Package ‘cellpypes’

January 27, 2024

Title Cell Type Pipes for Single-Cell RNA Sequencing Data

Version 0.3.0

Description Annotate single-cell RNA sequencing data manually based on
marker gene thresholds.
Find cell type rules (gene+threshold) through exploration,
use the popular piping operator '%>%' to reconstruct complex
cell type hierarchies.
'cellpypes' models technical noise to find positive and negative cells for
a given expression threshold and returns cell type labels or pseudobulks.
Cite this package as Frauhammer (2022) <doi:10.5281/zenodo.6555728> and
visit <https://github.com/FelixTheStudent/cellpypes> for tutorials and newest
features.

URL https://github.com/FelixTheStudent/cellpypes

BugReports https://github.com/FelixTheStudent/cellpypes/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.2.0

Suggests testthat (>= 3.0.0), knitr, rmarkdown, Seurat, DESeq2,
RcppAnnoy, tibble, SeuratObject

Config/testthat/edition 3

Imports scUtils, ggplot2, Matrix, rlang, viridis, cowplot, dplyr,
scales, methods, scattermore

Depends R (>= 2.10)

NeedsCompilation no

Author Felix Frauhammer [aut, cre]

Maintainer Felix Frauhammer <felixfrauhammer@gmail.com>

Repository CRAN

Date/Publication 2024-01-27 07:30:07 UTC

R topics documented:

`classify` *Classify cells on previously defined rules*

Description

Classify cells on previously defined rules

Usage

```
classify(
  obj,
  classes = NULL,
  knn_refine = 0,
  replace_overlap_with = "Unassigned",
  return_logical_matrix = FALSE,
  overdispersion = 0.01
)
```

Arguments

`obj`	A cellpypes object, see section **cellpypes Objects** below.
`classes`	Character vector with one or more class names. If NULL (the default), plots finest available cell types (all classes that are not parent of any other class).
`knn_refine`	Numeric between 0 and 1. If 0, do not refine labels obtained from UMI count pooling. If larger than 0 (recommended: 0.1), cellpypes will try to label unassigned cells by majority vote, see section **knn_refine** below.

`replace_overlap_with`
: Character string, by default: `"Unassigned"`. See section **Handling overlap**.

`return_logical_matrix`
: logical. If TRUE, a logical matrix with classes in columns and cells in rows is returned instead of resolving overlaps with `replace_overlap_with`. If a single class is supplied, the matrix has exactly one column and the user can pipe it into "drop" to convert it to a vector.

`overdispersion`
: Defaults to 0.01, only change it if you know what you are doing. If set to 0, the NB simplifies to the Poisson distribution, and larger values give more variance. The 0.01 default value follows the recommendation by Lause, Berens and Kobak (Genome Biology 2021) to use `size=100` in pnbinom for typical data sets.

Value

A factor with cell type labels.

cellpypes Objects

A cellpypes object is a list with four slots:

`raw` (sparse) matrix with genes in rows, cells in columns

`totalUMI` the colSums of obj$raw

`embed` two-dimensional embedding of the cells, provided as data.frame or tibble with two columns and one row per cell.

`neighbors` index matrix with one row per cell and k columns, where k is the number of nearest neighbors (we recommend 15<k<100, e.g. k=50). Here are two ways to get the neighbors index matrix:

- Use `find_knn(featureMatrix)$idx`, where featureMatrix could be principal components, latent variables or normalized genes (features in rows, cells in columns).
- use `as(seurat@graphs[["RNA_nn"]], "dgCMatrix")> .1` to extract the kNN graph computed on RNA. The `> .1` ensures this also works with RNA_snn, wknn/wsnn or any other available graph – check with `names(seurat@graphs)`.

Handling overlap

Overlap denotes all cells for which rules from multiple classes apply, and these cells will be labeled as `Unassigned` by default. If you are in fact interested in where the overlap is, set `return_logical_matrix`=TRUE and inspect the result. Note that it matters whether you call `classify("Tcell")` or `classify(c("Tcell","Bcell")` – any existing overlap between T and B cells is labelled as `Unassigned` in this second call, but not in the first.

Replacing overlap happens only between mutually exclusive labels (such as Tcell and Bcell), but not within a lineage. To make an example, overlap is NOT replaced between child (PD1+Ttox) and parent (Ttox) or any other ancestor (Tcell), but instead the most detailed cell type (PD1+Ttox) is returned.

All of the above is also true for `plot_classes`, as it wraps `classify`.

knn_refine

With knn_refine > 0, cellpypes refines cell type labels with a kNN classifier.

By default, cellpypes only assigns cells to a class if all relevant rules apply. In other words, all marker gene UMI counts in the cell's neighborhood all have to be clearly above/below their threshold. Since UMI counts are sparse (even after neighbor pooling done by cellpypes), this can leave many cells unassigned.

It is reasonable to assume an unassigned cell is of the same cell type as the majority of its nearest neighbors. Therefore, cellpypes implements a kNN classifier to further refine labels obtained by manually thresholding UMI counts. knn_refine = 0.3 means a cell is assigned the class label held by most of its neighbors unless no class gets more than 30 %. If most neighbors are unassigned, the cell will also be set to "Unassigned". Choosing knn_refine = 0.3 gives results reminiscent of clustering (which assigns all cells), while knn_refine = 0.5 leaves cells 'in between' two similar cell types unassigned.

We recommend looking at knn_refine = 0 first as it's faster and more directly tied to marker gene expression. If assigning all cells is desired, we recommend knn_refine = 0.3 or lower, while knn_refine = 0.5 makes cell types more 'crisp' by setting cells 'in between' related subtypes to "Unassigned".

Examples

```
classify(rule(simulated_umis, "Tcell", "CD3E", ">", 1))
```

---

class_to_deseq2 *Create DESeq2 object for a given cell type*

Description

Create a DESeq2 data set ('dds' in the DESeq2 vignette) for the specified class (cell type).

Usage

```
class_to_deseq2(obj, meta_df, class, design = ~condition)
```

Arguments

obj	A cellpypes object, see section **cellpypes Objects** below.
meta_df	Data frame where each column helps to identify a pseudobulk. Typical columns of meta_df are for example patient, treatment and cell type – anything that uniquely identifies a replicate / batch / 10x run. Each row in meta_df corresponds to a single cell in your raw count matrix.
class	The name of cellpypes class for which you want to test for differential expression.
design	A formula based on columns in meta_df. To test differential expression between two groups in meta_df$condition, use formula ~ condition. More complex formulas (e.g. with interactions) are possible, for example ~ genotype + treatment + genotype:treatment.

Value

A DESeq2 object (e.g. dds)

cellpypes Objects

A cellpypes object is a list with four slots:

`raw` (sparse) matrix with genes in rows, cells in columns

`totalUMI` the colSums of obj$raw

`embed` two-dimensional embedding of the cells, provided as data.frame or tibble with two columns and one row per cell.

`neighbors` index matrix with one row per cell and k columns, where k is the number of nearest neighbors (we recommend 15<k<100, e.g. k=50). Here are two ways to get the neighbors index matrix:

- Use `find_knn(featureMatrix)$idx`, where featureMatrix could be principal components, latent variables or normalized genes (features in rows, cells in columns).
- use `as(seurat@graphs[["RNA_nn"]], "dgCMatrix")> .1` to extract the kNN graph computed on RNA. The `> .1` ensures this also works with RNA_snn, wknn/wsnn or any other available graph – check with `names(seurat@graphs)`.

Examples

```
data("simulated_umis")
# Meta data
ncells <- ncol(simulated_umis$raw)
dummy_variable <- function(x) factor(sample(x, ncells, replace=TRUE))
meta_data <- data.frame(patient=dummy_variable(paste0("patient", 1:6)),
                        treatment=dummy_variable(c("control", "treated")))

obj <- rule(simulated_umis, "T", "CD3E",">", 1e-4)
 # > 5 s in CRAN check
dds <- class_to_deseq2(obj, meta_data, "T", ~ treatment)
```

feat *Feature plots: Color gene expression in 2D embeddings*

Description

Highlight gene expression in UMAP embeddings, for example.

Usage

```
feat(obj, features, fast = NULL, verbose = TRUE, ...)
```

Arguments

`obj` A cellpypes object, see section **cellpypes Objects** below.

`features` A vector of genes (features) to colour by.

`fast` Set this to TRUE if you want fast plotting in spite of many cells (using the scattermore package). If NULL (default), cellpypes decides automatically and fast plotting is done for more than 10k cells, if FALSE it always uses `geom_point`.

`verbose` feat ignores gene names not present in your object and warns you about them by default. `verbose`=FALSE will suppress the warning (not recommended in interactive use).

`...` Arguments passed to cowplot's plot_grid function, for example ncol or rel_widths.

Value

A ggplot object (assembled by cowplot).

cellpypes Objects

A cellpypes object is a list with four slots:

`raw` (sparse) matrix with genes in rows, cells in columns

`totalUMI` the colSums of obj$raw

`embed` two-dimensional embedding of the cells, provided as data.frame or tibble with two columns and one row per cell.

`neighbors` index matrix with one row per cell and k columns, where k is the number of nearest neighbors (we recommend 15<k<100, e.g. k=50). Here are two ways to get the neighbors index matrix:

- Use `find_knn(featureMatrix)$idx`, where featureMatrix could be principal components, latent variables or normalized genes (features in rows, cells in columns).
- use `as(seurat@graphs[["RNA_nn"]], "dgCMatrix")> .1` to extract the kNN graph computed on RNA. The `> .1` ensures this also works with RNA_snn, wknn/wsnn or any other available graph – check with `names(seurat@graphs)`.

Examples

```
feat(simulated_umis, "CD3E")
```

find_knn *Find approximate k-nearest neighbors*

Description

Implements RcppAnnoy's approximate nearest neighbor search (much faster than precise neighbors). Random search is made reproducible using `set.seed(seed)`. Hint: If you pass `find_knn`'s output directly to `uwot::umap` via the `nn_method` argument, make sure to set `umap`'s argument `n_sgd_threads` to <=1 to ensure the UMAP embedding is reproducible.

Usage

```
find_knn(featureMatrix, k = 50, n_trees = 50, seed = 42)
```

Arguments

`featureMatrix`	Numeric matrix with features in rows, cells in columns. Rows could be normalized genes or latent dimensions such as principal components.
`k`	Number of neighbors to find.
`n_trees`	RccpAnnoy builds a forest of `n_trees` trees. More trees gives higher precision when querying. Default: 50.
`seed`	Random seed for neighbor search, default: 42.

Value

List with two slots:

- `idx` A NxK matrix (N cells, K neighbors) containing the integer indexes of the approximate nearest neighbors in featureMatrix. Each cell is considered to be its own nearest neighbor, next to K-1 other neighbors.
- `dist` A NxK matrix containing the distances of the nearest neighbors.

Inspired by `uwot::umap`'s return value when setting `ret_nn=TRUE`.

Examples

```
  # Imagine we have 30 cells and 100 features:
 fmat <- matrix(rnorm(3000), ncol=30)
 nn <- find_knn(fmat,k=15)
 # nn$idx has 30 rows and 15 columns.
```

`is_classes` *Check if obj$classes looks as expected. is_class returns FALSE for example in these cases: is_classes(NULL) is_classes(data.frame()) is_classes(data.frame(class=c("T","T"), parent=c("..root..","..root..")))*

Description

Check if obj$classes looks as expected. is_class returns FALSE for example in these cases: is_classes(NULL) is_classes(data.frame()) is_classes(data.frame(class=c("T","T"), parent=c("..root..","..root..")))

Usage

```
is_classes(classes)
```

Arguments

`classes`	The obj$classes you want to check.

Value

logical scalar.

`is_rules` *Check if obj$rules looks as expected.*

Description

Check if obj$rules looks as expected.

Usage

```
is_rules(rules)
```

Arguments

`rules` The obj$rules slot of a cellpypes object.

Value

logical scalar

`knn_refine` *Refine cell type labels with knn classifier*

Description

Assigns the label that most neighbors have, given it is more than `min_knn_prob`. I've found empirically on the MALT data that `min_knn_prob=0.5` gives good results, whether you classify the entire data set or just a single cell type. It simply excludes some of the cells that have more than 2 cell types in their neighborhood and none is much stronger than the others, so this is a reasonable, conservative filtering.

Usage

```
knn_refine(labels, neighbors, min_knn_prob = 0.5)
```

Arguments

`labels` Cell type labels as character or factor.

`neighbors` Neighbor graph, pass `obj$neighbors`.

`min_knn_prob` Value between 0 and 1, defaults to 0.5. If the 'winning label' is below this proportion of kNN that have it, knn_refine will return "Unassigned".

Value

Character vector with refined labels.

`plot_classes` *Call and visualize 'classify' function*

Description

Call and visualize 'classify' function

Usage

```
plot_classes(
  obj,
  classes = NULL,
  knn_refine = 0,
  replace_overlap_with = "Unassigned",
  return_logical_matrix = FALSE,
  fast = NULL,
  point_size = 0.4,
  point_size_legend = 2,
  base_size = 15,
  overdispersion = 0.01
)
```

Arguments

`obj` A cellpypes object, see section **cellpypes Objects** below.

`classes` Character vector with one or more class names. If NULL (the default), plots finest available cell types (all classes that are not parent of any other class).

`knn_refine` Numeric between 0 and 1. If 0, do not refine labels obtained from UMI count pooling. If larger than 0 (recommended: 0.1), cellpypes will try to label unassigned cells by majority vote, see section **knn_refine** below.

`replace_overlap_with` Character string, by default: `"Unassigned"`. See section **Handling overlap**.

`return_logical_matrix` logical. If TRUE, a logical matrix with classes in columns and cells in rows is returned instead of resolving overlaps with `replace_overlap_with`. If a single class is supplied, the matrix has exactly one column and the user can pipe it into "drop" to convert it to a vector.

`fast` Set this to TRUE if you want fast plotting in spite of many cells (using the scattermore package). If NULL (default), cellpypes decides automatically and fast plotting is done for more than 10k cells, if FALSE it always uses `geom_point`.

`point_size` Dot size used by geom_point.

`point_size_legend` Dot size displayed in legend. Legend colors are easier to read with larger points.

`base_size` The base_size of theme_bw, i.e. how large text is displayed. Default: 15.

`overdispersion` Defaults to 0.01, only change it if you know what you are doing. If set to 0, the NB simplifies to the Poisson distribution, and larger values give more variance. The 0.01 default value follows the recommendation by Lause, Berens and Kobak (Genome Biology 2021) to use `size=100` in pnbinom for typical data sets.

Value

A ggplot2 object.

cellpypes Objects

A cellpypes object is a list with four slots:

`raw` (sparse) matrix with genes in rows, cells in columns

`totalUMI` the colSums of obj$raw

`embed` two-dimensional embedding of the cells, provided as data.frame or tibble with two columns and one row per cell.

`neighbors` index matrix with one row per cell and k columns, where k is the number of nearest neighbors (we recommend 15<k<100, e.g. k=50). Here are two ways to get the neighbors index matrix:

- Use `find_knn(featureMatrix)$idx`, where featureMatrix could be principal components, latent variables or normalized genes (features in rows, cells in columns).
- use `as(seurat@graphs[["RNA_nn"]], "dgCMatrix")> .1` to extract the kNN graph computed on RNA. The `> .1` ensures this also works with RNA_snn, wknn/wsnn or any other available graph – check with `names(seurat@graphs)`.

Handling overlap

Overlap denotes all cells for which rules from multiple classes apply, and these cells will be labeled as `Unassigned` by default. If you are in fact interested in where the overlap is, set `return_logical_matrix`=TRUE and inspect the result. Note that it matters whether you call `classify("Tcell")` or `classify(c("Tcell","Bcell")` – any existing overlap between T and B cells is labelled as `Unassigned` in this second call, but not in the first.

Replacing overlap happens only between mutually exclusive labels (such as Tcell and Bcell), but not within a lineage. To make an example, overlap is NOT replaced between child (PD1+Ttox) and parent (Ttox) or any other ancestor (Tcell), but instead the most detailed cell type (PD1+Ttox) is returned.

All of the above is also true for `plot_classes`, as it wraps `classify`.

knn_refine

With `knn_refine` > `0`, cellpypes refines cell type labels with a kNN classifier.

By default, cellpypes only assigns cells to a class if all relevant rules apply. In other words, all marker gene UMI counts in the cell's neighborhood all have to be clearly above/below their threshold. Since UMI counts are sparse (even after neighbor pooling done by cellpypes), this can leave many cells unassigned.

It is reasonable to assume an unassigned cell is of the same cell type as the majority of its nearest neighbors. Therefore, cellpypes implements a kNN classifier to further refine labels obtained by manually thresholding UMI counts. `knn_refine = 0.3` means a cell is assigned the class label held by most of its neighbors unless no class gets more than 30 %. If most neighbors are unassigned, the cell will also be set to "Unassigned". Choosing `knn_refine = 0.3` gives results reminiscent of clustering (which assigns all cells), while `knn_refine = 0.5` leaves cells 'in between' two similar cell types unassigned.

We recommend looking at `knn_refine = 0` first as it's faster and more directly tied to marker gene expression. If assigning all cells is desired, we recommend `knn_refine = 0.3` or lower, while `knn_refine = 0.5` makes cell types more 'crisp' by setting cells 'in between' related subtypes to "Unassigned".

Examples

```
plot_classes(rule(simulated_umis, "T", "CD3E",">", 1))
```

`plot_last` *Plot the last modified rule or class*

Description

Plot the last modified rule or class

Usage

```
plot_last(
  obj,
  show_feat = TRUE,
  what = "rule",
  fast = NULL,
  legend_rel_width = 0.3,
  overdispersion = 0.01
)
```

Arguments

`obj`	A cellpypes object, see section **cellpypes Objects** below.
`show_feat`	If TRUE (default), a second panel shows the feature plot of the relevant gene.
`what`	Either "rule" or "class".
`fast`	Set this to TRUE if you want fast plotting in spite of many cells (using the scattermore package). If NULL (default), cellpypes decides automatically and fast plotting is done for more than 10k cells, if FALSE it always uses `geom_point`.
`legend_rel_width`	Relative width compared to the other two plots (only relevant if `show_feat=TRUE`).
`overdispersion`	Defaults to 0.01, only change if you know what you are doing. See further classify.

Value

Returns a ggplot2 object with the plot.

cellpypes Objects

A cellpypes object is a list with four slots:

`raw` (sparse) matrix with genes in rows, cells in columns

`totalUMI` the colSums of obj$raw

`embed` two-dimensional embedding of the cells, provided as data.frame or tibble with two columns and one row per cell.

`neighbors` index matrix with one row per cell and k columns, where k is the number of nearest neighbors (we recommend 15<k<100, e.g. k=50). Here are two ways to get the neighbors index matrix:

- Use `find_knn(featureMatrix)$idx`, where featureMatrix could be principal components, latent variables or normalized genes (features in rows, cells in columns).
- use `as(seurat@graphs[["RNA_nn"]], "dgCMatrix")> .1` to extract the kNN graph computed on RNA. The `> .1` ensures this also works with RNA_snn, wknn/wsnn or any other available graph – check with `names(seurat@graphs)`.

Examples

```
plot_last(rule(simulated_umis, "T", "CD3E",">", 1))
```

pool_across_neighbors *Sum up x across neighbors in a nearest neighbor graph.*

Description

Neighbor pooling means that x is summed across the nearest neighbors.

Usage

```
pool_across_neighbors(x, neighbors)
```

Arguments

`x`	Numeric vector.
`neighbors`	Nearest neighbor graph provided as NxK index matrix (N observations, K neighbors) or NxN adjacency matrix. Index matrices can be obtained with find_knn (specifically the slot `idx` in the list it returns).

Value

Numeric vector of length x.

Examples

```
set.seed(42)
# simulate 30 cells without biological signal:
dummy_dat <- matrix(rpois(3000, .1), ncol=30)
# find 15 approximate nearest neighbors
neighbors <- find_knn(dummy_dat, k = 15)
# pool gene1 counts across neighbors:
neighbor_sum_gene1 <- pool_across_neighbors(dummy_dat[1,], neighbors$idx)
```

`pseudobulk` *Form pseudobulks from single cells.*

Description

Sum up cells in count matrix `raw` for bulk RNA methods such as DESeq2.

Usage

```
pseudobulk(raw, pseudobulk_id)
```

Arguments

`raw`	A matrix with raw UMI counts, cells in columns.
`pseudobulk_id`	A factor that identifies which cells should go to which pseudobulk. Generate pseudobulk_ids with the pseudobulk_id function!

Value

A matrix where each column is a pseudobulk and each row a gene.

Examples

```
# Create pseudobulk counts and coldata for DESeq2:
coldata <- data.frame(
  celltype = rep(c("X+Y-", "X+Y+", "X-Y+", "X-Y-"),
                 each = nrow(simulated_umis$embed)/4), # 4 cell types
  patient  = c("3", "500.", "*5", "/")
)
coldata$pseudobulk_id <- pseudobulk_id(coldata)
counts <- pseudobulk(simulated_umis$raw, coldata$pseudobulk_id)
# Use counts/coldata as input for DESeqDataSetFromMatrix (DESeq2).
```

pseudobulk_id *Generate unique IDs to identify your pseudobulks.*

Description

This function generates unique IDs that are valid colnames as well. Use these IDs in function pseudobulk.

Usage

```
pseudobulk_id(factor_df)
```

Arguments

`factor_df`	Data frame where each column helps to identify a pseudobulk. Each row in factor_df corresponds to a single cell in your raw count matrix. Typical columns of factor_df are for example patient, treatment and cell type – anything that uniquely identifies a replicate.

Details

Wraps make.names to generate syntactically valid IDs. Use these IDs in the pseudobulk function. Note that this function combines all columns in factor_df, so only include the columns that uniquely identify replicates. Cells from the same experimental unit

Value

Factor with syntactically valid and unique IDs.

Examples

```
# Create pseudobulk counts and coldata for DESeq2:
coldata <- data.frame(
  celltype = rep(c("X+Y-", "X+Y+", "X-Y+", "X-Y-"),
                 each = nrow(simulated_umis$embed)/4), # 4 cell types
  patient  = c("3", "500.", "*5", "/")
)
coldata$pseudobulk_id <- pseudobulk_id(coldata)
counts <- pseudobulk(simulated_umis$raw, coldata$pseudobulk_id)
# Use counts/coldata as input for DESeqDataSetFromMatrix (DESeq2).
```

pype_code_template *Generate code template for cellpype rules*

Description

This function rule code snippet with neat text alignment to the console. Paste this into your script and start changing the rules.

Usage

```
pype_code_template(n_rules = 3)
```

Arguments

n_rules	Number of lines (rules) to generate

Value

Prints rules to the consoles.

Examples

```
pype_code_template()
```

pype_from_seurat *Convert Seurat to cellpypes object.*

Description

Start cellpyping a Seurat object. This function saves the user from building his own cellpypes object, which is done with `list(umi, neighbors,embed, totalUMI)`.

Usage

```
pype_from_seurat(seurat, graph_name = NULL)
```

Arguments

seurat	A Seurat object.
graph_name	Supply one of the graphs. To see options, type `names(seurat@graphs)`. If left empty (`NULL`, the default), `pype_from_seurat` will try to guess the correct name for you.

Value

A cellpypes object.

cellpypes Objects

A cellpypes object is a list with four slots:

`raw` (sparse) matrix with genes in rows, cells in columns

`totalUMI` the colSums of obj$raw

`embed` two-dimensional embedding of the cells, provided as data.frame or tibble with two columns and one row per cell.

`neighbors` index matrix with one row per cell and k columns, where k is the number of nearest neighbors (we recommend 15<k<100, e.g. k=50). Here are two ways to get the neighbors index matrix:

- Use `find_knn(featureMatrix)$idx`, where featureMatrix could be principal components, latent variables or normalized genes (features in rows, cells in columns).
- use `as(seurat@graphs[["RNA_nn"]], "dgCMatrix")> .1` to extract the kNN graph computed on RNA. The `> .1` ensures this also works with RNA_snn, wknn/wsnn or any other available graph – check with `names(seurat@graphs)`.

---

rule *Add a cell type rule.*

Description

This is the heart of cellpypes and best used by piping from one rule into the next with `magrittr::%>%`. Check out examples at gitHub)!

Usage

```
rule(
  obj,
  class,
  feature,
  operator = ">",
  threshold,
  parent = NULL,
  use_CP10K = TRUE
)
```

Arguments

`obj`	A cellpypes object, see section **cellpypes Objects** below.
`class`	Character scalar with the class name. Typically, cellpypes classes are literature cell types ("T cell") or any subpopulation of interest ("CD3E+TNF+LAG3-").
`feature`	Character scalar naming the gene you'd like to threshold. Must be a row name in `obj$raw`.
`operator`	One of `c(">","<")`. Use ">" for positive (CD3E+) and "<" for negative markers (MS4A1-).

`threshold` Numeric scalar with the expression threshold separating positive from negative cells. Experiment with this value, until expression and selected cells agree well in UMAP (see examples on gitHub).

`parent` Character scalar with the parent class (e.g. "T cell" for "Cytotoxic T cells"). Only has to be specified once per class (else most recent one is taken), and defaults to "..root.." if NULL is passed in all rules.

`use_CP10K` If TRUE, `threshold` is taken to be counts per 10 thousand UMI counts, a measure for RNA molecule fractions. We recommend CP10K for human intuition (1 CP10K is roughly 1 UMI per cell), but the results are the exact same whether you use `threshold=1,CP10K=TRUE` or `threshold=1e-4,CP10K=FALSE`.

Details

Calling `rule` is computationally cheap because it only stores the cell type rule while all computations happen in classify. If you have classes with multiple rules, the most recent `parent` and `feature-threshold` combination counts. It is ok to mix rules with and without `use_CP10K=TRUE`.

Value

`obj` is returned, but with the rule and class stored in `obj$rules` and `obj$classes`, to be used by classify.

cellpypes Objects

A cellpypes object is a list with four slots:

`raw` (sparse) matrix with genes in rows, cells in columns

`totalUMI` the colSums of obj$raw

`embed` two-dimensional embedding of the cells, provided as data.frame or tibble with two columns and one row per cell.

`neighbors` index matrix with one row per cell and k columns, where k is the number of nearest neighbors (we recommend 15<k<100, e.g. k=50). Here are two ways to get the neighbors index matrix:

- Use `find_knn(featureMatrix)$idx`, where featureMatrix could be principal components, latent variables or normalized genes (features in rows, cells in columns).
- use `as(seurat@graphs[["RNA_nn"]], "dgCMatrix")> .1` to extract the kNN graph computed on RNA. The `> .1` ensures this also works with RNA_snn, wknn/wsnn or any other available graph – check with `names(seurat@graphs)`.

See Also

To have nicely formatted code in the end, copy the output of `pype_code_template()` to your script and start editing.

Examples

```
# T cells are CD3E+:
obj <- rule(simulated_umis, "T", "CD3E", ">", .1)
# T cells are MS4A1-:
obj <- rule(obj, "T", "MS4A1", "<", 1)
# Tregs are a subset of T cells:
obj <- rule(obj, "Treg", "FOXP3", ">", .1, parent="T")
```

simulated_umis *Simulated scRNAseq data*

Description

This data serves to develop cellpypes and to illustrate its functionality. I made it up entirely.

Usage

```
simulated_umis
```

Format

A list with 4 entries:

raw Raw (unnormalized) UMI counts for a handful of genes, last row are totalUMI.

neighbors Indices of each cell's 50 nearest neighbors.

embed Simulated UMAP embedding.

celltype Cell type label that I used to simulate the data.

Source

Very simple simulation (c.f. data-raw/simulated_umis.R in source code).

Index