Package 'clustree'

October 12, 2022

Type Package

Title Visualise Clusterings at Different Resolutions

Version 0.5.0

Date 2022-06-24

Maintainer Luke Zappia <luke@lazappi.id.au>

Description Deciding what resolution to use can be a difficult question when approaching a clustering analysis. One way to approach this problem is to look at how samples move as the number of clusters increases. This package allows you to produce clustering trees, a visualisation for interrogating clusterings as resolution increases.

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/lazappi/clustree

BugReports https://github.com/lazappi/clustree/issues

VignetteBuilder knitr

Depends R (>= 3.5), ggraph

Imports checkmate, igraph, dplyr, grid, ggplot2, viridis, methods, rlang, tidygraph, ggrepel

Suggests testthat (>= 2.1.0), knitr, rmarkdown, SingleCellExperiment, Seurat (>= 2.3.0), covr, SummarizedExperiment, pkgdown, spelling

RoxygenNote 7.2.0

Language en-GB

NeedsCompilation no

Author Luke Zappia [aut, cre] (<https://orcid.org/0000-0001-7744-8565>),
Alicia Oshlack [aut] (<https://orcid.org/0000-0001-9788-5690>),
Andrea Rau [ctb],
Paul Hoffman [ctb] (<https://orcid.org/0000-0002-7693-8957>)

Repository CRAN

Date/Publication 2022-06-25 20:50:02 UTC

R topics documented:

`clustree-package` *Clustree*

Description

Deciding what resolution to use can be a difficult question when approaching a clustering analysis. One way to approach this problem is to look at how samples move as the number of clusters increases. This package allows you to produce clustering trees, a visualisation for interrogating clusterings as resolution increases.

`add_node_labels` *Add node labels*

Description

Add node labels to a clustering tree plot with the specified aesthetics.

Usage

```
add_node_labels(
  node_label,
  node_colour,
  node_label_size,
  node_label_colour,
  node_label_nudge,
  allowed
)
```

Arguments

`node_label`	the name of a metadata column for node labels
`node_colour`	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
`node_label_size`	size of node label text
`node_label_colour`	colour of node_label text
`node_label_nudge`	numeric value giving nudge in y direction for node labels
`allowed`	vector of allowed node attributes to use as aesthetics

`add_node_points` *Add node points*

Description

Add node points to a clustering tree plot with the specified aesthetics.

Usage

```
add_node_points(node_colour, node_size, node_alpha, allowed)
```

Arguments

`node_colour`	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
`node_size`	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
`node_alpha`	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency
`allowed`	vector of allowed node attributes to use as aesthetics

`aggr_metadata` *Aggregate metadata*

Description

Aggregate a metadata column to get a summarized value for a cluster node

Usage

```
aggr_metadata(node_data, col_name, col_aggr, metadata, is_cluster)
```

Arguments

`node_data`	data.frame containing information about a set of cluster nodes
`col_name`	the name of the metadata column to aggregate
`col_aggr`	string naming a function used to aggregate the column
`metadata`	data.frame providing metadata on samples
`is_cluster`	logical vector indicating which rows of metadata are in the node to be summarized

Value

data.frame with aggregated data

`assert_colour_node_aes` *Assert colour node aesthetics*

Description

Raise error if an incorrect set of colour node parameters has been supplied.

Usage

```
assert_colour_node_aes(
  node_aes_name,
  prefix,
  metadata,
  node_aes,
  node_aes_aggr,
  min,
  max
)
```

Arguments

`node_aes_name`	name of the node aesthetic to check
`prefix`	string indicating columns containing clustering information
`metadata`	data.frame containing metadata on each sample that can be used as node aesthetics
`node_aes`	value of the node aesthetic to check
`node_aes_aggr`	aggregation function associated with the node aesthetic
`min`	minimum numeric value allowed
`max`	maximum numeric value allowed

`assert_node_aes` *Assert node aesthetics*

Description

Raise error if an incorrect set of node parameters has been supplied.

Usage

```
assert_node_aes(node_aes_name, prefix, metadata, node_aes, node_aes_aggr)
```

Arguments

`node_aes_name`	name of the node aesthetic to check
`prefix`	string indicating columns containing clustering information
`metadata`	data.frame containing metadata on each sample that can be used as node aesthetics
`node_aes`	value of the node aesthetic to check
`node_aes_aggr`	aggregation function associated with the node aesthetic

`assert_numeric_node_aes`

Assert numeric node aesthetics

Description

Raise error if an incorrect set of numeric node parameters has been supplied.

Usage

```
assert_numeric_node_aes(
  node_aes_name,
  prefix,
  metadata,
  node_aes,
  node_aes_aggr,
  min,
  max
)
```

Arguments

`node_aes_name`	name of the node aesthetic to check
`prefix`	string indicating columns containing clustering information
`metadata`	data.frame containing metadata on each sample that can be used as node aesthetics
`node_aes`	value of the node aesthetic to check
`node_aes_aggr`	aggregation function associated with the node aesthetic
`min`	minimum numeric value allowed
`max`	maximum numeric value allowed

`build_tree_graph` *Build tree graph*

Description

Build a tree graph from a set of clusterings, metadata and associated aesthetics

Usage

```
build_tree_graph(
  clusterings,
  prefix,
  count_filter,
  prop_filter,
  metadata,
  node_aes_list
)
```

Arguments

`clusterings`	numeric matrix containing clustering information, each column contains clustering at a separate resolution
`prefix`	string indicating columns containing clustering information
`count_filter`	count threshold for filtering edges in the clustering graph
`prop_filter`	in proportion threshold for filtering edges in the clustering graph
`metadata`	data.frame containing metadata on each sample that can be used as node aesthetics
`node_aes_list`	nested list containing node aesthetics

Value

tidygraph::tbl_graph object containing the tree graph

`calc_sc3_stability` *Calculate SC3 stability*

Description

Calculate the SC3 stability index for every cluster at every resolution in a set of clusterings. The index varies from 0 to 1, where 1 suggests that a cluster is more stable across resolutions. See `calc_sc3_stability_cluster()` for more details.

Usage

```
calc_sc3_stability(clusterings)
```

Arguments

`clusterings`	numeric matrix containing clustering information, each column contains clustering at a separate resolution

Value

matrix with stability score for each cluster

`calc_sc3_stability_cluster`

Calculate single SC3 stability

Description

Calculate the SC3 stability index for a single cluster in a set of clusterings. The index varies from 0 to 1, where 1 suggests that a cluster is more stable across resolutions.

Usage

```
calc_sc3_stability_cluster(clusterings, res, cluster)
```

Arguments

`clusterings`	numeric matrix containing clustering information, each column contains clustering at a separate resolution
`res`	resolution of the cluster to calculate stability for
`cluster`	index of the cluster to calculate stability for

Details

This index was originally introduced in the `SC3` package for clustering single-cell RNA-seq data. Clusters are awarded increased stability if they share the same samples as a cluster at another resolution and penalised at higher resolutions. We use a slightly different notation to describe the score but the results are the same:

$$s(c_{k,i}) = \frac{1}{size(L) + 1} \sum_{l \in L} \sum_{j \in N_l} \frac{size(c_{k,i} \cap c_{l,j})}{size(c_{l,j}) * size(N_l)^2}$$

Where:

- `c_{x, y}` is cluster `y` at resolution `x`
- `k` is the resolution of the cluster we want to score
- `i` is the index of the cluster we want to score
- `L` is the set of all resolutions except `k`
- `l` is a resolution in `L`
- `N_l` is the set of clusters at resolution `l` that share samples with `c_{k, i}`
- `j` is a cluster in `N_l`

Value

SC3 stability index

See Also

The documentation for the `calculate_stability` function in the SC3 package

`check_node_aes_list` *Check node aes list*

Description

Warn if node aesthetic names are incorrect

Usage

```
check_node_aes_list(node_aes_list)
```

Arguments

`node_aes_list`	List of node aesthetics

Value

Corrected node aesthetics list

`clustree` *Plot a clustering tree*

Description

Creates a plot of a clustering tree showing the relationship between clusterings at different resolutions.

Usage

```
clustree(x, ...)

## S3 method for class 'matrix'
clustree(
  x,
  prefix,
  suffix = NULL,
  metadata = NULL,
  count_filter = 0,
  prop_filter = 0.1,
  layout = c("tree", "sugiyama"),
  use_core_edges = TRUE,
  highlight_core = FALSE,
  node_colour = prefix,
  node_colour_aggr = NULL,
  node_size = "size",
  node_size_aggr = NULL,
```

```
  node_size_range = c(4, 15),
  node_alpha = 1,
  node_alpha_aggr = NULL,
  node_text_size = 3,
  scale_node_text = FALSE,
  node_text_colour = "black",
  node_text_angle = 0,
  node_label = NULL,
  node_label_aggr = NULL,
  node_label_size = 3,
  node_label_nudge = -0.2,
  edge_width = 1.5,
  edge_arrow = TRUE,
  edge_arrow_ends = c("last", "first", "both"),
  show_axis = FALSE,
  return = c("plot", "graph", "layout"),
  ...
)

## S3 method for class 'data.frame'
clustree(x, prefix, ...)

## S3 method for class 'SingleCellExperiment'
clustree(x, prefix, exprs = "counts", ...)

## S3 method for class 'seurat'
clustree(x, prefix = "res.", exprs = c("data", "raw.data", "scale.data"), ...)

## S3 method for class 'Seurat'
clustree(
  x,
  prefix = paste0(assay, "_snn_res."),
  exprs = c("data", "counts", "scale.data"),
  assay = NULL,
  ...
)
```

Arguments

`x`	object containing clustering data
`...`	extra parameters passed to other methods
`prefix`	string indicating columns containing clustering information
`suffix`	string at the end of column names containing clustering information
`metadata`	data.frame containing metadata on each sample that can be used as node aesthetics
`count_filter`	count threshold for filtering edges in the clustering graph
`prop_filter`	in proportion threshold for filtering edges in the clustering graph

`layout`	string specifying the "tree" or "sugiyama" layout, see `igraph::layout_as_tree()` and `igraph::layout_with_sugiyama()` for details
`use_core_edges`	logical, whether to only use core tree (edges with maximum in proportion for a node) when creating the graph layout, all (unfiltered) edges will still be displayed
`highlight_core`	logical, whether to increase the edge width of the core network to make it easier to see
`node_colour`	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
`node_colour_aggr`	if `node_colour` is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
`node_size`	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
`node_size_aggr`	if `node_size` is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
`node_size_range`	numeric vector of length two giving the maximum and minimum point size for plotting nodes
`node_alpha`	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency
`node_alpha_aggr`	if `node_aggr` is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
`node_text_size`	numeric value giving the size of node text if `scale_node_text` is `FALSE`
`scale_node_text`	logical indicating whether to scale node text along with the node size
`node_text_colour`	colour value for node text (and label)
`node_text_angle`	the rotation of the node text
`node_label`	additional label to add to nodes
`node_label_aggr`	if `node_label` is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
`node_label_size`	numeric value giving the size of node label text
`node_label_nudge`	numeric value giving nudge in y direction for node labels
`edge_width`	numeric value giving the width of plotted edges
`edge_arrow`	logical indicating whether to add an arrow to edges
`edge_arrow_ends`	string indicating which ends of the line to draw arrow heads if `edge_arrow` is `TRUE`, one of "last", "first", or "both"

`show_axis`	whether to show resolution axis
`return`	string specifying what to return, either "plot" (a `ggplot` object), "graph" (a `tbl_graph` object) or "layout" (a `ggraph` layout object)
`exprs`	source of gene expression information to use as node aesthetics, for `SingleCellExperiment` objects it must be a name in `assayNames(x)`, for a `seurat` object it must be one of `data`, `raw.data` or `scale.data` and for a `Seurat` object it must be one of `data`, `counts` or `scale.data`
`assay`	name of assay to pull expression and clustering data from for `Seurat` objects

Details

Data sources

Plotting a clustering tree requires information about which cluster each sample has been assigned to at different resolutions. This information can be supplied in various forms, as a matrix, data.frame or more specialised object. In all cases the object provided must contain numeric columns with the naming structure `PXS` where `P` is a prefix indicating that the column contains clustering information, `X` is a numeric value indicating the clustering resolution and `S` is any additional suffix to be removed. For `SingleCellExperiment` objects this information must be in the `colData` slot and for `Seurat` objects it must be in the `meta.data` slot. For all objects except matrices any additional columns can be used as aesthetics, for matrices an additional metadata data.frame can be supplied if required.

Filtering

Edges in the graph can be filtered by adjusting the `count_filter` and `prop_filter` parameters. The `count_filter` removes any edges that represent less than that number of samples, while the `prop_filter` removes edges that represent less than that proportion of cells in the node it points towards.

Node aesthetics

The aesthetics of the plotted nodes can be controlled in various ways. By default the colour indicates the clustering resolution, the size indicates the number of samples in that cluster and the transparency is set to 100%. Each of these can be set to a specific value or linked to a supplied metadata column. For a `SingleCellExperiment` or `Seurat` object the names of genes can also be used. If a metadata column is used than an aggregation function must also be supplied to combine the samples in each cluster. This function must take a vector of values and return a single value.

Layout

The clustering tree can be displayed using either the Reingold-Tilford tree layout algorithm or the Sugiyama layout algorithm for layered directed acyclic graphs. These layouts were selected as the are the algorithms available in the `igraph` package designed for trees. The Reingold-Tilford algorithm places children below their parents while the Sugiyama places nodes in layers while trying to minimise the number of crossing edges. See `igraph::layout_as_tree()` and `igraph::layout_with_sugiyama()` for more details. When `use_core_edges` is `TRUE` (default) only the core tree of the maximum in proportion edges for each node are used for constructing the layout. This can often lead to more attractive layouts where the core tree is more visible.

Value

a `ggplot` object (default), a `tbl_graph` object or a `ggraph` layout object depending on the value of `return`

Examples

```
data(nba_clusts)
clustree(nba_clusts, prefix = "K")
```

`clustree_overlay` *Overlay a clustering tree*

Description

Creates a plot of a clustering tree overlaid on a scatter plot of individual samples.

Usage

```
clustree_overlay(x, ...)

## S3 method for class 'matrix'
clustree_overlay(
  x,
  prefix,
  metadata,
  x_value,
  y_value,
  suffix = NULL,
  count_filter = 0,
  prop_filter = 0.1,
  node_colour = prefix,
  node_colour_aggr = NULL,
  node_size = "size",
  node_size_aggr = NULL,
  node_size_range = c(4, 15),
  node_alpha = 1,
  node_alpha_aggr = NULL,
  edge_width = 1,
  use_colour = c("edges", "points"),
  alt_colour = "black",
  point_size = 3,
  point_alpha = 0.2,
  point_shape = 18,
  label_nodes = FALSE,
  label_size = 3,
  plot_sides = FALSE,
  side_point_jitter = 0.45,
  side_point_offset = 1,
  ...
)
```

```
## S3 method for class 'data.frame'
clustree_overlay(x, prefix, ...)

## S3 method for class 'SingleCellExperiment'
clustree_overlay(
  x,
  prefix,
  x_value,
  y_value,
  exprs = "counts",
  red_dim = NULL,
  ...
)

## S3 method for class 'seurat'
clustree_overlay(
  x,
  x_value,
  y_value,
  prefix = "res.",
  exprs = c("data", "raw.data", "scale.data"),
  red_dim = NULL,
  ...
)

## S3 method for class 'Seurat'
clustree_overlay(
  x,
  x_value,
  y_value,
  prefix = paste0(assay, "_snn_res."),
  exprs = c("data", "counts", "scale.data"),
  red_dim = NULL,
  assay = NULL,
  ...
)
```

Arguments

`x`	object containing clustering data
`...`	extra parameters passed to other methods
`prefix`	string indicating columns containing clustering information
`metadata`	data.frame containing metadata on each sample that can be used as node aesthetics
`x_value`	numeric metadata column to use as the x axis
`y_value`	numeric metadata column to use as the y axis
`suffix`	string at the end of column names containing clustering information

`count_filter`	count threshold for filtering edges in the clustering graph
`prop_filter`	in proportion threshold for filtering edges in the clustering graph
`node_colour`	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
`node_colour_aggr`	if `node_colour` is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
`node_size`	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
`node_size_aggr`	if `node_size` is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
`node_size_range`	numeric vector of length two giving the maximum and minimum point size for plotting nodes
`node_alpha`	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency
`node_alpha_aggr`	if `node_aggr` is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
`edge_width`	numeric value giving the width of plotted edges
`use_colour`	one of "edges" or "points" specifying which element to apply the colour aesthetic to
`alt_colour`	colour value to be used for edges or points (whichever is NOT given by `use_colour`)
`point_size`	numeric value giving the size of sample points
`point_alpha`	numeric value giving the alpha of sample points
`point_shape`	numeric value giving the shape of sample points
`label_nodes`	logical value indicating whether to add labels to clustering graph nodes
`label_size`	numeric value giving the size of node labels is `label_nodes` is `TRUE`
`plot_sides`	logical value indicating whether to produce side on plots
`side_point_jitter`	numeric value giving the y-direction spread of points in side plots
`side_point_offset`	numeric value giving the y-direction offset for points in side plots
`exprs`	source of gene expression information to use as node aesthetics, for `SingleCellExperiment` objects it must be a name in `assayNames(x)`, for a `seurat` object it must be one of `data`, `raw.data` or `scale.data` and for a `Seurat` object it must be one of `data`, `counts` or `scale.data`
`red_dim`	dimensionality reduction to use as a source for x_value and y_value
`assay`	name of assay to pull expression and clustering data from for `Seurat` objects

Details

Data sources

Plotting a clustering tree requires information about which cluster each sample has been assigned to at different resolutions. This information can be supplied in various forms, as a matrix, data.frame or more specialised object. In all cases the object provided must contain numeric columns with the naming structure `PXS` where `P` is a prefix indicating that the column contains clustering information, `X` is a numeric value indicating the clustering resolution and `S` is any additional suffix to be removed. For `SingleCellExperiment` objects this information must be in the `colData` slot and for `Seurat` objects it must be in the `meta.data` slot. For all objects except matrices any additional columns can be used as aesthetics.

Filtering

Edges in the graph can be filtered by adjusting the `count_filter` and `prop_filter` parameters. The `count_filter` removes any edges that represent less than that number of samples, while the `prop_filter` removes edges that represent less than that proportion of cells in the node it points towards.

Node aesthetics

The aesthetics of the plotted nodes can be controlled in various ways. By default the colour indicates the clustering resolution, the size indicates the number of samples in that cluster and the transparency is set to 100%. Each of these can be set to a specific value or linked to a supplied metadata column. For a `SingleCellExperiment` or `Seurat` object the names of genes can also be used. If a metadata column is used than an aggregation function must also be supplied to combine the samples in each cluster. This function must take a vector of values and return a single value.

Colour aesthetic

The colour aesthetic can be applied to either edges or sample points by setting `use_colour`. If "edges" is selected edges will be coloured according to the clustering resolution they originate at. If "points" is selected they will be coloured according to the cluster they are assigned to at the highest resolution.

Dimensionality reductions

For `SingleCellExperiment` and `Seurat` objects precomputed dimensionality reductions can be used for x or y aesthetics. To do so `red_dim` must be set to the name of a dimensionality reduction in `reducedDimNames(x)` (for a `SingleCellExperiment`) or `x@dr` (for a `Seurat` object). `x_value` and `y_value` can then be set to `red_dimX` when `red_dim` matches the `red_dim` argument and `X` is the column of the dimensionality reduction to use.

Value

a `ggplot` object if `plot_sides` is `FALSE` or a list of `ggplot` objects if `plot_sides` is `TRUE`

Examples

```
data(nba_clusts)
clustree_overlay(nba_clusts, prefix = "K", x_value = "PC1", y_value = "PC2")
```

`get_tree_edges` *Get tree edges*

Description

Extract the edges from a set of clusterings

Usage

```
get_tree_edges(clusterings, prefix)
```

Arguments

`clusterings`	numeric matrix containing clustering information, each column contains clustering at a separate resolution
`prefix`	string indicating columns containing clustering information

Value

data.frame containing edge information

`get_tree_nodes` *Get tree nodes*

Description

Extract the nodes from a set of clusterings and add relevant attributes

Usage

```
get_tree_nodes(clusterings, prefix, metadata, node_aes_list)
```

Arguments

`clusterings`	numeric matrix containing clustering information, each column contains clustering at a separate resolution
`prefix`	string indicating columns containing clustering information
`metadata`	data.frame containing metadata on each sample that can be used as node aesthetics
`node_aes_list`	nested list containing node aesthetics

Value

data.frame containing node information

`nba_clusts` *Clustered NBA positions dataset*

Description

NBA positions dataset clustered using k-means with a range of values of `k`

Usage

```
nba_clusts
```

Format

`nba_clusts` is a data.frame containing the NBA positions dataset with additional columns holding k-means clusterings at different values of `k` and the first two principal components

- **Position** - Player position
- **TurnoverPct** - Turnover percentage
- **ReboundPct** - Rebound percentage
- **AssistPct** - Assist percentage
- **FieldGoalPct** - Field goal percentage
- **K1 - K5** - Results of k-means clustering
- **PC1** - First principal component
- **PC2** - Second principal component

Source

NBA positions downloaded from `https://github.com/lazappi/nba_positions`.

The source dataset is available from Kaggle at `https://www.kaggle.com/drgilermo/nba-players-stats/data?select=Seasons_Stats.csv` and was originally scraped from Basketball Reference.

See `https://github.com/lazappi/clustree/blob/master/data-raw/nba_clusts.R` for details of how clustering was performed.

`overlay_node_points` *Overlay node points*

Description

Overlay clustering tree nodes on a scatter plot with the specified aesthetics.

Usage

```
overlay_node_points(
  nodes,
  x_value,
  y_value,
  node_colour,
  node_size,
  node_alpha
)
```

Arguments

`nodes`	data.frame describing nodes
`x_value`	column of nodes to use for the x position
`y_value`	column of nodes to use for the y position
`node_colour`	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
`node_size`	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
`node_alpha`	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency

---

`plot_overlay_side` *Plot overlay side*

Description

Plot the side view of a clustree overlay plot. If the ordinary plot shows the tree from above this plot shows it from the side, highlighting either the x or y dimension and the clustering resolution.

Usage

```
plot_overlay_side(
  nodes,
  edges,
  points,
  prefix,
  side_value,
  graph_attr,
  node_size_range,
  edge_width,
  use_colour,
  alt_colour,
  point_size,
  point_alpha,
  point_shape,
  label_nodes,
  label_size,
  y_jitter,
  y_offset
)
```

Arguments

`nodes`	data.frame describing nodes
`edges`	data.frame describing edges
`points`	data.frame describing points
`prefix`	string indicating columns containing clustering information
`side_value`	string giving the metadata column to use for the x axis
`graph_attr`	list describing graph attributes
`node_size_range`	numeric vector of length two giving the maximum and minimum point size for plotting nodes
`edge_width`	numeric value giving the width of plotted edges
`use_colour`	one of "edges" or "points" specifying which element to apply the colour aesthetic to
`alt_colour`	colour value to be used for edges or points (whichever is NOT given by `use_colour`)
`point_size`	numeric value giving the size of sample points
`point_alpha`	numeric value giving the alpha of sample points
`point_shape`	numeric value giving the shape of sample points
`label_nodes`	logical value indicating whether to add labels to clustering graph nodes
`label_size`	numeric value giving the size of node labels is `label_nodes` is `TRUE`
`y_jitter`	numeric value giving the y-direction spread of points in side plots
`y_offset`	numeric value giving the y-direction offset for points in side plots

Value

ggplot object

`sc_example` *Simulated scRNA-seq dataset*

Description

A simulated scRNA-seq dataset generated using the `splatter` package and clustered using the `SC3` and `Seurat` packages.

Usage

```
sc_example
```

Format

`sc_example` is a list holding a simulated scRNA-seq dataset. Items in the list included the simulated counts, normalised log counts, tSNE dimensionality reduction and cell assignments from `SC3` and `Seurat` clustering.

Source

```
# Simulation
library("splatter") # Version 1.2.1

sim <- splatSimulate(batchCells = 200, nGenes = 10000,
                     group.prob = c(0.4, 0.2, 0.2, 0.15, 0.05),
                     de.prob = c(0.1, 0.2, 0.05, 0.1, 0.05),
                     method = "groups", seed = 1)
sim_counts <- counts(sim)[1:1000, ]

# SC3 Clustering
library("SC3") # Version 1.7.6
library("scater") # Version 1.6.2

sim_sc3 <- SingleCellExperiment(assays = list(counts = sim_counts))
rowData(sim_sc3)$feature_symbol <- rownames(sim_counts)
sim_sc3 <- normalise(sim_sc3)
sim_sc3 <- sc3(sim_sc3, ks = 1:8, biology = FALSE, n_cores = 1)
sim_sc3 <- runTSNE(sim_sc3)

# Seurat Clustering
library("Seurat") # Version 2.2.0

sim_seurat <- CreateSeuratObject(sim_counts)
sim_seurat <- NormalizeData(sim_seurat, display.progress = FALSE)
```

```
sim_seurat <- FindVariableGenes(sim_seurat, do.plot = FALSE,
                                display.progress = FALSE)
sim_seurat <- ScaleData(sim_seurat, display.progress = FALSE)
sim_seurat <- RunPCA(sim_seurat, do.print = FALSE)
sim_seurat <- FindClusters(sim_seurat, dims.use = 1:6,
                           resolution = seq(0, 1, 0.1),
                           print.output = FALSE)

sc_example <- list(counts = counts(sim_sc3),
                   logcounts = logcounts(sim_sc3),
                   tsne = reducedDim(sim_sc3),
                   sc3_clusters = as.data.frame(colData(sim_sc3)),
                   seurat_clusters = sim_seurat@meta.data)
```

`store_node_aes` *Store node aesthetics*

Description

Store the names of node attributes to use as aesthetics as graph attributes

Usage

```
store_node_aes(graph, node_aes_list, metadata)
```

Arguments

`graph`	graph to store attributes in
`node_aes_list`	nested list containing node aesthetics
`metadata`	data.frame containing metadata that can be used as aesthetics

Value

graph with additional attributes

Index