Package 'treeclimbR'

December 19, 2024

Type Package

Title An algorithm to find optimal signal levels in a tree

Version 1.3.0

Date 2024-10-11

Description The arrangement of hypotheses in a hierarchical structure appears in many research fields and often indicates different resolutions at which data can be viewed. This raises the question of which resolution level the signal should best be interpreted on. treeclimbR provides a flexible method to select optimal resolution levels (potentially different levels in different parts of the tree), rather than cutting the tree at an arbitrary level. treeclimbR uses a tuning parameter to generate candidate resolutions and from these selects the optimal one.

License Artistic-2.0

Encoding UTF-8

biocViews StatisticalMethod, CellBasedAssays

Depends R (>= 4.4.0)

Imports TreeSummarizedExperiment (>= 1.99.0), edgeR, methods, SummarizedExperiment, S4Vectors, dirmult, dplyr, tibble, tidyr, ape, diffcyt, ggnewscale, ggplot2 (>= 3.4.0), viridis, ggtree, stats, utils, rlang

Suggests knitr, rmarkdown, scales, testthat (>= 3.0.0), BiocStyle

RoxygenNote 7.3.2

VignetteBuilder knitr

URL https://github.com/csoneson/treeclimbR

BugReports https://github.com/csoneson/treeclimbR/issues

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/treeclimbR

git_branch devel

git_last_commit 653ec0b

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2024-12-18

Author Ruizhu Huang [aut] (ORCID: <https://orcid.org/0000-0003-3285-1945>), Charlotte Soneson [aut, cre] (ORCID: <https://orcid.org/0000-0003-3833-2169>)

Maintainer Charlotte Soneson <charlottesoneson@gmail.com>

Contents

`treeclimbR-package` *treeclimbR*

Description

The arrangement of hypotheses in a hierarchical structure appears in many research fields and often indicates different resolutions at which data can be viewed. This raises the question of which resolution level the signal should best be interpreted on. treeclimbR provides a flexible method to select optimal resolution levels (potentially different levels in different parts of the tree), rather than cutting the tree at an arbitrary level. treeclimbR uses a tuning parameter to generate candidate resolutions and from these selects the optimal one.

Author(s)

Ruizhu Huang

Charlotte Soneson

See Also

Useful links:

- https://github.com/csoneson/treeclimbR
- Report bugs at https://github.com/csoneson/treeclimbR/issues

---

aggDS *Aggregate observed data based on a tree*

Description

Aggregate observed values based on a column (sample) tree, e.g. for differential state analysis. The returned object will contain one abundance matrix for each node in the tree, with columns corresponding to sample IDs and rows corresponding to the same features as the rows of the input object. The nodes correspond to either the original sample clusters, or larger metaclusters encompassing several of the original clusters (defined by the provided column tree).

Usage

```
aggDS(
  TSE,
  assay = "counts",
  sample_id = "sample_id",
  group_id = "group_id",
  cluster_id = "cluster_id",
  FUN = sum,
  message = FALSE
)
```

Arguments

TSE	A TreeSummarizedExperiment object. Rows represent variables (e.g., genes) and columns represent observations (e.g., cells). The object must contain a column tree, whose tips represent initial cell clusters (the cluster_id annotation indicates which of these clusters a cell belongs to). The internal nodes represent increasingly coarse partitions of the cells obtained by successively merging the original clusters according to the provided column tree.
assay	The name or index of the assay from TSE to aggregate values from.
sample_id	A character scalar indicating the column of colData(TSE) that corresponds to the sample ID. These will be the columns of the output object.

group_id	A character scalar indicating the column of colData(TSE) that corresponds to the group/condition. This information will be propagated to the aggregated object.
cluster_id	A character scalar indicating the column of colData(TSE) that corresponds to the initial cluster ID for each cell.
FUN	The aggregation function.
message	A logical scalar, indicating whether progress messages should be printed to the console.

Value

A SummarizedExperiment object. Each assay represents the aggregated values for one node in the tree.

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ape)
    library(ggtree)
})

set.seed(1L)
tr <- rtree(3, tip.label = LETTERS[seq_len(3)])
ggtree(tr) +
    geom_text(aes(label = node), hjust = -1, vjust = 1) +
    geom_text(aes(label = label), hjust = -1, vjust = -1)

cc <- matrix(rpois(60, 10), nrow = 6)
rownames(cc) <- paste0("gene", seq_len(6))
colnames(cc) <- paste0("cell", seq_len(10))
cd <- data.frame(sid = rep(seq_len(2), each = 5),
                 gid = rep(letters[seq_len(2)], each = 5),
                 cid = sample(LETTERS[seq_len(3)], size = 10,
                              replace = TRUE),
                 stringsAsFactors = FALSE)
tse <- TreeSummarizedExperiment(assays = list(counts = cc),
                                colTree = tr,
                                colNodeLab = cd$cid,
                                colData = cd)

out <- aggDS(TSE = tse, assay = "counts", sample_id = "sid",
             group_id = "gid", cluster_id = "cid")

## Aggregated counts for the node 5
SummarizedExperiment::assay(out, "alias_5")
## This is equal to the sum of the counts from nodes 1 and 2
```

```
SummarizedExperiment::assay(out, "alias_1")
SummarizedExperiment::assay(out, "alias_2")
```

---

buildTree *Tree versions of diffcyt functions*

Description

A collection of functions from the diffcyt package have been generalized to work with data provided in a tree structure. The tree represents increasingly coarse clustering of the cells, and the leaves are the clusters from an initial, high-resolution clustering generated by diffcyt. Note that diffcyt represents data using SummarizedExperiment objects with cells in rows and features in columns.

Usage

```
buildTree(d_se, dist_method = "euclidean", hclust_method = "average")

calcMediansByTreeMarker(d_se, tree)

calcTreeCounts(d_se, tree)

calcTreeMedians(d_se, tree, message = FALSE)
```

Arguments

d_se	A SummarizedExperiment object, with cells as rows and features as columns. This should be the output from generateClusters. The colData is assumed to contain a factor named marker_class.
dist_method	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. Please refer to method in dist for more information.
hclust_method	The agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC). Please refer to method in hclust for more information.
tree	A phylo object from buildTree.
message	A logical scalar indicating whether progress messages should be printed.

Details

The data object is assumed to contain a factor marker_class in the column meta-data (see prepareData), which indicates the protein marker class for each column of data ("type", "state", or "none").

Variables id_type_markers and id_state_markers are saved in the metadata slot of the output object. These can be used to identify the 'cell type' and 'cell state' markers in the list of assays in the output TreeSummarizedExperiment object, which is useful in later steps of the 'diffcyt' pipeline.

- buildTree applies hierarchical clustering to build a tree starting from the high-resolution clustering created by generateClusters. The function calculates the median abundance for each (ID type) marker and cluster, and uses this data to further aggregate the initial clusters using hierarchical clustering.
- calcTreeCounts calculates the number of cells per cluster-sample combination (referred to as cluster cell 'counts', 'abundances', or 'frequencies'. This is a tree version of calcCounts.
- calcMediansByTreeMarker calculates the median value for each cluster-marker combination. A cluster is represented by a node on the tree. This is a tree version of calcMediansByClusterMarker.
- calcTreeMedians calculates the median expression for each cluster-sample-marker combination. This is a tree version of calcMedians.

Value

- For buildTree, a phylo object representing the hierarchical clustering of the initial high-resolution clusters.
- For calcTreeCounts, a TreeSummarizedExperiment object, with clusters (nodes of the tree) in rows, samples in columns and abundances (counts) in an assay.
- For calcMediansByTreeMarker, a TreeSummarizedExperiment object with clusters (nodes of the tree) in rows and markers in columns. The marker expression values are in the assay.
- For calcTreeMedians, a TreeSummarizedExperiment object with median marker expression for each cluster (each node of the tree) and each sample for each cluster (node of the tree). Each node is represented as a separate assay, with the number of columns equal to the number of samples. The metadata slot contains variables id_type_markers and id_state_markers.

Author(s)

Ruizhu Huang

Examples

```
## For a complete workflow example demonstrating each step in the 'diffcyt'
## pipeline, please see the diffcyt vignette.
suppressPackageStartupMessages({
    library(diffcyt)
    library(TreeSummarizedExperiment)
})

## Helper function to create random data (one sample)
d_random <- function(n = 20000, mean = 0, sd = 1, ncol = 20, cofactor = 5) {
    d <- sinh(matrix(rnorm(n, mean, sd), ncol = ncol)) * cofactor
    colnames(d) <- paste0("marker", sprintf("%02d", seq_len(ncol)))
    d
}

## Create random data (without differential signal)
set.seed(123)
d_input <- list(
    sample1 = d_random(), sample2 = d_random(),
    sample3 = d_random(), sample4 = d_random()
```

```
)

experiment_info <- data.frame(
    sample_id = factor(paste0("sample", seq_len(4))),
    group_id = factor(c("group1", "group1", "group2", "group2"))
)

marker_info <- data.frame(
    channel_name = paste0("channel", sprintf("%03d", seq_len(20))),
    marker_name = paste0("marker", sprintf("%02d", seq_len(20))),
    marker_class = factor(c(rep("type", 10), rep("state", 10)),
                          levels = c("type", "state", "none"))
)

# Prepare data
d_se <- diffcyt::prepareData(d_input, experiment_info, marker_info)

# Transform data
d_se <- diffcyt::transformData(d_se)

# Generate clusters
d_se <- diffcyt::generateClusters(d_se)

# Build a tree
tr <- buildTree(d_se)

## Calculate abundances for nodes in each sample
d_counts_tree <- calcTreeCounts(d_se = d_se, tree = tr)

## Calculate medians (by cluster and marker)
d_medians_by_cluster_marker <-
    calcMediansByTreeMarker(d_se = d_se, tree = tr)

## Calculate medians (by cluster, sample and marker)
d_medians_tree <- calcTreeMedians(d_se = d_se, tree = tr)
```

---

edgerWrp *Wrapper applying an edgeR differential analysis workflow*

Description

edgerWrp is a wrapper using functions from the edgeR package (Robinson et al. 2010, *Bioinformatics*; McCarthy et al. 2012, *Nucleic Acids Research*) to fit models and perform a moderated test for each entity.

Usage

```
edgerWrp(
  count,
  lib_size = NULL,
```

```
  option = c("glm", "glmQL"),
  design,
  contrast = NULL,
  normalize = TRUE,
  normalize_method = "TMM",
  ...
)
```

Arguments

count	A matrix with features (e.g., genes or microbes) in rows and samples in columns.
lib_size	A numeric vector with library sizes for each sample. If NULL (default), the column sums of count are used.
option	Either "glm" or "glmQL". If "glm", glmFit and glmLRT are used; otherwise, glmQLFit and glmQLFTest are used. Details about the difference between the two options can be found in the help pages of glmQLFit.
design	A numeric design matrix, e.g. created by model.matrix. Please refer to design in glmQLFit and glmFit for more details.
contrast	A numeric vector specifying one contrast of the linear model coefficients to be tested. Its length must equal the number of columns of design. If NULL, the last coefficient will be tested. Please refer to contrast in glmQLFTest and glmLRT for more details.
normalize	A logical scalar, specifying whether normalization factors should be calculated (using calcNormFactors).
normalize_method	Normalization method to be used. Please refer to method in calcNormFactors for more details.
...	More arguments to pass to glmFit (if option = "glm" or glmQLFit (if option = "glmQL").

Details

The function performs the following steps:

- Create a DGEList object. If lib_size is given, set the library sizes to these values, otherwise use the column sums of the count matrix.
- If normalize is TRUE, estimate normalization factors using calcNormFactors.
- Estimate dispersions with estimateDisp.
- Depending on the value of option, apply either the LRT or QLF edgeR workflows (i.e., either glmFit + glmLRT or glmQLFit + glmQLFTest), testing for the specified contrast.

Value

The output of glmQLFTest or glmLRT depending on the specified option.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
})
## Read example data
x <- readRDS(system.file("extdata/da_sim_100_30_18de.rds",
                         package = "treeclimbR"))

## Run differential abundance analysis
out <- edgerWrp(count = assay(x), option = "glm",
                design = model.matrix(~ group, data = colData(x)),
                contrast = c(0, 1))

## The output is an edgeR DGELRT object
class(out)
```

`evalCand` *Evaluate candidate levels and select the optimal one*

Description

Evaluate all candidate levels proposed by `getCand` and select the one with best performance. For more details about how the scoring is done, see Huang et al (2021): https://doi.org/10.1186/s13059-021-02368-1.

Usage

```
evalCand(
  tree,
  type = c("single", "multiple"),
  levels,
  score_data = NULL,
  node_column,
  p_column,
  sign_column,
  feature_column = NULL,
  method = "BH",
  limit_rej = 0.05,
  use_pseudo_leaf = FALSE,
  message = FALSE
)
```

Arguments

`tree`	A `phylo` object.
`type`	A character scalar indicating whether the evaluation is for a DA-type workflow (set `type="single"`) or a DS-type workflow (set `type="multiple"`).

levels	A list of candidate levels that are returned by getCand. If type = "single", elements in the list are candidate levels, and are named by the value of the tuning parameter. If type = "multiple", a nested list is required and the list should be named by the feature (e.g., genes). In that case, each element is a list of candidate levels for that feature.
score_data	A data.frame (type = "single") or a list of data.frames (type = "multiple"). Each data.frame must have at least one column containing the node IDs (defined by node_column), one column with p-values (defined by p_column), one column with the direction of change (defined by sign_column) and one optional column with the feature (feature_column, for type="multiple").
node_column	The name of the column that contains the node information.
p_column	The name of the column that contains p-values of nodes.
sign_column	The name of the column that contains the direction of the (estimated) change.
feature_column	The name of the column that contains information about the feature ID.
method	method The multiple testing correction method. Please refer to the argument method in p.adjust. Default is "BH".
limit_rej	The desired false discovery rate threshold.
use_pseudo_leaf	A logical scalar. If FALSE, the FDR is calculated on the leaf level of the tree; If TRUE, the FDR is calculated on the pseudo-leaf level. The pseudo-leaf level is the level on which entities have sufficient data to run analysis and the that is closest to the leaf level.
message	A logical scalar, indicating whether progress messages should be printed.

Value

A list with the following components:

candidate_best The best candidate level

output Node-level information for best candidate level

candidate_list A list of candidates

level_info Summary information of all candidates

FDR The specified FDR level

method The method to perform multiple test correction.

column_info A list with the specified node, p-value, sign and feature column names

More details about the columns in level_info:

- t The thresholds.
- r The upper limit of t to control FDR on the leaf level.
- is_valid Whether the threshold is in the range to control leaf FDR.
- limit_rej The specified FDR.
- level_name The name of the candidate level.
- rej_leaf The number of rejections on the leaf level.
- rej_pseudo_leaf The number of rejected pseudo-leaf nodes.
- rej_node The number of rejections on the tested candidate level (leaves or internal nodes).

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
})

## Generate example tree and assign p-values and signs to each node
data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
   geom_text2(aes(label = node)) +
   geom_hilight(node = 13, fill = "blue", alpha = 0.5) +
   geom_hilight(node = 18, fill = "orange", alpha = 0.5)
set.seed(1)
pv <- runif(19, 0, 1)
pv[c(seq_len(5), 13, 14, 18)] <- runif(8, 0, 0.001)

fc <- sample(c(-1, 1), 19, replace = TRUE)
fc[c(seq_len(3), 13, 14)] <- 1
fc[c(4, 5, 18)] <- -1
df <- data.frame(node = seq_len(19),
                 pvalue = pv,
                 logFoldChange = fc)

## Propose candidates
ll <- getCand(tree = tinyTree, score_data = df,
              node_column = "node",
              p_column = "pvalue",
              sign_column = "logFoldChange")

## Evaluate candidates
cc <- evalCand(tree = tinyTree, levels = ll$candidate_list,
              score_data = ll$score_data, node_column = "node",
              p_column = "pvalue", sign_column = "logFoldChange",
              limit_rej = 0.05)

## Best candidate
cc$candidate_best

## Details for best candidate
cc$output
```

`fdr` *Calculate false discovery rate (FDR) on a tree structure*

Description

Calculate the false discovery rate on a tree structure, at either leaf or node level.

Usage

```
fdr(tree, truth, found, only.leaf = TRUE)
```

Arguments

tree	A phylo object.
truth	True signal nodes (e.g., nodes that are truly differentially abundant between experimental conditions). **Note:** when the FDR is requested at the leaf level (only.leaf = TRUE), the descendant leaves of the given nodes will be found and the FDR will be estimated on the leaf level.
found	Detected signal nodes (e.g., nodes that have been found to be differentially abundant via a statistical testing procedure). **Note:** when the FDR is requested at the leaf level (only.leaf = TRUE), the descendant leaves of the given nodes will be found out and the FDR will be estimated on the leaf level.
only.leaf	A logical scalar. If TRUE, the false discovery rate is calculated at the leaf (tip) level; otherwise it is calculated at the node level.

Value

The estimated false discovery rate.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(ggtree)
    library(TreeSummarizedExperiment)
})

data(tinyTree)

## Two branches are truly differential
ggtree(tinyTree, branch.length = "none") +
   geom_text2(aes(label = node)) +
   geom_hilight(node = 16, fill = "orange", alpha = 0.3) +
   geom_hilight(node = 13, fill = "blue", alpha = 0.3)

## FDR at the leaf level if nodes 14 and 15 are called differential (1/8)
fdr(tree = tinyTree, truth = c(16, 13),
    found = c(15, 14), only.leaf = TRUE)

## FDR at the node level if nodes 14 and 15 are called differential (2/14)
fdr(tree = tinyTree, truth = c(16, 13),
```

```
    found = c(15, 14), only.leaf = FALSE)
```

---

findChild *Find the children of an internal node in a tree*

Description

Find the direct children of an internal node in a tree.

Usage

```
findChild(tree, node, use.alias = FALSE)
```

Arguments

tree	A phylo object.
node	Either the node number or node label of an internal node of tree.
use.alias	A logical scalar. If FALSE (default), the node label is used to name the output; otherwise, the alias of the node label is used. The alias of the node label is created by adding a prefix "alias_" to the node number.

Value

A vector of nodes. The numeric value is the node number, and the vector name is the corresponding node label. If a node has no label, it would have NA as name when use.alias = FALSE, and have the alias of the node label as name when use.alias = TRUE.

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(ggtree)
    library(TreeSummarizedExperiment)
})

data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
    geom_text2(aes(label = node), color = "darkblue",
               hjust = -0.5, vjust = 0.7) +
    geom_text2(aes(label = label), color = "darkorange",
               hjust = -0.1, vjust = -0.7)

## Specify node numbers
findChild(tree = tinyTree, node = c(17, 12))
```

```
## Name return values using aliases
findChild(tree = tinyTree, node = c(17, 12), use.alias = TRUE)

## Specify node labels
findChild(tree = tinyTree, node = c("Node_17", "Node_12"))

## Tips have no children
findChild(tree = tinyTree, node = "t4")
```

`findExcl` *Find branches that are non-overlapping with specified branches in a tree*

Description

Find all branches whose leaves do not overlap with those of the specified branches.

Usage

```
findExcl(tree, node, use.alias = FALSE)
```

Arguments

`tree`	A `phylo` object.
`node`	A numeric or character vector specifying the nodes.
`use.alias`	A logical scalar. If `TRUE`, the alias name is used to name the output vector.

Value

A vector of node numbers

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(ggtree)
    library(TreeSummarizedExperiment)
})

data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
    geom_text2(aes(label = node)) +
    geom_hilight(node = 17, fill = "blue", alpha = 0.3) +
    geom_hilight(node = 13, fill = "orange", alpha = 0.3)
```

```
## Find branches whose leaves do not overlap with the two colored branches.
## The returned branches are represented at the highest tree level
## possible without including any of the forbidden branches.
findExcl(tree = tinyTree, node = c(17, 13))
```

---

getCand *Generate candidates for different thresholds*

Description

Generate candidates for different thresholds (t). A candidate consists of a disjoint collection of leaves and internal branches, that collectively cover all leaves in the tree, and represents a specific aggregation pattern along the tree.

Usage

```
getCand(
  tree,
  t = NULL,
  score_data,
  node_column,
  p_column,
  sign_column,
  threshold = 0.05,
  pct_na = 0.5,
  message = FALSE
)
```

Arguments

tree	A phylo object.
t	A vector of threshold values used to search for candidates, in the range [0, 1]. The default (NULL) uses a sequence c(seq(0, 0.04, by = 0.01), seq(0.05, 1, by = 0.05))
score_data	A data.frame including at least one column with node IDs (specified with the node_column argument), one column with p-values (specified with the p_column argument) and one column with directions of change (specified with the sign_column argument).
node_column	The name of the column of score_data that contains the node information.
p_column	The name of the column of score_data that contains p-values for nodes.
sign_column	The name of the column of score_data that contains the direction of change (e.g., the log-fold change). Only the sign of this column will be used.

threshold	Numeric scalar; any internal node where the value of the p-value column is above this value will not be returned. The default is 0.05. The aim of this threshold is to avoid arbitrarily picking up internal nodes without true signal.
pct_na	Numeric scalar. In order for an internal node to be eligible for selection, more than pct_na of its direct child nodes must have a valid (i.e., non-missing) value in the p_column column. Hence, increasing this number implies a more strict selection (in terms of presence of explicit values).
message	A logical scalar, indicating whether progress messages should be printed to the console.

Value

A list with two elements: candidate_list and score_data. condidate_list is a list of candidates obtained for the different thresholds. score_data is a data.frame that includes columns from the input score_data and additional columns with q-scores for different thresholds.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
})

data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
   geom_text2(aes(label = node)) +
   geom_hilight(node = 13, fill = "blue", alpha = 0.3) +
   geom_hilight(node = 18, fill = "orange", alpha = 0.3)

## Simulate p-values and directions of change for nodes
## (Nodes 1, 2, 3, 4, 5, 13, 14, 18 have a true signal)
set.seed(1)
pv <- runif(19, 0, 1)
pv[c(seq_len(5), 13, 14, 18)] <- runif(8, 0, 0.001)

fc <- sample(c(-1, 1), 19, replace = TRUE)
fc[c(seq_len(3), 13, 14)] <- 1
fc[c(4, 5, 18)] <- -1
df <- data.frame(node = seq_len(19),
                 pvalue = pv,
                 logFoldChange = fc)

ll <- getCand(tree = tinyTree, score_data = df,
              t = c(0.01, 0.05, 0.1, 0.25, 0.75),
              node_column = "node", p_column = "pvalue",
              sign_column = "logFoldChange")
```

```
## Candidates
ll$candidate_list

## Score table
ll$score_data
```

---

getData *Extract data from a TreeHeatmap*

Description

Extract different elements of the data shown in a figure generated with TreeHeatmap, such as the heatmap itself, or the row and column names.

Usage

```
getData(
  tree_hm,
 type = c("heatmap", "row_name", "column_name", "title", "column_anno", "column_order",
    "column_split")
)
```

Arguments

tree_hm	The output of TreeHeatmap.
type	A character scalar indicating the type of information to extract. Should be one of "heatmap", "row_name", "column_name","title", "column_anno", "column_order".

Value

A data.frame (if type is not "column_order"), or a vector of column names (if type is "column_order").

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
    library(ggplot2)
    library(scales)
    library(dplyr)
    library(ggnewscale)
})
```

```
## Load example data (tiny tree with corresponding count matrix)
tse <- readRDS(system.file("extdata", "tinytree_counts.rds",
                           package = "treeclimbR"))

## Aggregate counts for each of the highlighted subtrees
tseagg <- aggTSE(
    tse,
    rowLevel = c(13, 18,
                 setdiff(showNode(tinyTree, only.leaf = TRUE),
                         unlist(findDescendant(tinyTree, node = c(13, 18),
                                               only.leaf = TRUE)))))
ct <- SummarizedExperiment::assay(tseagg, "counts")
col_split <- ifelse(colnames(ct) %in% paste0("S", seq_len(5)), "A", "B")
names(col_split) <- colnames(ct)

## Prepare the tree figure
tree_fig <- ggtree(tinyTree, branch.length = "none",
                   layout = "rectangular") +
    geom_hilight(node = 18, fill = "orange", alpha = 0.3) +
    geom_hilight(node = 13, fill = "blue", alpha = 0.3)
fig <- TreeHeatmap(
    tree = tinyTree, tree_fig = tree_fig, hm_data = ct,
    cluster_column = TRUE, column_split = col_split,
    column_anno = col_split, column_anno_gap = 0.6,
    column_anno_color = c(A = "red", B = "blue"),
    show_colnames = TRUE, colnames_position = "bottom",
    colnames_angle = 90, colnames_size = 2, colnames_offset_y = -0.2,
    show_title = TRUE, title_offset_y = 1.5, title_color = "blue"
)
fig

## Extract data for heatmap
df_hm <- getData(tree_hm = fig, type = "heatmap")

## Generate data to add a column annotation
ct <- df_hm |>
    dplyr::select(x, width, variable) |>
    dplyr::distinct()
set.seed(1)
ann <- matrix(sample(LETTERS[seq_len(2)], size = 3 * ncol(df_hm),
                     replace = TRUE),
              nrow = 3)
rownames(ann) <- paste0("g", seq_len(3))
colnames(ann) <- ct$variable
ann <- data.frame(ann) |>
       mutate(y = min(df_hm$y) - seq_len(nrow(ann)),
       label = rownames(ann))
df_ann <- tidyr::pivot_longer(
    ann, names_to = "variable",
    values_to = "value", cols = -c("y", "label")) |>
    left_join(ct)

## Add new column annotation
```

```
fig +
    new_scale_fill() +
    geom_tile(data = df_ann, aes(x, y-0.5,
                                 width = width, fill = value)) +
    scale_fill_viridis_d() +
    geom_text(data = df_ann, aes(x = min(x) - 1, y = y - 0.5,
                label = label))
```

`getLevel` *Search for a target level on the tree via a specified score*

Description

Search for the target level of the tree via a specified score. The score value needs to be provided for each node of the tree.

Usage

```
getLevel(
  tree,
  score_data,
  drop,
  score_column,
  node_column,
  get_max,
  parent_first = TRUE,
  message = FALSE
)
```

Arguments

`tree`	A `phylo` object.
`score_data`	A `data.frame` providing scores for all nodes in the tree. The data frame should have at least 2 columns, one with information about nodes (the node number) and the other with the score for each node.
`drop`	A logical expression indicating elements or rows to keep. Missing values are taken as `FALSE`.
`score_column`	The name of the column of `score_data` that contains the original scores of the nodes.
`node_column`	The name of the column of `score_data` that contains the node numbers.
`get_max`	A logical scalar. If `TRUE`, search for nodes that has higher score value than its descendants; otherwise, search for nodes that has lower score value than its descendants.
`parent_first`	A logical scalar. If `TRUE`, the parent node is selected if tied values occur on the parent node and some of the children nodes.
`message`	A logical scalar indicating whether progress messages should be printed.

Value

A `data.frame` similar to `score_data` but with an additional column named `keep` indicating which nodes to retain.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
})

data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
    geom_text2(aes(label = node), color = "darkblue",
           hjust = -0.5, vjust = 0.7) +
    geom_text2(aes(label = label), color = "darkorange",
               hjust = -0.1, vjust = -0.7) +
    geom_hilight(node = 13, fill = "blue", alpha = 0.3) +
    geom_hilight(node = 16, fill = "orange", alpha = 0.3)

## Generate score for each node
pv <- rep(0.1, 19)
pv[c(16, 13, 17)] <- c(0.01, 0.05, 0.005)
out <- data.frame(node = 1:19, pvalue = pv)

## Search nodes
final <- getLevel(tree = tinyTree,
                  score_data = out,
                  drop =  pvalue > 0.05,
                  score_column = "pvalue",
                  node_column = "node",
                  get_max = FALSE,
                  parent_first = TRUE,
                  message = FALSE)

## Nodes to keep
final$node[final$keep]
```

`infoCand` *Get information of candidates*

Description

Extract information about candidates.

Usage

```
infoCand(object)
```

Arguments

object	An output object from evalCand.

Value

A data.frame with information about candidates.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
})

## Simulate some data
data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
   geom_text2(aes(label = node)) +
   geom_hilight(node = 13, fill = "blue", alpha = 0.3) +
   geom_hilight(node = 18, fill = "orange", alpha = 0.3)
set.seed(1)
pv <- runif(19, 0, 1)
pv[c(seq_len(5), 13, 14, 18)] <- runif(8, 0, 0.001)

fc <- sample(c(-1, 1), 19, replace = TRUE)
fc[c(seq_len(3), 13, 14)] <- 1
fc[c(4, 5, 18)] <- -1
df <- data.frame(node = seq_len(19),
                 pvalue = pv,
                 logFoldChange = fc)

## Get candidates
ll <- getCand(tree = tinyTree, score_data = df,
              node_column = "node",
              p_column = "pvalue",
              sign_column = "logFoldChange")

## Evaluate candidates
cc <- evalCand(tree = tinyTree, levels = ll$candidate_list,
              score_data = df, node_column = "node",
              p_column = "pvalue", sign_column = "logFoldChange",
              limit_rej = 0.05)

## Get summary info about candidates
```

```
out <- infoCand(object = cc)
out
```

`isConnect` *Check whether nodes are contained in the same path from a leaf to the root in a tree*

Description

Perform an elementwise check of whether two vectors of nodes are "connected" in specific ways in a tree. A pair of nodes are considered to be connected if they are part of the same path from a leaf to the root of the tree. They are considered directly connected if one node is the parent of the other, and indirectly connected otherwise.

Usage

```
isConnect(tree, node_a, node_b, connect = "any")
```

Arguments

`tree`	A `phylo` object.
`node_a`, `node_b`	The two vectors of nodes (either node numbers or node labels) to check for connections. The vectors should have the same length (if not, the shorter one will be recycled), as the check for connectivity is done elementwise.
`connect`	One of "any", "direct", "indirect", the type of connections to search for.

Value

A logical vector of the same length as `node_a` and `node_b`, where each element indicates whether the corresponding elements of `node_a` and `node_b` are connected in the specified way.

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(ggtree)
    library(TreeSummarizedExperiment)
})

data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
    geom_text2(aes(label = node))

node_a <- c(4, 18, 19, 2)
```

```
node_b <- c(4, 5, 6, 3)

isConnect(tree = tinyTree, node_a = node_a,
          node_b = node_b, connect = "any")
```

`medianByClusterMarker` *Calculate median values of markers for each cluster*

Description

Calculate median value of each marker in each cluster.

Usage

```
medianByClusterMarker(
  SE,
  assay = 1,
  marker_in_column = TRUE,
  column_cluster = "cluster_id",
  use_marker = NULL
)
```

Arguments

`SE`	A `SummarizedExperiment` object.
`assay`	A numeric index or assay name indicating with assay of `SE` to use to calculate medians.
`marker_in_column`	A logical scalar, indicating whether markers (genes, features) are in the columns of `SE` or not.
`column_cluster`	The name of the column of `colData(SE)` that contains the cluster assignment of each sample.
`use_marker`	A logical or numeric vector such that `SE[use_marker, ]` (if `marker_in_column = FALSE`) or `SE[, use_marker]` (if `marker_in_column = TRUE`) subsets `SE` to the markers that should be retained. If `NULL` (default), all markers are used.

Value

A `SummarizedExperiment` object containing the median value of each marker in each cluster.

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(SummarizedExperiment)
})

## Simulate data with 100 cells and 10 markers (5 type, 5 state markers)
set.seed(1)
count <- matrix(rpois(n = 1000, lambda = 10), nrow = 100)
colnames(count) <- paste0("mk", 1:10)
rowD <- data.frame("cluster" = sample(seq_len(6), 100, replace = TRUE))
colD <- data.frame(type_marker = rep(c(FALSE, TRUE), each = 5))

## SE with markers in columns
d_se <- SummarizedExperiment(assays = list(counts = count),
                             rowData = rowD,
                             colData = colD)
medianByClusterMarker(SE = d_se, marker_in_column = TRUE,
                      column_cluster = "cluster",
                      use_marker = colData(d_se)$type_marker)

## SE with markers in rows
d_se <- SummarizedExperiment(assays = list(counts = t(count)),
                             rowData = colD,
                             colData = rowD)
medianByClusterMarker(SE = d_se, marker_in_column = FALSE,
                      column_cluster = "cluster",
                      use_marker = rowData(d_se)$type_marker)
```

`nodeResult` *Extract table with node-level DA/DS results*

Description

Extract a table with the top-ranked nodes from a DA/DS analysis output (generated by `runDA` or `runDS`).

Usage

```
nodeResult(
  object,
  n = 10,
  type = c("DA", "DS"),
  adjust_method = "BH",
  sort_by = "PValue",
  p_value = 1
)
```

Arguments

object	The output from runDA or runDS.
n	An integer indicating the maximum number of entities to return.
type	Either "DA" (for **object** from runDA) or "DS" (for **object** from runDS).
adjust_method	A character string specifying the method used to adjust p-values for multiple testing. See p.adjust for possible values.
sort_by	A character string specifying the sorting method. This will be passed to topTags. Possibilities are "PValue" for p-value, "logFC" for absolute log-fold change or "none" for no sorting.
p_value	A numeric cutoff value for adjusted p-values. This will be passed to topTags. Only entities with adjusted p-values equal or lower than specified are returned.

Value

A data frame with results for all nodes passing the imposed thresholds. The columns **logFC**, **logCPM**, **PValue**, **FDR**, **F** (or **LR**) are from (the output table of) topTags. The **node** column stores the node number for each entity. Note: **FDR** is corrected over all features and nodes when the specified type = "DS".

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
})

lse <- readRDS(system.file("extdata", "da_sim_100_30_18de.rds",
                           package = "treeclimbR"))
tse <- aggTSE(lse, rowLevel = showNode(tree = rowTree(lse),
                                       only.leaf = FALSE))
dd <- model.matrix( ~ group, data = colData(tse))
out <- runDA(TSE = tse, feature_on_row = TRUE,
             assay = "counts", option = "glmQL",
             design = dd, contrast = NULL,
             normalize = TRUE)

## Top 10 nodes with DA
nodeResult(out, n = 10)
```

parEstimate *Parameter estimation for Dirichlet-multinomial distribution*

Description

parEstimate is a wrapper of the function dirmult with default settings for init, initscalar, epsilon, trace and mode. It allows the input obj to be either a matrix or a TreeSummarizedExperiment and outputs the estimated values of pi and theta.

Usage

```
parEstimate(obj, assay = NULL)
```

Arguments

obj	A matrix or TreeSummarizedExperiment, with samples in the columns and entities in the rows.
assay	If obj is a TreeSummarizedExperiment, the name or index of the assay to use to estimate Dirichlet multinomial parameters. If NULL, the first assay will be used.

Value

A list including the estimates of pi (a vector with one element per row in obj) and theta (a scalar).

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
})

set.seed(1L)
y <- matrix(rnbinom(200, size = 1, mu = 10), nrow = 10)
colnames(y) <- paste("S", seq_len(20), sep = "")
rownames(y) <- tinyTree$tip.label
toy_tse <- TreeSummarizedExperiment(rowTree = tinyTree,
                                    assays = list(y))
res <- parEstimate(obj = toy_tse)
metadata(res)$assays.par
```

runDA *Test for differential abundance using edgeR*

Description

Test for differential abundance of entities using functions from the edgeR package. This adapts edgerWrp to accept input as a TreeSummarizedExperiment (TSE) object instead of a matrix. Features could be represented in either rows or columns. By default, features are in the rows. Then, samples are in columns and the sample information is in colData. The tree that stores the hierarchical information about features is in rowTree. Each row of the assays can be mapped to a node of the tree. Data on rows that are mapped to internal nodes is generated from data on leaf nodes. Normalization for samples is automatically performed by edgeR and the library size is calculated using features that are mapped to leaf nodes.

Usage

```
runDA(
  TSE,
  feature_on_row = TRUE,
  assay = NULL,
  option = c("glm", "glmQL"),
  design = NULL,
  contrast = NULL,
  filter_min_count = 10,
  filter_min_total_count = 15,
  filter_large_n = 10,
  filter_min_prop = 0.7,
  normalize = TRUE,
  normalize_method = "TMM",
  group_column = "group",
  design_terms = "group",
  ...
)
```

Arguments

TSE	A TreeSummarizedExperiment object.
feature_on_row	A logical scalar. If TRUE (default), features or entities (e.g. genes, OTUs) are in rows of the assays tables, and samples are in columns; otherwise, it's the other way around.
assay	A numeric index or assay name to specify which assay from assays is used for analysis.
option	Either "glm" or "glmQL". If "glm", glmFit and glmLRT are used; otherwise, glmQLFit and glmQLFTest are used. Details about the difference between two options are in the help page of glmQLFit.

`design` A numeric design matrix. If `NULL`, all columns of the sample annotation will be used to create the design matrix.

`contrast` A numeric vector specifying one contrast of the linear model coefficients to be tested equal to zero. Its length must equal to the number of columns of design. If `NULL`, the last coefficient will be tested equal to zero.

`filter_min_count`
A numeric value, passed to **min.count** of `filterByExpr`.

`filter_min_total_count`
A numeric value, passed to **min.total.count** of `filterByExpr`.

`filter_large_n` A numeric value, passed to **large.n** of `filterByExpr`.

`filter_min_prop`
A numeric value, passed to **min.prop** of `filterByExpr`.

`normalize` A logical scalar indicating whether to estimate normalization factors (using `calcNormFactors`).

`normalize_method`
Normalization method to be used. See `calcNormFactors` for more details.

`group_column` The name of the column in the sample annotation providing group labels for samples (currently not used).

`design_terms` The names of columns from the sample annotation that will be used to generate the design matrix. This is ignored if **design** is provided.

`...` More arguments to pass to `glmFit` (`option = "glm"` or `glmQLFit` (`option = "glmQL"`).

Details

The experimental design is specified by a design matrix and provided via the argument `design`. More details about the calculation of normalization factor could be found from `calcNormFactors`.

Value

A list with entries **edgeR_results**, **tree**, and **nodes_drop**.

edgeR_results The output of `glmQLFTest` or `glmLRT` depending on the specified `option`.

tree The hierarchical structure of entities that was stored in the input `TSE`.

nodes_drop A vector storing the alias node labels of entities that are filtered before analysis due to low counts.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
})
```

```
## Load example data set
lse <- readRDS(system.file("extdata", "da_sim_100_30_18de.rds",
                           package = "treeclimbR"))

## Aggregate counts on internal nodes
nodes <- showNode(tree = tinyTree, only.leaf = FALSE)
tse <- aggTSE(x = lse, rowLevel = nodes)

dd <- model.matrix(~ group, data = colData(tse))
out <- runDA(TSE = tse, feature_on_row = TRUE,
             assay = 1, option = "glmQL",
             design = dd, contrast = NULL,
             normalize = TRUE, filter_min_count = 2)
names(out)
out$nodes_drop
edgeR::topTags(out$edgeR_results, sort.by = "PValue")
```

---

runDS *Test for differential state using edgeR*

Description

Test for differential state of entities using functions from the edgeR package. This adapts edgerWrp to accept input as a SummarizedExperiment (SE) object instead of matrix. Each assay should correspond to data for one node of the tree. Samples are in columns and features are in rows. The sample information is in colData. The tree that stores the hierarchical relation between the assays is provided via the argument tree.

Usage

```
runDS(
  SE,
  tree,
  option = c("glm", "glmQL"),
  design = NULL,
  contrast = NULL,
  filter_min_count = 1,
  filter_min_total_count = 15,
  filter_large_n = 10,
  filter_min_prop = 1,
  min_cells = 10,
  normalize = TRUE,
  normalize_method = "TMM",
  group_column = "group_id",
  design_terms = "group_id",
  message = TRUE,
  ...
)
```

Arguments

SE	A SummarizedExperiment object, typically generated by aggDS.
tree	A phylo object. Each **assay** of SE stores data for one node of the tree.
option	Either "glm" or "glmQL". If "glm", glmFit and glmLRT are used; otherwise, glmQLFit and glmQLFTest are used. Details about the difference between two options are in the help page of glmQLFit.
design	A numeric design matrix. If NULL, all columns of the sample annotation will be used to create the design matrix.
contrast	A numeric vector specifying one contrast of the linear model coefficients to be tested equal to zero. Its length must equal to the number of columns of design. If NULL, the last coefficient will be tested equal to zero.
filter_min_count	A numeric value, passed to **min.count** of filterByExpr.
filter_min_total_count	A numeric value, passed to **min.total.count** of filterByExpr.
filter_large_n	A numeric value, passed to **large.n** of filterByExpr.
filter_min_prop	A numeric value, passed to **min.prop** of filterByExpr.
min_cells	A numeric scalar specifying the minimal number of cells in a node required to include a node in the analysis. The information about the number of cells per node and sample should be available in metadata(SE)$n_cells. A node is retained if at least half of the samples have at least min_cells cells belonging to the node.
normalize	A logical scalar indicating whether to estimate normalization factors (using calcNormFactors).
normalize_method	Normalization method to be used. See calcNormFactors for more details.
group_column	The name of the column in the sample annotation providing group labels for samples. This annotation is used for filtering.
design_terms	The names of columns from the sample annotation that will be used to generate the design matrix. This is ignored if **design** is provided.
message	A logical scalar, indicating whether progress messages should be printed.
...	More arguments to pass to glmFit (option = "glm" or glmQLFit (option = "glmQL").

Value

A list with entries **edgeR_results**, **tree**, and **nodes_drop**.

edgeR_results A list. Each of the elements contains the output of glmQLFTest or glmLRT for one node, depending on the specified option.

tree The hierarchical structure of entities that was stored in the input SE.

nodes_drop A vector storing the alias node labels of entities that are filtered before analysis due to low counts.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
})
## Load example data
ds_tse <- readRDS(system.file("extdata", "ds_sim_20_500_8de.rds",
                              package = "treeclimbR"))
ds_se <- aggDS(TSE = ds_tse, assay = "counts", sample_id = "sample_id",
               group_id = "group", cluster_id = "cluster_id", FUN = sum)
## Information about the number of cells is provided in the metadata
S4Vectors::metadata(ds_se)$n_cells

ds_res <- runDS(SE = ds_se, tree = colTree(ds_tse), option = "glmQL",
                group_column = "group", contrast = c(0, 1),
                filter_min_count = 0, filter_min_total_count = 1,
                design = model.matrix(~ group, data = colData(ds_se)),
                filter_min_prop = 0, min_cells = 5, message = FALSE)
## Top differential features (across nodes)
nodeResult(ds_res, type = "DS")
```

---

selNode *Select branches meeting certain criteria*

Description

Select branches in a tree meeting the specified criteria in terms of number of leaves and the count proportion. Note that only internal branch nodes are considered - no individual leaves will be returned.

Usage

```
selNode(
  pr = NULL,
  obj = NULL,
  assay = 1,
  data = NULL,
  tree = NULL,
  minTip = 0,
  maxTip = Inf,
  minPr = 0,
  maxPr = 1,
  skip = NULL,
  all = FALSE
)
```

Arguments

pr	A named numeric vector to provide proportions of entities. If this is provided, obj and data will be ignored.
obj	A TreeSummarizedExperiment object. Only used if pr is NULL.
assay	The index or name of the assay of obj to use for estimating node count proportions. Only used if obj is not NULL.
data	Either a count table with entities in rows and samples in columns, or a list with pi and theta estimates (the output of parEstimate). Only used if pr and obj are NULL.
tree	A phylo object. If obj is used as input, the tree will be extracted from the rowTree of obj.
minTip	the minimum number of leaves in the selected branch.
maxTip	The maximum number of leaves in the selected branch.
minPr	The minimum count proportion of the selected branch in a sample. A value between 0 and 1.
maxPr	The maximum count proportion of the selected branch in a sample. A value between 0 and 1.
skip	A character vector of node labels. These nodes can not be descendants or the ancestors of the selected branch.
all	A logical scalar. If FALSE (default), the branch node of a single branch, which meets the requirements and has the minimum count proportion of branches meeting the requirements, is returned; otherwise branch nodes of all branches meeting the requirements are returned.

Value

A data.frame with node information for the selected internal node(s).

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
})

## Generate example data
set.seed(1)
data(tinyTree)
toyTable <- matrix(rnbinom(40, size = 1, mu = 10), nrow = 10)
colnames(toyTable) <- paste(rep(LETTERS[seq_len(2)], each = 2),
                            rep(seq_len(2), 2), sep = "_")
rownames(toyTable) <- tinyTree$tip.label

## Estimate entity proportions from count matrix under a Dirichlet
```

```
## Multinomial framework, and use this as the input for selNode
dat <- parEstimate(obj = toyTable)
selNode(tree = tinyTree, data = dat, all = TRUE)
selNode(tree = tinyTree, data = dat,
        minTip = 4, maxTip = 9, minPr = 0, maxPr = 0.8, all = TRUE)

## Alternatively, directly provide the proportions vector
selNode(tree = tinyTree, pr = dat$pi, all = TRUE)

## Return only branch with lowest proportion among valid ones
selNode(tree = tinyTree, pr = dat$pi, all = FALSE)

## Start instead from a TreeSummarizedExperiment object
lse <- TreeSummarizedExperiment(rowTree = tinyTree,
                                assays = list(counts = toyTable))
selNode(obj = lse, assay = "counts", all = TRUE)

## Don't allow node 1 to be included
selNode(obj = lse, assay = "counts", skip = 1, all = TRUE)
```

`simData` *Simulate different scenarios of abundance change in entities*

Description

Simulate a data set with different abundance patterns for entities under different conditions. These entities have their corresponding nodes on a tree.

Usage

```
simData(
  tree = NULL,
  data = NULL,
  obj = NULL,
  assay = NULL,
  scenario = "BS",
  from.A = NULL,
  from.B = NULL,
  minTip.A = 0,
  maxTip.A = Inf,
  minTip.B = 0,
  maxTip.B = Inf,
  minPr.A = 0,
  maxPr.A = 1,
  ratio = 4,
  adjB = NULL,
  pct = 0.6,
  nSam = c(50, 50),
```

```
  mu = 10000,
  size = NULL,
  n = 1,
  FUN = sum,
  message = FALSE
)
```

Arguments

tree	A phylo object. Only used when obj is NULL.
data	A count matrix with entities corresponding to tree leaves in the rows and samples in the columns. Only used when obj is NULL.
obj	A TreeSummarizedExperiment object with observed data to use as the input for the simulation. If NULL, data and \ tree must be provided instead.
assay	If obj is not NULL, a numeric index or character scalar indicating which assay of the object to use as the basis for simulation. If assay is NULL, the first assay in the object is used.
scenario	The simulation scenario, either "BS", "US", or "SS" (see **Details**).
from.A, from.B	The branch node labels of branches A and B for which the signal will be swapped. By default, both are NULL, in which case they will be chosen based on the restrictions provided (minTip.A, maxTip.A, minTip.B, maxTip.B, minPr.A, maxPr.A, ratio). Note: If from.A is NULL, from.B is also set to NULL.
minTip.A	The minimum number of leaves allowed in branch A.
maxTip.A	The maximum number of leaves allowed in branch A.
minTip.B	The minimum number of leaves allowed in branch B.
maxTip.B	The maximum number of leaves allowed in branch B.
minPr.A	A numeric value in [0, 1]. The minimum abundance proportion of leaves in branch A.
maxPr.A	A numeric value in [0, 1]. The maximum abundance proportion of leaves in branch A.
ratio	A numeric value. The proportion ratio of branch B to branch A. This value is used to select branches(see **Details**). If there are no branches having exactly this ratio, the pair with the value closest to ratio will be selected.
adjB	A numeric value in [0, 1] (only for scenario "SS"), or NULL. If NULL, branch A and the selected part of branch B swap their proportions. If a numeric value, e.g. 0.1, then the counts for the selected part of branch B decreases to 10 the original value, and this decrease is added to branch A. For example, assume there are two experimental conditions (C1 & C2), branch A has a count of 10 and branch B has a count of 40 in C1. If adjB is set to 0.1, then in C2 branch B becomes 4 and branch A 46 so that the total count of the two branches stays the same.
pct	The percentage of leaves in branch B that have differential abundance under different conditions (only for scenario "SS").
nSam	A numeric vector of length 2, indicating the sample size for each of the two simulated conditions.

mu, size
: The parameters of the Negative Binomial distribution. (see mu and size in rnbinom). These parameters are used to generate the library size for each simulated sample. If size is not specified, mu should be a vector of numbers from which the library size is sampled with replacement.

n
: A numeric value to specify how many count tables would be generated with the same settings. The default is 1, i.e., one count table would be obtained at the end. If greater than 1, the output is a list of matrices.

FUN
: A function to calculate the aggregated count at each internal node based on its descendant leaves (e.g., sum, mean). The argument of the function should be a numeric vector with the counts of an internal node's descendant leaves.

message
: A logical scalar, indicating whether progress messages should be printed to the console.

Details

Simulate a count table for entities which are corresponding to the nodes of a tree. The entities are in rows and the samples from different groups or conditions are in columns. The library size of each sample is sampled from a Negative Binomial distribution with mean and size specified by the arguments mu and size. The counts of entities, that are mapped to the leaf nodes, in a sample are assumed to follow a Dirichlet-Multinomial distribution. The parameters for the Dirichlet-Multinomial distribution are estimated from a real data set specified by data via the function dirmult (see dirmult). To generate different abundance patterns under different conditions, we provide three different scenarios, "BS", "US", and "SS" (specified via scenario).

- BS: two branches are selected to swap their proportions, and leaves on the same branch have the same fold change.
- US: two branches are selected to swap their proportions. Leaves in the same branch have different fold changes but same direction (either increase or decrease).
- SS: two branches are selected. One branch has its proportion swapped with the proportion of some leaves from the other branch.

Value

a TreeSummarizedExperiment object.

- **assays** A list of count matrices, with entities in rows and samples in columns. Each row can be mapped to a node of the tree.
- **rowData** Annotation data for the rows.
- **colData** Annotation data for the columns.
- **rowTree** The tree structure of entities.
- **rowLinks** The link between rows and nodes on the tree.
- **metadata** More details about the simulation.
  - **FC** the fold change of entities corresponding to the tree leaves.
  - **Branch** the information about two selected branches.
    - **A** The branch node label (or number) of branch A.
    - **B** The branch node label (or number) of branch B.

- **ratio** The count proportion ratio of branch B to branch A.
- **A_tips** The number of leaves on branch A.
- **B_tips** The number of leaves on branch B.
- **A_prop** The count proportion of branch A.
- **B_prop** The count proportion of branch B.

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
})
## Generate data to use as the starting point (this would usually be a
## real data set)
set.seed(1L)
y <- matrix(rnbinom(120, size = 1, mu = 10), nrow = 10)
colnames(y) <- paste("S", seq_len(12), sep = "")
rownames(y) <- tinyTree$tip.label

toy_lse <- TreeSummarizedExperiment(rowTree = tinyTree,
                                    assays = list(counts = y))
simData(obj = toy_lse, ratio = 2, scenario = "BS", pct = 0.5)
```

---

topNodes *Generate a table of top-ranked entities (nodes)*

Description

Generate a table of top-ranked nodes from the optimal resolution candidate of entities on a tree.

Usage

```
topNodes(
  object,
  n = 10,
  sort_by = NULL,
  sort_decreasing = FALSE,
  sort_by_absolute = FALSE,
  p_value = 1
)
```

Arguments

`object`	An output object from evalCand.
`n`	An integer, the maximum number of entities to return.
`sort_by`	A character string specifying the column of `object$output` to sort by. Set to `NULL` to return without sorting.
`sort_decreasing`	A logical value indicating whether to sort by decreasing value of the `sort_by` column.
`sort_by_absolute`	A logical value indicating whether to take the absolute value of the `sort_by` column before sorting.
`p_value`	A numeric cutoff value for adjusted p-values. Only entities with adjusted p-values equal or lower than specified are returned.

Value

A `data.frame` with test results. The **node** column stores the node number for each entity.

Author(s)

Ruizhu Huang, Charlotte Soneson

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
})

data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
   geom_text2(aes(label = node)) +
   geom_hilight(node = 13, fill = "blue", alpha = 0.3) +
   geom_hilight(node = 18, fill = "orange", alpha = 0.3)
set.seed(1)
pv <- runif(19, 0, 1)
pv[c(seq_len(5), 13, 14, 18)] <- runif(8, 0, 0.001)

fc <- sample(c(-1, 1), 19, replace = TRUE)
fc[c(seq_len(3), 13, 14)] <- 1
fc[c(4, 5, 18)] <- -1
df <- data.frame(node = seq_len(19),
                 pvalue = pv,
                 logFoldChange = fc)
ll <- getCand(tree = tinyTree, score_data = df,
              node_column = "node",
              p_column = "pvalue",
              sign_column = "logFoldChange")
cc <- evalCand(tree = tinyTree, levels = ll$candidate_list,
              score_data = df, node_column = "node",
```

```
                p_column = "pvalue", sign_column = "logFoldChange",
                limit_rej = 0.05)

## Unsorted result table
topNodes(cc)

## Sort by p-value in increasing order
topNodes(cc, sort_by = "pvalue")
```

---

tpr *Calculate true positive rate (TPR) on a tree structure*

Description

Calculate the true positive rate on a tree structure, at either leaf or node level.

Usage

```
tpr(tree, truth, found, only.leaf = TRUE)
```

Arguments

tree	A phylo object.
truth	True signal nodes (e.g., nodes that are truly differentially abundant between experimental conditions). **Note:** when the TPR is requested at the leaf level (only.leaf = TRUE), the descendant leaves of the given nodes will be found and the TPR will be estimated on the leaf level.
found	Detected signal nodes (e.g., nodes that have been found to be differentially abundant via a statistical testing procedure). **Note:** when the TPR is requested at the leaf level (only.leaf = TRUE), the descendant leaves of the given nodes will be found out and the TPR will be estimated on the leaf level.
only.leaf	A logical scalar. If TRUE, the false discovery rate is calculated at the leaf (tip) level; otherwise it is calculated at the node level.

Value

The estimated true positive rate.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(ggtree)
    library(TreeSummarizedExperiment)
})

data("tinyTree")

## Two branches are truly differential
ggtree(tinyTree, branch.length = "none") +
   geom_text2(aes(label = node)) +
   geom_hilight(node = 16, fill = "orange", alpha = 0.3) +
   geom_hilight(node = 13, fill = "blue", alpha = 0.3)

## TPR at the leaf level if nodes 14 and 15 are called differential  (7/8)
tpr(tree = tinyTree, truth = c(16, 13),
    found = c(15, 14), only.leaf = TRUE)

## TPR at the node level if nodes 14 and 15 are called differential (12/14)
tpr(tree = tinyTree, truth = c(16, 13),
    found = c(15, 14), only.leaf = FALSE)
```

TreeHeatmap *Generate a heatmap corresponding to an arbitrary aggregation level of a tree*

Description

Generate a heatmap corresponding to an arbitrary aggregation level of a tree.

Usage

```
TreeHeatmap(
  tree,
  tree_fig,
  hm_data,
  tree_hm_gap = 0,
  rel_width = 1,
  cell_line_color = NA,
  cell_line_size = 0,
  column_order = NULL,
  column_split = NULL,
  column_split_gap = 0.2,
  column_split_label = NULL,
  split_label_fontface = "bold",
  split_label_color = "black",
  split_label_size = 3,
```

```
  split_label_angle = 0,
  split_label_offset_x = 0,
  split_label_offset_y = 2,
  split_label_hjust = 0.5,
  split_label_vjust = 0,
  column_anno = NULL,
  column_anno_size = 1,
  column_anno_color = NULL,
  column_anno_gap = 0.1,
  legend_title_hm = "Expression",
  legend_title_column_anno = "group",
  show_colnames = FALSE,
  colnames_position = "top",
  colnames_angle = 0,
  colnames_offset_x = 0,
  colnames_offset_y = 0,
  colnames_size = 4,
  colnames_hjust = 0.5,
  show_rownames = FALSE,
  rownames_position = "right",
  rownames_angle = 0,
  rownames_offset_x = 0,
  rownames_offset_y = 0,
  rownames_size = 4,
  rownames_hjust = 0.5,
  rownames_label = NULL,
  show_title = FALSE,
  title_hm = "First heatmap",
  title_fontface = "bold",
  title_color = "black",
  title_size = 3,
  title_angle = 0,
  title_offset_x = 0,
  title_offset_y = 2,
  title_hjust = 0.5,
  cluster_column = FALSE,
  dist_method = "euclidean",
  hclust_method = "ave",
  show_row_tree = TRUE
)
```

Arguments

tree	A phylo object.
tree_fig	A ggtree object corresponding to tree. This will be used to represent the tree in the resulting figure.
hm_data	A data.frame with the values to show in the heatmap. The row names should correspond to the nodes of **tree**.

`tree_hm_gap` A numeric scalar specifying the gap between the tree and the heatmap.

`rel_width` A numeric scalar specifying the width of heatmap relative to the width of the tree. For example, if `rel_width = 1`, the width of the heatmap is the same as the width of the tree.

`cell_line_color`
A color for the lines separating cells in the heatmap.

`cell_line_size` A numeric scalar specifying the line width for lines separating cells in the heatmap.

`column_order` A character vector specifying the display order of the columns in the heatmap. Should correspond to the column names of `hm_data`. Ignored when **column_split** is provided.

`column_split` A named character vector that provides the grouping information used to split the columns in the heatmap. The names should correspond to the column names of `hm_data`.

`column_split_gap`
A numeric scalar specifying the gap between the groups of split columns in the heatmap.

`column_split_label`
A named character vector to label the column split. The names should correspond to the values in `column_split`.

`split_label_fontface`
The fontface of the labels of the column split.

`split_label_color`
The color of the the labels of the column split.

`split_label_size`
The size of the the labels of the column split.

`split_label_angle`
The angle of the the labels of the column split.

`split_label_offset_x`
A numeric value to shift the labels of the column split along the x-axis.

`split_label_offset_y`
A numeric value to shift the labels of the column split along the y-axis.

`split_label_hjust`
The horizontal justification for the labels of the column split: e.g. 0 (left aligned); 0.5 (centered); 1 (right aligned).

`split_label_vjust`
Similar to `split_label_hjust`, but controls vertical justification.

`column_anno` A named vector to specify labels that are used to annotate the columns of heatmap.

`column_anno_size`
A numeric value to specify the size of the annotation bar.

`column_anno_color`
A named vector to specify colors that are used to annotate the columns of the heatmap.

`column_anno_gap`
A numeric value to specify the gap between the column annotation bar and the heatmap.

`legend_title_hm`
: The legend title of the heatmap.

`legend_title_column_anno`
: The legend title of the column annotation.

`show_colnames`
: A logical value to specify whether column names should be displayed.

`colnames_position`
: The position of column names, either "top" or "bottom".

`colnames_angle`
: A numeric scalar specifying the angle of column names.

`colnames_offset_x`
: A numeric value to shift column names on the x-axis.

`colnames_offset_y`
: A numeric value to shift column names on the y-axis.

`colnames_size`
: A numeric value to specify the size of column names.

`colnames_hjust`
: The horizontal justification for column names: e.g. 0 (left aligned); 0.5 (centered); 1 (right aligned).

`show_rownames`
: A logical value to specify whether row names should be displayed.

`rownames_position`
: The position of the row names, either "right" or "left".

`rownames_angle`
: A numeric value specifying the angle of row names.

`rownames_offset_x`
: A numeric value to shift row names on the x-axis.

`rownames_offset_y`
: A numeric value to shift row names on the y-axis.

`rownames_size`
: A numeric value to specify the size of row names.

`rownames_hjust`
: The horizontal justification for row names: e.g. 0 (left aligned); 0.5 (centered); 1 (right aligned).

`rownames_label`
: A named vector to annotate the rows of the heatmap instead of the row names of **hm_data**.

`show_title`
: A logical value to specify whether the title should be displayed.

`title_hm`
: The title of the heatmap.

`title_fontface`
: The fontface of the title.

`title_color`
: The color of the title.

`title_size`
: The size of the title.

`title_angle`
: The angle of the title.

`title_offset_x`
: A numeric value to shift the title along the x-axis.

`title_offset_y`
: A numeric value to shift the title along the y-axis.

`title_hjust`
: The horizontal justification for the title: e.g. 0 (left aligned); 0.5 (centered); 1 (right aligned).

`cluster_column`
: A logical scalar, specifying whether columns of the heatmap should be clustered by similarity. This is ignored when **column_order** is given.

`dist_method`
: See **method** in `dist`. The distance method used for clustering columns.

`hclust_method`
: See **method** in `hclust`. The clustering method used for clustering columns.

`show_row_tree`
: A logical scalar (default `TRUE`). If `FALSE`, the figure provided in `tree_fig` is not shown.

Value

A ggtree object.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
    library(ggplot2)
    library(scales)
})

## Load example data (tiny tree with corresponding count matrix)
tse <- readRDS(system.file("extdata", "tinytree_counts.rds",
                           package = "treeclimbR"))

## Prepare the tree figure
tree_fig <- ggtree(rowTree(tse), branch.length = "none",
                   layout = "rectangular") +
    geom_hilight(node = 18, fill = "orange", alpha = 0.3) +
    geom_hilight(node = 13, fill = "blue", alpha = 0.3)
tree_fig

## Simple heatmap with tree
TreeHeatmap(tree = rowTree(tse), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tse, "counts"))

## Aggregate counts for each of the highlighted subtrees
tseagg <- aggTSE(
    tse,
    rowLevel = c(13, 18,
                 setdiff(showNode(tinyTree, only.leaf = TRUE),
                         unlist(findDescendant(tinyTree, node = c(13, 18),
                                               only.leaf = TRUE)))))

## Visualize aggregated heatmap with tree
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tseagg, "counts"))

## Cluster columns
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tseagg, "counts"),
            cluster_column = TRUE)

## Split columns
col_split <- ifelse(colnames(tseagg) %in% paste0("S", seq_len(5)), "A", "B")
names(col_split) <- colnames(tseagg)
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
```

```
            hm_data = SummarizedExperiment::assay(tseagg, "counts"),
            cluster_column = TRUE, column_split = col_split)

## Annotate columns
col_anno <- col_split
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tseagg, "counts"),
            cluster_column = TRUE, column_split = col_split,
            column_anno = col_anno, column_anno_gap = 1)

## Change annotation colors
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tseagg, "counts"),
            cluster_column = TRUE, column_split = col_split,
            column_anno = col_anno, column_anno_gap = 1,
            column_anno_color = c(A = "red", B = "blue"))

## Add column names
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tseagg, "counts"),
            cluster_column = TRUE, column_split = col_split,
            column_anno = col_anno, column_anno_gap = 1,
            column_anno_color = c(A = "red", B = "blue"),
            show_colnames = TRUE, colnames_position = "bottom",
            colnames_angle = 90, colnames_size = 2,
            colnames_offset_y = -0.4)

## Add title
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tseagg, "counts"),
            cluster_column = TRUE, column_split = col_split,
            column_anno = col_anno, column_anno_gap = 1,
            column_anno_color = c(A = "red", B = "blue"),
            show_colnames = TRUE, colnames_position = "bottom",
            colnames_angle = 90, colnames_size = 2,
            colnames_offset_y = -0.4,
            show_title = TRUE, title_offset_y = 2,
            title_color = "blue")

## Change colors
TreeHeatmap(tree = rowTree(tseagg), tree_fig = tree_fig,
            hm_data = SummarizedExperiment::assay(tseagg, "counts"),
            cluster_column = TRUE, column_split = col_split,
            column_anno = col_anno, column_anno_gap = 1,
            column_anno_color = c(A = "red", B = "blue"),
            show_colnames = TRUE, colnames_position = "bottom",
            colnames_angle = 90, colnames_size = 2,
            colnames_offset_y = -0.4,
            show_title = TRUE, title_offset_y = 2,
            title_color = "blue") +
            scale_fill_gradientn(
                colours = c("blue", "yellow", "red"),
                values = scales::rescale(c(5, 8, 10)),
```

```
                guide = "colorbar", limits = c(5, 10))
```

`treeScore` *Generate weighted tree score accounting for the family effect*

Description

`treeScore` takes the tree structure into account when calculating the score for an internal node. If an internal node A has two children B and C, (`A->B`, `A->C`), the new score at node A would be calculated as the weighted mean of the scores in the whole family (A, B and C). The weights are based on the number of descendant leaves. For example, if the node B has 2 descendant leaves, and C has 3 descendant leaves, then A would have 5. The calculation would be $(Score\_A * 5 + Score\_B * 2 + Score\_C * 3)/10$. The generation starts from the leaves and the new generated scores are used to update those in higher level of the tree until the root is reached.

Usage

```
treeScore(tree, score_data, node_column, score_column, new_score)
```

Arguments

`tree`	A `phylo` object.
`score_data`	A data frame that includes at least two columns. One column stores the number of the node, and the other stores the original score of the corresponding node.
`node_column`	The name of the column of `score_data` that contains the numbers of the nodes.
`score_column`	The name of the column of `score_data` that contains the original scores of the nodes.
`new_score`	The name of the column that stores the generated score.

Value

A `data.frame` similar to `score_data`, but with an extra column (named `new_score`) containing the weighted scores.

Author(s)

Ruizhu Huang

Examples

```
suppressPackageStartupMessages({
    library(TreeSummarizedExperiment)
    library(ggtree)
    library(dplyr)
})
```

```
## tree
data(tinyTree)
ggtree(tinyTree, branch.length = "none") +
    geom_text2(aes(label = node))

## score
exScore <- data.frame(nodeNum = seq_len(19), score = (seq_len(19))/10)

## Calculate new score based on the tree
newScore <- treeScore(tree = tinyTree, score_data = exScore,
                      node_column = "nodeNum",
                      score_column = "score",
                      new_score = "wScore")

## Visualize the result
## The original scores are in black texts and the new ones in blue
df <- newScore |>
      rename(node = nodeNum) |>
      mutate(score = round(score, 3),
             wScore = round(wScore, 3))
ggtree(tinyTree) %<+%
   df +
   geom_text2(aes(label = score), hjust = -0.05) +
   geom_text2(aes(label = wScore, hjust = -1.2),
   color = "blue")
```

Index