Package ‘digitalDLSorteR’

October 13, 2022

Type Package
Title Deconvolution of Bulk RNA-Seq Data Based on Deep Learning
Version 0.3.1
Maintainer Diego Mañanes <dmananes@cnic.es>
Description Deconvolution of bulk RNA-Seq data using context-specific deconvolution models based on Deep Neural Networks using scRNA-Seq data as input. These models are able to make accurate estimates of the cell composition of bulk RNA-Seq samples from the same context using the advances provided by Deep Learning and the meaningful information provided by scRNA-Seq data. See Torroja and Sanchez-Cabo (2019) <doi:10.3389/fgene.2019.00978> for more details.
License GPL-3
BugReports https://github.com/diegommcc/digitalDLSorteR/issues
Encoding UTF-8
Depends R (>= 4.0.0)
Imports rlang, Matrix, methods, tidyr, SingleCellExperiment, SummarizedExperiment, zinbwave, stats, pbapply, S4Vectors, dplyr, tools, reshape2, gtools, reticulate, keras, tensorflow, ggplot2, ggpubr, RColorBrewer
Suggests knitr, rmarkdown, BiocParallel, rhdf5, DelayedArray, DelayedMatrixStats, HDF5Array, testthat
SystemRequirements Python (>= 2.7.0), TensorFlow (https://www.tensorflow.org/)
RoxygenNote 7.1.2
Collate 'AllClasses.R' 'AllGenerics.R' 'digitalDLSorteR.R'
  'dnnModel.R' 'evalMetrics.R' 'loadData.R' 'simBulk.R'
  'simSingleCell.R' 'utils.R'
VignetteBuilder knitr
NeedsCompilation no
### R topics documented:

- barErrorPlot .................................................. 3
- barPlotCellTypes ............................................ 5
- blandAltmanLehPlot ......................................... 7
- bulk.simul ..................................................... 10
- calculateEvalMetrics ....................................... 10
- cell.names ..................................................... 12
- cell.types ..................................................... 12
- corrExpPredPlot ............................................. 13
- deconv.data ................................................. 16
- deconv.results .............................................. 16
- deconv.DigitalDLSorter .................................... 17
- deconv.DigitalDLSorterObj ................................ 20
- digitalDLSorterR ............................................ 22
- DigitalDLSorter-class ..................................... 23
- DigitalDLSorterDNN-class ................................ 25
- distErrorPlot ................................................ 26
- estimateZinbwaveParams .................................. 29
- features ....................................................... 31
- generateBulkCellMatrix ................................... 32
- getProbMatrix ................................................ 35
- installTFpython ............................................. 36
- listToDDL ..................................................... 37
- listToDDLSDNN ............................................... 37
- loadDeconvData ............................................ 38
- loadSCPProfiles ............................................. 39
- loadTrainedModelFromH5 .................................. 41
- method ......................................................... 42
- model .......................................................... 42
- plots .......................................................... 43
- plotTrainingHistory ....................................... 43
- preparingToSave .......................................... 44
- prob.cell.types ............................................. 45
- prob.matrix .................................................. 45
- ProbMatrixCellTypes-class ................................ 46
- project ......................................................... 47
- saveRDS ........................................................ 47
- saveTrainedModelAsH5 ..................................... 48
- set .............................................................. 49
- set.list ........................................................ 50
Description

Generate bar error plots by cell type (CellType) or by number of different cell types (nCellTypes) on test pseudo-bulk samples.

Usage

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

Arguments

object  DigitalDLSorter object with trained.model slot containing metrics in test.deconv.metrics slot.
error   'MAE' or 'MSE'.
by      Variable used to display errors. Available options are: 'nCellTypes', 'CellType'.
dispersion Standard error ('se') or standard deviation ('sd'). The former is the default.
filter.sc Boolean indicating whether single-cell profiles are filtered out and only correlation of results associated with bulk samples are displayed (TRUE by default).
barErrorPlot

title Title of the plot.
angle Angle of ticks.
theme ggplot2 theme.

Value
A ggplot object with the mean and dispersion of the errors

See Also
calculateEvalMetrics corrExpPredPlot distErrorPlot blandAltmanLehPlot

Examples

## Not run:
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))
  )
)
DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)
probMatrixValid <- data.frame(
  Cell_Type = paste0("CellType", seq(6)),
  from = c(1, 1, 1, 15, 15, 30),
  to = c(15, 15, 30, 50, 50, 70)
)
DDLS <- generateBulkCellMatrix(
  object = DDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  prob.design = probMatrixValid,
  num.bulk.samples = 50,
  verbose = TRUE
)
# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
barPlotCellTypes

DDLS <- trainDigitalDLSorterModel(
  object = DDLS,
  on.the.fly = TRUE,
  batch.size = 15,
  num.eepochs = 5
)
# evaluation using test data
DDLS <- calculateEvalMetrics(
  object = DDLS
)
# bar error plots
barErrorPlot(
  object = DDLS,
  error = "MSE",
  by = "CellType"
)
barErrorPlot(
  object = DDLS,
  error = "MAE",
  by = "nCellTypes"
)

## End(Not run)

barPlotCellTypes

Bar plot of deconvoluted cell type proportions in bulk RNA-Seq samples

Description

Bar plot of deconvoluted cell type proportions in bulk RNA-Seq samples.

Usage

barPlotCellTypes(
  data,
  colors = NULL,
  simplify = NULL,
  color.line = NA,
  x.label = "Bulk samples",
  rm.x.text = FALSE,
  title = "Results of deconvolution",
  legend.title = "Cell types",
  angle = 90,
  theme = NULL,
  ...
)
## S4 method for signature 'DigitalDLSorter'

```r
barPlotCellTypes(
  data,
  colors = NULL,
  simplify = NULL,
  color.line = NA,
  x.label = "Bulk samples",
  rm.x.text = FALSE,
  title = "Results of deconvolution",
  legend.title = "Cell types",
  angle = 90,
  theme = NULL,
  name.data = NULL
)
```

## S4 method for signature 'ANY'

```r
barPlotCellTypes(
  data,
  colors,
  color.line = NA,
  x.label = "Bulk samples",
  rm.x.text = FALSE,
  title = "Results of deconvolution",
  legend.title = "Cell types",
  angle = 90,
  theme = NULL
)
```

### Arguments

- **data** 
  - `DigitalDLSorter` object with `deconv.results` slot or a data frame/matrix with cell types as columns and samples as rows.
- **colors**
  - Vector of colors to be used.
- **simplify**
  - Type of simplification performed during deconvolution. Can be `simpli.set` or `simpli.maj` (NULL by default). It is only for `DigitalDLSorter` objects.
- **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.
- **...**
  - Other arguments for specific methods.
- **name.data**
  - If a `DigitalDLSorter` is given, name of the element that stores the results in the `deconv.results` slot.
Value

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

See Also

decovDigitalDLSorter  deconvDigitalDLSorterObj

Examples

# matrix of simulated proportions (same structure as deconvolution results)
deconvResults <- gtools::rdirichlet(n = 20, alpha = c(1, 1, 1, 0.5, 0.1))
colnames(deconvResults) <- paste("CellType", seq(ncol(deconvResults)))
rownames(deconvResults) <- paste("BulkSample", seq(nrow(deconvResults)))
barPlotCellTypes(deconvResults)

# Using a DigitalDLSorter object
DDLS <- DigitalDLSorter(deconv.results = list(Example = deconvResults))
barPlotCellTypes(DDLS)


... }

Arguments

object DigitalDLSorter object with trained.model slot containing metrics in test.deconv.metrics slot.

colors Vector of colors to be used. Only vectors with a number of colors equal to or greater than the levels of color.by will be accepted. By default a custom color list is used.

color.by Variable used to color data. Options are nCellTypes and CellType.

facet.by Variable used to display 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 display 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 bulk samples are displayed (TRUE by default).

density Boolean indicating whether density lines must be displayed (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.

Value

A ggplot object with Bland-Altman agreement plots between expected and actual proportions.

See Also

calculateEvalMetrics corrExpPredPlot distErrorPlot barErrorPlot

Examples

## Not run:
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))
)

DDLS <- loadSCProfiles(
    single.cell.data = sce,
    cell.ID.column = "Cell_ID",
    gene.ID.column = "Gene_ID"
)

probMatrixValid <- data.frame(
    Cell_Type = paste0("CellType", seq(6)),
    from = c(1, 1, 1, 15, 15, 30),
    to = c(15, 15, 30, 50, 50, 70)
)

DDLS <- generateBulkCellMatrix(
    object = DDLS,
    cell.ID.column = "Cell_ID",
    cell.type.column = "Cell_Type",
    prob.design = probMatrixValid,
    num.bulk.samples = 50,
    verbose = TRUE
)

# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
DDLS <- trainDigitalDLSorterModel(
    object = DDLS,
    on.the.fly = TRUE,
    batch.size = 15,
    num.epochs = 5
)

# evaluation using test data
DDLS <- calculateEvalMetrics(
    object = DDLS
)

# Bland-Altman plot by cell type
blandAltmanLehPlot(
    object = DDLS,
    facet.by = "CellType",
    color.by = "CellType"
)

# Bland-Altman plot of all samples mixed
blandAltmanLehPlot(
    object = DDLS,
    facet.by = NULL,
    color.by = "CellType",
    alpha.point = 0.3,
calculateEvalMetrics

```r
log2 = TRUE
)

## End(Not run)
```

--

bulk.simul  

*Get and set* bulk.simul slot in a *DigitalDLSorter object*

**Description**

Get and set bulk.simul slot in a **DigitalDLSorter** object

**Usage**

```r
bulk.simul(object, type.data = "both")
bulk.simul(object, type.data = "both") <- value
```

**Arguments**

- **object**  
  DigitalDLSorter object.

- **type.data**  
  Element of the list. Can be 'train', 'test' or 'both' (the last by default).

- **value**  
  List with two elements, train and test, each one being a **SummarizedExperiment** object with simulated bulk RNA-Seq samples.

---

calculateEvalMetrics  

*Calculate evaluation metrics for bulk RNA-Seq samples from test data*

**Description**

Calculate evaluation metrics for bulk RNA-seq samples from test data to understand model performance. By default, absolute error (AbsErr), proportional absolute error (ppAbsErr), squared error (SqrErr) and proportional squared error (ppSqrErr) are calculated for each test sample. In addition, each of these metrics is aggregated using their mean values according to three criteria: each cell type (CellType), probability bins in ranges of 0.1 (pBin) and number of different cell types present in the sample nCellTypes. Finally, the process is repeated only considering bulk samples (filtering out single-cell profiles from the evaluation). The evaluation metrics will be available in the test.deconv.metrics slot of the **DigitalDLSorterDNN** object (trained.model slot of the **DigitalDLSorter** object).

**Usage**

```r
calculateEvalMetrics(object, metrics = c("MAE", "MSE"))
```
calculateEvalMetrics

Arguments

- **object**  
  DigitalDLSorter object with a trained model in the trained.model slot and the actual cell proportions of pseudo-bulk samples in prob.cell.matrix slot.

- **metrics**  
  Metrics used to evaluate the model performance. Mean absolute error ("MAE") and mean squared error ("MSE") by default.

Value

A DigitalDLSorter object with the trained.model slot containing a DigitalDLSorterDNN object with the test.deconv.metrics slot. The last contains the metrics calculated.

See Also

distErrorPlot corrExpPredPlot blandAltmanLehPlot barErrorPlot

Examples

```r
## Not run:
set.seed(123)
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(
    counts = matrix(
      rpois(300, 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))
  )
)
DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)
probMatrixValid <- data.frame(
  Cell_Type = paste0("CellType", seq(6)),
  from = c(1, 1, 1, 15, 15, 30),
  to = c(15, 15, 30, 50, 50, 70)
)
DDLS <- generateBulkCellMatrix(
  object = DDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  prob.design = probMatrixValid,
  num.bulk.samples = 50,
)"
verbose = TRUE
)
# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
DDLS <- trainDigitalDLSorterModel(
  object = DDLS,
  on.the.fly = TRUE,
  batch.size = 15,
  num.epochs = 5
)
# evaluation using test data
DDLS <- calculateEvalMetrics(
  object = DDLS
)
## End(Not run)

---

**cell.names**

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

**Description**

Get and set cell.names slot in a ProbMatrixCellTypes object

**Usage**

```r
cell.names(object)
```

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

**Arguments**

- **object**: ProbMatrixCellTypes object.
- **value**: Matrix containing the name of the pseudo-bulk samples to be simulated as rows and the cells to be used to simulate them as columns (n.cell argument)

---

**cell.types**

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

**Description**

Get and set cell.types slot in a DigitalDLSorterDNN object
Usage

cell.types(object)

cell.types(object) <- value

Arguments

object                  DigitalDLSorterDNN object.
value                   Vector with cell types considered by the Deep Neural Network model.

Description

Generate correlation plots between predicted and expected cell type proportions from test data. Correlation plots can be displayed all mixed or split by cell type (CellType) or number of cell types present in the samples (nCellTypes). See the facet.by argument and examples for more information. Moreover, a user-selected correlation value is displayed as an annotation on the plots. See the corr argument for details.

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

DigitalDLSorter object with trained.model slot containing metrics in the test.deconv.metrics slot of a DigitalDLSorterDNN object.

colors

Vector of colors to be used. Only vectors with a number of colors equal to or greater than the levels of color.by will be accepted. By default, a custom color list is used.

facet.by

Variable used to display 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).

color.by

Variable used to color data. Options are nCellTypes and CellType.

corr

Correlation value displayed as an annotation on the plot. Available metrics are Pearson's correlation coefficient ('pearson') and concordance correlation coefficient ('ccc'). The argument can be 'pearson', 'ccc' or 'both' (by default).

filter.sc

Boolean indicating whether single-cell profiles are filtered out and only errors associated with pseudo-bulk samples are displayed (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 from gplots if facet.by is not NULL.

Value

A ggplot object with the correlation plots between expected and actual proportions.

See Also

calculateEvalMetrics distErrorPlot blandAltmanLehPlot barErrorPlot

Examples

## Not run:
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))
)
)
DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)
probMatrixValid <- data.frame(
  Cell_Type = paste0("CellType", seq(6)),
  from = c(1, 1, 1, 15, 15, 30),
  to = c(15, 15, 30, 50, 50, 70)
)
DDLS <- generateBulkCellMatrix(
  object = DDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  prob.design = probMatrixValid,
  num.bulk.samples = 50,
  verbose = TRUE
)
# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
DDLS <- trainDigitalDLSorferModel(
  object = DDLS,
  on.the.fly = TRUE,
  batch.size = 15,
  num.epochs = 5
)
# evaluation using test data
DDLS <- calculateEvalMetrics(
  object = DDLS
)
# correlations by cell type
corrExpPredPlot(
  object = DDLS,
  facet.by = "CellType",
  color.by = "CellType",
  corr = "both"
)
# correlations of all samples mixed
corrExpPredPlot(
  object = DDLS,
  facet.by = NULL,
  color.by = "CellType",
  corr = "ccc"
deconv.data

### Get and set deconv.data slot in a DigitalDLSorter object

**Description**

Get and set deconv.data slot in a DigitalDLSorter object

**Usage**

```r
deconv.data(object, name.data = NULL)
```

```r
deconv.data(object, name.data = NULL) <- value
```

**Arguments**

- **object** 
  - DigitalDLSorter object.
- **name.data** 
  - Name of the data. If NULL (by default), all data contained in the deconv.data slot are returned.
- **value** 
  - List whose names are the reference of the stored data.

deconv.results

### Get and set deconv.results slot in a DigitalDLSorter object

**Description**

Get and set deconv.results slot in a DigitalDLSorter object

**Usage**

```r
deconv.results(object, name.data = NULL)
```

```r
deconv.results(object, name.data = NULL) <- value
```

**Arguments**

- **object** 
  - DigitalDLSorter object.
- **name.data** 
  - Name of the data. If NULL (by default), all results contained in the deconv.results slot are returned.
- **value** 
  - List whose names are the reference of the stored results.
Deconvolute bulk RNA-Seq samples using a pre-trained DigitalDL-Sorter model

Description

Deconvolute bulk gene expression samples (bulk RNA-Seq) to enumerate and quantify the proportion of cell types present in a bulk sample using Deep Neural Network models. This function is intended for users who want to use pre-trained models integrated in the package. So far, the available models allow to deconvolute the immune infiltration of breast cancer (using data from Chung et al., 2017) and the immune infiltration of colorectal cancer (using data from Li et al., 2017) samples. For the former, two models are available at two different levels of specificity: specific cell types (breast.chung.specific) and generic cell types (breast.chung.generic). See breast.chung.generic, breast.chung.specific, and colorectal.li documentation from the digitalDLSorterData package for more details.

Usage

deconvDigitalDLSorter(
  data,
  model = NULL,
  batch.size = 128,
  normalize = TRUE,
  scaling = "standarize",
  simplify.set = NULL,
  simplify.majority = NULL,
  verbose = TRUE
)

Arguments

data Matrix or data frame with bulk RNA-Seq samples with genes as rows in SYMBOL notation and samples as columns.
model Pre-trained DNN model to use to deconvolute data. Up to now, the available models are intended to deconvolute samples from breast cancer (breast.chung.generic and breast.chung.specific) and colorectal cancer (colorectal.li). These pre-trained models are stored in the digitalDLSorterData package, so it must be installed together with digitalDLSorter to use this function.
batch.size Number of samples loaded into RAM each time during the deconvolution process. If not specified, batch.size will be set to 128.
normalize Normalize data before deconvolution (TRUE by default).
scaling How to scale data before training. It may be: "standarize" (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 is not scaled.
simplify.set List specifying which cell types should be compressed into a new label whose name will be the list name item. See examples and vignettes for details.

simplify.majority List specifying which cell types should be compressed into the cell type with the highest proportion in each sample. Unlike simplify.set, this argument allows to maintain the complexity of the results while compressing the information, as no new labels are created.

verbose Show informative messages during execution.

Details

This function is intended for users who want to use digitalDLSorteR to deconvolute their bulk RNA-Seq samples using pre-trained models. For users who want to build their own models from other scRNA-Seq datasets, see the loadSCProfiles and deconvDigitalDLSorterObj functions.

Value

A data frame with samples \( (i) \) as rows and cell types \( (j) \) as columns. Each entry represents the predicted proportion of cell type \( j \) in sample \( i \).

References


See Also
deconvDigitalDLSorterObj

Examples

```r
## Not run:
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))
  )
)
DDLS <- loadSCProfiles(
```
single.cell.data = sce,
cell.ID.column = "Cell_ID",
gene.ID.column = "Gene_ID"
)
probMatrixValid <- data.frame(
  Cell_Type = paste0("CellType", seq(6)),
  from = c(1, 1, 1, 15, 15, 30),
  to = c(15, 15, 30, 50, 50, 70)
)
DDLS <- generateBulkCellMatrix(
  object = DDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  prob.design = probMatrixValid,
  num.bulk.samples = 50,
  verbose = TRUE
)
# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
DDLS <- trainDigitalDLSorterModel(
  object = DDLS,
  on.the.fly = TRUE,
  batch.size = 15,
  num.epochs = 5
)
# simulating bulk RNA-Seq data
countsBulk <- matrix(
  stats::rpois(100, lambda = sample(seq(4, 10), size = 100, replace = TRUE)),
  nrow = 40, ncol = 15,
  dimnames = list(paste0("Gene", seq(40)), paste0("Bulk", seq(15)))
)
# this is only an example. See vignettes to see how to use pre-trained models
# from the digitalDLSorterRmodels data package
results1 <- deconvDigitalDLSorter(
  data = countsBulk,
  model = trained.model(DDLS),
  normalize = TRUE
)
# simplify arguments
simplify <- list(CellGroup1 = c("CellType1", "CellType2", "CellType4"),
  CellGroup2 = c("CellType3", "CellType5"))
# in this case the names of the list will be the new labels
results2 <- deconvDigitalDLSorter(
  countsBulk,
  model = trained.model(DDLS),
  normalize = TRUE,
  simplify.set = simplify
)
# in this case the cell type with the highest proportion will be the new label
results3 <- deconvDigitalDLSorter(
  countsBulk,
  model = trained.model(DDLS),
  normalize = TRUE,
Deconvolute bulk gene expression samples (bulk RNA-Seq)

Description

Deconvolute bulk gene expression samples (bulk RNA-Seq). This function requires a `DigitalDLSorter` object with a trained Deep Neural Network model (\texttt{trained.model} slot) and the new bulk RNA-Seq samples to be deconvoluted in the \texttt{deconv.data} slot. See \texttt{?loadDeconvData} for more details.

Usage

\begin{verbatim}
deconvDigitalDLSorterObj(
  object,
  name.data,
  batch.size = 128,
  normalize = TRUE,
  scaling = "standarize",
  simplify.set = NULL,
  simplify.majority = NULL,
  verbose = TRUE
)
\end{verbatim}

Arguments

\begin{itemize}
  \item \textbf{object} \hspace{1cm} `DigitalDLSorter` object with \texttt{trained.data} and \texttt{deconv.data} slots.
  \item \textbf{name.data} \hspace{1cm} Name of the data stored in the `DigitalDLSorter` object. If not provided, the first data set will be used.
  \item \textbf{batch.size} \hspace{1cm} Number of samples per gradient update. If not specified, \texttt{batch.size} will default to 128.
  \item \textbf{normalize} \hspace{1cm} Normalize data before deconvolution (\texttt{TRUE} by default).
  \item \textbf{scaling} \hspace{1cm} How to scale data before training. It may be: "standarize" (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 \texttt{normalize = FALSE}, data is not scaled.
  \item \textbf{simplify.set} \hspace{1cm} List specifying which cell types should be compressed into a new label whose name will be the list item. See examples for details. If provided, results are stored in a list with 'raw' and 'simpli.set' results.
\end{itemize}
simplify.majority
List specifying which cell types should be compressed into the cell type with the highest proportion in each sample. Unlike simplify.set, it allows to maintain the complexity of the results while compressing the information, as no new labels are created. If provided, the results are stored in a list with 'raw' and 'simpli.majority' results.

verbose
Show informative messages during the execution.

Details
This function is intended for users who have built a devonvolution model using their own single-cell RNA-Seq data. If you want to use a pre-trained model to deconvolute your samples, see ?deconvDigitalDLSorter.

Value
DigitalDLSorter object with deconv.results slot. The resulting information is a data frame with samples (i) as rows and cell types (j) as columns. Each entry represents the proportion of j cell type in i sample. If simplify.set or/simplify.majority are provided, the deconv.results slot will contain a list with raw and simplified results.

References

See Also
trainDigitalDLSorterModel DigitalDLSorter

Examples
## Not run:
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))
  )
)
DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)

probMatrixValid <- data.frame(
  Cell_Type = paste0("CellType", seq(6)),
  from = c(1, 1, 1, 15, 15, 30),
  to = c(15, 15, 30, 50, 50, 70)
)

DDLS <- generateBulkCellMatrix(
  object = DDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  prob.design = probMatrixValid,
  num.bulk.samples = 50,
  verbose = TRUE
)

# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
DDLS <- trainDigitalDLSorterModel(
  object = DDLS,
  on.the.fly = TRUE,
  batch.size = 15,
  num.epochs = 5
)

# simulating bulk RNA-Seq data
countsBulk <- matrix(
  stats::rpois(100, lambda = sample(seq(4, 10), size = 100, replace = TRUE)),
  nrow = 40, ncol = 15,
  dimnames = list(paste0("Gene", seq(40)), paste0("Bulk", seq(15)))
)

seBulk <- SummarizedExperiment(assay = list(counts = countsBulk))

DDLS <- loadDeconvData(
  object = DDLS,
  data = seBulk,
  name.data = "Example"
)

# simplify arguments
simplify <- list(CellGroup1 = c("CellType1", "CellType2", "CellType4"),
  CellGroup2 = c("CellType3", "CellType5"))

DDLS <- deconvDigitalDLSorterObj(
  object = DDLS,
  name.data = "Example",
  simplify.set = simplify,
  simplify.majority = simplify
)

## End(Not run)
**digitalDLSorter**

**digitalDLSorter**: an R package to deconvolute bulk RNA-Seq samples using single-cell RNA-Seq data and Deep Learning

---

**Description**

**digitalDLSorter** is an R package that allows to deconvolute bulk RNA-Seq data using context-specific deconvolution models based on single-cell RNA-Seq data and Deep Neural Networks. These models are able to make accurate estimates of the cell composition of bulk RNA-Seq samples from the same context using the advances provided by Deep Learning and the meaningful information provided by scRNA-Seq data. See Torroja and Sanchez-Cabo (2019) (doi: 10.3389/fgene.2019.00978) for more details.

**Details**

The foundation of the method consists of a process that starts from single-cell RNA-Seq data and, after a few steps, a Deep Neural Network (DNN) model is trained with simulated pseudo-bulk RNA-Seq samples whose cell composition is known. These trained models are able to deconvolute any bulk RNA-Seq sample from the same biological context by determining the proportion of present cell types. The main advantage is the possibility to build deconvolution models trained with real data from certain biological environments. For example, to quantify the proportion of tumor infiltrated lymphocytes (TILs) in breast cancer, a specific model for this type of samples can be obtained by using this package. This overcomes the limitation of other methods, since stromal and immune cells may significantly change their transcriptional profiles depending on tissue and disease context.

The package can be used in two ways: deconvoluting bulk RNA-Seq samples using pre-trained models available on the digitalDLSorterRmodels R package or building your own models trained with your own single-cell RNA-Seq data. These new models may be published to make them available for other users working with similar data. So far, available models allow deconvoluting breast and colorectal cancer samples. See vignettes and [https://diegommcc.github.io/digitalDLSorter/](https://diegommcc.github.io/digitalDLSorter/) for more details.

---

**DigitalDLSorter-class**

**The DigitalDLSorter Class**

---

**Description**

The DigitalDLSorter object is the core of digitalDLSorter. This object stores different intermediate data resulting from the creation of new context-specific deconvolution models from single-cell data. It is only used in the case of building new deconvolution models. To deconvolute bulk samples using pre-trained models, see `deconvDigitalDLSorter` function and the package digitalDLSorterData.
Details

This object uses other classes to store the different types of data produced during the process:

- **SingleCellExperiment** class for single-cell RNA-Seq data, using sparse matrix from the **Matrix** package (dgCMatrix class) or HDF5Array class in the case of using HDF5 files as back-end (see below for more information).
- **ZinbModel** class with estimated parameters for the simulation of new single-cell profiles.
- **SummarizedExperiment** class for large bulk RNA-Seq data storage.
- **ProbMatrixCellTypes** class for the compositional cell matrices constructed during the process. See ?ProbMatrixCellTypes for details.
- **DigitalDLSorterDNN** class to store the information related to Deep Neural Network models. This step is performed using keras. See ?DigitalDLSorterDNN for details.

digitalDLSorteR can be used in two ways: to build new deconvolution models from single-cell RNA-Seq data or to deconvolute bulk RNA-Seq samples using pre-trained models available at digitalDLSorteRdata package. If you want to build new models, see loadSCProfiles function. On the other hand, if you want to use pre-trained models, see deconvDigitalDLSorter function.

In order to provide a way to work with large amounts of data on RAM-constrained machines, we provide the possibility to use HDF5 files as back-end to store count matrices of both real/simulated single-cell and bulk RNA-Seq profiles. For this, the package uses the HDF5Array and DelayedArray classes from the homonymous packages.

Once the Deep Neural Network model has been trained, it is possible to save it as RDS or HDF5 files. Please see DigitalDLSorterDNN for more details.

Slots

- **single.cell.real** Real single-cell data stored in a SingleCellExperiment object. The count matrix is stored as dgCMatrix or HDF5Array objects.
- **zinb.params** ZinbModel object with estimated parameters for the simulation of new single-cell expression profiles.
- **single.cell.simul** Simulated single-cell expression profiles from the ZINB-WaVE model.
- **prob.cell.types** ProbMatrixCellTypes class with cell composition matrices built for the simulation of pseudo-bulk RNA-Seq profiles with known cell composition.
- **bulk.simul** A list of simulated train and test bulk RNA-Seq samples. Each entry is a SummarizedExperiment object. The count matrices can be stored as HDF5Array files using HDF5 files as back-end in case of RAM limitations.
- **trained.model** DigitalDLSorterDNN object with all the information related to the trained model. See ?DigitalDLSorterDNN for more details.
- **deconv.data** List of SummarizedExperiment objects where it is possible to store new bulk RNA-Seq experiments for deconvolution. The name of the entries corresponds to the name of the data provided. See deconvDigitalDLSorterObj for details.
- **deconv.results** Slot containing the deconvolution results of applying the deconvolution model to the data present in the deconv.data slot. It is a list in which the names corresponds to the data from which they come.
- **project** Name of the project.
- **version** Version of DigitalDLSorteR this object was built under.
The DigitalDLSorterDNN Class

Description

The DigitalDLSorterDNN object stores all the information related to Deep Neural Network models. It contains the trained model, the training history and the results of prediction 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` package which uses the R6 classes system. If you want to save the object as an RDS file, `digitalDLSorterR` provides a `saveRDS` generic function that transforms the model stored as an R6 object 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. That is the minimum information needed to use the model as predictor. If you want to keep the optimizer state, see `?saveTrainedModelAsH5`. If you want to store `DigitalDLSorter` object on disk as an RDA file, 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.

test.metrics List with the evolution of the selected metrics during training.

test.pred Deconvolution results on test data. Columns are cell types, rows are samples and each entry corresponds to the proportion of this cell type in this sample.

cell.types Vector with cell types to deconvolute.

features Vector with the features used during training. These features will be used in subsequent predictions (the nomenclature used in new bulk RNA-Seq samples must be the same).

test.deconv.metrics Performance of the model on each sample of test data compared to known cell proportions. This slot is used after `calculateEvalMetrics` (see `?calculateEvalMetrics` for more details).
distErrorPlot

Generate box plots or violin plots to show how the errors are distributed

Description

Generate violin plots or box plots to show how the errors are distributed by proportion bins of 0.1. Errors can be displayed all mixed or split by cell type (CellType) or number of cell types present in the samples (nCellTypes). See the facet.by argument and examples for more details.

Usage

```r
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**: DigitalDLSorter object with trained.model slot containing metrics in the test.deconv.metrics slot of a DigitalDLSorterDNN object.
- **error**: The error to be represented. Available errors are absolute error ("AbsErr"), proportional absolute error ("ppAbsErr"), squared error ("SqrErr") and proportional squared error ("ppSqrErr").
- **colors**: Vector of colors to be used. Only vectors with a number of colors equal to or greater than the levels of color.by will be accepted. By default, a custom color list is used.
- **x.by**: Variable used for the X-axis. When facet.by is not NULL, the best choice is pBin (probability bins). The options are nCellTypes (number of different cell types), CellType (cell type) and pBin.
distErrorPlot

- **facet.by**: Variable used to display data in different panels. If NULL, the plot is not split into different panels. Options are nCellTypes (number of different cell types) and CellType (cell type).
- **color.by**: Variable used to color the data. Options are nCellTypes and CellType.
- **filter.sc**: Boolean indicating whether single-cell profiles are filtered out and only errors associated with pseudo-bulk samples are displayed (TRUE by default).
- **error.label**: Boolean indicating whether to display 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 from ggplot2 if facet.by is not NULL.

**Value**

A ggplot object with the representation of the desired errors.

**See Also**

calculateEvalMetrics corrExpPredPlot blandAltmanLehPlot barErrorPlot

**Examples**

```r
## Not run:
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))
)

DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)

probMatrixValid <- data.frame(
  Cell_Type = paste0("CellType", seq(6)),
  from = c(1, 1, 1, 15, 15, 30),
  to = c(15, 15, 30, 50, 50, 70)
)

DDLS <- generateBulkCellMatrix(
  object = DDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  prob.design = probMatrixValid,
  num.bulk.samples = 50,
  verbose = TRUE
)

# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
DDLS <- trainDigitalDLSorterModel(
  object = DDLS,
  on.the.fly = TRUE,
  batch.size = 15,
  num.epochs = 5
)

# evaluation using test data
DDLS <- calculateEvalMetrics(
  object = DDLS
)

# representation, for more examples, see the vignettes
distErrorPlot(
  object = DDLS,
  error = "AbsErr",
  facet.by = "CellType",
  color.by = "nCellTypes",
  error.label = TRUE
)
distErrorPlot(
  object = DDLS,
  error = "AbsErr",
  x.by = "CellType",
  facet.by = NULL,
  filter.sc = FALSE,
  color.by = "CellType",
  error.label = TRUE
)

## End(Not run)
estimateZinbwaveParams

Estimate the 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 optional, only is needed if the size of you dataset is too small or there are underrepresented cell types in order to train the Deep Neural Network model in a more balanced way. 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  DigitalDLSorter 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).
### estimateZinbwaveParams

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subset.cells</td>
<td>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.</td>
</tr>
<tr>
<td>proportional</td>
<td>If TRUE, the original cell type proportions in the subset of cells generated by <code>subset.cells</code> 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).</td>
</tr>
<tr>
<td>set.type</td>
<td>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.</td>
</tr>
<tr>
<td>threads</td>
<td>Number of threads used for estimation (1 by default). To set up the parallel environment, the <code>BiocParallel</code> package must be installed.</td>
</tr>
<tr>
<td>verbose</td>
<td>Show informative messages during the execution (TRUE by default).</td>
</tr>
</tbody>
</table>

**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, `digitalDLSorter` 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 `DigitalDLSorter` 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-Se'q data.

**References**


**See Also**

`simSCProfiles`
features

 Examples

```r
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))
  )
)
DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)
DDLS <- estimateZinbwaveParams(
  object = DDLS,
  cell.type.column = "Cell_Type",
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID",
  subset.cells = 2,
  verbose = TRUE
)
```

features (object)

Get and set features slot in a DigitalDLSorterDNN object

Description

Get and set features slot in a DigitalDLSorterDNN object

Usage

```r
features(object)
features(object) <- value
```

Arguments

<table>
<thead>
<tr>
<th>argument</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>DigitalDLSorterDNN object.</td>
</tr>
<tr>
<td>value</td>
<td>Vector with features (genes) considered by the Deep Neural Network model.</td>
</tr>
</tbody>
</table>
**generateBulkCellMatrix**

*Generate training and test cell composition matrices*

**Description**

Generate training and test cell composition matrices for the simulation of pseudo-bulk RNA-Seq samples with known cell composition using single-cell expression profiles. The resulting `ProbMatrixCellTypes` object contains a matrix that determines the proportion of the different cell types that will compose the simulated pseudo-bulk samples. In addition, this object also contains other information relevant for the process. This function does not simulate pseudo-bulk samples, this task is performed by the `simBulkProfiles` or `trainDigitalDLSorterModel` functions (see Documentation).

**Usage**

```r
generateBulkCellMatrix(
  object,
  cell.ID.column,
  cell.type.column,
  prob.design,
  num.bulk.samples,
  n.cells = 100,
  train.freq.cells = 2/3,
  train.freq.bulk = 2/3,
  proportions.train = c(10, 5, 20, 15, 35, 15),
  proportions.test = c(10, 5, 20, 15, 35, 15),
  prob.zero = c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5),
  balanced.type.cells = FALSE,
  verbose = TRUE
)
```

**Arguments**

- **object** `DigitalDLSorter` object with `single.cell.real` slot and, optionally, with `single.cell.simul` slot.
- **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.
- **prob.design** Data frame with the expected frequency ranges for each cell type present in the experiment. This information can be estimated from literature or from the single-cell experiment itself. This data frame must be constructed by three columns with specific headings (see examples):
  - A cell type column with the same name of the cell type column in cells metadata (cell.type.column). If the name of the column is not the same,
the function will return an error. All cell types must appear in the cells metadata.

- A second column called 'from' with the start frequency for each cell type.
- A third column called 'to' with the ending frequency for each cell type.

num.bulk.samples
Number of bulk RNA-Seq sample proportions (and thus simulated bulk RNA-Seq samples) to be generated taking into account training and test data. We recommend setting this value according to the number of single-cell profiles available in DigitalDSorter object avoiding an excessive re-sampling, but generating a large number of samples for better training.

n.cells
Number of cells that will be aggregated in order to simulate one bulk RNA-Seq sample (100 by default).

train.freq.cells
Proportion of cells used to simulate training pseudo-bulk samples (2/3 by default).

train.freq.bulk
Proportion of bulk RNA-Seq samples to the total number (num.bulk.samples) used for the training set (2/3 by default).

proportions.train
Vector of six integers that determines the proportions of bulk samples generated by the different methods (see Details and Torroja and Sanchez-Cabo, 2019. for more information). This vector represents proportions, so its entries must add up 100. By default, a majority of random samples will be generated without using predefined ranges.

proportions.test
proportions.train for test samples.

prob.zero
Probability of producing cell type proportions equal to zero. It is a vector of six elements corresponding to the six methods of producing cell type proportions (see proportions.train for more details).

balanced.type.cells
Boolean indicating whether the training and test cells will be split in a balanced way considering the cell types (FALSE by default).

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

Details
First, the available single-cell profiles are split into training and test subsets (2/3 for training and 1/3 for test by default (see train.freq.cells)) to avoid falsifying the results during model evaluation. Next, num.bulk.samples bulk samples proportions are built and the single-cell profiles to be used to simulate each pseudo-bulk RNA-Seq sample are set, being 100 cells per bulk sample by default (see n.cells argument). The proportions of training and test pseudo-bulk samples are set by train.freq.bulk (2/3 for training and 1/3 for testing by default). Finally, in order to avoid biases due to the composition of the pseudo-bulk RNA-Seq samples, cell type proportions \( w_1, \ldots, w_k \), where \( k \) is the number of cell types available in single-cell profiles) are randomly generated by using six different approaches:
1. Cell proportions are randomly sampled from a truncated uniform distribution with predefined limits according to a priori knowledge of the abundance of each cell type (see prob.design argument). This information can be inferred from the single-cell experiment itself or from the literature.

2. A second set is generated by randomly permuting cell type labels from a distribution generated by the previous method.

3. Cell proportions are randomly sampled as by method 1 without replacement.

4. Using the last method for generating proportions, cell types labels are randomly sampled.

5. Cell proportions are randomly sampled from a Dirichlet distribution.

6. Pseudo-bulk RNA-Seq samples composed of the same cell type are generated in order to provide 'pure' pseudo-bulk samples.

If you want to inspect the distribution of cell type proportions generated by each method during the process, they can be visualized by the showProbPlot function (see Documentation).

Value

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

References


See Also

simBulkProfiles ProbMatrixCellTypes

Examples

set.seed(123)  # reproducibility
# simulated data
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))
  )

getProbMatrix

### Description

Getter function for the cell composition matrix. This function allows to access to the cell composition matrix of simulated training or test pseudo-bulk RNA-Seq data.

### Usage

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

### Arguments

- **object** `DigitalDLSorter` object with `prob.cell.types` slot.
- **type.data** Subset of data to be shown: train or test.

### Value

A matrix object with the desired cell proportion matrix.

### See Also

- `generateBulkCellMatrix`
installTFpython

Install Python dependencies for digitalDLSorteR

Description

This is a helper function to install Python dependencies needed: a Python interpreter with TensorFlow Python library and its dependencies. It is performed using the reticulate package and the installer of the tensorflow R package. The available options are virtual or conda environments. The new environment is called digitaldlsorter-env. In any case, this installation can be manually done as it is explained in https://diegommcc.github.io/digitalDLSorteR/articles/kerasIssues.html, but we recommend using this function.

Usage

installTFpython(conda = "auto", install.conda = FALSE, miniconda.path = NULL)

Arguments

condaphPath to a conda executable. Use "auto" (by default) allows reticulate to automatically find an appropriate conda binary.
install.condaBoolean indicating if install miniconda automatically using reticulate. If TRUE, conda argument is ignored. FALSE by default.
miniconda.pathIf 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 make easier the installation of the requirements needed to use digitalDLSorteR. It will automatically install Miniconda (if wanted, see Parameters) and create an environment called 'digitaldlsorter-env'. If you want to use other python/conda environment, see ?tensorflow::use_condaenv and/or the vignettes.

Value

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

Examples

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

# End(Not run)
listToDDLS

Transform DigitalDLSorter-like list into an actual DigitalDLSorterDNN object

Description
Transform DigitalDLSorter-like list into an actual DigitalDLSorter object. This function allows to generate the examples and the vignettes of digitalDLSorter package as these are the data used. These data are stored in the digitalDLSorterData package.

Usage
listToDDLS(listTo)

Arguments
listTo A list in which each element must correspond to each slot of an DigitalDLSorter object. The names must be the same as the slot names.

Value
DigitalDLSorter object the data provided in the original list.

See Also
listToDDLSDNN

listToDDLSDNN

Transform DigitalDLSorterDNN-like list into an actual DigitalDLSorterDNN object

Description
Transform DigitalDLSorterDNN-like list into an actual DigitalDLSorterDNN object. This function allows to use pre-trained models in the digitalDLSorter package. These models are stored in the digitalDLSorterModels package.

Usage
listToDDLSDNN(listTo)

Arguments
listTo A list in which each element must correspond to each slot of an DigitalDLSorterDNN object. The names must be the same as the slot names.
loadDeconvData

Value

DigitalDLSorterDNN object with the data provided in the original list.

See Also

listToDDLs

loadDeconvData

Load data to be deconvoluted into a DigitalDLSorter object

Description

Load data to be deconvoluted. Data can be provided from a file path of a tabulated text file (tsv and tsv.gz formats are accepted) or a SummarizedExperiment object.

Usage

loadDeconvData(object, data, name.data = NULL)

## S4 method for signature 'DigitalDLSorter,character'
loadDeconvData(object, data, name.data = NULL)

## S4 method for signature 'DigitalDLSorter,SummarizedExperiment'
loadDeconvData(object, data, name.data = NULL)

Arguments

object       DigitalDLSorter object with trained.model slot.
data         File path where the data is stored or a SummarizedExperiment object.
name.data    Name under which the data is stored in the DigitalDLSorter object. When data is a file path and name.data is not provided, the base name of file will be used.

Value

A DigitalDLSorter object with deconv.data slot with the new bulk-RNA-Seq samples loaded.

See Also

trainDigitalDLSorterModel deconvDigitalDLSorterObj
Create a DigitalDLSorter object from single-cell RNA-seq data

Description

Create a DigitalDLSorter object from single-cell RNA-seq data from files (formats allowed: tsv, tsv.gz, mtx (sparse matrix) and hdf5) or a SingleCellExperiment object. The data will be stored in single.cell.real slot. The data provided should 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 the data is 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 the data is provided as a SingleCellExperiment object, it must contain single-cell counts in the assay slot, cells metadata in the colData slot and genes metadata in the rowData slot. The data must be provided without any transformation (e.g. log-transformation) and raw counts are preferred.

Usage

loadSCProfiles(
  single.cell.data,
  cell.ID.column,
  gene.ID.column,
  name.dataset.h5,
  min.counts = 0,
  min.cells = 0,
  file.backend = NULL,
  name.dataset.backend = NULL,
  compression.level = NULL,
  chunk.dims = NULL,
  block.processing = FALSE,
  verbose = TRUE,
  project = "DigitalDLSorterProject"
)

Arguments

single.cell.data

If data is provided from files, single.cell.real must be a vector of three elements: single-cell counts, cells metadata and genes metadata. If data is 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.
cell.ID.column  Name or number of the column in the cells metadata corresponding to cell names in expression matrix.
gene.ID.column  Name or number of the column in the genes metadata corresponding to the names used for features/genes.
name.dataset.h5  Name of the data set if HDF5 file is provided.
min.counts  Minimum gene counts to filter (0 by default).
min.cells  Minimum of cells with more than min.counts (0 by default).
file.backend  Valid file path where to store the loaded data as HDF5 file. If provided, data is stored in HDF5 files as back-end using 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.
name.dataset.backend  Name of the dataset of the 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.
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 DigitalDLSorter 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.
block.processing  Boolean indicating whether data should be treated as blocks (only if data is provided as HDF5 file). FALSE by default. Note that using this functionality is suitable for cases where is not possible to load the 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 DigitalDLSorter object.

Details

This data can be used to simulate new single-cell profiles using the ZINB-WaVE framework with the estimateZinbwaveParams function. In this way, it is possible to increase the signal of cell types that are underrepresented in the original dataset. If this step is not necessary, these profiles will be used directly to simulate pseudo-bulk RNA-seq samples with known cell composition.

Value

A DigitalDLSorter object with the single-cell RNA-seq data provided loaded into the single.cell.real slot as a SingleCellExperiment object.
loadTrainedModelFromH5

Load from an HDF5 file a trained Deep Neural Network model into a DigitalDLSorter object

Description

Load from an HDF5 file a trained Deep Neural Network model into a DigitalDLSorter object. Note that HDF5 file must be a valid trained model (keras object).

Usage

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

- **object**: `DigitalDLSorter` 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 `DigitalDLSorterDNN` object will be formed, but will not contain other slots (FALSE by default).

Value

`DigitalDLSorter` object with trained.model slot with the new keras DNN model incorporated.

See Also

`trainDigitalDLSorterModel`, `deconvDigitalDLSorterObj`, `saveTrainedModelAsH5`

---

**method**

*Get and set method slot in a `ProbMatrixCellTypes` object*

**Description**

Get and set method slot in a `ProbMatrixCellTypes` object

**Usage**

```r
method(object)

method(object) <- value
```

**Arguments**

- **object**: `ProbMatrixCellTypes` object.
- **value**: Vector with names of cells present in the object.

---

**model**

*Get and set model slot in a `DigitalDLSorterDNN` object*

**Description**

Get and set model slot in a `DigitalDLSorterDNN` object

**Usage**

```r
model(object)

model(object) <- value
```
plots

Arguments

| object   | DigitalDLSorterDNN object. |
| value    | keras.engine.sequential.Sequential object with a trained Deep Neural Network model. |

Description

Get and set plots slot in a ProbMatrixCellTypes object

Usage

```r
plots(object)
plots(object) <- value
```

Arguments

| object   | ProbMatrixCellTypes object. |
| value    | List of lists with plots showing the distribution of the cell proportions generated by each method during the process. |

plotTrainingHistory

Plot training history of a trained DigitalDLSorter Deep Neural Network model

Description

Plot training history of a trained DigitalDLSorter Deep Neural Network model.

Usage

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

Arguments

| object   | DigitalDLSorter object with trained.model slot. |
| title    | Title of plot. |
| metrics  | Metrics to be plotted. If NULL (by default), all metrics available in the DigitalDLSorterDNN object will be plotted. |
Description
Prepare a DigitalDLSorter object that has a DigitalDLSorterDNN object with a trained DNN model. Keras models cannot be stored natively as R objects (e.g., RData or RDS files). By saving the structure as a JSON-like character object and the weights as a list, it is possible to retrieve the model and make predictions. Note: with this option, the state of the optimizer is not saved, only the architecture and weights.

Usage
preparingToSave(object)

Arguments
object DigitalDLSorter object with the trained.data slot.

Details
It is possible to save the entire model as an HDF5 file with the saveTrainedModelAsH5 function and to load it into a DigitalDLSorter object with the loadTrainedModelFromH5 function.

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

Value
A DigitalDLSorter or DigitalDLSorterDNN object with its trained keras model transformed from a keras.engine.sequential.Sequential class into a list with the architecture as a JSON-like character object and the weights as a list.

See Also
saveRDS saveTrainedModelAsH5

preparingToSave

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

See Also
trainDigitalDLsorterModel deconvDigitalDLsorterObj
**prob.cell.types**  Get and set prob.cell.types slot in a DigitalDLSorter object

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

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>DigitalDLSorter object.</td>
</tr>
<tr>
<td>type.data</td>
<td>Element of the list. Can be 'train', 'test' or 'both' (the last by default).</td>
</tr>
<tr>
<td>value</td>
<td>List with two elements, train and test, each one with a ProbMatrixCellTypes object.</td>
</tr>
</tbody>
</table>

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

**Description**
Get and set prob.matrix slot in a ProbMatrixCellTypes object

**Usage**
prob.matrix(object)
prob.matrix(object) <- value

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>ProbMatrixCellTypes object.</td>
</tr>
<tr>
<td>value</td>
<td>Matrix with cell types as columns and samples as rows.</td>
</tr>
</tbody>
</table>
The Class `ProbMatrixCellTypes`

Description

The `ProbMatrixCellTypes` class is a data storage class that contains information related to the cell composition matrices used for the simulation of pseudo-bulk RNA-Seq samples. The matrix is stored in the `prob.matrix` slot. The other of slots contain additional information generated during the process and required in subsequent steps.

Details

As described in Torroja and Sánchez-Cabo, 2019, the proportions are constructed using six different methods in order to avoid biases due to the composition of the simulated bulk samples. In `plots` slot, plots are stored that visually represent the distribution of these probabilities in order to provide a way to monitor the different sets of samples generated. These plots can be shown using the `showProbPlot` function (see `?showProbPlot` for more details).

Slots

- `prob.matrix` Matrix of cell proportions generated for the simulation of bulk samples. Rows correspond to the bulk samples to be generated \((i)\), columns are the cell types present in the provided single-cell data \((j)\) and each entry is the proportion of \(j\) cell type in \(i\) sample.
- `cell.names` Matrix containing the names of the cells that will make up each simulated pseudo-bulk sample.
- `set.list` List of cells sorted according to the cell type they belong to.
- `set` Vector containing the cell names present in the object.
- `plots` List of lists with plots showing the distribution of the cell proportions generated by each method during the process. In each list, boxplot, violinplot, linesplot or ncelltypes can be found. Please see `showProbPlot` for more details.
- `type.data` Character with the type of data contained: training or test.

References

**project**

Get and set project slot in a DigitalDLSorter object

**Description**

Get and set project slot in a DigitalDLSorter object

**Usage**

```r
project(object)
```

```r
project(object) <- value
```

**Arguments**

- `object` : DigitalDLSorter object.
- `value` : Character indicating the name of the project.

---

**saveRDS**

Save DigitalDLSorter objects as RDS files

**Description**

Save DigitalDLSorter and DigitalDLSorterDNN objects as RDS files. Keras models cannot be stored natively as R objects (e.g. RData or RDS files). By saving the structure as a JSON-like character object and the weights as a list, it is possible to retrieve the model and make 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 the architecture and weights. It is possible to save the entire model as an HDF5 file with the saveTrainedModelAsH5 function and to load it into a DigitalDLSorter object with the loadTrainedModelFromH5 function. See documentation for details.

**Usage**

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

```r
## S4 method for signature 'DigitalDLSorterDNN'
saveRDS(
  object,
```
saveTrainedModelAsH5

```r
saveTrainedModelAsH5 = function(file, ascii = FALSE, version = NULL, compress = TRUE, refhook = NULL)

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

### Arguments

- **object**: `DigitalDLSorter` or `DigitalDLSorterDNN` 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 `DigitalDLSorter` object as an RDS file on disk.

### See Also

- `DigitalDLSorter`
- `saveRDS`
- `saveTrainedModelAsH5`

### Description

Save a trained `DigitalDLSorter` Deep Neural Network model to disk as an HDF5 file. Note that this function does not save the `DigitalDLSorterDNN` object, but 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  DigitalDLSorter 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

trainDigitalDLSorterModel loadTrainedModelFromH5

set

Get and set set slot in a ProbMatrixCellTypes object

Description

Get and set set slot in a ProbMatrixCellTypes object

Usage

set(object)
set(object) <- value

Arguments

object  ProbMatrixCellTypes object.
value   Vector with names of cells present in the object.
**set.list**  
*Get and set set.list slot in a ProbMatrixCellTypes object*

**Description**
Get and set set.list slot in a ProbMatrixCellTypes object

**Usage**

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

**Arguments**

- **object**  
  ProbMatrixCellTypes object.

- **value**  
  List of cells sorted according to the cell type to which they belong.

**showProbPlot**  
*Show distribution plots of the cell proportions generated by generateBulkCellMatrix*

**Description**
Show distribution plots of the cell proportions generated by generateBulkCellMatrix. These frequencies will determine the proportion of different cell types used during the simulation of pseudo-bulk RNA-Seq samples. There are 6 subsets of proportions generated by different approaches that 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'.

**Usage**

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

**Arguments**

- **object**  
  DigitalDLSorter 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 plots are only for diagnostic purposes. This is the reason because they are generated without any parameter introduced by the user.

Value

A ggplot object.

See Also

generateBulkCellMatrix

Examples

# simulating data
set.seed(123) # reproducibility
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))
  )
)
DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)
probMatrix <- data.frame(
  Cell_Type = paste0("CellType", seq(4)),
  from = c(1, 1, 1, 30),
  to = c(15, 15, 50, 70)
)
DDLS <- generateBulkCellMatrix(
  object = DDLS,
  cell.ID.column = "Cell_ID",
  cell.type.column = "Cell_Type",
  prob.design = probMatrix,
  num.bulk.samples = 60
)
lapply(
  X = 1:6, FUN = function(x) {
    showProbPlot(
      x
    )
  })
Description

Simulate training and test pseudo-bulk RNA-Seq profiles using the cell composition matrices generated by the `generateBulkCellMatrix` function. The samples are generated under the assumption that the expression level of the $i$ gene in the $j$ bulk sample is given by the sum of the expression levels of the cell types $X_{ijk}$ that make them up weighted by the proportions of these $k$ cell types in each sample. In practice, as described in Torroja and Sanchez-Cabo, 2019, these profiles are generated by summing a number of cells of different cell types determined by proportions from a matrix of known cell composition. The number of simulated pseudo-bulk RNA-Seq samples and the number of cells composing each sample are determined by `generateBulkCellMatrix` (see Documentation) **Note:** this step can be avoided by using the `on.the.fly` argument in the `trainDigitalDLSorterModel` function. See Documentation for more information.

Usage

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

Arguments

- **object** `DigitalDLSorter` object with `single.cell.real/single.cell.simul` and `prob.cell.types` slots.
- **type.data** Type of data to generate between 'train', 'test' or 'both' (the last by default).
- **pseudobulk.function** Function used to build pseudo-bulk samples. It may be:
• "MeanCPM": single-cell profiles (raw counts) are transformed into CPMs and cross-cell averages are calculated. Then, \( \log_2(CPM + 1) \) is calculated.
• "AddCPM": single-cell profiles (raw counts) are transformed into CPMs and are added up across cells. Then, log-CPMs are calculated.
• "AddRawCount": single-cell profiles (raw counts) are added up across cells. Then, log-CPMs are calculated.

**file.backend**  
Valid file path to store the simulated single-cell expression profiles as an HDF5 file (NULL by default). If provided, the data is 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.

**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 pseudo-bulk 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).

**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 **DigitalDLSorter** 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 the simulation of pseudo-bulk samples (1 by default). Set according to computational resources and avoid it if block.size will be used.

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

**Details**

**digitalDLSorter** allows the use of HDF5 files as back-end to store the resulting data using the **DelayedArray** and **HDF5Array** packages. This functionality allows to work without keeping the data loaded into RAM, which could be of vital importance 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. The data will be accessible from R without being loaded into memory. This option slightly slows down execution times, as subsequent transformations of the data will be done in blocks rather than using all the data.
We recommend this option according to the computational resources available and the number of pseudo-bulk samples to be generated.

Note that if you use the file.backend argument with block.processing = FALSE, all pseudo-bulk profiles will be simulated in one step and, therefore, loaded into RAM. Then, the data will be written to an HDF5 file. To avoid the RAM collapse, pseudo-bulk profiles can be simulated and written to HDF5 files in blocks of block.size size by setting block.processing = TRUE.

It is possible to avoid this step by using the on.the.fly argument in the trainDigitalDLSorterModel function. In this way, data is generated 'on the fly' during the neural network training. For more details, see ?trainDigitalDLSorterModel.

Value

A DigitalDLSorter object with bulk.simul slot containing a list with one or two entries (depending on selected type.data argument): 'train' and 'test'. Each entry contains a SummarizedExperiment object with simulated bulk samples in the assay slot, sample names in the colData slot and feature names in the rowData slot.

References


See Also

generateBulkCellMatrix ProbMatrixCellTypes trainDigitalDLSorterModel

Examples

set.seed(123) # reproducibility
# simulated data
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))
  )
)
DDLS <- loadSCProfiles(
single.cell.data = sce,
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**

```r
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)
```
simSCProfiles

**Arguments**

- **object** \(\text{DigitalDLSorter}\) 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 \text{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 (\text{NULL} by default). If provided, the data is stored in HDF5 files used as back-end by using the \text{DelayedArray}, \text{HDF5Array} and \text{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 \text{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 \text{?getHDF5DumpCompressionLevel} from the \text{HDF5Array} package for more information.
- **block.processing** Boolean indicating whether the data should be simulated in blocks (only if file.backend is used, \text{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 = \text{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 \text{NULL}, the default value is a vector of two items: the number of genes considered by the \text{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 (\text{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 $\mu$ and $\theta$ estimated parameters and inserting dropouts by sampling from a binomial distribution with probability $p_i$. 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 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 `DigitalDLSorter` object with `single.cell.simul` slot containing a `SingleCellExperiment` object with the simulated single-cell expression profiles.

**References**


**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))
)

DDLS <- loadSCProfiles(
  single.cell.data = sce,
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID"
)

DDLS <- estimateZinbwaveParams(
  object = DDLS,
  cell.type.column = "Cell_Type",
  cell.ID.column = "Cell_ID",
  gene.ID.column = "Gene_ID",
  subset.cells = 4,
  verbose = FALSE
)

DDLS <- simSCProfiles(
  object = DDLS,
  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 DigitalDLSorter object

Description

Get and set single.cell.real slot in a DigitalDLSorter object

Usage

single.cell.real(object)

single.cell.real(object) <- value

Arguments

object  DigitalDLSorter object.
value  SingleCellExperiment object with real single-cell profiles.
**single.cell.simul**

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

**Description**

Get and set single.cell.simul slot in a DigitalDLSorter object

**Usage**

```r
single.cell.simul(object)
```

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

**Arguments**

- **object**  
  *DigitalDLSorter* object.

- **value**  
  *SingleCellExperiment* object with simulated single-cell profiles.

---

**test.deconv.metrics**

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

**Description**

Get and set test.deconv.metrics slot in a DigitalDLSorterDNN object

**Usage**

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

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

**Arguments**

- **object**  
  *DigitalDLSorterDNN* object.

- **metrics**  
  Metrics to show ('All' by default)

- **value**  
  List with evaluation metrics used to assess the performance of the model on each sample of test data.
test.metrics  
*Get and set test.metrics slot in a DigitalDLSorterDNN object*

**Description**

Get and set test.metrics slot in a DigitalDLSorterDNN object

**Usage**

```r
test.metrics(object)

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

**Arguments**

- `object`  
  DigitalDLSorterDNN object.
- `value`  
  List object with the resulting metrics after prediction on test data with the Deep Neural Network model.

---

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

**Description**

Get and set test.pred slot in a DigitalDLSorterDNN object

**Usage**

```r
test.pred(object)

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

**Arguments**

- `object`  
  DigitalDLSorterDNN object.
- `value`  
  Matrix object with the prediction results on test data.
trainDigitalDLSorterModel

*Train Deep Neural Network model*

**Description**

Train a Deep Neural Network model using the training data from `DigitalDLSorter` object. In addition, the trained model is evaluated with test data and prediction results are obtained to determine its performance (see `calculateEvalMetrics`). Training and evaluation can be performed using simulated profiles stored in the `DigitalDLSorter` object or 'on the fly' by simulating the pseudo-bulk profiles at the same time as the training/evaluation is performed (see Details).

**Usage**

```r
trainDigitalDLSorterModel(
  object,
  combine = "both",
  batch.size = 64,
  num.epochs = 10,
  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"),
  scaling = "standarize",
  custom.model = NULL,
  shuffle = FALSE,
  on.the.fly = FALSE,
  pseudobulk.function = "MeanCPM",
  threads = 1,
  view.metrics.plot = TRUE,
  verbose = TRUE
)
```

**Arguments**

- **object** `DigitalDLSorter` object with `single.cell.real/single.cell.simul/prob.cell.matrix` and `bulk.simul` slots.
- **combine** Type of profiles to be used for training. Can be 'both', 'single-cell' or 'bulk' ('both' by default). For test data, both types of profiles will be used.
- **batch.size** Number of samples per gradient update. If not specified, batch.size will default to 64.
- **num.epochs** Number of epochs to train the model (10 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 num.hidden.layers argument.

activation.fun
Activation function to use ('relu' by default). See the keras documentation to know available activation functions.

dropout.rate
Float between 0 and 1 indicating the fraction of the input neurons to drop in layer dropouts (0.25 by default). By default, digitalDLSorterR 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.

scaling
How to scale data before training. It may be: "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).

custom.model
Allows to use a custom neural network. It must be a keras.engine.sequential.Sequential 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). Note that if bulk.simul is not NULL, the data already has been shuffled and shuffle will be ignored.

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

pseudobulk.function
Function used to build pseudo-bulk samples. It may be:

- "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.
- "AddRawCount": single-cell profiles (raw counts) are added up across cells. Then, log-CPMs are calculated.

threads
Number of threads used during simulation of pseudo-bulk samples if on.the.fly = TRUE (1 by default).

view.metrics.plot
Boolean indicating whether to show plots of loss and metrics progression during training (TRUE by default). keras for R allows to see the progression of the model 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

**Keras/Tensorflow environment**

All Deep Learning related steps in the `digitalDLSorteR` package are performed by using the `keras` package, an API in R for `keras` in Python available on CRAN. We recommend using the installation guide available at [https://tensorflow.rstudio.com/](https://tensorflow.rstudio.com/) in order to set a more customized configuration.

**Simulation of bulk RNA-Seq profiles 'on the fly'**

`trainDigitalDLsorterModel` allows to avoid storing bulk RNA-Seq profiles by using `on.the.fly` argument. This functionality aims to avoid execution times and memory usage of the `simBulkProfiles` function, as the simulated pseudo-bulk profiles are built in each batch during training/evaluation.

**Neural network architecture**

By default, `trainDigitalDLsorterModel` implements the architecture selected in Torroja and Sánchez-Cabo, 2019. However, as the default architecture may not produce good results depending on the dataset, it is possible to change its parameters by using the corresponding argument: 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 in 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 `DigitalDLSorter` object with `trained.model` slot containing a `DigitalDLSorterDNN` object. For more information about the structure of this class, see `?DigitalDLSorterDNN`.

**References**


**See Also**

`plotTrainingHistory` `deconvDigitalDLsorter` `deconvDigitalDLsorterObj`

**Examples**

```r
## Not run:
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))
)
)
DDLS <- loadSCProfiles(
    single.cell.data = sce,
    cell.ID.column = "Cell_ID",
    gene.ID.column = "Gene_ID"
)
probMatrixValid <- data.frame(
    Cell_Type = paste0("CellType", seq(2)),
    from = c(1, 30),
    to = c(15, 70)
)
DDLS <- generateBulkCellMatrix(
    object = DDLS,
    cell.ID.column = "Cell_ID",
    cell.type.column = "Cell_Type",
    prob.design = probMatrixValid,
    num.bulk.samples = 30,
    verbose = TRUE
)
# training of DDLS model
tensorflow::tf$compat$v1$disable_eager_execution()
DDLS <- trainDigitalDLSorterModel(
    object = DDLS,
    on.the.fly = TRUE,
    batch.size = 12,
    num.epochs = 5
)
## End(Not run)

---

**trained.model**

Get and set trained.model slot in a DigitalDLSorter object

### Description

Get and set trained.model slot in a DigitalDLSorter object

### Usage

```r
trained.model(object)
trained.model(object) <- value
```
training.history

Arguments

object DigitalDLSorter object.

value DigitalDLSorterDNN object.

Description

Get and set training.history slot in a DigitalDLSorterDNN object

Usage

training.history(object)

training.history(object) <- value

Arguments

object DigitalDLSorterDNN object.

value keras_training_history object with the training history of the Deep Neural Network model

zinb.params

Description

Get and set zinb.params slot in a DigitalDLSorter object

Usage

zinb.params(object)

zinb.params(object) <- value

Arguments

object DigitalDLSorter object.

value ZinbParametersModel object with a valid ZinbModel object.
The Class ZinbParametersModel

Description

The ZinbParametersModel class is a wrapper class of the ZinbModel class from zinbwave package.

Details

This is a wrapper class of the ZinbModel class. It consists of only one slot (zinbwave.mode) that contains the ZinbModel object.

Slots

zinbwave.model A valid ZinbModel object.

References


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.
Index

barErrorPlot, 3, 8, 11, 14, 27
barPlotCellTypes, 5
barPlotCellTypes, ANY-method
   (barPlotCellTypes), 5
barPlotCellTypes, DigitalDLSorter-method
   (barPlotCellTypes), 5
blandAltmanLehPlot, 4, 7, 11, 14, 27
bulk.simul, 10
bulk.simul, DigitalDLSorter-method
   (bulk.simul), 10
bulk.simul<<-(bulk.simul), 10
bulk.simul<<-, DigitalDLSorter-method
   (bulk.simul), 10
calculateEvalMetrics, 4, 8, 10, 14, 25, 27, 61
cell.names, 12
cell.names, ProbMatrixCellTypes-method
   (cell.names), 12
cell.names<<-(cell.names), 12
cell.names<<-, ProbMatrixCellTypes-method
   (cell.names), 12
cell.types, 12
cell.types, DigitalDLSorterDNN-method
   (cell.types), 12
cell.types<<-(cell.types), 12
cell.types<<-, DigitalDLSorterDNN-method
   (cell.types), 12
corrExpPredPlot, 4, 8, 11, 13, 27
deconv.data, 16, 24
deconv.data, DigitalDLSorter-method
   (deconv.data), 16
deconv.data<<-(deconv.data), 16
deconv.data<<-, DigitalDLSorter-method
   (deconv.data), 16
deconv.results, 16
deconv.results, DigitalDLSorter-method
   (deconv.results), 16
deconv.results<<-(deconv.results), 16
deconv.results<<-, DigitalDLSorter-method
   (deconv.results), 16
deconv.results<<-, DigitalDLSorterDNN-method
   (deconv.results), 16
deconv.results<<-, DigitalDLSorterDNN-method
   (deconv.results), 16
deconv.results<<-, DigitalDLSorterObj, 7, 18, 20, 24,
loadDeconvData, DigitalDLSorter, character-method
(loadDeconvData), 38
loadDeconvData, DigitalDLSorter, SummarizedExperiment-method
(loadDeconvData), 38
loadSCPProfiles, 18, 24, 39
loadTrainedModelFromH5, 41, 44, 47, 49

method, 42
method, ProbMatrixCellTypes-method
(method), 42
method <- (method), 42
model, 42
model, DigitalDLSorterDNN-method
(model), 42
model <- (model), 42
model <-, DigitalDLSorterDNN-method
(model), 42

plots, 43
plots, ProbMatrixCellTypes-method
(plots), 43
plots <- (plots), 43
plotTrainingHistory, 43, 63
preparingToSave, 25, 44, 48
prob.cell.types, 45
prob.cell.types, DigitalDLSorterDNN-method
(prob.cell.types), 45
prob.cell.types <- (prob.cell.types), 45
prob.cell.types <-, DigitalDLSorterDNN-method
(prob.cell.types), 45
prob.matrix, 45
prob.matrix, ProbMatrixCellTypes-method
(prob.matrix), 45
prob.matrix <- (prob.matrix), 45
prob.matrix <-, ProbMatrixCellTypes-method
(prob.matrix), 45
ProbMatrixCellTypes, 12, 24, 32, 34, 42, 43, 45, 49, 50, 54
ProbMatrixCellTypes
(ProbMatrixCellTypes-class), 46
ProbMatrixCellTypes-class, 46
project, 47
project, DigitalDLSorter-method
(project), 47
project <- (project), 47

save, 48
saveRDS, 44, 47, 48
saveRDS, DigitalDLSorterDNN-method
(saveRDS), 47
saveRDS, saveRDS-method (saveRDS), 47
saveTrainedModelAsH5, 25, 42, 44, 47, 48, 48
set, 49
set, ProbMatrixCellTypes-method (set), 49
set.list, 50
set.list, ProbMatrixCellTypes-method
(set.list), 50
set.list <- (set.list), 50
set.list <-, ProbMatrixCellTypes-method
(set.list), 50
set <- (set), 49
set <-, ProbMatrixCellTypes-method (set), 49
showProbPlot, 34, 46, 50
simBulkProfiles, 32, 34, 52
simSCPProfiles, 29, 30, 55
single.cell.real, 58
single.cell.real, DigitalDLSorterDNN-method
(single.cell.real), 58
single.cell.real <- (single.cell.real), 58
single.cell.real <-, DigitalDLSorterDNN-method
(single.cell.real), 58
single.cell.simul, 59
single.cell.simul, DigitalDLSorterDNN-method
(single.cell.simul), 59
single.cell.simul <- (single.cell.simul), 59
single.cell.simul <-, DigitalDLSorterDNN-method
(single.cell.simul), 59
SingleCellExperiment, 24, 39, 40, 57–59
SummarizedExperiment, 10, 24, 38, 54

test.deconv.metrics, 59
test.deconv.metrics, DigitalDLSorterDNN-method
(test.deconv.metrics), 59
test.deconv.metrics <- (test.deconv.metrics), 59
test.deconv.metrics <-, DigitalDLSorterDNN-method
(test.deconv.metrics), 59
test.metrics, 60
test.metrics,DigitalDLSorterDNN-method  
(test.test.metrics), 60

test.metrics<-(test.metrics), 60

testmetrics<-,DigitalDLSorterDNN-method  
(test.test.metrics), 60

test.pred, 60

test.pred,DigitalDLSorterDNN-method  
(test.test.pred), 60

test.pred<-(test.test.pred), 60

test.pred<-,DigitalDLSorterDNN-method  
(test.test.pred), 60

trainDigitalDLSorterModel, 21, 32, 38, 42, 44, 49, 52, 54, 61, 63

trained.model, 20, 64

tained.model,DigitalDLSorter-method  
(trained.model), 64

trained.model<-(trained.model), 64

trained.model<-,DigitalDLSorter-method  
(trained.model), 64

training.history, 65

training.history,DigitalDLSorterDNN-method  
(training.history), 65

training.history<-(training.history), 65

training.history<-,DigitalDLSorterDNN-method  
(training.history), 65

zinb.params, 65

zinb.params,DigitalDLSorter-method  
(zinb.params), 65

zinb.params<-(zinb.params), 65

zinb.params<-,DigitalDLSorter-method  
(zinb.params), 65

zinbf, 30

ZinbModel, 24, 65, 66

ZinbParametersModel, 30, 65, 66

ZinbParametersModel  
(ZinbParametersModel-class), 66

ZinbParametersModel-class, 66

zinbwave.model, 66

zinbwave.model,ZinbParametersModel-method  
(zinbwave.model), 66

zinbwave.model<-(zinbwave.model), 66

zinbwave.model<-,ZinbParametersModel-method  
(zinbwave.model), 66