Package 'SpatialDDLS'

July 21, 2025

Type Package

Title Deconvolution of Spatial Transcriptomics Data Based on Neural Networks

Version 1.0.3

Maintainer Diego Mañanes <dmananesc@cnic.es>

Description Deconvolution of spatial transcriptomics data based on neural networks and single-cell RNA-seq data. SpatialDDLS implements a workflow to create neural network models able to make accurate estimates of cell composition of spots from spatial transcriptomics data using deep learning and the meaningful information provided by single-cell RNA-seq data. See Torroja and Sanchez-Cabo (2019) <doi:10.3389/fgene.2019.00978> and Mañanes et al. (2024) <doi:10.1093/bioinformatics/btae072> to get an o amples of its performance.

License GPL-3

URL https://diegommcc.github.io/SpatialDDLS/, https://github.com/diegommcc/SpatialDDLS

BugReports https://github.com/diegommcc/SpatialDDLS/issues

Encoding UTF-8

Depends R (>= 4.0.0)

Imports rlang, grr, Matrix, methods, SpatialExperiment, SingleCellExperiment, SummarizedExperiment, zinbwave, stats, pbapply, S4Vectors, dplyr, reshape2, gtools, reticulate, keras, tensorflow, FNN, ggplot2, ggpubr, scran, scuttle

Suggests knitr, rmarkdown, BiocParallel, rhdf5, DelayedArray, DelayedMatrixStats, HDF5Array, testthat, ComplexHeatmap, grid, bluster, lsa, irlba

SystemRequirements Python (>= 2.7.0), TensorFlow (https://www.tensorflow.org/)

RoxygenNote 7.3.2

Collate 'AllClasses.R' 'AllGenerics.R' 'SpatialDDLS.R' 'dnnModel.R' 'evalMetrics.R' 'interGradientsDL.R' 'loadData.R' 'plotSpatialCoor.R' 'simMixedSpots.R' 'simSingleCell.R' 'spatialClustering.R' 'utils.R'

VignetteBuilder knitr

NeedsCompilation no

Author Diego Mañanes [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7247-6794>),
Carlos Torroja [aut] (ORCID: <https://orcid.org/0000-0001-8914-3400>),
Fatima Sanchez-Cabo [aut] (ORCID:
<https://orcid.org/0000-0003-1881-1664>)

Repository CRAN

Date/Publication 2024-10-31 09:40:06 UTC

Contents

`barErrorPlot` *Generate bar error plots*

Description

Generate bar error plots by cell type (`CellType`) or by number of different cell types (`nCellTypes`) on test mixed transcriptional profiles.

Usage

```
barErrorPlot(
  object,
  error = "MSE",
  by = "CellType",
  dispersion = "se",
  filter.sc = TRUE,
  title = NULL,
  angle = NULL,
  theme = NULL
)
```

Arguments

object	SpatialDDLS object with trained.model slot containing metrics in the test.deconv.metrics slot of a DeconvDLModel object.
error	'MAE' or 'MSE'.
by	Variable used to show errors. Available options are: 'nCellTypes', 'CellType'.
dispersion	Standard error ('se') or standard deviation ('sd'). The former by default.
filter.sc	Boolean indicating whether single-cell profiles are filtered out and only correlation of results associated with mixed transcriptional profiles are shown (TRUE by default).
title	Title of the plot.
angle	Angle of ticks.
theme	**ggplot2** theme.

Value

A ggplot object.

See Also

calculateEvalMetrics corrExpPredPlot distErrorPlot blandAltmanLehPlot

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 20,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(20)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(20)),
    Cell_Type = sample(x = paste0("CellType", seq(6)), size = 20,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
```

```
  cell.type.column = "Cell_Type",
  num.sim.spots = 100,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
# training of DDLS model
SDDLS <- trainDeconvModel(
  object = SDDLS,
  batch.size = 10,
  num.epochs = 5
)
# evaluation using test data
SDDLS <- calculateEvalMetrics(object = SDDLS)
# bar error plots
barErrorPlot(
  object = SDDLS,
  error = "MSE",
  by = "CellType"
)
barErrorPlot(
  object = SDDLS,
  error = "MAE",
  by = "nCellTypes"
)
```

---

barPlotCellTypes *Bar plot of deconvoluted cell type proportions*

Description

Bar plot of deconvoluted cell type proportions.

Usage

```
barPlotCellTypes(
  data,
  colors = NULL,
  set = NULL,
  prediction = "Regularized",
  color.line = NA,
  x.label = "Spots",
  rm.x.text = FALSE,
  title = "Results of deconvolution",
  legend.title = "Cell types",
  angle = 90,
```

```
  theme = NULL,
  index.st = NULL
)
```

Arguments

data	SpatialDDLS object with the deconv.spots slot containing predicted cell type proportions.
colors	Vector of colors to be used.
set	Type of simplification performed during deconvolution. It can be simpli.set or simpli.maj (NULL by default).
prediction	Set of predicted cell proportions to be plotted. It can be "Regularized", "Intrinsic" or "Extrinsic".
color.line	Color of the border bars.
x.label	Label of x-axis.
rm.x.text	Logical value indicating whether to remove x-axis ticks (name of samples).
title	Title of the plot.
legend.title	Title of the legend plot.
angle	Angle of text ticks.
theme	**ggplot2** theme.
index.st	Name or index of the element wanted to be shown in the deconv.spots slot.

Value

A ggplot object with the provided cell proportions represented as a bar plot.

See Also

deconvSpatialDDLS

---

blandAltmanLehPlot *Generate Bland-Altman agreement plots between predicted and expected cell type proportions of test data*

Description

Generate Bland-Altman agreement plots between predicted and expected cell type proportions from test data. The Bland-Altman agreement plots can be shown all mixed or split by either cell type (CellType) or the number of cell types present in spots (nCellTypes). See the facet.by argument and examples for more information.

Usage

```
blandAltmanLehPlot(
  object,
  colors,
  color.by = "CellType",
  facet.by = NULL,
  log.2 = FALSE,
  filter.sc = TRUE,
  density = TRUE,
  color.density = "darkblue",
  size.point = 0.05,
  alpha.point = 1,
  ncol = NULL,
  nrow = NULL,
  title = NULL,
  theme = NULL,
  ...
)
```

Arguments

object	SpatialDDLS object with trained.model slot containing metrics in the test.deconv.metrics slot of a DeconvDLModel object.
colors	Vector of colors to be used.
color.by	Variable used to color data. Options are nCellTypes and CellType.
facet.by	Variable used to show the data in different panels. If NULL, the plot is not split into different panels. Options are nCellTypes (by number of different cell types) and CellType (by cell type).
log.2	Whether to show the Bland-Altman agreement plot in log2 space (FALSE by default).
filter.sc	Boolean indicating whether single-cell profiles are filtered out and only correlations of results associated with mixed spot profiles are shown (TRUE by default).
density	Boolean indicating whether density lines should be shown (TRUE by default).
color.density	Color of density lines if the density argument is TRUE.
size.point	Size of the points (0.1 by default).
alpha.point	Alpha of the points (0.1 by default).
ncol	Number of columns if facet.by is used.
nrow	Number of rows if facet.by is used.
title	Title of the plot.
theme	**ggplot2** theme.
...	Additional argument for the facet_wrap function of **ggplot2** if facet.by is not NULL.

Value

A ggplot object.

See Also

calculateEvalMetrics corrExpPredPlot distErrorPlot barErrorPlot

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 20,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(20)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(20)),
    Cell_Type = sample(x = paste0("CellType", seq(6)), size = 20,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
# training of DDLS model
SDDLS <- trainDeconvModel(
  object = SDDLS,
  batch.size = 15,
  num.epochs = 5
)
# evaluation using test data
SDDLS <- calculateEvalMetrics(object = SDDLS)
# Bland-Altman plot by cell type
blandAltmanLehPlot(
```

```
  object = SDDLS,
  facet.by = "CellType",
  color.by = "CellType"
)
# Bland-Altman plot of all samples mixed
blandAltmanLehPlot(
  object = SDDLS,
  facet.by = NULL,
  color.by = "CellType",
  alpha.point = 0.3,
  log2 = TRUE
)
```

`calculateEvalMetrics` *Calculate evaluation metrics on test mixed transcriptional profiles*

Description

Calculate evaluation metrics on test mixed transcriptional profiles. By default, absolute error (`AbsErr`), proportional absolute error (`ppAbsErr`), squared error (`SqrErr`), and proportional squared error (`ppSqrErr`) are calculated for each test mixed profile. In addition, each of these metrics is aggregated according to three criteria: cell type (`CellType`), probability bins in ranges of 0.1 (`pBin`), and number of different cell types present in the spot (`nCellTypes`).

Usage

```
calculateEvalMetrics(object)
```

Arguments

`object`	`SpatialDDLS` object with a trained model in `trained.model` slot and the actual cell proportions of test mixed profiles in `prob.cell.types` slot.

Value

A `SpatialDDLS` object with is a `DeconvDLModel` object. The calculated metrics are stored in the `test.deconv.metrics` slot of the `DeconvDLModel` object.

See Also

`distErrorPlot` `corrExpPredPlot` `blandAltmanLehPlot` `barErrorPlot`

cell.names *Get and set* cell.names *slot in a* PropCellTypes *object*

Description

Get and set cell.names slot in a PropCellTypes object

Usage

```
cell.names(object)

cell.names(object) <- value
```

Arguments

object	PropCellTypes object.
value	Matrix containing names of the mixed transcriptional profiles to be simulated as rows and cells to be used to simulate them as columns.

cell.types *Get and set* cell.types *slot in a* DeconvDLModel *object*

Description

Get and set cell.types slot in a DeconvDLModel object

Usage

```
cell.types(object)

cell.types(object) <- value
```

Arguments

object	DeconvDLModel object.
value	Vector with cell types considered by the deep neural network model.

corrExpPredPlot *Generate correlation plots between predicted and expected cell type proportions of test data*

Description

Generate correlation plots between predicted and expected cell type proportions of test data. Correlation plots can be shown all mixed or either split by cell type (`CellType`) or the number of different cell types present in the spots (`nCellTypes`).

Usage

```
corrExpPredPlot(
  object,
  colors,
  facet.by = NULL,
  color.by = "CellType",
  corr = "both",
  filter.sc = TRUE,
  pos.x.label = 0.01,
  pos.y.label = 0.95,
  sep.labels = 0.15,
  size.point = 0.1,
  alpha.point = 1,
  ncol = NULL,
  nrow = NULL,
  title = NULL,
  theme = NULL,
  ...
)
```

Arguments

`object`	`SpatialDDLS` object with `trained.model` slot containing metrics in the `test.deconv.metrics` slot of a `DeconvDLModel` object.
`colors`	Vector of colors to be used.
`facet.by`	Show data in different panels. Options are `nCellTypes` (number of different cell types) and `CellType` (cell type) (`NULL` by default).
`color.by`	Variable used to color data. Options are `nCellTypes` and `CellType`.
`corr`	Correlation value shown as an annotation on the plot. Available metrics are Pearson's correlation coefficient (`'pearson'`) and concordance correlation coefficient (`'ccc'`). It can be `'pearson'`, `'ccc'` or `'both'` (by default).
`filter.sc`	Boolean indicating whether single-cell profiles are filtered out and only mixed transcriptional profile errors are shown (`TRUE` by default).
`pos.x.label`	X-axis position of correlation annotations (0.95 by default).

pos.y.label	Y-axis position of correlation annotations (0.1 by default).
sep.labels	Space separating annotations if corr is equal to 'both' (0.15 by default).
size.point	Size of points (0.1 by default).
alpha.point	Alpha of points (0.1 by default).
ncol	Number of columns if facet.by is other than NULL.
nrow	Number of rows if facet.by is different from NULL.
title	Title of the plot.
theme	**ggplot2** theme.
...	Additional arguments for the facet_wrap function of **ggplot2** if facet.by is not NULL.

Value

A ggplot object.

See Also

calculateEvalMetrics distErrorPlot blandAltmanLehPlot barErrorPlot

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 20,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(20)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(20)),
    Cell_Type = sample(x = paste0("CellType", seq(6)), size = 20,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
```

```
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
# training of DDLS model
SDDLS <- trainDeconvModel(
  object = SDDLS,
  batch.size = 15,
  num.epochs = 5
)
# evaluation using test data
SDDLS <- calculateEvalMetrics(object = SDDLS)
# correlations by cell type
corrExpPredPlot(
  object = SDDLS,
  facet.by = "CellType",
  color.by = "CellType",
  corr = "both"
)
# correlations of all samples mixed
corrExpPredPlot(
  object = SDDLS,
  facet.by = NULL,
  color.by = "CellType",
  corr = "ccc",
  pos.x.label = 0.2,
  alpha.point = 0.3
)
```

`createSpatialDDLSobject` *Create a* `SpatialDDLS` *object*

Description

Create a `SpatialDDLS` object by providing single-cell RNA-seq data. Additionally, spatial transcriptomics data contained in `SpatialDDLS` objects can also be provided. It is recommended to provide both types of data to only use genes shared between both experiments.

Usage

```
createSpatialDDLSobject(
  sc.data,
  sc.cell.ID.column,
  sc.cell.type.column,
  sc.gene.ID.column,
```

```
  st.data,
  st.spot.ID.column,
  st.gene.ID.column,
  filter.mt.genes = "^mt-",
  sc.filt.genes.cluster = TRUE,
  sc.min.mean.counts = 1,
  sc.n.genes.per.cluster = 300,
  top.n.genes = 2000,
  sc.log.FC = TRUE,
  sc.min.counts = 1,
  sc.min.cells = 1,
  st.min.counts = 1,
  st.min.spots = 1,
  st.n.slides = 3,
  shared.genes = TRUE,
  sc.name.dataset.h5 = NULL,
  sc.file.backend = NULL,
  sc.name.dataset.backend = NULL,
  sc.compression.level = NULL,
  sc.chunk.dims = NULL,
  sc.block.processing = FALSE,
  verbose = TRUE,
  project = "SpatialDDLS-Proj"
)
```

Arguments

`sc.data`	Single-cell RNA-seq profiles to be used as reference. If data are provided from files, `single.cell.real` must be a vector of three elements: single-cell counts, cells metadata and genes metadata. On the other hand, If data are provided from a `SingleCellExperiment` object, single-cell counts must be present in the `assay` slot, cells metadata in the `colData` slot, and genes metadata in the `rowData` slot.
`sc.cell.ID.column`	Name or number of the column in cells metadata corresponding to cell names in expression matrix (single-cell RNA-seq data).
`sc.cell.type.column`	Name or column number corresponding to cell types in cells metadata.
`sc.gene.ID.column`	Name or number of the column in genes metadata corresponding to the names used for features/genes (single-cell RNA-seq data).
`st.data`	Spatial transcriptomics datasets to be deconvoluted. It can be a single `SpatialExperiment` object or a list of them.
`st.spot.ID.column`	Name or number of the column in spots metadata corresponding to spot names in expression matrix (spatial transcriptomics data).

st.gene.ID.column
: Name or number of the column in the genes metadata corresponding to the names used for features/genes (spatial transcriptomics data).

filter.mt.genes
: Regular expression matching mitochondrial genes to be ruled out (`^mt-` by default). If `NULL`, no filtering is performed.

sc.filt.genes.cluster
: Whether to filter single-cell RNA-seq genes according to a minimum threshold of non-zero average counts per cell type (`sc.min.mean.counts`). `TRUE` by default.

sc.min.mean.counts
: Minimum non-zero average counts per cluster to filter genes. 1 by default.

sc.n.genes.per.cluster
: Top n genes with the highest logFC per cluster (300 by default). See Details section for more details.

top.n.genes
: Maximum number of genes used for downstream steps (2000 by default). In case the number of genes after filtering is greater than `top.n.genes`, these genes will be set according to variability across the whole single-cell dataset.

sc.log.FC
: Whether to filter genes with a logFC less than 0.5 when `sc.filt.genes.cluster = TRUE` (`TRUE` by default).

sc.min.counts
: Minimum gene counts to filter (1 by default; single-cell RNA-seq data).

sc.min.cells
: Minimum of cells with more than `min.counts` (1 by default; single-cell RNA-seq data).

st.min.counts
: Minimum gene counts to filter (1 by default; spatial transcriptomics data).

st.min.spots
: Minimum of cells with more than `min.counts` (1 by default; spatial transcriptomics data).

st.n.slides
: Minimum number of slides (`SpatialExperiment` objects) in which a gene has to be expressed in order to keep it. This parameter is applicable only when multiple `SpatialExperiment` objects are provided. Genes not present in at least `st.n.slides` will be discarded. If no filtering is desired, set `st.n.slides = 1`.

shared.genes
: If set to `TRUE`, only genes present in both the single-cell and spatial transcriptomics data will be retained for further processing (`TRUE` by default).

sc.name.dataset.h5
: Name of the data set if HDF5 file is provided for single-cell RNA-seq data.

sc.file.backend
: Valid file path where to store the loaded for single-cell RNA-seq data as HDF5 file. If provided, data are stored in a HDF5 file as back-end using the **DelayedArray** and **HDF5Array** packages instead of being loaded into RAM. This is suitable for situations where you have large amounts of data that cannot be stored in memory. Note that operations on these data will be performed by blocks (i.e subsets of determined size), which may result in longer execution times. `NULL` by default.

sc.name.dataset.backend
: Name of the HDF5 file dataset to be used. Note that it cannot exist. If `NULL` (by default), a random dataset name will be generated.

`sc.compression.level`
: The compression level used if `sc.file.backend` is provided. It is an integer value between 0 (no compression) and 9 (highest and slowest compression). See ?`getHDF5DumpCompressionLevel` from the **HDF5Array** package for more information.

`sc.chunk.dims`
: Specifies dimensions that HDF5 chunk will have. If `NULL`, the default value is a vector of two items: the number of genes considered by `SpatialDDLS` object during the simulation, and only one sample in order to increase read times in the following steps. A larger number of columns written in each chunk may lead to longer read times.

`sc.block.processing`
: Boolean indicating whether single-cell RNA-seq data should be treated as blocks (only if data are provided as HDF5 file). `FALSE` by default. Note that using this functionality is suitable for cases where it is not possible to load data into RAM and therefore execution times will be longer.

`verbose`
: Show informative messages during the execution (`TRUE` by default).

`project`
: Name of the project for `SpatialDDLS` object.

Details

Filtering genes

In order to reduce the number of dimensions used for subsequent steps, `createSpatialDDLSobject` implements different strategies aimed at removing useless genes for deconvolution:

- Filtering at the cell level: genes less expressed than a determined cutoff in N cells are removed. See `sc.min.cells`/`st.min.cells` and `sc.min.counts`/`st.min.cells` parameters.
- Filtering at the cluster level (only for scRNA-seq data): if `sc.filt.genes.cluster == TRUE`, `createSpatialDDLSobject` sets a cutoff of non-zero average counts per cluster (`sc.min.mean.counts` parameter) and take only the `sc.n.genes.per.cluster` genes with the highest logFC per cluster. LogFCs are calculated using normalized logCPM of each cluster with respect to the average in the whole dataset). Finally, if the number of remaining genes is greater than `top.n.genes`, genes are ranked based on variance and the `top.n.genes` most variable genes are used for downstream analyses.

Single-cell RNA-seq data

Single-cell RNA-seq data can be provided from files (formats allowed: tsv, tsv.gz, mtx (sparse matrix) and hdf5) or a `SingleCellExperiment` object. Data will be stored in the `single.cell.real` slot, and must consist of three pieces of information:

- Single-cell counts: genes as rows and cells as columns.
- Cells metadata: annotations (columns) for each cell (rows).
- Genes metadata: annotations (columns) for each gene (rows).

If data are provided from files, `single.cell.real` argument must be a vector of three elements ordered so that the first file corresponds to the count matrix, the second to the cells metadata, and the last to the genes metadata. On the other hand, if data are provided as a `SingleCellExperiment` object, it must contain single-cell counts in `assay`, cells metadata in `colData`, and genes metadata in

rowData. Data must be provided without any transformation (e.g. log-transformation), raw counts are preferred.

Spatial transcriptomics data

It must be a SpatialExperiment object (or a list of them if more than one slide is going to be deconvoluted) containing the same information as the single-cell RNA-seq data: the count matrix, spots metadata, and genes metadata. Please, make sure the gene identifiers used the spatial and single-cell transcriptomics data are consistent.

Value

A SpatialDDLS object with the single-cell RNA-seq data provided loaded into the single.cell.real slot as a SingleCellExperiment object. If spatial transcriptomics data are provided, they will be loaded into the spatial.experiments slot.

See Also

estimateZinbwaveParams genMixedCellProp

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(100, lambda = 5), nrow = 40, ncol = 30,
      dimnames = list(paste0("Gene", seq(40)), paste0("RHC", seq(30)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(30)),
    Cell_Type = sample(x = paste0("CellType", seq(4)), size = 30,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(40))
  )
)
counts <- matrix(
  rpois(30, lambda = 5), ncol = 6,
  dimnames = list(paste0("Gene", 1:5), paste0("Spot", 1:6))
)
coordinates <- matrix(
  c(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3), ncol = 2
)
ste <- SpatialExperiment::SpatialExperiment(
  assays = list(counts = as.matrix(counts)),
  rowData = data.frame(Gene_ID = paste0("Gene", 1:5)),
  colData = data.frame(Cell_ID = paste0("Spot", 1:6)),
  spatialCoords = coordinates
)
```

```
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  st.data = ste,
  st.spot.ID.column = "Cell_ID",
  st.gene.ID.column = "Gene_ID",
  project = "Simul_example",
  sc.filt.genes.cluster = FALSE
)
```

`deconv.spots` *Get and set* `deconv.spots` *slot in a* `SpatialExperiment` *object*

Description

Get and set `deconv.spots` slot in a `SpatialExperiment` object

Usage

```
deconv.spots(object, index.st = NULL)

deconv.spots(object, index.st = NULL) <- value
```

Arguments

`object`	`SpatialExperiment` object.
`index.st`	Name or index of predicted cell proportions (same as for the `spatial.experiments` slot). If `NULL` (by default), all results are returned.
`value`	List of predicted cell type proportions for the experiments stored in the `spatial.experiments` slot.

`DeconvDLModel-class` *The DeconvDLModel Class*

Description

The `DeconvDLModel` object stores all the information related to deep neural network models. It consists of the trained model, the training history, and the predictions on test data. After running `calculateEvalMetrics`, it is possible to find the performance evaluation of the model on test data (see ?`calculateEvalMetrics` for details).

Details

The steps related to Deep Learning are carried out using the **keras** and **tensorflow** packages, which use the R6 classes system. If you want to save the `DeconvDLModel` object as an RDS file, **SpatialDDLS** provides a `saveRDS` generic function that transforms the R6 object containing the trained model into a native valid R object. Specifically, the model is converted into a list with the architecture of the network and the weights learned during training, which is the minimum information needed to use the model as a predictor. If you want to keep the optimizer state, see `?saveTrainedModelAsH5`. If you want to store either the `DeconvDLModel` or the `SpatialDDLS` objects on disk as RDA files, see `?preparingToSave`.

Slots

`model` Trained deep neural network. This slot can contain an R6 `keras.engine.sequential.Sequential` object or a list with two elements: the architecture of the model and the resulting weights after training.

`training.history` List with the evolution of the selected metrics during training.

`test.metrics` Performance of the model on test data.

`test.pred` Predicted cell type proportions on test data.

`cell.types` Vector with cell types considered by the model.

`features` Vector with features (genes) considered by the model. These features will be used for subsequent predictions.

`test.deconv.metrics` Performance of the model on test data by cell type. This slot is generated by the `calculateEvalMetrics` function (see `?calculateEvalMetrics` for more details).

`interpret.gradients` Gradients for interpretation. **SpatialDDLS** provides some functions to better understand prediction made by the model (see `?interGradientsDL` for more details). This slot is a list of either one or two elements: gradients of either the loss function or the predicted class with respect to the input variables using pure (only one cell type) mixed transcriptional profiles. These gradients can be interpreted as to what extent the model is using these variables to predict each cell type proportions.

`deconvSpatialDDLS` *Deconvolute spatial transcriptomics data using trained model*

Description

Deconvolute spatial transcriptomics data using the trained model in the `SpatialDDLS` object. The trained model is used to predict cell proportions of two mirrored transcriptional profiles:

- 'Intrinsic' profiles: transcriptional profiles of each spot in the ST dataset.
- 'Extrinsic' profiles: profiles simulated from the surrounding spots of each spot.

After prediction, cell proportions from the intrinsic profiles (intrinsic cell proportions) are regularized based on the similarity between intrinsic and extrinsic profiles in order to maintain spatial consistency. This approach leverages both transcriptional and spatial information. For more details, see Mañanes et al., 2023 and the Details section.

Usage

```
deconvSpatialDDLS(
  object,
  index.st,
  normalize = TRUE,
  scaling = "standardize",
  k.spots = 4,
  pca.space = TRUE,
  fast.pca = TRUE,
  pcs.num = 50,
  pca.var = 0.8,
  metric = "euclidean",
  alpha.cutoff = "mean",
  alpha.quantile = 0.5,
  simplify.set = NULL,
  simplify.majority = NULL,
  use.generator = FALSE,
  batch.size = 64,
  verbose = TRUE
)
```

Arguments

object	SpatialDDLS object with trained.model and spatial.experiments slots.
index.st	Name or index of the dataset/slide stored in the SpatialDDLS object (spatial.experiments slot) to be deconvolute. If missing, all datasets will be deconvoluted.
normalize	Normalize data (logCPM) before deconvolution (TRUE by default).
scaling	How to scale data before training. Options include "standardize" (values are centered around the mean with a unit standard deviation) or "rescale" (values are shifted and rescaled so that they end up ranging between 0 and 1). If normalize = FALSE, data are not scaled.
k.spots	Number of nearest spots considered for each spot during regularization and simulation of extrinsic transcriptional profiles. The greater, the smoother the regularization will be (4 by default).
pca.space	Whether to use PCA space to calculate distances between intrinsic and extrinsic transcriptional profiles (TRUE by default).
fast.pca	Whether using the **irlba** implementation. If TRUE, the number of PCs used is defined by the parameter. If FALSE, the PCA implementation from the **stats** R package is used instead (TRUE by default).
pcs.num	Number of PCs used to calculate distances if fast.pca == TRUE (50 by default).
pca.var	Threshold of explained variance (between 0.2 and 1) used to choose the number of PCs used if pca.space == TRUE and fast.pca == FALSE (0.8 by default).
metric	Metric used to measure distance/similarity between intrinsic and extrinsic transcriptional profiles. It may be 'euclidean', 'cosine' or 'pearson' ('euclidean' by default).

alpha.cutoff	Minimum distance for regularization. It may be 'mean' (spots with transcriptional distances shorter than the mean distance of the dataset will be modified) or 'quantile' (spots with transcriptional distances shorter than the alpha.quantile quantile are used). 'mean' by default.
alpha.quantile	Quantile used if alpha.cutoff == 'quantile'. 0.5 by default.
simplify.set	List specifying which cell types should be compressed into a new label with the name of the list item. See examples for details. If provided, results are stored in a list with 'raw' and 'simpli.set' elements.
simplify.majority	List specifying which cell types should be compressed into the cell type with the highest proportion in each spot. Unlike simplify.set, no new labels are created. If provided, results are stored in a list with 'raw' and 'simpli.majority' elements.
use.generator	Boolean indicating whether to use generators for prediction (FALSE by default).
batch.size	Number of samples per batch. Only when use.generator = TRUE.
verbose	Show informative messages during the execution.

Details

The deconvolution process involves two main steps: predicting cell proportions based on transcriptome using the trained neural network model, and regularization of cell proportions based on the spatial location of each spot. In the regularization step, a mirrored version of each spot is simulated based on its N-nearest spots. We refer to these profiles as 'extrinsic' profiles, whereas the transcriptional profiles of each spot are called 'intrinsic' profiles. Extrinsic profiles are used to regularize predictions based on intrinsic profiles. The rationale is that spots surrounded by transcriptionally similar spots should have similar cell compositions, and therefore predicted proportions can be smoothed to preserve their spatial consistency. On the other hand, spots surrounded by dissimilar spots cannot be predicted by their neighbors, and thus they can only be predicted by their own transcriptional profiles likely due to presenting very specific cell compositions.

Regarding the working os **SpatialDDLS**: first, extrinsic profiles are simulated based on the N-nearest spots for each spot by summing their transcriptomes. Distances between extrinsic and intrinsic profiles of each spot are calculated so that similar/dissimilar spots are identified. These two sets of transcriptional profiles are used as input for the trained neural network model, and according to the calculated distances, a weighted mean between the predicted proportions for each spot is calculated. Spots with distances between intrinsic and extrinsic profiles greater than alpha.cutoff are not regularized, whereas spots with distances less than alpha.cutoff contribute to the weighted mean. Weights are calculated by rescaling distances less than alpha.cutoff between 0 and 0.5, so that the maximum extent to which a extrinsic profile can modified the predictions based on intrinsic profiles is 0.5 (a regular mean). For more details, see Mañanes et al., 2023.

This function requires a SpatialDDLS object with a trained deep neural network model (trained.model slot, and the spatial transcriptomics datasets to be deconvoluted in the spatial.experiments slot. See ?createSpatialDDLSobject or ?loadSTProfiles for more details.

Value

SpatialDDLS object with a deconv.spots slot. The output is a list containing 'Regularized', 'Intrinsic' and 'Extrinsic' deconvoluted cell proportions, 'Distances' between intrinsic and extrinsic

transcriptional profiles, and 'Weight.factors' with the final weights used to regularize intrinsic cell proportions. If `simplify.set` and/or `simplify.majority` are provided, the `deconv.spots` slot will contain a list with raw and simplified results.

References

Mañanes, D., Rivero-García, I., Jimenez-Carretero, D., Torres, M., Sancho, D., Torroja, C., Sánchez-Cabo, F. (2023). SpatialDDLS: An R package to deconvolute spatial transcriptomics data using neural networks. biorxiv. doi: doi:10.1101/2023.08.31.555677.

See Also

`trainDeconvModel` `SpatialDDLS`

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
     rpois(30, lambda = 5), nrow = 15, ncol = 20,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(20)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(20)),
    Cell_Type = sample(x = paste0("CellType", seq(6)), size = 20,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
# training of SDDLS model
SDDLS <- trainDeconvModel(
  object = SDDLS,
```

```
  batch.size = 15,
  num.epochs = 5
)
# simulating spatial data
ngenes <- sample(3:40, size = 1)
ncells <- sample(10:40, size = 1)
counts <- matrix(
  rpois(ngenes * ncells, lambda = 5), ncol = ncells,
  dimnames = list(paste0("Gene", seq(ngenes)), paste0("Spot", seq(ncells)))
)
coordinates <- matrix(
  rep(c(1, 2), ncells), ncol = 2
)
st <- SpatialExperiment::SpatialExperiment(
  assays = list(counts = as.matrix(counts)),
  rowData = data.frame(Gene_ID = paste0("Gene", seq(ngenes))),
  colData = data.frame(Cell_ID = paste0("Spot", seq(ncells))),
  spatialCoords = coordinates
)
SDDLS <- loadSTProfiles(
  object = SDDLS,
  st.data = st,
  st.spot.ID.column = "Cell_ID",
  st.gene.ID.column = "Gene_ID"
)
# simplify arguments
simplify <- list(CellGroup1 = c("CellType1", "CellType2", "CellType4"),
                 CellGroup2 = c("CellType3", "CellType5"))
SDDLS <- deconvSpatialDDLS(
  object = SDDLS,
  index.st = 1,
  simplify.set = simplify,
  simplify.majority = simplify
)
```

`distErrorPlot` *Generate box or violin plots showing error distribution*

Description

Generate box or violin plots to show how errors are distributed. Errors can be shown all mixed or either split by cell type (`CellType`) or number of cell types present in the spots (`nCellTypes`). See the `facet.by` argument and examples for more details.

Usage

```
distErrorPlot(
  object,
  error,
```

```
  colors,
  x.by = "pBin",
  facet.by = NULL,
  color.by = "nCellTypes",
  filter.sc = TRUE,
  error.label = FALSE,
  pos.x.label = 4.6,
  pos.y.label = NULL,
  size.point = 0.1,
  alpha.point = 1,
  type = "violinplot",
  ylimit = NULL,
  nrow = NULL,
  ncol = NULL,
  title = NULL,
  theme = NULL,
  ...
)
```

Arguments

object	SpatialDDLS object with trained.model slot containing metrics in the test.deconv.metrics slot of a DeconvDLModel object.
error	Error to be represented. Available metric errors are: absolute error ('AbsErr'), proportional absolute error ('ppAbsErr'), squared error ('SqrErr'), and proportional squared error ('ppSqrErr').
colors	Vector of colors to be used.
x.by	Variable used for the X-axis. When facet.by is not NULL, the best choice is pBin (probability bins). Options: nCellTypes (number of different cell types), CellType (cell type), and pBin.
facet.by	Show data in different panels. Options are nCellTypes (number of different cell types) and CellType (cell type) (NULL by default).
color.by	Variable used to color data. Options are nCellTypes and CellType.
filter.sc	Boolean indicating whether single-cell profiles are filtered out and only mixed transcriptional profile errors are shown (TRUE by default).
error.label	Boolean indicating whether to show the average error as a plot annotation (FALSE by default).
pos.x.label	X-axis position of error annotations.
pos.y.label	Y-axis position of error annotations.
size.point	Size of points (0.1 by default).
alpha.point	Alpha of points (0.1 by default).
type	Type of plot: 'boxplot' or 'violinplot' (the latter by default).
ylimit	Upper limit in Y-axis if it is required (NULL by default).
nrow	Number of rows if facet.by is not NULL.

ncol	Number of columns if facet.by is not NULL.
title	Title of the plot.
theme	**ggplot2** theme.
...	Additional arguments for the facet_wrap function of **ggplot2** if facet.by is not NULL.

Value

A ggplot object.

See Also

calculateEvalMetrics corrExpPredPlot blandAltmanLehPlot barErrorPlot

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 20,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(20)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(20)),
    Cell_Type = sample(
      x = paste0("CellType", seq(6)), size = 20, replace = TRUE
    )
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
# training of DDLS model
```

```
SDDLS <- trainDeconvModel(
  object = SDDLS,
  batch.size = 15,
  num.epochs = 5
)
# evaluation using test data
SDDLS <- calculateEvalMetrics(object = SDDLS)
# representation, for more examples, see the vignettes
distErrorPlot(
  object = SDDLS,
  error = "AbsErr",
  facet.by = "CellType",
  color.by = "nCellTypes",
  error.label = TRUE
)
distErrorPlot(
  object = SDDLS,
  error = "AbsErr",
  x.by = "CellType",
  facet.by = NULL,
  color.by = "CellType",
  error.label = TRUE
)
```

`estimateZinbwaveParams` *Estimate parameters of the ZINB-WaVE model to simulate new single-cell RNA-Seq expression profiles*

Description

Estimate the parameters of the ZINB-WaVE model using a real single-cell RNA-Seq data set as reference to simulate new single-cell profiles and increase the signal of underrepresented cell types. This step is only is needed if the size of the single-cell RNA-seq dataset is too small or there are underrepresented cell types. After this step, the `simSCProfiles` function will use the estimated parameters to simulate new single-cell profiles. See ?`simSCProfiles` for more information.

Usage

```
estimateZinbwaveParams(
  object,
  cell.type.column,
  cell.ID.column,
  gene.ID.column,
  cell.cov.columns,
  gene.cov.columns,
  subset.cells = NULL,
```

```
  proportional = TRUE,
  set.type = "All",
  threads = 1,
  verbose = TRUE
)
```

Arguments

object	SpatialDDLS object with a single.cell.real slot.
cell.type.column	Name or column number corresponding to the cell type of each cell in cells metadata.
cell.ID.column	Name or column number corresponding to the cell names of expression matrix in cells metadata.
gene.ID.column	Name or column number corresponding to the notation used for features/genes in genes metadata.
cell.cov.columns	Name or column number(s) in cells metadata to be used as covariates during model fitting (if no covariates are used, set to empty or NULL).
gene.cov.columns	Name or column number(s) in genes metadata that will be used as covariates during model fitting (if no covariates are used, set to empty or NULL).
subset.cells	Number of cells to fit the ZINB-WaVE model. Useful when the original data set is too large to fit the model. Set a value according to the original data set and the resources available on your computer. If NULL (by default), all cells will be used. Must be an integer greater than or equal to the number of cell types considered and less than or equal to the total number of cells.
proportional	If TRUE, the original cell type proportions in the subset of cells generated by subset.cells will not be altered as far as possible. If FALSE, all cell types will have the same number of cells as far as possible (TRUE by default).
set.type	Cell type(s) to evaluate ('All' by default). It is recommended fitting the model to all cell types rather than using only a subset of them to capture the total variability present in the original experiment even if only a subset of cell types is simulated.
threads	Number of threads used for estimation (1 by default). To set up the parallel environment, the **BiocParallel** package must be installed.
verbose	Show informative messages during the execution (TRUE by default).

Details

ZINB-WaVE is a flexible model for zero-inflated count data. This function carries out the model fit to real single-cell data modeling Y_{ij} (the count of feature j for sample i) as a random variable following a zero-inflated negative binomial (ZINB) distribution. The estimated parameters will be used for the simulation of new single-cell expression profiles by sampling a negative binomial distribution and inserting dropouts from a binomial distribution. To do so, **SpatialDDLS** uses the zinbFit function from the **zinbwave** package (Risso et al., 2018). For more details about the model, see Risso et al., 2018.

Value

A SpatialDDLS object with zinb.params slot containing a ZinbParametersModel object. This object contains a slot with the estimated ZINB-WaVE parameters from the real single-cell RNA-Seq data.

References

Risso, D., Perraudeau, F., Gribkova, S. et al. (2018). A general and flexible method for signal extraction from single-cell RNA-seq data. Nat Commun 9, 284. doi: doi:10.1038/s41467017-025545.

Torroja, C. and Sánchez-Cabo, F. (2019). digitalDLSorter: A Deep Learning algorithm to quantify immune cell populations based on scRNA-Seq data. Frontiers in Genetics 10, 978. doi: doi:10.3389/fgene.2019.00978.

See Also

simSCProfiles

Examples

```
set.seed(123) # reproducibility
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 10,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(10)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(10)),
    Cell_Type = sample(x = paste0("CellType", seq(2)), size = 10,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  project = "Simul_example",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- estimateZinbwaveParams(
  object = SDDLS,
  cell.type.column = "Cell_Type",
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID",
  subset.cells = 2,
  verbose = TRUE
```

```
)
```

`features` *Get and set* `features` *slot in a* `DeconvDLModel` *object*

Description

Get and set `features` slot in a `DeconvDLModel` object

Usage

```
features(object)

features(object) <- value
```

Arguments

`object`	`DeconvDLModel` object.
`value`	Vector with features (genes) considered by the deep neural network model.

`genMixedCellProp` *Generate training and test cell type composition matrices*

Description

Generate training and test cell type composition matrices for the simulation of mixed transcriptional profiles with known cell composition using single-cell expression profiles. The resulting `PropCellTypes` object will contain all the information needed to simulate new mixed transcriptional profiles. Note this function does not simulate the mixed profiles, this task is performed by the `simMixedProfiles` or `trainDeconvModel` functions (see Documentation).

Usage

```
genMixedCellProp(
  object,
  cell.ID.column,
  cell.type.column,
  num.sim.spots,
  n.cells = 50,
  train.freq.cells = 3/4,
  train.freq.spots = 3/4,
  proportion.method = c(0, 0, 1),
  prob.sparity = 1,
  min.zero.prop = NULL,
  balanced.type.cells = TRUE,
  verbose = TRUE
)
```

Arguments

object	SpatialDDLS object with single.cell.real slot and, optionally, with single.cell.simul slot.
cell.ID.column	Name or column number corresponding to cell names in cells metadata.
cell.type.column	Name or column number corresponding to cell types in cells metadata.
num.sim.spots	Number of mixed profiles to be simulated. It is recommended to adjust this number according to the number of available single-cell profiles.
n.cells	Specifies the number of cells to be randomly selected and combined to generate the simulated mixed profiles. By default, it is set to 50 It controls the level of noise present in the simulated data, as it determines how many single-cell profiles will be combined to produce each spot.
train.freq.cells	Proportion of cells used to simulate training mixed transcriptional profiles (3/4 by default).
train.freq.spots	Proportion of mixed transcriptional profiles to be used for training, relative to the total number of simulated spots (num.sim.spots). The default value is 3/4.
proportion.method	Vector with three elements that controls the proportion of simulated proportions generated by each method: random sampling of a Dirichlet distribution, "pure" spots (1 cell type), and spots generated from a random sampling of a Dirichlet distribution but with a specified number of different cell types (determined by min.zero.prop), respectively. By default, all samples are generated by the last method.
prob.sparity	It only affects the proportions generated by the first method (Dirichlet distribution). It determines the probability of having missing cell types in each simulated spot, as opposed to a mixture of all cell types. A higher value for this parameter will result in more sparse simulated samples.
min.zero.prop	This parameter controls the minimum number of cell types that will be absent in each simulated spot. If NULL (by default), this value will be half of the total number of different cell types, but increasing it will result in more spots composed of fewer cell types. This helps to create more sparse proportions and cover a wider range of situations during model training.
balanced.type.cells	Boolean indicating whether training and test cells will be split in a balanced way considering cell types (TRUE by default).
verbose	Show informative messages during the execution (TRUE by default).

Details

First, the single-cell profiles are randomly divided into two subsets, with 2/3 of the data for training and 1/3 for testing. The default setting for this ratio can be changed using the train.freq.cells parameter. Next, a total of num.sim.spots mixed proportions are simulated using a Dirichlet distribution. This simulation takes into account the probability of missing cell types in each spot, which

can be adjusted using the `prob.sparity` parameter. For each mixed sample, `n.cells` single-cell profiles are randomly selected and combined to generate the simulated mixed sample. In addition to the Dirichlet-based proportions, pure spots (containing only one cell type) and spots containing a specified number of different cell types (determined by the `min.zero.prop` parameter) are also generated in order to cover situations with only a few cell types present. The proportion of simulated spots generated by each method can be controlled using the `proportion.method` parameter. To visualize the distribution of cell type proportions generated by each method, the `showProbPlot` function can be used.

Value

A `SpatialDDLS` object with `prob.cell.types` slot containing a `list` with two `PropCellTypes` objects (training and test). For more information about the structure of this class, see ?`PropCellTypes`.

See Also

`simMixedProfiles` `PropCellTypes`

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(100, lambda = 5), nrow = 40, ncol = 30,
      dimnames = list(paste0("Gene", seq(40)), paste0("RHC", seq(30)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(30)),
    Cell_Type = sample(x = paste0("CellType", seq(4)), size = 30,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(40))
  )
)

SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE,
  project = "Simul_example"
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 10,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
```

```
  verbose = TRUE
)
```

`getProbMatrix` *Getter function for the cell composition matrix*

Description

Getter function for the cell composition matrix. This function allows to access to the cell composition matrix of simulated mixed transcriptional profiles.

Usage

```
getProbMatrix(object, type.data)
```

Arguments

`object`	SpatialDDLS object with `prob.cell.types` slot.
`type.data`	Subset of data to return: `train` or `test`.

Value

Cell type proportion matrix.

See Also

`genMixedCellProp`

`installTFpython` *Install Python dependencies for SpatialDDLS*

Description

This function facilitates the installation of the required Python dependencies for the **SpatialDDLS** R package, as it requires a Python interpreter with the TensorFlow Python library and its dependencies.

Usage

```
installTFpython(
  conda = "auto",
  python.version = "3.8",
  tensorflow.version = "2.6",
  install.conda = FALSE,
  miniconda.path = NULL
)
```

Arguments

conda	Path to a conda executable. Using "auto" (by default) allows **reticulate** to automatically find an appropriate conda binary.
python.version	Python version to be installed in the environment ("3.8" by default). We recommend keeping this version as it has been tested to be compatible with tensorflow 2.6.
tensorflow.version	Tensorflow version to be installed in the environment ("2.6" by default).
install.conda	Boolean indicating if installing miniconda automatically by using **reticulate**. If TRUE, conda argument is ignored. FALSE by default.
miniconda.path	If install.conda is TRUE, you can set the path where miniconda will be installed. If NULL, conda will find automatically the proper place.

Details

This function is intended to simplify the installation process for **SpatialDDLS** by automatically installing Miniconda and creating a new environment named SpatialDDLS-env with all **SpatialDDLS**' dependencies covered. For users who wish to use a different Python or conda environment, see the tensorflow::use_condaenv function for more information.

Value

No return value, called for side effects: installation of conda environment with a Python interpreter and Tensorflow

Examples

```
## Not run:
notesInstallation <- installTFpython(
  conda = "auto", install.conda = TRUE
)

## End(Not run)
```

---

interGradientsDL *Calculate gradients of predicted cell types/loss function with respect to input features for interpreting trained deconvolution models*

Description

This function enables users to gain insights into the interpretability of the deconvolution model. It calculates the gradients of classes/loss function with respect to the input features used in training. These numeric values are calculated per gene and cell type in pure mixed transcriptional profiles, providing information on the extent to which each feature influences the model's prediction of cell proportions for each cell type.

Usage

```
interGradientsDL(
  object,
  method = "class",
  normalize = TRUE,
  scaling = "standardize",
  verbose = TRUE
)
```

Arguments

object	SpatialDDLS object containing a trained deconvolution model (trained.model slot) and pure mixed transcriptional profiles (mixed.profiles slot).
method	Method to calculate gradients with respect to inputs. It can be 'class' (gradients of predicted classes w.r.t. inputs), 'loss' (gradients of loss w.r.t. inputs) or 'both'.
normalize	Whether to normalize data using logCPM (TRUE by default). This parameter is only considered when the method used to simulate the mixed transcriptional profiles (simMixedProfiles function) was "AddRawCount". Otherwise, data were already normalized. This parameter should be set according to the transformation used to train the model.
scaling	How to scale data. It can be: "standardize" (values are centered around the mean with a unit standard deviation), "rescale" (values are shifted and rescaled so that they end up ranging between 0 and 1, by default) or "none" (no scaling is performed). This parameter should be set according to the transformation used to train the model.
verbose	Show informative messages during the execution (TRUE by default).

Details

Gradients of classes / loss function with respect to the input features are calculated exclusively using pure mixed transcriptional profiles composed of a single cell type. Consequently, these numbers can be interpreted as the extent to which each feature is being used to predict each cell type proportion. Gradients are calculated at the sample level for each gene, but only mean gradients by cell type are reported. For additional details, see Mañanes et al., 2023.

Value

Object containing gradients in the interpret.gradients slot of the DeconvDLModel object (trained.model slot).

See Also

deconvSpatialDDLS plotTrainingHistory

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 10,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(10)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(10)),
    Cell_Type = sample(x = paste0("CellType", seq(2)), size = 10,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
SDDLS <- trainDeconvModel(
  object = SDDLS,
  batch.size = 12,
  num.epochs = 5
)
## calculating gradients
SDDLS <- interGradientsDL(SDDLS)
```

---

loadSTProfiles *Loads spatial transcriptomics data into a SpatialDDLS object*

Description

This function loads a SpatialExperiment object (or a list with several SpatialExperiment objects) into a SpatialDDLS object.

Usage

```
loadSTProfiles(
  object,
  st.data,
  st.spot.ID.column,
  st.gene.ID.column,
  st.min.counts = 0,
  st.min.spots = 0,
  st.n.slides = 3,
  verbose = TRUE
)
```

Arguments

object	A SpatialDDLS object.
st.data	A SpatialExperiment object (or a list with several SpatialExperiment objects) to be deconvoluted.
st.spot.ID.column	Name or number of the column in spots metadata corresponding to spot names in the expression matrix.
st.gene.ID.column	Name or number of the column in genes metadata corresponding to names used for features/genes.
st.min.counts	Minimum gene counts to filter (0 by default).
st.min.spots	Minimum of spots with more than min.counts (0 by default).
st.n.slides	Minimum number of slides (SpatialExperiment objects) in which a gene has to be expressed in order to keep it. This parameter is applicable only when multiple SpatialExperiment objects are provided. Genes not present in at least st.n.slides will be discarded. If no filtering is desired, set st.n.slides = 1.
verbose	Show informative messages during execution (TRUE by default).

Details

It is recommended to perform this step when creating the SpatialDDLS object using the createSpatialDDLSobject function in order to only keep genes shared between the spatial transcriptomics and the single-cell transcriptomics data used as reference. In addition, please, make sure the gene identifiers used the spatial and single-cell transcriptomics data are consistent.

Value

A SpatialDDLS object with the provided spatial trainscriptomics data loaded into the spatial.experiments slot.

See Also

createSpatialDDLSobject trainDeconvModel

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(100, lambda = 5), nrow = 40, ncol = 30,
      dimnames = list(paste0("Gene", seq(40)), paste0("RHC", seq(30)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(30)),
    Cell_Type = sample(x = paste0("CellType", seq(4)), size = 30,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(40))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)

## simulating a SpatialExperiment object
counts <- matrix(rpois(30, lambda = 5), ncol = 6)
rownames(counts) <- paste0("Gene", 1:5)
colnames(counts) <- paste0("Spot", 1:6)
coordinates <- matrix(
  c(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3), ncol = 2
)
ste <- SpatialExperiment::SpatialExperiment(
  assays = list(counts = as.matrix(counts)),
  rowData = data.frame(Gene_ID = paste0("Gene", 1:5)),
  colData = data.frame(Cell_ID = paste0("Spot", 1:6)),
  spatialCoords = coordinates
)

## previous SpatialDDLS object
SDDLS <- loadSTProfiles(
  object = SDDLS,
  st.data = ste,
  st.spot.ID.column = "Cell_ID",
  st.gene.ID.column = "Gene_ID"
)
```

loadTrainedModelFromH5

Load from an HDF5 file a trained deep neural network model into a SpatialDDLS *object*

Description

Load from an HDF5 file a trained deep neural network model into a SpatialDDLS object. Note that HDF5 file must be a valid trained model (**keras** object).

Usage

```
loadTrainedModelFromH5(object, file.path, reset.slot = FALSE)
```

Arguments

object	SpatialDDLS object with trained.model slot.
file.path	Valid file path where the model are stored.
reset.slot	Deletes trained.slot if it already exists. A new DeconvDLModel object will be formed, but will not contain other slots (FALSE by default).

Value

SpatialDDLS object with trained.model slot with the new **keras** DNN model incorporated.

See Also

trainDeconvModel saveTrainedModelAsH5

method

Get and set method *slot in a* PropCellTypes *object*

Description

Get and set method slot in a PropCellTypes object

Usage

```
method(object)

method(object) <- value
```

Arguments

object	PropCellTypes object.
value	Vector containing the method by which cell type proportions were generated.

mixed.profiles *Get and set* mixed.profiles *slot in a* SpatialExperiment *object*

Description

Get and set mixed.profiles slot in a SpatialExperiment object

Usage

```
mixed.profiles(object, type.data = "both")

mixed.profiles(object, type.data = "both") <- value
```

Arguments

object	SpatialExperiment object.
type.data	Type of data to return. It can be 'both' (default), 'train', or 'test'.
value	List with two SummarizedExperiment objects, train and test, each one containing simulated mixed transcriptional profiles.

model *Get and set* model *slot in a* DeconvDLModel *object*

Description

Get and set model slot in a DeconvDLModel object

Usage

```
model(object)

model(object) <- value
```

Arguments

object	DeconvDLModel object.
value	keras.engine.sequential.Sequential object with a trained deep neural network model.

plotDistances *Plot distances between intrinsic and extrinsic profiles*

Description

Color spots on the spatial coordinates according to distances between intrinsic and extrinsic transcriptional profiles.

Usage

```
plotDistances(
  object,
  index.st,
  mid.scale = "mean",
  size.point = 1,
  title = NULL,
  theme = NULL
)
```

Arguments

object	A SpatialDDLS object.
index.st	Index of the spatial transcriptomics data to be plotted. It can be either a position or a name if a named list was provided.
mid.scale	The midpoint of the diverging scale. it may be 'mean' or 'median' (the former by default).
size.point	Size of points (0.1 by default).
title	Title of plot.
theme	**ggplot2** theme.

Value

A ggplot object.

See Also

deconvSpatialDDLS trainDeconvModel

plotHeatmapGradsAgg *Plot a heatmap of gradients of classes / loss function wtih respect to the input*

Description

Plot a heatmap showing the top positive and negative gene average gradients per cell type.

Usage

```
plotHeatmapGradsAgg(
  object,
  method = "class",
  top.n.genes = 15,
  scale.gradients = TRUE
)
```

Arguments

object	SpatialDDLS object with a DeconvDLModel object containing gradients in the interpret.gradients slot.
method	Method to calculate gradients with respect to input features. It can be 'class' (gradients of predicted classes w.r.t. input features) or 'loss' (gradients of loss w.r.t. input features) ('class' by default).
top.n.genes	Top n genes (positive and negative) taken per cell type.
scale.gradients	Whether to calculate feature-wise z-scores of gradients (TRUE by default).

Value

A list of Heatmap-class objects, one for top positive and another one for top negative gradients.

See Also

interGradientsDL trainDeconvModel

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 10,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(10)))
    )
  ),
  colData = data.frame(
```

```
    Cell_ID = paste0("RHC", seq(10)),
    Cell_Type = sample(x = paste0("CellType", seq(2)), size = 10,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
SDDLS <- trainDeconvModel(
  object = SDDLS,
  batch.size = 12,
  num.epochs = 5
)
## calculating gradients
SDDLS <- interGradientsDL(SDDLS)
plotHeatmapGradsAgg(SDDLS, top.n.genes = 2)
```

`plots` *Get and set* `plots` *slot in a* `PropCellTypes` *object*

Description

Get and set `plots` slot in a `PropCellTypes` object

Usage

```
plots(object)

plots(object) <- value
```

Arguments

object	PropCellTypes object.
value	List of lists with plots showing the distribution of cell proportions generated by each method.

---

plotSpatialClustering *Plot results of clustering based on predicted cell proportions*

Description

Color spots on the spatial coordinates according to the results of clustering based on predicted proportions.

Usage

```
plotSpatialClustering(
  object,
  index.st,
  method,
  k.nn,
  k.centers,
  colors,
  size.point = 1,
  title = NULL,
  theme = NULL
)
```

Arguments

object	A SpatialDDLS object.
index.st	Index of the spatial transcriptomics data to be plotted. It can be either a position or a name if a named list of SpatialExperiment objects was provided.
method	Clustering method results to plot. It can be "graph" or "k.means". If missing, the first configuration found in the object will be plotted.
k.nn	Number of nearest neighbors used if method == "graph".
k.centers	Number of k centers used if method == "k.means".
colors	Vector of colors to be used.
size.point	Size of points (0.1 by default).
title	Title of plot.
theme	**ggplot2** theme.

Value

A ggplot object.

See Also

spatialPropClustering deconvSpatialDDLS

---

plotSpatialGeneExpr *Plot normalized gene expression data (logCPM) in spatial coordinates*

Description

Color spots on the spatial coordinates according to the logCPM values of a particular gene.

Usage

```
plotSpatialGeneExpr(
  object,
  index.st,
  gene,
  colors = "spectral",
  size.point = 1,
  title = NULL,
  theme = NULL
)
```

Arguments

object	A SpatialDDLS object.
index.st	Index of the spatial transcriptomics data to be plotted. It can be either a position or a name if a named list of SpatialExperiment objects was provided.
gene	Gene to color spots by.
colors	Color scale to be used. It can be "blues" or "spectral" (the latter by default).
size.point	Size of points (0.1 by default).
title	Title of plot.
theme	**ggplot2** theme.

Value

A ggplot object.

See Also

interGradientsDL topGradientsCellType

plotSpatialProp *Plot predicted proportions for a specific cell type using spatial coordinates of spots*

Description

Color spots on the spatial coordinates according to the predicted proportions of a particular cell type. Color scale is adapted depending on the range of predicted proportions.

Usage

```
plotSpatialProp(
  object,
  index.st,
  cell.type,
  colors = "blues",
  set = "raw",
  prediction = "Regularized",
  limits = NULL,
  size.point = 1,
  title = NULL,
  theme = NULL
)
```

Arguments

object	A SpatialDDLS object.
index.st	Index of the spatial transcriptomics data to be plotted. It can be either a position or a name if a named list of SpatialExperiment objects was provided.
cell.type	Cell type predicted proportions to color spots by.
colors	Color scale to be used. It can be "blues" or "spectral" (the former by default).
set	If results were simplified (see ?deconvSpatialDDLS for details), what results to plot (raw by default).
prediction	It can be "Regularized", "Intrinsic" or "Extrinsic" ("Regularized" by default).
limits	A vector of two elements indicating wanted limits for color scale. If NULL (by default), color scale is adjusted to max and min predicted proportions.
size.point	Size of points (0.1 by default).
title	Title of plot.
theme	**ggplot2** theme.

Value

A ggplot object.

See Also

plotSpatialPropAll deconvSpatialDDLS trainDeconvModel

---

plotSpatialPropAll *Plot predicted proportions for all cell types using spatial coordinates of spots*

Description

Color spots on the spatial coordinates plot according to their predicted cell type proportions. All cell types are represented together using the same color scale from 0 to 1.

Usage

```
plotSpatialPropAll(
  object,
  index.st,
  colors = "blues",
  set = "raw",
  prediction = "Regularized",
  size.point = 0.1,
  title = NULL,
  nrow = NULL,
  ncol = NULL,
  theme = NULL
)
```

Arguments

object	A SpatialDDLS object.
index.st	Index of the spatial transcriptomics data to be plotted. It can be either a position or a name if a named list of SpatialExperiment objects was provided.
colors	Color scale to be used. It can be "blues" or "spectral" (the former by default).
set	If results were simplified (see ?deconvSpatialDDLS for details), which results to plot (raw by default).
prediction	It can be "Regularized", "Intrinsic" or "Extrinsic" ("Regularized" by default).
size.point	Size of points (0.1 by default).
title	Title of plot.
nrow	Number of rows in the split plot.
ncol	Number of columns in the split plot.
theme	**ggplot2** theme.

Value

A ggplot object.

See Also

plotSpatialProp deconvSpatialDDLS trainDeconvModel

`plotTrainingHistory` *Plot training history of a trained SpatialDDLS deep neural network model*

Description

Plot training history of a trained SpatialDDLS deep neural network model.

Usage

```
plotTrainingHistory(
  object,
  title = "History of metrics during training",
  metrics = NULL
)
```

Arguments

`object`	SpatialDDLS object with a `trained.model` slot.
`title`	Title of plot.
`metrics`	Metrics to be plotted. If `NULL` (by default), all metrics available in the DeconvDLModel object will be plotted.

Value

A ggplot object with the progression of the selected metrics during training.

See Also

trainDeconvModel

preparingToSave *Prepare SpatialDDLS object to be saved as an RDA file*

Description

This function prepares a SpatialDDLS object to be saved as an RDA file when contains a DeconvDLModel object with a trained DNN model.

Usage

```
preparingToSave(object)
```

Arguments

object	SpatialDDLS object with a trained.data slot containing a DeconvDLModel object with a trained DNN model.

Details

Since **keras** models cannot be saved natively as R objects, this function saves the structure of the model as a JSON-like character object and its weights as a list. This allows for the retrieval of the model and making predictions. It is important to note that the state of the optimizer is not saved, only the model's architecture and weights. To save the entire model, please see the saveTrainedModelAsH5 and loadTrainedModelFromH5 functions.

It is also possible to save a SpatialDDLS object as an RDS file with the saveRDS function without any preparation.

Value

A SpatialDDLS or DeconvDLModel object with its trained keras model transformed from a keras.engine.sequential.Sequ class into a list with its architecture as a JSON-like character object, and its weights as a list.

See Also

saveRDS saveTrainedModelAsH5

prob.cell.types *Get and set* prob.cell.types *slot in a* SpatialExperiment *object*

Description

Get and set prob.cell.types slot in a SpatialExperiment object

Usage

```
prob.cell.types(object, type.data = "both")

prob.cell.types(object, type.data = "both") <- value
```

Arguments

object	SpatialExperiment object.
type.data	Type of data to return. It can be 'both' (default), 'train', or 'test'.
value	List with two PropCellTypes objects corresponding to train and test data.

---

prob.matrix *Get and set* prob.matrix *slot in a* PropCellTypes *object*

Description

Get and set prob.matrix slot in a PropCellTypes object

Usage

```
prob.matrix(object)

prob.matrix(object) <- value
```

Arguments

object	PropCellTypes object.
value	Matrix with cell types as columns and samples as rows.

---

project *Get and set* project *slot in a* SpatialExperiment *object*

Description

Get and set project slot in a SpatialExperiment object

Usage

```
project(object)

project(object) <- value
```

Arguments

object	SpatialExperiment object.
value	Character indicating the name of the project.

PropCellTypes-class *The PropCellTypes Class*

Description

The PropCellTypes class is a data storage class which contains information related to cell type composition matrices used to simulate mixed transcriptional profiles. This matrix is stored in the prob.matrix slot while the other slots contain additional information generated during the process and required for subsequent steps.

Details

See ?genMixedCellProp function for information about how cell type composition matrices are generated. Plots of cell type proportion distributions can be accessed using the showProbPlot function (see ?showProbPlot for more details).

Slots

prob.matrix Matrix of cell type proportions to simulate mixed transcriptional profiles.

cell.names Matrix containing cells used to generate the simulated mixed transcriptional profiles.

set.list List of cells sorted by cell type.

set Vector containing cell names present in the object.

method Vector indicating the method by which cell type proportions were generated.

plots Plots showing cell type proportion distributions. See ?showProbPlot for more details.

type.data Character indicating the type of data contained: 'train' or 'test'.

saveRDS *Save SpatialExperiment objects as RDS files*

Description

Save SpatialExperiment and DeconvDLModel objects as RDS files. **keras** models cannot be stored natively as R objects (e.g. RData or RDS files). By saving the architecture as a JSON-like character object and the weights as a list, it is possible to retrieve a functional model and make new predictions. If the trained.model slot is empty, the function will behave as usual. **Note:** with this option, the state of optimizer is not saved, only model's architecture and weights. It is possible to save the entire model as an HDF5 file with the saveTrainedModelAsH5 function and load it into a SpatialExperiment object with the loadTrainedModelFromH5 function. See documentation for details.

Usage

```
saveRDS(
  object,
  file,
  ascii = FALSE,
  version = NULL,
  compress = TRUE,
  refhook = NULL
)

## S4 method for signature 'DeconvDLModel'
saveRDS(
  object,
  file,
  ascii = FALSE,
  version = NULL,
  compress = TRUE,
  refhook = NULL
)

## S4 method for signature 'SpatialDDLS'
saveRDS(
  object,
  file,
  ascii = FALSE,
  version = NULL,
  compress = TRUE,
  refhook = NULL
)
```

Arguments

object	SpatialExperiment or DeconvDLModel object to be saved
file	File path where the object will be saved
ascii	a logical. If TRUE or NA, an ASCII representation is written; otherwise (default), a binary one is used. See the comments in the help for save.
version	the workspace format version to use. NULL specifies the current default version (3). The only other supported value is 2, the default from R 1.4.0 to R 3.5.0.
compress	a logical specifying whether saving to a named file is to use "gzip" compression, or one of "gzip", "bzip2" or "xz" to indicate the type of compression to be used. Ignored if file is a connection.
refhook	a hook function for handling reference objects.

Value

No return value, saves a SpatialExperiment object as an RDS file on disk.

See Also

SpatialExperiment saveTrainedModelAsH5

---

saveTrainedModelAsH5 *Save a trained* SpatialDDLS *deep neural network model to disk as an HDF5 file*

Description

Save a trained SpatialDDLS deep neural network model to disk as an HDF5 file. Note that this function does not save the DeconvDLModel object, only the trained **keras** model. This is the alternative to the saveRDS and preparingToSave functions if you want to keep the state of the optimizer.

Usage

```
saveTrainedModelAsH5(object, file.path, overwrite = FALSE)
```

Arguments

object	SpatialDDLS object with trained.model slot.
file.path	Valid file path where to save the model to.
overwrite	Overwrite file if it already exists.

Value

No return value, saves a **keras** DNN trained model as HDF5 file on disk.

See Also

trainDeconvModel loadTrainedModelFromH5

---

set *Get and set* set *slot in a* PropCellTypes *object*

Description

Get and set set slot in a PropCellTypes object

Usage

```
set(object)

set(object) <- value
```

Arguments

object	PropCellTypes object.
value	A vector containing the names of cells that are present in the object.

set.list — *Get and set* set.list *slot in a* PropCellTypes *object*

Description

Get and set set.list slot in a PropCellTypes object

Usage

```
set.list(object)

set.list(object) <- value
```

Arguments

object	PropCellTypes object.
value	List of cells sorted by their corresponding cell type.

showProbPlot — *Show distribution plots of the cell proportions generated by* genMixedCellProp

Description

Show distribution plots of the cell proportions generated by the genMixedCellProp function.

Usage

```
showProbPlot(object, type.data, set, type.plot = "boxplot")
```

Arguments

object	SpatialDDLS object with prob.cell.types slot with plot slot.
type.data	Subset of data to show: train or test.
set	Integer determining which of the 6 different subsets to display.
type.plot	Character determining which type of visualization to display. It can be 'boxplot', 'violinplot', 'linesplot' or 'ncelltypes'. See Description for more information.

Details

These frequencies will determine the proportion of different cell types used during the simulation of mixed transcriptional profiles. Proportions generated by each method (see ?genMixedCellProp) can be visualized in three ways: box plots, violin plots, and lines plots. You can also plot the probabilities based on the number of different cell types present in the samples by setting type.plot = 'nCellTypes'.

Value

A ggplot object.

See Also

genMixedCellProp

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(100, lambda = 5), nrow = 40, ncol = 30,
      dimnames = list(paste0("Gene", seq(40)), paste0("RHC", seq(30)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(30)),
    Cell_Type = sample(x = paste0("CellType", seq(4)), size = 30,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(40))
  )
)

SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  project = "Simul_example",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 10,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
showProbPlot(
   SDDLS,
   type.data = "train",
   set = 1,
   type.plot = "boxplot"
 )
```

`simMixedProfiles` *Simulate training and test mixed spot profiles*

Description

Simulate training and test mixed spot transcriptional profiles using cell composition matrices generated by the `genMixedCellProp` function.

Usage

```
simMixedProfiles(
  object,
  type.data = "both",
  mixing.function = "AddRawCount",
  file.backend = NULL,
  compression.level = NULL,
  block.processing = FALSE,
  block.size = 1000,
  chunk.dims = NULL,
  threads = 1,
  verbose = TRUE
)
```

Arguments

`object`	`SpatialDDLS` object with `single.cell.real`/`single.cell.simul`, and `prob.cell.types` slots.
`type.data`	Type of data to generate: `'train'`, `'test'` or `'both'` (the last by default).
`mixing.function`	Function used to build mixed transcriptional profiles. It may be: • `"AddRawCount"`: single-cell profiles (raw counts) are added up across cells. Then, log-CPMs are calculated (by default). • `"MeanCPM"`: single-cell profiles (raw counts) are transformed into CPMs and cross-cell averages are calculated. Then, `log2(CPM + 1)` is calculated. • `"AddCPM"`: single-cell profiles (raw counts) are transformed into CPMs and are added up across cells. Then, log-CPMs are calculated.
`file.backend`	Valid file path to store simulated mixed expression profiles as an HDF5 file (`NULL` by default). If provided, data are stored in HDF5 files used as back-end by using the **DelayedArray**, **HDF5Array** and **rhdf5** packages instead of loading all data into RAM. Note that operations on this matrix will be performed in blocks (i.e subsets of determined size) which may result in longer execution times.
`compression.level`	The compression level used if `file.backend` is provided. It is an integer value between 0 (no compression) and 9 (highest and slowest compression). See `?getHDF5DumpCompressionLevel` from the **HDF5Array** package for more information.

`block.processing`
: Boolean indicating whether data should be simulated in blocks (only if `file.backend` is used, `FALSE` by default). This functionality is suitable for cases where it is not possible to load all data into memory, and it leads to longer execution times.

`block.size`
: Only if `block.processing = TRUE`. Number of mixed expression profiles that will be simulated in each iteration. Larger numbers result in higher memory usage but shorter execution times. Set accordingly to available computational resources (1000 by default).

`chunk.dims`
: Specifies the dimensions that HDF5 chunk will have. If `NULL`, the default value is a vector of two items: the number of genes considered by `SpatialDDLS` object during the simulation, and a single sample to reduce read times in the following steps. A larger number of columns written in each chunk can lead to longer read times.

`threads`
: Number of threads used during simulation (1 by default).

`verbose`
: Show informative messages during the execution (`TRUE` by default).

Details

Mixed profiles are generated under the assumption that the expression level of a particular gene in a given spot is the sum of the expression levels of the cell types that make it up weighted by their proportions. In practice, as described in Torroja and Sanchez-Cabo, 2019, these profiles are generated by summing gene expression levels of a determined number of cells specified by a known cell composition matrix. The number of simulated spots and cells used to simulate each spot are determined by the `genMixedCellProp` function. This step can be avoided by using the `on.the.fly` argument in the `trainDeconvModel` function.

SpatialDDLS allows to use HDF5 files as back-end to store simulated data using the **DelayedArray** and **HDF5Array** packages. This functionality allows to work without keeping the data loaded into RAM, which could be useful during some computationally heavy steps such as neural network training on RAM-limited machines. You must provide a valid file path in the `file.backend` argument to store the resulting file with the '.h5' extension. This option slows down execution times, as subsequent transformations of the data will be done in blocks. Note that if you use the `file.backend` argument with `block.processing = FALSE`, all mixed profiles will be simulated in one step and, thus, loaded into RAM. Then, the matrix will be written to an HDF5 file. To avoid the RAM collapse, these profiles can be simulated and written to HDF5 files in blocks of `block.size` size by setting `block.processing = TRUE`. We recommend this option accordingly to the computational resources available and the number of simulated spots to be generated, but, in most of the cases, it is not necessary.

Value

A `SpatialDDLS` object with `mixed.profiles` slot containing a list with one or two entries (depending on selected `type.data` argument): `'train'` and `'test'`. Each entry consists of a `SummarizedExperiment` object with the simulated mixed slot profiles.

References

Fischer B, Smith M and Pau, G (2020). rhdf5: R Interface to HDF5. R package version 2.34.0.

Pagès H, Hickey P and Lun A (2020). DelayedArray: A unified framework for working transparently with on-disk and in-memory array-like datasets. R package version 0.16.0.

Pagès H (2020). HDF5Array: HDF5 backend for DelayedArray objects. R package version 1.18.0.

See Also

genMixedCellProp PropCellTypes trainDeconvModel

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(100, lambda = 5), nrow = 40, ncol = 30,
      dimnames = list(paste0("Gene", seq(40)), paste0("RHC", seq(30)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(30)),
    Cell_Type = sample(x = paste0("CellType", seq(4)), size = 30,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(40))
  )
)

SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE,
  project = "Simul_example"
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 10,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS, verbose = TRUE)
```

`simSCProfiles` *Simulate new single-cell RNA-Seq expression profiles using the ZINB-WaVE model parameters*

Description

Simulate single-cell expression profiles by randomly sampling from a negative binomial distribution and inserting dropouts by sampling from a binomial distribution using the ZINB-WaVE parameters estimated by the `estimateZinbwaveParams` function.

Usage

```
simSCProfiles(
  object,
  cell.ID.column,
  cell.type.column,
  n.cells,
  suffix.names = "_Simul",
  cell.types = NULL,
  file.backend = NULL,
  name.dataset.backend = NULL,
  compression.level = NULL,
  block.processing = FALSE,
  block.size = 1000,
  chunk.dims = NULL,
  verbose = TRUE
)
```

Arguments

`object`	`SpatialDDLS` object with `single.cell.real` and `zinb.params` slots.
`cell.ID.column`	Name or column number corresponding to the cell names of expression matrix in cells metadata.
`cell.type.column`	Name or column number corresponding to the cell type of each cell in cells metadata.
`n.cells`	Number of simulated cells generated per cell type (i.e. if you have 10 different cell types in your dataset, if `n.cells = 100`, then 1000 cell profiles will be simulated).
`suffix.names`	Suffix used on simulated cells. This suffix must be unique in the simulated cells, so make sure that this suffix does not appear in the real cell names.
`cell.types`	Vector indicating the cell types to simulate. If `NULL` (by default), `n.cells` single-cell profiles for all cell types will be simulated.
`file.backend`	Valid file path to store the simulated single-cell expression profiles as an HDF5 file (`NULL` by default). If provided, the data are stored in HDF5 files used as

back-end by using the **DelayedArray**, **HDF5Array** and **rhdf5** packages instead of loading all data into RAM memory. This is suitable for situations where you have large amounts of data that cannot be loaded into memory. Note that operations on this data will be performed in blocks (i.e subsets of determined size) which may result in longer execution times.

`name.dataset.backend`
: Name of the dataset in HDF5 file to be used. Note that it cannot exist. If `NULL` (by default), a random dataset name will be used.

`compression.level`
: The compression level used if `file.backend` is provided. It is an integer value between 0 (no compression) and 9 (highest and slowest compression). See ?`getHDF5DumpCompressionLevel` from the **HDF5Array** package for more information.

`block.processing`
: Boolean indicating whether the data should be simulated in blocks (only if `file.backend` is used, `FALSE` by default). This functionality is suitable for cases where is not possible to load all data into memory and it leads to larger execution times.

`block.size`
: Only if `block.processing = TRUE`. Number of single-cell expression profiles that will be simulated in each iteration during the process. Larger numbers result in higher memory usage but shorter execution times. Set according to available computational resources (1000 by default). Note that it cannot be greater than the total number of simulated cells.

`chunk.dims`
: Specifies the dimensions that HDF5 chunk will have. If `NULL`, the default value is a vector of two items: the number of genes considered by the ZINB-WaVE model during the simulation and a single sample in order to reduce read times in the following steps. A larger number of columns written in each chunk can lead to longer read times in subsequent steps. Note that it cannot be greater than the dimensions of the simulated matrix.

`verbose`
: Show informative messages during the execution (`TRUE` by default).

Details

Before this step, see ?`estimateZinbwaveParams`. As described in Torroja and Sanchez-Cabo, 2019, this function simulates a given number of transcriptional profiles for each cell type provided by randomly sampling from a negative binomial distribution with μ and θ estimated parameters and inserting dropouts by sampling from a binomial distribution with probability pi. All parameters are estimated from single-cell real data using the `estimateZinbwaveParams` function. It uses the ZINB-WaVE model (Risso et al., 2018). For more details about the model, see ?`estimateZinbwaveParams` and Risso et al., 2018.

The `file.backend` argument allows to create a HDF5 file with simulated single-cell profiles to be used as back-end to work with data stored on disk instead of loaded into RAM. If the `file.backend` argument is used with `block.processing = FALSE`, all the single-cell profiles will be simulated in one step and, therefore, loaded into in RAM memory. Then, data will be written in HDF5 file. To avoid to collapse RAM memory if too many single-cell profiles are goin to be simulated, single-cell profiles can be simulated and written to HDF5 files in blocks of `block.size` size by setting `block.processing = TRUE`.

Value

A SpatialDDLS object with single.cell.simul slot containing a SingleCellExperiment object with the simulated single-cell expression profiles.

References

Risso, D., Perraudeau, F., Gribkova, S. et al. (2018). A general and flexible method for signal extraction from single-cell RNA-seq data. Nat Commun 9, 284. doi: doi:10.1038/s41467017-025545.

Torroja, C. and Sánchez-Cabo, F. (2019). digitalDLSorter: A Deep Learning algorithm to quantify immune cell populations based on scRNA-Seq data. Frontiers in Genetics 10, 978. doi: doi:10.3389/fgene.2019.00978.

See Also

estimateZinbwaveParams

Examples

```
set.seed(123) # reproducibility
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 10,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(10)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(10)),
    Cell_Type = sample(x = paste0("CellType", seq(2)), size = 10,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE,
  project = "Simul_example"
)
SDDLS <- estimateZinbwaveParams(
  object = SDDLS,
  cell.type.column = "Cell_Type",
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID",
  subset.cells = 2,
  verbose = TRUE
)
```

```
SDDLS <- simSCProfiles(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  n.cells = 2,
  verbose = TRUE
)
```

`single.cell.real` *Get and set* `single.cell.real` *slot in a* `SpatialExperiment` *object*

Description

Get and set `single.cell.real` slot in a `SpatialExperiment` object

Usage

```
single.cell.real(object)

single.cell.real(object) <- value
```

Arguments

`object`	`SpatialExperiment` object.
`value`	`SingleCellExperiment` object with real single-cell profiles.

`single.cell.simul` *Get and set* `single.cell.simul` *slot in a* `SpatialExperiment` *object*

Description

Get and set `single.cell.simul` slot in a `SpatialExperiment` object

Usage

```
single.cell.simul(object)

single.cell.simul(object) <- value
```

Arguments

`object`	`SpatialExperiment` object.
`value`	`SingleCellExperiment` object with simulated single-cell profiles.

spatial.experiments *Get and set* spatial.experiments *slot in a* SpatialExperiment *object*

Description

Get and set spatial.experiments slot in a SpatialExperiment object

Usage

```
spatial.experiments(object, index.st = NULL)

spatial.experiments(object, index.st = NULL) <- value
```

Arguments

object	SpatialExperiment object.
index.st	Index of the spatial transcriptomics data within the list. It can be either an position or a name if a named list was provided. If NULL (by default), all data contained in the spatial.experiments slot are returned.
value	List in which each element is a SpatialExperiment object. It can be a named list.

SpatialDDLS-class *The SpatialDDLS Class*

Description

The SpatialDDLS object is the core of the **SpatialDDLS** package. This object stores different intermediate data needed for the construction of new deconvolution models, the spatial transcriptomics profiles to be deconvoluted, and the predicted cell type proportions.

Details

This object uses other classes to store different types of data generated during the workflow:

- SingleCellExperiment class for single-cell RNA-Seq data storage, using sparse matrix from the **Matrix** package (dgCMatrix class) or HDF5Array class in case of using HDF5 files as back-end (see below for more information).
- SpatialExperiment class for spatial transcriptomics data storage.
- zinbModel class with estimated parameters for the simulation of new single-cell profiles.
- SummarizedExperiment class for simulated mixed transcriptional profiles storage.
- PropCellTypes class for composition cell type matrices. See ?PropCellTypes for details.

- DeconvDLModel class to store information related to deep neural network models. See ?DeconvDLModel for details.

In order to provide a way to work with large amounts of data in RAM-constrained machines, we provide the possibility of using HDF5 files as back-end to store count matrices of both real and simulated single-cell profiles by using the **HDF5Array** and **DelayedArray** classes from the homonymous packages.

Slots

`single.cell.real` Real single-cell data stored in a `SingleCellExperiment` object. The count matrix is stored either as `dgCMatrix` or `HDF5Array` objects.

`spatial.experiments` List of `SpatialExperiment` objects to be deconvoluted.

`zinb.params` `zinbModel` object with estimated parameters for the simulation of new single-cell expression profiles.

`single.cell.simul` Simulated single-cell expression profiles using the ZINB-WaVE model.

`prob.cell.types` `PropCellTypes` class with cell composition matrices built for the simulation of mixed transcriptional profiles with known cell composition.

`mixed.profiles` List of simulated train and test mixed transcriptional profiles. Each entry is a `SummarizedExperiment` object. Count matrices can be stored as `HDF5Array` objects using HDF5 files as back-end in case of RAM limitations.

`trained.model` `DeconvDLModel` object with information related to the deconvolution model. See ?`DeconvDLModel` for more details.

`deconv.spots` Deconvolution results. It consists of a list where each element corresponds to the results for each `SpatialExperiment` object contained in the `spatial.experiments` slot.

`project` Name of the project.

`version` Version of **SpatialDDLS** this object was built under.

`SpatialDDLS-Rpackage` *SpatialDDLS: an R package to deconvolute spatial transcriptomics data using deep neural networks*

Description

SpatialDDLS is an R package that provides a neural network-based solution for cell type deconvolution of spatial transcriptomics data. The package takes advantage of single-cell RNA sequencing (scRNA-seq) data to simulate mixed transcriptional profiles with known cell composition and train fully-connected neural networks to predict the cell type composition of spatial transcriptomics spots. The resulting trained models can be applied to new spatial transcriptomics data to predict cell type proportions, allowing for more accurate cell type identification and characterization of spatially-resolved transcriptomic data. Finally, predictions are forced to keep spatial consistency through a process we refer to as spatial regularization. Overall, **SpatialDDLS** is a powerful tool for cell type deconvolution in spatial transcriptomics data, providing a reliable, fast and flexible solution for researchers in the field. See Mañanes et al. (2024) (doi:10.1093/bioinformatics/btae072) and some examples (`https://diegommcc.github.io/SpatialDDLS/`) for more details.

spatialPropClustering *Cluster spatial data based on predicted cell proportions*

Description

Cluster spatial transcriptomics data according to the cell proportions predicted in each spot. It allows to segregate ST data into niches with similar cell composition.

Usage

```
spatialPropClustering(
  object,
  index.st,
  method = "graph",
  k.nn = 10,
  k.centers = 5,
  verbose = TRUE
)
```

Arguments

object	SpatialDDLS object with deconvoluted ST datasets.
index.st	Name or index of the dataset/slide already deconvoluted to be clustered. If missing, all datasets already deconvoluted will be clustered.
method	Clustering method. It can be graph (a nearest neighbor graph is created and Louvain algorithm is used to detect communities) or k.means (k-means algorithm is run with the specified number of centers (k.centers parameter)).
k.nn	An integer specifying the number of nearest neighbors to be used during graph construction (10 by default). Only if method == "graph".
k.centers	An integer specifying the number of centers for k-means algorithm (5 by default). Only if method == "k.means".
verbose	Show informative messages during the execution (TRUE by default).

Value

A SpatialDDLS object containing computed clusters as a column in the slot colData of the SpatialExperiment objects.

See Also

plotTrainingHistory deconvSpatialDDLS

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 10,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(10)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(10)),
    Cell_Type = sample(x = paste0("CellType", seq(2)), size = 10,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
SDDLS <- trainDeconvModel(
  SDDLS,
  batch.size = 12,
  num.epochs = 5
)
# simulating spatial data
ngenes <- sample(3:40, size = 1)
ncells <- sample(10:40, size = 1)
counts <- matrix(
  rpois(ngenes * ncells, lambda = 5), ncol = ncells,
  dimnames = list(paste0("Gene", seq(ngenes)), paste0("Spot", seq(ncells)))
)
coordinates <- matrix(
  rep(c(1, 2), ncells), ncol = 2
)
st <- SpatialExperiment::SpatialExperiment(
  assays = list(counts = as.matrix(counts)),
  rowData = data.frame(Gene_ID = paste0("Gene", seq(ngenes))),
```

```
  colData = data.frame(Cell_ID = paste0("Spot", seq(ncells))),
  spatialCoords = coordinates
)
SDDLS <- loadSTProfiles(
  object = SDDLS,
  st.data = st,
  st.spot.ID.column = "Cell_ID",
  st.gene.ID.column = "Gene_ID"
)
SDDLS <- deconvSpatialDDLS(
  SDDLS,
  index.st = 1
)
SDDLS <- spatialPropClustering(SDDLS, index.st = 1, k.nn = 5)
```

`test.deconv.metrics` *Get and set* `test.deconv.metrics` *slot in a* `DeconvDLModel` *object*

Description

Get and set `test.deconv.metrics` slot in a `DeconvDLModel` object

Usage

```
test.deconv.metrics(object, metrics = "All")

test.deconv.metrics(object, metrics = "All") <- value
```

Arguments

`object`	`DeconvDLModel` object.
`metrics`	Metrics to show (`'All'` by default)
`value`	List with evaluation metrics to assess the performance of the model on each sample of test data.

`test.metrics` *Get and set* `test.metrics` *slot in a* `DeconvDLModel` *object*

Description

Get and set `test.metrics` slot in a `DeconvDLModel` object

Usage

```
test.metrics(object)

test.metrics(object) <- value
```

Arguments

object	DeconvDLModel object.
value	List with evaluation metrics after prediction on test data.

---

test.pred *Get and set* test.pred *slot in a* DeconvDLModel *object*

Description

Get and set test.pred slot in a DeconvDLModel object

Usage

```
test.pred(object)

test.pred(object) <- value
```

Arguments

object	DeconvDLModel object.
value	Matrix object with prediction results on test data.

---

topGradientsCellType *Get top genes with largest/smallest gradients per cell type*

Description

Retrieve feature names with the largest/smallest gradients per cell type. These genes can be used to visualize their spatial expression in the ST data (plotGeneSpatial function) or to plot the calculated gradients as a heatmap (plotGradHeatmap function).

Usage

```
topGradientsCellType(object, method = "class", top.n.genes = 15)
```

Arguments

object	SpatialDDLS object with a DeconvDLModel object containing gradients in the interpret.gradients slot.
method	Method gradients were calculated by. It can be either 'class' (gradients of predicted classes w.r.t. inputs) or 'loss' (gradients of loss w.r.t. input features).
top.n.genes	Top n genes (positive and negative) taken per cell type.

Value

List of gene names with the top positive and negative gradients per cell type.

See Also

interGradientsDL trainDeconvModel

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 10,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(10)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(10)),
    Cell_Type = sample(x = paste0("CellType", seq(2)), size = 10,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
SDDLS <- trainDeconvModel(
```

```
  object = SDDLS,
  batch.size = 12,
  num.epochs = 5
)
## calculating gradients
SDDLS <- interGradientsDL(SDDLS)
listGradients <- topGradientsCellType(SDDLS)
lapply(listGradients, head, n = 5)
```

`trainDeconvModel` *Train deconvolution model for spatial transcriptomics data*

Description

Train a deep neural network model using training data from the `SpatialDDLS` object. This model will be used to deconvolute spatial transcriptomics data from the same biological context as the single-cell RNA-seq data used to train it. In addition, the trained model is evaluated using test data, and prediction results are obtained to determine its performance (see `?calculateEvalMetrics`).

Usage

```
trainDeconvModel(
  object,
  type.data.train = "mixed",
  type.data.test = "mixed",
  batch.size = 64,
  num.epochs = 60,
  num.hidden.layers = 2,
  num.units = c(200, 200),
  activation.fun = "relu",
  dropout.rate = 0.25,
  loss = "kullback_leibler_divergence",
  metrics = c("accuracy", "mean_absolute_error", "categorical_accuracy"),
  normalize = TRUE,
  scaling = "standardize",
  norm.batch.layers = TRUE,
  custom.model = NULL,
  shuffle = TRUE,
  sc.downsampling = NULL,
  use.generator = FALSE,
  on.the.fly = FALSE,
  agg.function = "AddRawCount",
  threads = 1,
  view.metrics.plot = TRUE,
  verbose = TRUE
)
```

Arguments

object	SpatialDDLS object with single.cell.real/single.cell.simul, prob.cell.types, and mixed.profiles slots (the last only if on.the.fly = FALSE).
type.data.train	Type of profiles to be used for training. It can be 'both', 'single-cell' or 'mixed' ('mixed' by default).
type.data.test	Type of profiles to be used for evaluation. It can be 'both', 'single-cell' or 'mixed' ('mixed' by default).
batch.size	Number of samples per gradient update (64 by default).
num.epochs	Number of epochs to train the model (60 by default).
num.hidden.layers	Number of hidden layers of the neural network (2 by default). This number must be equal to the length of num.units argument.
num.units	Vector indicating the number of neurons per hidden layer (c(200, 200) by default). The length of this vector must be equal to the num.hidden.layers argument.
activation.fun	Activation function ('relu' by default). See the keras documentation to know available activation functions.
dropout.rate	Float between 0 and 1 indicating the fraction of input neurons to be dropped in layer dropouts (0.25 by default). By default, **SpatialDDLS** implements 1 dropout layer per hidden layer.
loss	Character indicating loss function selected for model training ('kullback_leibler_divergence' by default). See the keras documentation to know available loss functions.
metrics	Vector of metrics used to assess model performance during training and evaluation (c("accuracy", "mean_absolute_error", "categorical_accuracy") by default). See the keras documentation to know available performance metrics.
normalize	Whether to normalize data using logCPM (TRUE by default). This parameter is only considered when the method used to simulate mixed transcriptional profiles (simMixedProfiles function) was "AddRawCount". Otherwise, data were already normalized.
scaling	How to scale data before training. It can be: "standardize" (values are centered around the mean with a unit standard deviation), "rescale" (values are shifted and rescaled so that they end up ranging between 0 and 1) or "none" (no scaling is performed). "standardize" by default.
norm.batch.layers	Whether to include batch normalization layers between each hidden dense layer (TRUE by default).
custom.model	It allows to use a custom neural network architecture. It must be a keras.engine.sequential.Sequenti object in which the number of input neurons is equal to the number of considered features/genes, and the number of output neurons is equal to the number of cell types considered (NULL by default). If provided, the arguments related to the neural network architecture will be ignored.
shuffle	Boolean indicating whether data will be shuffled (TRUE by default).

`sc.downsampling`
: It is only used if `type.data.train` is equal to `'both'` or `'single-cell'`. It allows to set a maximum number of single-cell profiles of a specific cell type for training to avoid an unbalanced representation of classes (`NULL` by default).

`use.generator`
: Boolean indicating whether to use generators during training and test. Generators are automatically used when `on.the.fly` = `TRUE` or HDF5 files are used, but it can be activated by the user on demand (`FALSE` by default).

`on.the.fly`
: Boolean indicating whether simulated data will be generated 'on the fly' during training (`FALSE` by default).

`agg.function`
: If `on.the.fly` == `TRUE`, function used to build mixed transcriptional profiles. It may be:
  - `"AddRawCount"` (by default): single-cell profiles (raw counts) are added up across cells. Then, log-CPMs are calculated.
  - `"MeanCPM"`: single-cell profiles (raw counts) are transformed into logCPM and cross-cell averages are calculated.
  - `"AddCPM"`: single-cell profiles (raw counts) are transformed into CPMs and are added up across cells. Then, log-CPMs are calculated.

`threads`
: Number of threads used during simulation of mixed transcriptional profiles if `on.the.fly` = `TRUE` (1 by default).

`view.metrics.plot`
: Boolean indicating whether to show plots of loss and evaluation metrics during training (`TRUE` by default). **keras** for R allows to see model progression during training if you are working in RStudio.

`verbose`
: Boolean indicating whether to display model progression during training and model architecture information (`TRUE` by default).

Details

Simulation of mixed transcriptional profiles 'on the fly'

`trainDeconvModel` can avoid storing simulated mixed spot profiles by using the `on.the.fly` argument. This functionality aims at reducing the the `simMixedProfiles` function's memory usage: simulated profiles are built in each batch during training/evaluation.

Neural network architecture

It is possible to change the model's architecture: number of hidden layers, number of neurons for each hidden layer, dropout rate, activation function, and loss function. For more customized models, it is possible to provide a pre-built model through the `custom.model` argument (a `keras.engine.sequential.Sequential` object) where it is necessary that the number of input neurons is equal to the number of considered features/genes, and the number of output neurons is equal to the number of considered cell types.

Value

A `SpatialDDLS` object with `trained.model` slot containing a `DeconvDLModel` object. For more information about the structure of this class, see `?DeconvDLModel`.

See Also

`plotTrainingHistory` `deconvSpatialDDLS`

Examples

```
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(30, lambda = 5), nrow = 15, ncol = 10,
      dimnames = list(paste0("Gene", seq(15)), paste0("RHC", seq(10)))
    )
  ),
  colData = data.frame(
    Cell_ID = paste0("RHC", seq(10)),
    Cell_Type = sample(x = paste0("CellType", seq(2)), size = 10,
                       replace = TRUE)
  ),
  rowData = data.frame(
    Gene_ID = paste0("Gene", seq(15))
  )
)
SDDLS <- createSpatialDDLSobject(
  sc.data = sce,
  sc.cell.ID.column = "Cell_ID",
  sc.gene.ID.column = "Gene_ID",
  sc.filt.genes.cluster = FALSE
)
SDDLS <- genMixedCellProp(
  object = SDDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  num.sim.spots = 50,
  train.freq.cells = 2/3,
  train.freq.spots = 2/3,
  verbose = TRUE
)
SDDLS <- simMixedProfiles(SDDLS)
SDDLS <- trainDeconvModel(
  object = SDDLS,
  batch.size = 12,
  num.epochs = 5
)
```

`trained.model` *Get and set* `trained.model` *slot in a* `SpatialExperiment` *object*

Description

Get and set `trained.model` slot in a `SpatialExperiment` object

Usage

```
trained.model(object)

trained.model(object) <- value
```

Arguments

object	SpatialExperiment object.
value	DeconvDLModel object.

---

training.history *Get and set* training.history *slot in a* DeconvDLModel *object*

Description

Get and set training.history slot in a DeconvDLModel object

Usage

```
training.history(object)

training.history(object) <- value
```

Arguments

object	DeconvDLModel object.
value	keras_training_history object with the training history of the deep neural network model.

---

zinb.params *Get and set* zinb.params *slot in a* SpatialExperiment *object*

Description

Get and set zinb.params slot in a SpatialExperiment object

Usage

```
zinb.params(object)

zinb.params(object) <- value
```

Arguments

object	SpatialExperiment object.
value	ZinbParametersModel object with a valid zinbModel object.

ZinbParametersModel-class *The Class ZinbParametersModel*

Description

The ZinbParametersModel class is a wrapper class for the zinbModel class from the **zinbwave** package.

Details

This wrapper class contains the zinbwave.model slot, which holds a valid zinbModel object.

Slots

zinbwave.model A valid zinbModel object.

References

Risso, D., Perraudeau, F., Gribkova, S. et al. (2018). A general and flexible method for signal extraction from single-cell RNA-seq data. Nat Commun 9, 284. doi: doi:10.1038/s41467017-025545.

zinbwave.model *Get and set* zinbwave.model *slot in a* ZinbParametersModel *object*

Description

Get and set zinbwave.model slot in a ZinbParametersModel object

Usage

```
zinbwave.model(object)

zinbwave.model(object) <- value
```

Arguments

object	ZinbParametersModel object.
value	zinbModel object with the estimated parameters to simulate new single-cell profiles.

Index