index higher than ari.cutoff (numeric). By default 0.3. A value that can range between 0 (include all) and lower than 1.

verbose — A logical value to print verbose during the ICP run in case. Default is FALSE. Verbose might help debugging errors by printing intermediate ICP projection results.

RNGseed — Seed number passed to the parallel backend via BiocParallel to ensure reproducibility. Defaults to 123. If the BPPARAM parameter is provided, RNGseed is ignored and should be set within BPPARAM.

BPPARAM — A BiocParallelParam object specifying the parallel backend to use. This controls how tasks are distributed across workers. Use MulticoreParam (for Unix-like systems) and SnowParam (for Windows or cross-platform). If not specified, i.e., NULL, the default backend uses MulticoreParam for Unix-like systems and SnowParam for Windows.

Value

A SingleCellExperiment object.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 2, L = 25, C = 1, train.k.nn = 10,
    train.k.nn.prop = NULL, use.cluster.seed = FALSE,
```

```
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)

# Integrated PCA
set.seed(125) # to ensure reproducibility for the default 'irlba' method
sce <- RunPCA(object = sce, assay.name = "joint.probability", p = 10)

# Plot result
cowplot::plot_grid(
    PlotDimRed(
        object = sce, color.by = "Batch",
        legend.nrow = 1
    ),
    PlotDimRed(
        object = sce, color.by = "Species",
        legend.nrow = 1
    ),
    ncol = 2
)
```

RunPCA *Principal Component Analysis*

Description

Perform principal component analysis using assays or the joint probability matrix as input.

Usage

```
RunPCA.SingleCellExperiment(
  object,
  assay.name,
  p,
  scale,
  center,
  threshold,
  pca.method,
  return.model,
  select.icp.tables,
  features,
  dimred.name
)

## S4 method for signature 'SingleCellExperiment'
RunPCA(
  object,
  assay.name = "joint.probability",
```

```
  p = 50,
  scale = TRUE,
  center = TRUE,
  threshold = 0,
  pca.method = "irlba",
  return.model = FALSE,
  select.icp.tables = NULL,
  features = NULL,
  dimred.name = "PCA"
)
```

Arguments

object	A SingleCellExperiment object.
assay.name	Name of the assay to compute PCA. One of assayNames(object) or joint.probability. By default joint.probability is used. Use joint.probability to obtain an integrated embedding after running RunParallelDivisiveICP. One of the assays in assayNames(object) can be provided before performing integration to assess if data requires integration.
p	A positive integer denoting the number of principal components to calculate and select. Default is 50.
scale	A logical specifying whether the probabilities should be standardized to unit-variance before running PCA. Default is TRUE.
center	A logical specifying whether the probabilities should be centered before running PCA. Default is TRUE.
threshold	A threshold for filtering out ICP runs before PCA with the lower terminal projection accuracy below the threshold. Default is 0.
pca.method	A character specifying the PCA method. One of "irlba" (default), "RSpectra" or "stats". Set seed before, if the method is "irlba" to ensure reproducibility.
return.model	A logical specifying if the PCA model should or not be retrieved. By default FALSE. Only implemented for pca.method = "stats". If TRUE, the pca.method is coerced to "stats".
select.icp.tables	Select the ICP cluster probability tables to perform PCA. By default NULL, i.e., all are used, except if the ICP tables were obtained with the function RunParallelDivisiveICP, in which the ICP tables correspond to the last round of divisive clustering for every epoch. A vector of integers should be given otherwise.
features	A character of feature names matching row.names(object) to select from before computing PCA. Only used if assay.name is one of the assays in assayNames(object), otherwise it is ignored.
dimred.name	Dimensional reduction name given to the returned PCA. By default "PCA".

Value

object of SingleCellExperiment class

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 2, L = 25, C = 1, train.k.nn = 10,
    train.k.nn.prop = NULL, use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)

# Integrated PCA
set.seed(125) # to ensure reproducibility for the default 'irlba' method
sce <- RunPCA(object = sce, assay.name = "joint.probability", p = 10)

# Plot result
cowplot::plot_grid(
    PlotDimRed(
        object = sce, color.by = "Batch",
        legend.nrow = 1
    ),
    PlotDimRed(
        object = sce, color.by = "Species",
        legend.nrow = 1
    ),
    ncol = 2
)
```

RunTSNE *Barnes-Hut implementation of t-Distributed Stochastic Neighbor Embedding (t-SNE)*

Description

Run nonlinear dimensionality reduction using t-SNE with the PCA-transformed consensus matrix as input.

Usage

```
RunTSNE.SingleCellExperiment(
  object,
  dims,
  dimred.type,
  perplexity,
  dimred.name,
  ...
)

## S4 method for signature 'SingleCellExperiment'
RunTSNE(
  object,
  dims = NULL,
  dimred.type = "PCA",
  perplexity = 30,
  dimred.name = "TSNE",
  ...
)
```

Arguments

object	Object of SingleCellExperiment class.
dims	Dimensions to select from dimred.type. By default NULL, i.e., all the dimensions are selected. Provide a numeric vector to select a specific range, e.g., dims = 1:10 to select the first 10 dimensions.
dimred.type	Dimensional reduction type to use. By default "PCA".
perplexity	Perplexity of t-SNE.
dimred.name	Dimensional reduction name given to the returned t-SNE. By default "TSNE".
...	Parameters to be passed to the Rtsne function. The parameters given should match the parameters accepted by the Rtsne function. Check possible parameters with ?Rtsne::Rtsne.

Value

A SingleCellExperiment object.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
```

```
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Run PCA
set.seed(125) # to ensure reproducibility for the default 'irlba' method
sce <- RunPCA(
    object = sce, assay.name = "logcounts",
    pca.method = "stats", p = nrow(sce)
)

# Run t-SNE
set.seed(125) # to ensure reproducibility for the default 'irlba' method
sce <- RunTSNE(object = sce, dimred.type = "PCA", check_duplicates = FALSE)

# Plot result
cowplot::plot_grid(
    PlotDimRed(
        object = sce, color.by = "Batch",
        legend.nrow = 1
    ),
    PlotDimRed(
        object = sce, color.by = "Species",
        legend.nrow = 1
    ),
    ncol = 2
)
```

RunUMAP *Uniform Manifold Approximation and Projection (UMAP)*

Description

Run nonlinear dimensionality reduction using UMAP with a dimensional reduction as input.

Usage

```
RunUMAP.SingleCellExperiment(
  object,
  dims,
  dimred.type,
```

```
  return.model,
  umap.method,
  dimred.name,
  ...
)

## S4 method for signature 'SingleCellExperiment'
RunUMAP(
  object,
  dims = NULL,
  dimred.type = "PCA",
  return.model = FALSE,
  umap.method = "umap",
  dimred.name = "UMAP",
  ...
)
```

Arguments

object	An object of SingleCellExperiment class.
dims	Dimensions to select from dimred.type. By default NULL, i.e., all the dimensions are selected. Provide a numeric vector to select a specific range, e.g., dims = 1:10 to select the first 10 dimensions.
dimred.type	Dimensional reduction type to use. By default "PCA".
return.model	Return UMAP model. By default FALSE.
umap.method	UMAP method to use: "umap" or "uwot". By default "umap".
dimred.name	Dimensional reduction name given to the returned UMAP. By default "UMAP".
...	Parameters to be passed to the umap function. The parameters given should match the parameters accepted by the umap function depending on the umap.method given. Check possible parameters with ?umap::umap or ?uwot::umap depending if umap.method is "umap" or "uwot".

Value

A SingleCellExperiment object.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
```

```
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 2, L = 25, C = 1, train.k.nn = 10,
    train.k.nn.prop = NULL, use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)

# Integrated PCA
set.seed(125) # to ensure reproducibility for the default 'irlba' method
sce <- RunPCA(object = sce, assay.name = "joint.probability", p = 10)

# Plot result
cowplot::plot_grid(
    PlotDimRed(
        object = sce, color.by = "Batch",
        legend.nrow = 1
    ),
    PlotDimRed(
        object = sce, color.by = "Species",
        legend.nrow = 1
    ),
    ncol = 2
)

# Run UMAP
set.seed(123)
sce <- RunUMAP(sce, dimred.type = "PCA")

# Plot results
# Plot result
cowplot::plot_grid(
    PlotDimRed(
        object = sce, color.by = "Batch",
        legend.nrow = 1
    ),
    PlotDimRed(
        object = sce, color.by = "Species",
        legend.nrow = 1
    ),
    ncol = 2
)
```

SummariseCellClusterProbability
Summarise ICP cell cluster probability

Description

Summarise ICP cell cluster probability table(s)

Usage

```
SummariseCellClusterProbability.SingleCellExperiment(
  object,
  icp.run,
  icp.round,
  funs,
  scale.funs,
  save.in.sce
)

## S4 method for signature 'SingleCellExperiment'
SummariseCellClusterProbability(
  object,
  icp.run = NULL,
  icp.round = NULL,
  funs = c("mean", "median"),
  scale.funs = TRUE,
  save.in.sce = TRUE
)
```

Arguments

object	An object of SingleCellExperiment class with ICP cell cluster probability tables saved in metadata(object)$coralysis$joint.probability. After running RunParallelDivisiveICP.
icp.run	ICP run(s) to retrieve from metadata(object)$coralysis$joint.probability. By default NULL, i.e., all are retrieved. Specify a numeric vector to retrieve a specific set of tables.
icp.round	ICP round(s) to retrieve from metadata(object)$coralysis$joint.probability. By default NULL, i.e., all are retrieved.
funs	Functions to summarise ICP cell cluster probability: "mean" and/or "median". By default c("mean", "median"), i.e, both mean and median are calculated. Set to NULL to not estimate any.
scale.funs	Scale in the range 0-1 the summarised probability obtained with funs. By default TRUE, i.e., summarised probability will be scaled in the 0-1 range.

save.in.sce Save the data frame into the cell metadata from the SingleCellExperiment object or return the data frame. By default TRUE, i.e., the summary of probabilities retrieved is save in the SCE object in colData(object).

Value

A data frame or a SingleCellExperiment object with ICP cell cluster probability summarised.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Prepare SCE object for analysis
sce <- PrepareData(sce)

# Multi-level integration (just for highlighting purposes; use default parameters)
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "Batch",
    k = 2, L = 25, C = 1, train.k.nn = 10,
    train.k.nn.prop = NULL, use.cluster.seed = FALSE,
    build.train.set = FALSE, ari.cutoff = 0.1,
    threads = 2, RNGseed = 1024
)

# Integrated PCA
set.seed(125) # to ensure reproducibility for the default 'irlba' method
sce <- RunPCA(object = sce, assay.name = "joint.probability", p = 10)

# Summarise cluster probability
sce <- SummariseCellClusterProbability(
    object = sce, icp.round = 1,
    save.in.sce = TRUE
) # saved in 'colData'

# Plot the clustering result for ICP run no. 3
PlotDimRed(object = sce, color.by = "icp_run_round_3_1_clusters")
```

```
# Plot Coralysis mean cell cluster probabilities
PlotExpression(
    object = sce, color.by = "mean_probs",
    color.scale = "viridis"
)
```

TabulateCellBinsByGroup

Tabulate cell bins by group

Description

Frequency of cells per cell cluster probability bin by group for each label. The label has to be specified beforehand to the function `BinCellClusterProbability()`.

Usage

```
TabulateCellBinsByGroup.SingleCellExperiment(object, group, relative, margin)

## S4 method for signature 'SingleCellExperiment'
TabulateCellBinsByGroup(object, group, relative = FALSE, margin = 1)
```

Arguments

`object`	An object of `SingleCellExperiment` class obtained with the function `BinCellClusterProbability()`.
`group`	Character specifying the `colData` variable from the `SingleCellExperiment` object provided to the function `BinCellClusterProbability()` to use as categorical group variable.
`relative`	Logical specifying if relative proportions of cell bins per group should be returned. By default `FALSE`, i.e., absolute values are returned.
`margin`	If `relative` is `TRUE`, proportions should be calculated by: rows (`1`, the default); columns (`2`); or overall (`NULL`).

Value

A list of tables with the frequency of cells per bin of cell cluster probability by group for each label.

Examples

```
# Packages
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Import data from Zenodo
data.url <- "https://zenodo.org/records/14845751/files/pbmc_10Xassays.rds?download=1"
sce <- readRDS(file = url(data.url))

# Prepare data
```

```
sce <- PrepareData(object = sce)

# Multi-level integration - 'L = 4' just for highlighting purposes
set.seed(123)
sce <- RunParallelDivisiveICP(
    object = sce, batch.label = "batch", L = 4,
    threads = 2
)

# Cell states SCE object for a given cell type annotation or clustering
cellstate.sce <- BinCellClusterProbability(
    object = sce, label = "cell_type",
    icp.round = 4, bins = 20
)
cellstate.sce

# Tabulate cell bins by group
# give an interesting variable to the "group" parameter
cellbins.tables <- TabulateCellBinsByGroup(
    object = cellstate.sce,
    group = "batch",
    relative = TRUE,
    margin = 1
)
```

VlnPlot *Visualization of feature expression using violin plots*

Description

The VlnPlot function enables visualizing expression levels of feature(s), across clusters using violin plots.

Usage

```
VlnPlot.SingleCellExperiment(
  object,
  clustering.label,
  features,
  return.plot,
  rotate.x.axis.labels
)

## S4 method for signature 'SingleCellExperiment'
VlnPlot(
  object,
  clustering.label,
```

```
  features,
  return.plot = FALSE,
  rotate.x.axis.labels = FALSE
)
```

Arguments

object	of SingleCellExperiment class
clustering.label	A variable name (of class character) available in the cell metadata colData(object) with the clustering labels (character or factor) to use.
features	Feature names to plot by cluster (character) matching row.names(object).
return.plot	return.plot whether to return the ggplot2 object. Default is FALSE.
rotate.x.axis.labels	a logical denoting whether the x-axis labels should be rotated 90 degrees or just draw it. Default is FALSE.

Value

A ggplot2 object if return.plot=TRUE.

Examples

```
# Import package
suppressPackageStartupMessages(library("SingleCellExperiment"))

# Create toy SCE data
batches <- c("b1", "b2")
set.seed(239)
batch <- sample(x = batches, size = nrow(iris), replace = TRUE)
sce <- SingleCellExperiment(
    assays = list(logcounts = t(iris[, 1:4])),
    colData = DataFrame(
        "Species" = iris$Species,
        "Batch" = batch
    )
)
colnames(sce) <- paste0("samp", 1:ncol(sce))

# Plot features by clustering/grouping variable
VlnPlot(sce,
    clustering.label = "Species",
    features = row.names(sce)[1:4],
    rotate.x.axis.labels = TRUE
)
```

Index