Package ‘ClustAssess’

January 31, 2025

Type Package

Title Tools for Assessing Clustering

Version 1.0.0

Description A set of tools for evaluating clustering robustness using proportion of ambiguously clustered pairs (Senbabaoglu et al. (2014) <doi:10.1038/srep06207>), as well as similarity across methods and method stability using element-centric clustering comparison (Gates et al. (2019) <doi:10.1038/s41598-019-44892-y>). Additionally, this package enables stability-based parameter assessment for graph-based clustering pipelines typical in single-cell data analysis.

License MIT + file LICENSE

Encoding UTF-8

Additional_repositories `https://blaserlab.r-universe.dev`

biocViews Software, SingleCell, RNASeq, ATACSeq, Normalization, Preprocessing, DimensionReduction, Visualization, QualityControl, Clustering, Classification, Annotation, GeneExpression, DifferentialExpression

Depends R (>= 4.0.0), methods, stats

Imports dplyr, DT, fastcluster, foreach, glue, Gmedian, ggnewscale, ggplot2, ggrastr, ggrepel, ggtext, gprofiler2, igraph, jsonlite, leiden, Matrix (>= 1.5.0), matrixStats, progress, stringr, paletteer, plotly, qualpalr, RANN, reshape2, rlang, Seurat, shiny, shinyjs, shinyLP, shinyWidgets, utils, uwot, vioplot

RoxygenNote 7.3.2

LinkingTo Rcpp, RcppEigen

Suggests BiocManager, colourpicker, ComplexHeatmap, data.table, DelayedMatrixStats, devtools, doParallel, knitr, leidenbase, monocle3, patchwork, ragg, reticulate, rhdf5, RhpcBLASctl, rmarkdown, scales, SeuratObject, SharedObject, styler, testthat (>= 3.0.0)

URL https://github.com/Core-Bioinformatics/ClustAssess,
https://core-bioinformatics.github.io/ClustAssess/

BugReports https://github.com/Core-Bioinformatics/ClustAssess/issues

Config/testthat/edition 3

NeedsCompilation yes

Author Andi Munteanu [aut, cre],
Arash Shahsavari [aut],
Rafael Kollyfas [ctb],
Miguel Larraz Lopez de Novales [aut],
Liviu Ciortuz [ctb],
Irina Mohorianu [aut]

Maintainer Andi Munteanu <am3019@cam.ac.uk>

Repository CRAN

Date/Publication 2025-01-31 22:40:06 UTC

Contents

add_metadata *Add metadata to ClustAssess ShinyApp*

Description

Adds new metadata into the ClustAssess ShinyApp without having to update the object and re-create the app.

Usage

```
add_metadata(app_folder, metadata, qualpalr_colorspace = "pretty")
```

Arguments

`app_folder`	The folder containing the ClustAssess ShinyApp
`metadata`	The new metadata to be added. This parameter should be a dataframe that follows the same row ordering as the already existing metadata from the ClustAssess app.
`qualpalr_colorspace`	The colorspace to be used for the metadata

Value

NULL - the metadata object is updated in the app folder

`assess_clustering_stability`

Assessment of Stability for Graph Clustering

Description

Evaluates the stability of different graph clustering methods in the clustering pipeline. The method will iterate through different values of the resolution parameter and compare, using the EC Consistency score, the partitions obtained at different seeds.

Usage

```
assess_clustering_stability(
  graph_adjacency_matrix,
  resolution,
  n_repetitions = 100,
  seed_sequence = NULL,
  ecs_thresh = 1,
  clustering_algorithm = 1:3,
  clustering_arguments = list(),
  verbose = TRUE
)
```

Arguments

`graph_adjacency_matrix`	A square adjacency matrix based on which an igraph object will be built. The matrix should have rownames and colnames that correspond to the names of the cells.
`resolution`	A sequence of resolution values. The resolution parameter controls the coarseness of the clustering. The higher the resolution, the more clusters will be obtained. The resolution parameter is used in the community detection algorithms.
`n_repetitions`	The number of repetitions of applying the pipeline with different seeds; ignored if seed_sequence is provided by the user. Defaults to `100`.

`seed_sequence`	A custom seed sequence; if the value is NULL, the sequence will be built starting from 1 with a step of 100.
`ecs_thresh`	The ECS threshold used for merging similar clusterings.
`clustering_algorithm`	An index or a list of indexes indicating which community detection algorithm will be used: Louvain (1), Louvain refined (2), SLM (3) or Leiden (4). More details can be found in the Seurat's `FindClusters` function. Defaults to `1:3`.
`clustering_arguments`	A list of additional arguments that will be passed to the clustering method. More details can be found in the Seurat's `FindClusters` function.
`verbose`	Boolean value used for displaying the progress bar.

Value

A list having two fields:

- `all` - a list that contains, for each clustering method and each resolution value, the EC consistency between the partitions obtained by changing the seed
- `filtered` - similar to `all`, but for each configuration, we determine the number of clusters that appears the most and use only the partitions with this size

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_embedding <- matrix(runif(100 * 30), nrow = 100)
rownames(pca_embedding) <- paste0("cell_", seq_len(nrow(pca_embedding)))
colnames(pca_embedding) <- paste0("PC_", 1:30)


adj_matrix <- getNNmatrix(
    RANN::nn2(pca_embedding, k = 10)$nn.idx,
    10,
    0,
    -1
)$nn
rownames(adj_matrix) <- paste0("cell_", seq_len(nrow(adj_matrix)))
colnames(adj_matrix) <- paste0("cell_", seq_len(ncol(adj_matrix)))

# alternatively, the adj_matrix can be calculated
# using the `Seurat::FindNeighbors` function.

clust_diff_obj <- assess_clustering_stability(
    graph_adjacency_matrix = adj_matrix,
    resolution = c(0.5, 1),
    n_repetitions = 10,
    clustering_algorithm = 1:2,
    verbose = TRUE
)
plot_clustering_overall_stability(clust_diff_obj)
```

assess_feature_stability

Assess the stability for configurations of feature types and sizes

Description

Evaluate the stability of clusterings obtained based on incremental subsets of a given feature set.

Usage

```
assess_feature_stability(
  data_matrix,
  feature_set,
  steps,
  feature_type,
  resolution,
  n_repetitions = 100,
  seed_sequence = NULL,
  graph_reduction_type = "PCA",
  ecs_thresh = 1,
  matrix_processing = function(dt_mtx, actual_npcs = 30, ...) {
     actual_npcs <-
    min(actual_npcs, ncol(dt_mtx)%/%2)

    RhpcBLASctl::blas_set_num_threads(foreach::getDoParWorkers())
     embedding <-
    stats::prcomp(x = dt_mtx, rank. = actual_npcs)$x

    RhpcBLASctl::blas_set_num_threads(1)
     rownames(embedding) <- rownames(dt_mtx)

     colnames(embedding) <- paste0("PC_", seq_len(ncol(embedding)))

    return(embedding)
 },
  umap_arguments = list(),
  prune_value = -1,
  clustering_algorithm = 1,
  clustering_arguments = list(),
  verbose = FALSE
)
```

Arguments

data_matrix	A data matrix having the features on the rows and the observations on the columns.
feature_set	A set of feature names that can be found on the rownames of the data matrix.

`steps`	Vector containing the sizes of the subsets; negative values will be interpreted as using all features.
`feature_type`	A name associated to the feature_set.
`resolution`	A vector containing the resolution values used for clustering.
`n_repetitions`	The number of repetitions of applying the pipeline with different seeds; ignored if seed_sequence is provided by the user. Defaults to `100`.
`seed_sequence`	A custom seed sequence; if the value is NULL, the sequence will be built starting from 1 with a step of 100. Defaults to `NULL`.
`graph_reduction_type`	The graph reduction type, denoting if the graph should be built on either the PCA or the UMAP embedding. Defaults to `PCA`.
`ecs_thresh`	The ECS threshold used for merging similar clusterings. We recommend using the 1 value. Defaults to `1`.
`matrix_processing`	A function that will be used to process the data matrix by using a dimensionality reduction technique. The function should have one parameter, the data matrix, and should return an embedding describing the reduced space. By default, the function will use the precise PCA method with `prcomp`.
`umap_arguments`	A list containing the arguments that will be passed to the UMAP function. Refer to the `uwot::umap` function for more details.
`prune_value`	Argument indicating whether to prune the SNN graph. If the value is 0, the graph won't be pruned. If the value is between 0 and 1, the edges with weight under the pruning value will be removed. If the value is -1, the highest pruning value will be calculated automatically and used.
`clustering_algorithm`	An index indicating which community detection algorithm will be used: Louvain (1), Louvain refined (2), SLM (3) or Leiden (4). More details can be found in the Seurat's `FindClusters` function.
`clustering_arguments`	A list containing the arguments that will be passed to the community detection algorithm, such as the number of iterations and the number of starts. Refer to the Seurat's `FindClusters` function for more details.
`verbose`	A boolean indicating if the intermediate progress will be printed or not.

Value

A list having one field associated with a step value. Each step contains a list with three fields:

- ecc - the EC-Consistency of the partitions obtained on all repetitions
- embedding - one UMAP embedding generated on the feature subset
- most_frequent_partition - the most common partition obtained across repetitions

Note

The algorithm assumes that the feature_set is already sorted when performing the subsetting based on the steps values. For example, if the user wants to analyze highly variable feature set, they should provide them sorted by their variability.

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(100 * 10), runif(100 * 10, min = 3, max = 4)),
    nrow = 200, byrow = TRUE
)
rownames(expr_matrix) <- as.character(1:200)
colnames(expr_matrix) <- paste("feature", 1:10)

feature_stability_result <- assess_feature_stability(
    data_matrix = t(expr_matrix),
    feature_set = colnames(expr_matrix),
    steps = 5,
    feature_type = "feature_name",
    resolution = c(0.1, 0.5, 1),
    n_repetitions = 10,
    umap_arguments = list(
        # the following parameters are used by the umap function
        # and are not mandatory
        n_neighbors = 3,
        approx_pow = TRUE,
        n_epochs = 0,
        init = "random",
        min_dist = 0.3
    ),
    clustering_algorithm = 1
)
plot_feature_overall_stability_boxplot(feature_stability_result)
```

`assess_nn_stability` *Assess the stability for Graph Building Parameters*

Description

Evaluates clustering stability when changing the values of different parameters involved in the graph building step, namely the base embedding, the graph type and the number of neighbours.

Usage

```
assess_nn_stability(
  embedding,
  n_neigh_sequence,
  n_repetitions = 100,
  seed_sequence = NULL,
  graph_reduction_type = "PCA",
  ecs_thresh = 1,
  graph_type = 2,
  prune_value = -1,
  clustering_algorithm = 1,
  clustering_arguments = list(),
  umap_arguments = list()
)
```

Arguments

`embedding`	A matrix associated with a PCA embedding. Embeddings from other dimensionality reduction techniques (such as LSI) can be used.
`n_neigh_sequence`	A sequence of the number of nearest neighbours.
`n_repetitions`	The number of repetitions of applying the pipeline with different seeds; ignored if seed_sequence is provided by the user.
`seed_sequence`	A custom seed sequence; if the value is NULL, the sequence will be built starting from 1 with a step of 100.
`graph_reduction_type`	The graph reduction type, denoting if the graph should be built on either the PCA or the UMAP embedding.
`ecs_thresh`	The ECS threshold used for merging similar clusterings.
`graph_type`	Argument indicating whether the graph should be unweighted (0), weighted (1) or both (2).
`prune_value`	Argument indicating whether to prune the SNN graph. If the value is 0, the graph won't be pruned. If the value is between 0 and 1, the edges with weight under the pruning value will be removed. If the value is -1, the highest pruning value will be calculated automatically and used.
`clustering_algorithm`	An index indicating which community detection algorithm will be used: Louvain (1), Louvain refined (2), SLM (3) or Leiden (4). More details can be found in the Seurat's `FindClusters` function.
`clustering_arguments`	A list of arguments that will be passed to the clustering algorithm. See the `FindClusters` function in Seurat for more details.
`umap_arguments`	Additional arguments passed to the the `uwot::umap` method.

Value

A list having three fields:

- `n_neigh_k_corresp` - list containing the number of the clusters obtained by running the pipeline multiple times with different seed, number of neighbours and graph type (weighted vs unweigted)
- `n_neigh_ec_consistency` - list containing the EC consistency of the partitions obtained at multiple runs when changing the number of neighbours or the graph type
- `n_different_partitions` - the number of different partitions obtained by each number of neighbours

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_emb <- matrix(runif(100 * 30), nrow = 100, byrow = TRUE)
rownames(pca_emb) <- as.character(1:100)
colnames(pca_emb) <- paste0("PC_", 1:30)

nn_stability_obj <- assess_nn_stability(
    embedding = pca_emb,
    n_neigh_sequence = c(10, 15, 20),
    n_repetitions = 10,
    graph_reduction_type = "PCA",
    clustering_algorithm = 1
)
plot_n_neigh_ecs(nn_stability_obj)
```

---

automatic_stability_assessment

Assessment of Stability for Graph Clustering

Description

Evaluates the stability of different graph clustering methods in the clustering pipeline. The method will iterate through different values of the resolution parameter and compare, using the EC Consistency score, the partitions obtained at different seeds.

Usage

```
automatic_stability_assessment(
  expression_matrix,
  n_repetitions,
  n_neigh_sequence,
  resolution_sequence,
  features_sets,
  steps,
  seed_sequence = NULL,
  graph_reduction_embedding = "PCA",
  include_umap_nn_assessment = FALSE,
  n_top_configs = 3,
  ranking_criterion = "iqr",
  overall_summary = "median",
  ecs_threshold = 1,
  matrix_processing = function(dt_mtx, actual_npcs = 30, ...) {
     actual_npcs <-
    min(actual_npcs, ncol(dt_mtx)%/%2)

    RhpcBLASctl::blas_set_num_threads(foreach::getDoParWorkers())
     embedding <-
```

```
        stats::prcomp(x = dt_mtx, rank. = actual_npcs)$x

        RhpcBLASctl::blas_set_num_threads(1)
         rownames(embedding) <- rownames(dt_mtx)

         colnames(embedding) <- paste0("PC_", seq_len(ncol(embedding)))

        return(embedding)
     },
      umap_arguments = list(),
      prune_value = -1,
      algorithm_dim_reduction = 1,
      algorithm_graph_construct = 1,
      algorithms_clustering_assessment = 1:3,
      clustering_arguments = list(),
      verbose = TRUE,
      temp_file = NULL,
      save_temp = TRUE
    )
```

Arguments

`expression_matrix`	An expression matrix having the features on the rows and the cells on the columns.
`n_repetitions`	The number of repetitions of applying the pipeline with different seeds; ignored if seed_sequence is provided by the user. Defaults to `100`.
`n_neigh_sequence`	A sequence of the number of nearest neighbours.
`resolution_sequence`	A sequence of resolution values. The resolution parameter controls the coarseness of the clustering. The higher the resolution, the more clusters will be obtained. The resolution parameter is used in the community detection algorithms.
`features_sets`	A list of the feature sets. A feature set is a list of genes from the expression matrix that will be used in the dimensionality reduction.
`steps`	A list with the same names as `feature_sets`. Each name has assigned a ector containing the sizes of the subsets; negative values will be interpreted as using all features.
`seed_sequence`	A custom seed sequence; if the value is NULL, the sequence will be built starting from 1 with a step of 100.
`graph_reduction_embedding`	The type of dimensionality reduction used for the graph construction. The options are "PCA" and "UMAP". Defaults to `PCA`.
`include_umap_nn_assessment`	A boolean value indicating if the UMAP embeddings will be used for the nearest neighbours assessment. Defaults to `FALSE`.
`n_top_configs`	The number of top configurations that will be used for the downstream analysis in the dimensionality reduction step. Defaults to `3`.

`ranking_criterion`	The criterion used for ranking the configurations from the dimensionality reduction step. The options are "iqr", "median", "max", "top_qt", "top_qt_max", "iqr_median", "iqr_median_coeff" and "mean". Defaults to `iqr`.
`overall_summary`	A function used to summarize the stability of the configurations from the dimensionality reduction step across the different resolution values. The options are "median", "max", "top_qt", "top_qt_max", "iqr", "iqr_median", "iqr_median_coeff" and "mean". Defaults to `median`.
`ecs_threshold`	The ECS threshold used for merging similar clusterings.
`matrix_processing`	A function that will be used to process the data matrix by using a dimensionality reduction technique. The function should have one parameter, the data matrix, and should return an embedding describing the reduced space. By default, the function will use the precise PCA method with `prcomp`.
`umap_arguments`	A list containing the arguments that will be passed to the UMAP function. Refer to the `uwot::umap` function for more details.
`prune_value`	Argument indicating whether to prune the SNN graph. If the value is 0, the graph won't be pruned. If the value is between 0 and 1, the edges with weight under the pruning value will be removed. If the value is -1, the highest pruning value will be calculated automatically and used.
`algorithm_dim_reduction`	An index indicating the community detection algorithm that will be used in the Dimensionality reduction step.
`algorithm_graph_construct`	An index indicating the community detection algorithm that will be used in the Graph construction step.
`algorithms_clustering_assessment`	An index indicating which community detection algorithm will be used for the clustering step: Louvain (1), Louvain refined (2), SLM (3) or Leiden (4). More details can be found in the Seurat's `FindClusters` function.
`clustering_arguments`	A list containing the arguments that will be passed to the community detection algorithm, such as the number of iterations and the number of starts. Refer to the Seurat's `FindClusters` function for more details.
`verbose`	Boolean value used for displaying the progress of the assessment.
`temp_file`	The path to the file where the object will be saved.
`save_temp`	A boolean value indicating if the object will be saved to a file.

Value

A list having two fields:

- all - a list that contains, for each clustering method and each resolution value, the EC consistency between the partitions obtained by changing the seed
- filtered - similar to `all`, but for each configuration, we determine the number of clusters that appears the most and use only the partitions with this size

Examples

```
## Not run:
set.seed(2024)
# create an already-transposed artificial expression matrix
expr_matrix <- matrix(
    c(runif(20 * 10), runif(30 * 10, min = 3, max = 4)),
    nrow = 10, byrow = FALSE
)
colnames(expr_matrix) <- as.character(seq_len(ncol(expr_matrix)))
rownames(expr_matrix) <- paste("feature", seq_len(nrow(expr_matrix)))

autom_object <- automatic_stability_assessment(
    expression_matrix = expr_matrix,
    n_repetitions = 3,
    n_neigh_sequence = c(5),
    resolution_sequence = c(0.1, 0.5),
    features_sets = list(
        "set1" = rownames(expr_matrix)
    ),
    steps = list(
        "set1" = c(5, 7)
    ),
    umap_arguments = list(
        # the following parameters have been modified
        # from the default values to ensure that
        # the function will run under 5 seconds
        n_neighbors = 3,
        approx_pow = TRUE,
        n_epochs = 0,
        init = "random",
        min_dist = 0.3
    ),
    n_top_configs = 1,
    algorithms_clustering_assessment = 1,
    save_temp = FALSE,
    verbose = FALSE
)

# the object can be further used to plot the assessment results
plot_feature_overall_stability_boxplot(autom_object$feature_stability)
plot_n_neigh_ecs(autom_object$set1$"5"$nn_stability)
plot_k_n_partitions(autom_object$set1$"5"$clustering_stability)

## End(Not run)
```

`calculate_markers` *Calculate markers*

Description

Performs the Wilcoxon rank sum test to identify differentially expressed genes between two groups of cells.

Usage

```
calculate_markers(
  expression_matrix,
  cells1,
  cells2,
  logfc_threshold = 0,
  min_pct_threshold = 0.1,
  avg_expr_threshold_group1 = 0,
  min_diff_pct_threshold = -Inf,
  rank_matrix = NULL,
  feature_names = NULL,
  used_slot = "data",
  norm_method = "SCT",
  pseudocount_use = 1,
  base = 2,
  adjust_pvals = TRUE,
  check_cells_set_diff = TRUE
)
```

Arguments

`expression_matrix`	A matrix of gene expression values having genes in rows and cells in columns.
`cells1`	A vector of cell indices for the first group of cells.
`cells2`	A vector of cell indices for the second group of cells.
`logfc_threshold`	The minimum absolute log fold change to consider a gene as differentially expressed. Defaults to `0`, meaning all genes are taken into considereation.
`min_pct_threshold`	The minimum fraction of cells expressing a gene form each cell population to consider the gene as differentially expressed. Increasing the value will speed up the function. Defaults to `0.1`.
`avg_expr_threshold_group1`	The minimum average expression that a gene should have in the first group of cells to be considered as differentially expressed. Defaults to `0`.
`min_diff_pct_threshold`	The minimum difference in the fraction of cells expressing a gene between the two cell populations to consider the gene as differentially expressed. Defaults to `-Inf`.
`rank_matrix`	A matrix where the cells are ranked based on their expression levels with respect to each gene. Defaults to `NULL`, in which case the function will calculate the rank matrix. We recommend calculating the rank matrix beforehand and passing it to the function to speed up the computation.

`feature_names`	A vector of gene names. Defaults to `NULL`, in which case the function will use the row names of the expression matrix as gene names.
`used_slot`	Parameter that provides additional information about the expression matrix, whether it was scaled or not. The value of this parameter impacts the calculation of the fold change. If `data`, the function will calculates the fold change as the fraction between the log value of the average of the expression raised to exponential for the two cell groups. If `scale.data`, the function will calculate the fold change as the fraction between the average of the expression values for the two cell groups. Other options will default to calculating the fold change as the fraction between the log value of the average of the expression values for the two cell groups. Defaults to `data`.
`norm_method`	The normalization method used to normalize the expression matrix. The value of this parameter impacts the calculation of the average expression of the genes when `used_slot = "data"`. If `LogNormalize`, the log fold change will be calculated as described for the `used_slot` parameter. Otherwise, the log fold change will be calculated as the fraction between the log value of the average of the expression values for the two cell groups. Defaults to `SCT`.
`pseudocount_use`	The pseudocount to add to the expression values when calculating the average expression of the genes, to avoid the 0 value for the denominator. Defaults to `1`.
`base`	The base of the logharithm. Defaults to `2`.
`adjust_pvals`	A logical value indicating whether to adjust the p-values for multiple testing using the Bonferonni method. Defaults to `TRUE`.
`check_cells_set_diff`	A logical value indicating whether to check if thw two cell groups are disjoint or not. Defaults to `TRUE`.

Value

A data frame containing the following columns:

- `gene`: The gene name.
- `avg_log2FC`: The average log fold change between the two cell groups.
- `p_val`: The p-value of the Wilcoxon rank sum test.
- `p_val_adj`: The adjusted p-value of the Wilcoxon rank sum test.
- `pct.1`: The fraction of cells expressing the gene in the first cell group.
- `pct.2`: The fraction of cells expressing the gene in the second cell group.
- `avg_expr_group1`: The average expression of the gene in the first cell group.

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(100 * 50), runif(100 * 50, min = 3, max = 4)),
    ncol = 200, byrow = FALSE
)
colnames(expr_matrix) <- as.character(1:200)
rownames(expr_matrix) <- paste("feature", 1:50)

calculate_markers(
    expression_matrix = expr_matrix,
    cells1 = 101:200,
    cells2 = 1:100
)
# TODO should be rewritten such that you don't create new matrix objects inside
# just
```

---

calculate_markers_shiny

Calculate markers - Shiny

Description

Performs the Wilcoxon rank sum test to identify differentially expressed genes between two groups of cells in the shiny context. The method can be also used outside the shiny context, as long as the expression matrix is stored in a h5 file.

Usage

```
calculate_markers_shiny(
  cells1,
  cells2,
  logfc_threshold = 0,
  min_pct_threshold = 0.1,
  average_expression_threshold = 0,
  average_expression_group1_threshold = 0,
  min_diff_pct_threshold = -Inf,
  used_slot = "data",
  norm_method = "SCT",
  expression_h5_path = "expression.h5",
  pseudocount_use = 1,
  base = 2,
  verbose = TRUE,
  check_difference = TRUE
)
```

Arguments

cells1	A vector of cell indices for the first group of cells.
cells2	A vector of cell indices for the second group of cells.
logfc_threshold	The minimum absolute log fold change to consider a gene as differentially expressed. Defaults to `0`, meaning all genes are taken into considereation.

`min_pct_threshold`	The minimum fraction of cells expressing a gene form each cell population to consider the gene as differentially expressed. Increasing the value will speed up the function. Defaults to `0.1`.
`average_expression_threshold`	The minimum average expression that a gene should have in order to be considered as differentially expressed.
`average_expression_group1_threshold`	The minimum average expression that a gene should have in the first group of cells to be considered as differentially expressed. Defaults to `0`.
`min_diff_pct_threshold`	The minimum difference in the fraction of cells expressing a gene between the two cell populations to consider the gene as differentially expressed. Defaults to `-Inf`.
`used_slot`	Parameter that provides additional information about the expression matrix, whether it was scaled or not. The value of this parameter impacts the calculation of the fold change. If `data`, the function will calculates the fold change as the fraction between the log value of the average of the expression raised to exponential for the two cell groups. If `scale.data`, the function will calculate the fold change as the fraction between the average of the expression values for the two cell groups. Other options will default to calculating the fold change as the fraction between the log value of the average of the expression values for the two cell groups. Defaults to `data`.
`norm_method`	The normalization method used to normalize the expression matrix. The value of this parameter impacts the calculation of the average expression of the genes when `used_slot = "data"`. If `LogNormalize`, the log fold change will be calculated as described for the `used_slot` parameter. Otherwise, the log fold change will be calculated as the fraction between the log value of the average of the expression values for the two cell groups. Defaults to `SCT`.
`expression_h5_path`	The path to the h5 file containing the expression matrix. The h5 file should contain the following fields: `expression_matrix`, `rank_matrix`, `average_expression`, `genes`. The file path defaults to `expression.h5`.
`pseudocount_use`	The pseudocount to add to the expression values when calculating the average expression of the genes, to avoid the 0 value for the denominator. Defaults to `1`.
`base`	The base of the logharithm. Defaults to `2`.
`verbose`	Whether to print messages about the progress of the function. Defaults to TRUE.
`check_difference`	Whether to perform set difference between the two cells. Defaults to TRUE.

Value

A data frame containing the following columns:

- `gene`: The gene name.
- `avg_log2FC`: The average log fold change between the two cell groups.

- `p_val`: The p-value of the Wilcoxon rank sum test.
- `p_val_adj`: The adjusted p-value of the Wilcoxon rank sum test.
- `pct.1`: The fraction of cells expressing the gene in the first cell group.
- `pct.2`: The fraction of cells expressing the gene in the second cell group.
- `avg_expr_group1`: The average expression of the gene in the first cell group.
- `avg_expr`: The average expression of the gene.

`choose_stable_clusters` *Choose stable clusters based on ECC and frequency*

Description

Filter the list of clusters obtained by the automatic ClustAssess pipeline using the ECC and frequency thresholds. The ECC threshold is meant to filter out the partitions that are highly sensitive to the change of the random seed, while the purpose of the frequency threshold is to assure a statistical significance of the inferred stability.

Usage

```
choose_stable_clusters(
  clusters_list,
  ecc_threshold = 0.9,
  freq_threshold = 30,
  summary_function = mean
)
```

Arguments

`clusters_list`	List of clusters obtained from the `get_clusters_from_clustassess_object` function.
`ecc_threshold`	Minimum ECC value to consider a cluster as stable. Default is 0.9.
`freq_threshold`	Minimum total frequency of the partitions to consider. Default is 30.
`summary_function`	Function to summarize the ECC values. Default is `mean`. To match the results from the ClustAssess Shiny App, use `median`.

Value

A list of stable clusters that satisfy the ECC and frequency.

`consensus_cluster` *Consensus Clustering and Proportion of Ambiguously Clustered Pairs*

Description

Calculate consensus clustering and proportion of ambiguously clustered pairs (PAC) with hierarchical clustering.

Usage

```
consensus_cluster(
  x,
  k_min = 3,
  k_max = 100,
  n_reps = 100,
  p_sample = 0.8,
  p_feature = 1,
  p_minkowski = 2,
  dist_method = "euclidean",
  linkage = "complete",
  lower_lim = 0.1,
  upper_lim = 0.9,
  verbose = TRUE
)
```

Arguments

`x`	A samples x features normalized data matrix.
`k_min`	The minimum number of clusters calculated.
`k_max`	The maximum number of clusters calculated.
`n_reps`	The total number of subsamplings and reclusterings of the data; this value needs to be high enough to ensure PAC converges; convergence can be assessed with pac_convergence.
`p_sample`	The proportion of samples included in each subsample.
`p_feature`	The proportion of features included in each subsample.
`p_minkowski`	The power of the Minkowski distance.
`dist_method`	The distance measure for the distance matrix used in hclust; must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".
`linkage`	The linkage method used in hclust; must be one of "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid"
`lower_lim`	The lower limit for determining whether a pair is clustered ambiguously; the lower this value, the higher the PAC.
`upper_lim`	The upper limit for determining whether a pair is clustered ambiguously; the higher this value, the higher the PAC.
`verbose`	Logical value used for choosing to display a progress bar or not.

Value

A data.frame with PAC values across iterations, as well as parameter values used when calling the method.

References

Monti, S., Tamayo, P., Mesirov, J., & Golub, T. (2003). Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. Machine learning, 52(1), 91-118. https://doi.org/10.1023/A:1023949509487

Senbabaoglu, Y., Michailidis, G., & Li, J. Z. (2014). Critical limitations of consensus clustering in class discovery. Scientific reports, 4(1), 1-13. https://doi.org/10.1038/srep06207

Examples

```
pac.res <- consensus_cluster(iris[, 1:4], k_max = 20)
pac_convergence(pac.res, k_plot = c(3, 5, 7, 9))
```

---

create_monocle_default *Create monocle object*

Description

Use a normalized expression matrix and, potentially, an already generated PCA / UMAP embedding, to create a Monocle object.

Usage

```
create_monocle_default(
  normalized_expression_matrix,
  count_matrix = NULL,
  pca_embedding = NULL,
  umap_embedding = NULL,
  metadata_df = NULL
)
```

Arguments

`normalized_expression_matrix`	The normalized expression matrix having genes on rows and cells on columns.
`count_matrix`	The count matrix having genes on rows and cells on columns. If NULL, the normalized_expression_matrix will be used.
`pca_embedding`	The PCA embedding of the expression matrix. If NULL, the pca will be created using the `monocle3` package (default parameters).
`umap_embedding`	The UMAP embedding of the expression matrix. If NULL, the umap will be created using the `monocle3` package (default parameters).
`metadata_df`	The metadata dataframe having the cell names as rownames. If NULL, a dataframe with a single column named `identical_ident` will be created.

Value

A Monocle object of the expression matrix, having the stable number of clusters identified by ClustAssess.

Examples

```
## Not run:
set.seed(2024)
# create an already-transposed artificial expression matrix
expr_matrix <- matrix(
    c(runif(20 * 10), runif(30 * 10, min = 3, max = 4)),
    nrow = 10, byrow = FALSE
)
colnames(expr_matrix) <- as.character(seq_len(ncol(expr_matrix)))
rownames(expr_matrix) <- paste("feature", seq_len(nrow(expr_matrix)))

# uncomment to create the monocle object
mon_obj <- create_monocle_default(
    normalized_expression_matrix = expr_matrix,
    pca_emb = NULL,
    umap_emb = NULL,
    metadata_df = NULL
)

## End(Not run)
```

`create_monocle_from_clustassess`

Create monocle object from a ClustAssess object

Description

Use the object generated using the ClustAssess `automatic_stability_assessment` function to create a Monocle object which has the stable number of clusters.

Usage

```
create_monocle_from_clustassess(
  normalized_expression_matrix,
  count_matrix = NULL,
  clustassess_object,
  metadata_df,
  stable_feature_type,
  stable_feature_set_size,
  stable_clustering_method,
  stable_n_clusters = NULL,
  use_all_genes = FALSE
)
```

Arguments

`normalized_expression_matrix`	The normalized expression matrix having genes on rows and cells on columns.
`count_matrix`	The count matrix having genes on rows and cells on columns. If NULL, the normalized_expression_matrix will be used.
`clustassess_object`	The output of the `automatic_stability_assessment`.
`metadata_df`	The metadata dataframe having the cell names as rownames. If NULL, a dataframe with a single column named `identical_ident` will be created.
`stable_feature_type`	The feature type which leads to stable clusters.
`stable_feature_set_size`	The feature size which leads to stable clusters.
`stable_clustering_method`	The clustering method which leads to stable clusters.
`stable_n_clusters`	The number of clusters that are stable. If NULL, all the clusters will be provided. Defaults to `NULL`.
`use_all_genes`	A boolean value indicating if the expression matrix should be truncated to the genes used in the stability assessment. Defaults to `FALSE`.

Value

A Monocle object of the expression matrix, having the stable number of clusters identified by ClustAssess.

Examples

```
## Not run:
set.seed(2024)
# create an already-transposed artificial expression matrix
expr_matrix <- matrix(
    c(runif(20 * 10), runif(30 * 10, min = 3, max = 4)),
    nrow = 10, byrow = FALSE
)
colnames(expr_matrix) <- as.character(seq_len(ncol(expr_matrix)))
rownames(expr_matrix) <- paste("feature", seq_len(nrow(expr_matrix)))

autom_object <- automatic_stability_assessment(
    expression_matrix = expr_matrix,
    n_repetitions = 3,
    n_neigh_sequence = c(5),
    resolution_sequence = c(0.1, 0.5),
    features_sets = list(
        "set1" = rownames(expr_matrix)
    ),
    steps = list(
        "set1" = c(5, 7)
    ),
    umap_arguments = list(
        # the following parameters have been modified
        # from the default values to ensure that the function
        # will run under 5 seconds
        n_neighbors = 3,
        approx_pow = TRUE,
        n_epochs = 0,
        init = "random",
        min_dist = 0.3
    ),
    n_top_configs = 1,
    algorithms_clustering_assessment = 1,
    save_temp = FALSE,
    verbose = FALSE
)


# uncomment to create the monocle object
# mon_obj <- create_monocle_from_clustassess(
#     normalized_expression_matrix = expr_matrix,
#     clustassess_object = autom_object,
#     metadata = NULL,
#     stable_feature_type = "set1",
#     stable_feature_set_size = "5",
#     stable_clustering_method = "Louvain"
# )

## End(Not run)
```

`create_monocle_from_clustassess_app`

Create monocle object from a ClustAssess shiny app

Description

Use the files generated in the ClustAssess app to create a Monocle object which has the stable number of clusters.

Usage

```
create_monocle_from_clustassess_app(
  app_folder,
  stable_feature_type,
  stable_feature_set_size,
  stable_clustering_method,
  stable_n_clusters = NULL,
  use_all_genes = FALSE
)
```

Arguments

`app_folder`	Path pointing to the folder containing a ClustAssess app.
`stable_feature_type`	The feature type which leads to stable clusters.
`stable_feature_set_size`	The feature size which leads to stable clusters.
`stable_clustering_method`	The clustering method which leads to stable clusters.
`stable_n_clusters`	The number of clusters that are stable. If NULL, all the clusters will be provided. Defaults to `NULL`.
`use_all_genes`	A boolean value indicating if the expression matrix should be truncated to the genes used in the stability assessment. Defaults to `FALSE`.

Value

A Monocle object of the expression matrix, having the stable number of clusters identified by ClustAssess.

`create_seurat_object_default` *Create Seurat object*

Description

Use a normalized expression matrix and, potentially, an already generated PCA / UMAP embedding, to create a Seurat object.

Usage

```
create_seurat_object_default(
  normalized_expression_matrix,
  count_matrix = NULL,
  pca_embedding = NULL,
  umap_embedding = NULL,
  metadata_df = NULL
)
```

Arguments

`normalized_expression_matrix`	The normalized expression matrix having genes on rows and cells on columns.
`count_matrix`	The count matrix having genes on rows and cells on columns. If NULL, the normalized_expression_matrix will be used.
`pca_embedding`	The PCA embedding of the expression matrix. If NULL, the pca will be created using the `Seurat` package (default parameters).

`umap_embedding`	The UMAP embedding of the expression matrix. If NULL, the umap will be created using the `Seurat` package (default parameters).
`metadata_df`	The metadata dataframe having the cell names as rownames. If NULL, a dataframe with a single column named `identical_ident` will be created.

Value

A Seurat object of the expression matrix, having the stable number of clusters identified by ClustAssess.

`create_seurat_object_from_clustassess_app`
Create Seurat object from a ClustAssess shiny app

Description

Use the files generated in the ClustAssess app to create a Seurat object which has the stable number of clusters.

Usage

```
create_seurat_object_from_clustassess_app(
  app_folder,
  stable_feature_type,
  stable_feature_set_size,
  stable_clustering_method,
  stable_n_clusters = NULL,
  use_all_genes = FALSE
)
```

Arguments

`app_folder`	Path pointing to the folder containing a ClustAssess app.
`stable_feature_type`	The feature type which leads to stable clusters.
`stable_feature_set_size`	The feature size which leads to stable clusters.
`stable_clustering_method`	The clustering method which leads to stable clusters.
`stable_n_clusters`	The number of clusters that are stable. If NULL, all the clusters will be provided. Defaults to `NULL`.
`use_all_genes`	A boolean value indicating if the expression matrix should be truncated to the genes used in the stability assessment. Defaults to `FALSE`.

Value

A Seurat object of the expression matrix, having the stable number of clusters identified by ClustAssess.

`element_agreement` *Element-Wise Average Agreement Between a Set of Clusterings*

Description

Inspect how consistently of a set of clusterings agree with a reference clustering by calculating their element-wise average agreement.

Usage

```
element_agreement(
  reference_clustering,
  clustering_list,
  alpha = 0.9,
  r = 1,
  rescale_path_type = "max",
  ppr_implementation = "prpack",
  dist_rescaled = FALSE,
  row_normalize = TRUE
)
```

Arguments

`reference_clustering`
: The reference clustering, that each clustering in clustering_list is compared to. It can be either:
  - A numeric/character/factor vector of cluster labels for each element.
  - A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
  - An hclust object.

`clustering_list`
: The list of clustering results, each of which is either:
  - A numeric/character/factor vector of cluster labels for each element.
  - A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
  - An hclust object.

`alpha`
: A numeric giving the personalized PageRank damping factor; 1 - alpha is the restart probability for the PPR random walk.

`r`
: A numeric hierarchical scaling parameter.

`rescale_path_type`
: A string; rescale the hierarchical height by:
  - "max" : the maximum path from the root.

- "min" : the minimum path form the root.
- "linkage" : use the linkage distances in the clustering.

`ppr_implementation`	Choose a implementation for personalized page-rank calculation: • "prpack": use PPR algorithms in igraph. • "power_iteration": use power_iteration method.
`dist_rescaled`	A logical: if TRUE, the linkage distances are linearly rescaled to be in-between 0 and 1.
`row_normalize`	Whether to normalize all rows in clustering_result so they sum to one before calculating ECS. It is recommended to set this to TRUE, which will lead to slightly different ECS values compared to clusim.

Value

A vector containing the element-wise average agreement.

References

Gates, A. J., Wood, I. B., Hetrick, W. P., & Ahn, Y. Y. (2019). Element-centric clustering comparison unifies overlaps and hierarchy. Scientific reports, 9(1), 1-13. https://doi.org/10.1038/s41598-019-44892-y

Examples

```
# perform k-means clustering across 20 random seeds
reference.clustering <- iris$Species
clustering.list <- lapply(1:20, function(x) kmeans(iris[, 1:4], centers = 3)$cluster)
element_agreement(reference.clustering, clustering.list)
```

`element_consistency` *Element-Wise Consistency Between a Set of Clusterings*

Description

Inspect the consistency of a set of clusterings by calculating their element-wise clustering consistency (also known as element-wise frustration).

Usage

```
element_consistency(
  clustering_list,
  alpha = 0.9,
  r = 1,
  rescale_path_type = "max",
  ppr_implementation = "prpack",
  dist_rescaled = FALSE,
  row_normalize = TRUE
)
```

Arguments

`clustering_list`
The list of clustering results, each of which is either:

- A numeric/character/factor vector of cluster labels for each element.
- A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
- An hclust object.

`alpha` A numeric giving the personalized PageRank damping factor; 1 - alpha is the restart probability for the PPR random walk.

`r` A numeric hierarchical scaling parameter.

`rescale_path_type`
A string; rescale the hierarchical height by:

- "max" : the maximum path from the root.
- "min" : the minimum path form the root.
- "linkage" : use the linkage distances in the clustering.

`ppr_implementation`
Choose a implementation for personalized page-rank calculation:

- "prpack": use PPR algorithms in igraph.
- "power_iteration": use power_iteration method.

`dist_rescaled` A logical: if TRUE, the linkage distances are linearly rescaled to be in-between 0 and 1.

`row_normalize` Whether to normalize all rows in clustering_result so they sum to one before calculating ECS. It is recommended to set this to TRUE, which will lead to slightly different ECS values compared to clusim.

Value

A vector containing the element-wise consistency. If `calculate_sim_matrix` is set to `TRUE`, the element similarity matrix will be returned as well.

References

Gates, A. J., Wood, I. B., Hetrick, W. P., & Ahn, Y. Y. (2019). Element-centric clustering comparison unifies overlaps and hierarchy. Scientific reports, 9(1), 1-13. https://doi.org/10.1038/s41598-019-44892-y

Examples

```
# cluster across 20 random seeds
clustering.list <- lapply(1:20, function(x) kmeans(mtcars, centers = 3)$cluster)
element_consistency(clustering.list)
```

`element_sim` *The Element-Centric Clustering Similarity*

Description

Calculates the average element-centric similarity between two clustering results

Usage

```
element_sim(
  clustering1,
  clustering2,
  alpha = 0.9,
  r_cl1 = 1,
  rescale_path_type_cl1 = "max",
  ppr_implementation_cl1 = "prpack",
  dist_rescaled_cl1 = FALSE,
  row_normalize_cl1 = TRUE,
  r_cl2 = 1,
  rescale_path_type_cl2 = "max",
  ppr_implementation_cl2 = "prpack",
  dist_rescaled_cl2 = FALSE,
  row_normalize_cl2 = TRUE
)
```

Arguments

`clustering1` The first clustering result, which can be one of:

- A numeric/character/factor vector of cluster labels for each element.
- A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
- An hclust object.

`clustering2` The second clustering result, which can be one of:

- A numeric/character/factor vector of cluster labels for each element.
- A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
- An hclust object.

`alpha` A numeric giving the personalized PageRank damping factor; 1 - alpha is the restart probability for the PPR random walk.

`r_cl1` A numeric hierarchical scaling parameter for the first clustering.

`rescale_path_type_cl1`
A string; rescale the hierarchical height of the first clustering by:

- "max" : the maximum path from the root.
- "min" : the minimum path form the root.
- "linkage" : use the linkage distances in the clustering.

`ppr_implementation_cl1`
: Choose a implementation for personalized page-rank calculation for the first clustering:

  - "prpack": use PPR algorithms in igraph.
  - "power_iteration": use power_iteration method.

`dist_rescaled_cl1`
: A logical: if TRUE, the linkage distances of the first clustering are linearly rescaled to be in-between 0 and 1.

`row_normalize_cl1`
: Whether to normalize all rows in the first clustering so they sum to one before calculating ECS. It is recommended to set this to TRUE, which will lead to slightly different ECS values compared to clusim.

`r_cl2`
: A numeric hierarchical scaling parameter for the second clustering.

`rescale_path_type_cl2`
: A string; rescale the hierarchical height of the second clustering by:

  - "max" : the maximum path from the root.
  - "min" : the minimum path form the root.
  - "linkage" : use the linkage distances in the clustering.

`ppr_implementation_cl2`
: Choose a implementation for personalized page-rank calculation for the second clustering:

  - "prpack": use PPR algorithms in igraph.
  - "power_iteration": use power_iteration method.

`dist_rescaled_cl2`
: A logical: if TRUE, the linkage distances of the second clustering are linearly rescaled to be in-between 0 and 1.

`row_normalize_cl2`
: Whether to normalize all rows in the second clustering so they sum to one before calculating ECS. It is recommended to set this to TRUE, which will lead to slightly different ECS values compared to clusim.

Value

The average element-wise similarity between the two Clusterings.

Examples

```
km.res <- kmeans(mtcars, centers = 3)$cluster
hc.res <- hclust(dist(mtcars))
element_sim(km.res, hc.res)
```

`element_sim_elscore` *The Element-Centric Clustering Similarity for each Element*

Description

Calculates the element-wise element-centric similarity between two clustering results.

Usage

```
element_sim_elscore(
  clustering1,
  clustering2,
  alpha = 0.9,
  r_cl1 = 1,
  rescale_path_type_cl1 = "max",
  ppr_implementation_cl1 = "prpack",
  dist_rescaled_cl1 = FALSE,
  row_normalize_cl1 = TRUE,
  r_cl2 = 1,
  rescale_path_type_cl2 = "max",
  ppr_implementation_cl2 = "prpack",
  dist_rescaled_cl2 = FALSE,
  row_normalize_cl2 = TRUE
)
```

Arguments

`clustering1` The first clustering result, which can be one of:

- A numeric/character/factor vector of cluster labels for each element.
- A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
- An hclust object.

`clustering2` The second clustering result, which can be one of:

- A numeric/character/factor vector of cluster labels for each element.
- A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
- An hclust object.

`alpha` A numeric giving the personalized PageRank damping factor; 1 - alpha is the restart probability for the PPR random walk.

`r_cl1` A numeric hierarchical scaling parameter for the first clustering.

`rescale_path_type_cl1`
A string; rescale the hierarchical height of the first clustering by:

- "max" : the maximum path from the root.
- "min" : the minimum path form the root.
- "linkage" : use the linkage distances in the clustering.

`ppr_implementation_cl1`
: Choose a implementation for personalized page-rank calculation for the first clustering:
  - "prpack": use PPR algorithms in igraph.
  - "power_iteration": use power_iteration method.

`dist_rescaled_cl1`
: A logical: if TRUE, the linkage distances of the first clustering are linearly rescaled to be in-between 0 and 1.

`row_normalize_cl1`
: Whether to normalize all rows in the first clustering so they sum to one before calculating ECS. It is recommended to set this to TRUE, which will lead to slightly different ECS values compared to clusim.

`r_cl2`
: A numeric hierarchical scaling parameter for the second clustering.

`rescale_path_type_cl2`
: A string; rescale the hierarchical height of the second clustering by:
  - "max" : the maximum path from the root.
  - "min" : the minimum path form the root.
  - "linkage" : use the linkage distances in the clustering.

`ppr_implementation_cl2`
: Choose a implementation for personalized page-rank calculation for the second clustering:
  - "prpack": use PPR algorithms in igraph.
  - "power_iteration": use power_iteration method.

`dist_rescaled_cl2`
: A logical: if TRUE, the linkage distances of the second clustering are linearly rescaled to be in-between 0 and 1.

`row_normalize_cl2`
: Whether to normalize all rows in the second clustering so they sum to one before calculating ECS. It is recommended to set this to TRUE, which will lead to slightly different ECS values compared to clusim.

Value

Vector of element-centric similarity between the two clusterings for each element.

References

Gates, A. J., Wood, I. B., Hetrick, W. P., & Ahn, Y. Y. (2019). Element-centric clustering comparison unifies overlaps and hierarchy. Scientific reports, 9(1), 1-13. https://doi.org/10.1038/s41598-019-44892-y

Examples

```
km.res <- kmeans(iris[, 1:4], centers = 8)$cluster
hc.res <- hclust(dist(iris[, 1:4]))
element_sim_elscore(km.res, hc.res)
```

`element_sim_matrix` *Pairwise Comparison of Clusterings*

Description

Compare a set of clusterings by calculating their pairwise average element-centric clustering similarities.

Usage

```
element_sim_matrix(
  clustering_list,
  output_type = "matrix",
  alpha = 0.9,
  r = 1,
  rescale_path_type = "max",
  ppr_implementation = "prpack",
  dist_rescaled = FALSE,
  row_normalize = TRUE
)
```

Arguments

`clustering_list`	The list of clustering results, each of which is either: • A numeric/character/factor vector of cluster labels for each element. • A samples x clusters matrix/Matrix::Matrix of nonzero membership values. • An hclust object.
`output_type`	A string specifying whether the output should be a matrix or a data.frame.
`alpha`	A numeric giving the personalized PageRank damping factor; 1 - alpha is the restart probability for the PPR random walk.
`r`	A numeric hierarchical scaling parameter.
`rescale_path_type`	A string; rescale the hierarchical height by: • "max" : the maximum path from the root. • "min" : the minimum path form the root. • "linkage" : use the linkage distances in the clustering.
`ppr_implementation`	Choose a implementation for personalized page-rank calculation: • "prpack": use PPR algorithms in igraph. • "power_iteration": use power_iteration method.
`dist_rescaled`	A logical: if TRUE, the linkage distances are linearly rescaled to be in-between 0 and 1.
`row_normalize`	Whether to normalize all rows in clustering_result so they sum to one before calculating ECS. It is recommended to set this to TRUE, which will lead to slightly different ECS values compared to clusim.

Value

A matrix or data.frame containing the pairwise ECS values.

References

Gates, A. J., Wood, I. B., Hetrick, W. P., & Ahn, Y. Y. (2019). Element-centric clustering comparison unifies overlaps and hierarchy. Scientific reports, 9(1), 1-13. https://doi.org/10.1038/s41598-019-44892-y

Examples

```
# cluster across 20 random seeds
clustering.list <- lapply(1:20, function(x) kmeans(mtcars, centers = 3)$cluster)
element_sim_matrix(clustering.list, output_type = "matrix")
```

`getNNmatrix` *Computes the NN adjacency matrix given the neighbours*

Description

Computes the NN adjacency matrix given the neighbours

Usage

```
getNNmatrix(nnRanked, k = -1L, start = 0L, prune = 0)
```

Arguments

`nnRanked`	A matrix with the lists of the nearest neighbours for each point
`k`	The number of neighbours to consider. Defaults to `-1`, which means all neighbours.
`start`	The index of the first neighbour to consider. Defaults to `0`.
`prune`	The threshold to prune the SNN matrix. If -1, the function will only return the NN matrix. Defaults to `0`.

Value

A list with the NN and SNN adjacency matrices.

get_clusters_from_clustassess_object

Extract config-specific clusters from a ClustAssess object

Description

Given the output of the `automatic_stability_assessment` function, extract the clusters that are specific to a particular configuration of feature type, feature size, clustering method and, optionally, the number of clusters.

Usage

```
get_clusters_from_clustassess_object(
  clustassess_object,
  feature_type = NULL,
  feature_size = NULL,
  clustering_method = NULL,
  nclusters = NULL
)
```

Arguments

`clustassess_object`	Output of the `automatic_stability_assessment`.
`feature_type`	Type of feature used for dimensionality reduction. If `NULL`, it will select the first available feature.
`feature_size`	Size of the feature set used for clustering. If `NULL`, it will select the first available feature size.
`clustering_method`	Clustering method used. If `NULL`, it will select the first available clustering method.
`nclusters`	Number of clusters to extract. If `NULL`, all clusters are returned.

Value

A list of clusters that are specific to the given configuration. Each number of cluster will contain the list of partitions with that specific k and the ECC value indicating the overall stability of k.

get_highest_prune_param

Calculate the highest pruning parameter for the SNN graph given NN matrix

Description

Given a NN adjacency matrix, the function calculates the highest pruning parameter for the SNN graph that preserves the connectivity of the graph.

Usage

```
get_highest_prune_param(nn_matrix, n_neigh)
```

Arguments

`nn_matrix`	The adjacency matrix of the nearest neighbour graph.
`n_neigh`	The number of nearest neighbours.

Value

A list with the following fields:

- `prune_value`: The value of the highest pruning parameter.
- `adj_matrix`: The adjacency matrix of the SNN graph after pruning.

Note

Given the way the SNN graph is built, the possible values for the pruning parameter are limited and can be determined by the formula `i / (2 * n_neigh - i)`, where `i` is a number of nearest neighbours between 0 and `n_neigh`.

Examples

```
set.seed(2024)
# create an artificial pca embedding
pca_embedding <- matrix(
    c(runif(100 * 10), runif(100 * 10, min = 3, max = 4)),
    nrow = 200, byrow = TRUE
)
rownames(pca_embedding) <- as.character(1:200)
colnames(pca_embedding) <- paste("PC", 1:10)

# calculate the nn adjacency matrix
nn_matrix <- getNNmatrix(
    RANN::nn2(pca_embedding, k = 5)$nn.idx,
    5,
    0,
    -1
)$nn

get_highest_prune_param(nn_matrix, 5)$prune_value
```

---

get_highest_prune_param_embedding

Calculate the highest pruning parameter for the SNN graph given Embedding

---

Description

Given an embedding, the function calculates the highest pruning parameter for the SNN graph that preserves the connectivity of the graph.

Usage

```
get_highest_prune_param_embedding(embedding, n_neigh)
```

Arguments

`embedding`	A matrix associated with a PCA embedding. Embeddings from other dimensionality reduction techniques (such as LSI) can be used.
`n_neigh`	The number of nearest neighbours.

Value

The value of the highest pruning parameter.

Note

Given the way the SNN graph is built, the possible values for the pruning parameter are limited and can be determined by the formula `i / (2 * n_neigh - i)`, where `i` is a number of nearest neighbours between 0 and `n_neigh`.

Examples

```
set.seed(2024)
# create an artificial pca embedding
pca_embedding <- matrix(
    c(runif(100 * 10), runif(100 * 10, min = 3, max = 4)),
    nrow = 200, byrow = TRUE
)
rownames(pca_embedding) <- as.character(1:200)
colnames(pca_embedding) <- paste("PC", 1:10)

get_highest_prune_param_embedding(pca_embedding, 5)
```

`get_nn_conn_comps` *Relationship Between Nearest Neighbours and Connected Components*

Description

One of the steps in the clustering pipeline is building a k-nearest neighbour graph on a reduced-space embedding. This method assesses the relationship between different number of nearest neighbours and the connectivity of the graph. In the context of graph clustering, the number of connected components can be used as a lower bound for the number of clusters. The calculations are performed multiple times by changing the seed at each repetition.

Usage

```
get_nn_conn_comps(
  embedding,
  n_neigh_sequence,
  n_repetitions = 100,
  seed_sequence = NULL,
  include_umap = FALSE,
  umap_arguments = list()
)
```

Arguments

`embedding`	A matrix associated with a PCA embedding. Embeddings from other dimensionality reduction techniques (such as LSI) can be used.
`n_neigh_sequence`	A sequence of the number of nearest neighbours.
`n_repetitions`	The number of repetitions of applying the pipeline with different seeds; ignored if seed_sequence is provided by the user. Defaults to ‘100“.
`seed_sequence`	A custom seed sequence; if the value is NULL, the sequence will be built starting from 1 with a step of 100.
`include_umap`	A boolean value indicating whether to calculate the number of connected components for the UMAP embedding. Defaults to `FALSE`.
`umap_arguments`	Additional arguments passed to the the `uwot::umap` method.

Value

A list having one field associated with a number of nearest neighbours. Each value contains an array of the number of connected components obtained on the specified number of repetitions.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_emb <- matrix(runif(100 * 30), nrow = 100, byrow = TRUE)
rownames(pca_emb) <- as.character(1:100)
colnames(pca_emb) <- paste0("PCA_", 1:30)

nn_conn_comps_obj <- get_nn_conn_comps(
    embedding = pca_emb,
    n_neigh_sequence = c(2, 5),
    n_repetitions = 3,
    # arguments that are passed to the uwot function
    umap_arguments = list(
        min_dist = 0.3,
        metric = "cosine"
    )
)
plot_connected_comps_evolution(nn_conn_comps_obj)
```

---

marker_overlap *Cell-Wise Marker Gene Overlap*

Description

Calculates the per-cell overlap of previously calculated marker genes.

Usage

```
marker_overlap(
  markers1,
  markers2,
  clustering1,
  clustering2,
  n = 25,
  overlap_type = "jsi",
  rank_by = "-p_val",
  use_sign = TRUE
)
```

Arguments

`markers1`	The first data frame of marker genes, must contain columns called 'gene' and 'cluster'.
`markers2`	The second data frame of marker genes, must contain columns called 'gene' and 'cluster'.
`clustering1`	The first vector of cluster assignments.
`clustering2`	The second vector of cluster assignments.

`n`	The number of top n markers (ranked by rank_by) to use when calculating the overlap.
`overlap_type`	The type of overlap to calculated: must be one of 'jsi' for Jaccard similarity index and 'intersect' for intersect size.
`rank_by`	A character string giving the name of the column to rank marker genes by. Note the sign here: to rank by lowest p-value, preface the column name with a minus sign; to rank by highest value, where higher value indicates more discriminative genes (for example power in the ROC test), no sign is needed.
`use_sign`	A logical: should the sign of markers match for overlap calculations? So a gene must be a positive or a negative marker in both clusters being compared. If TRUE, markers1 and markers2 must have a 'avg_logFC' or 'avg_log2FC' column, from which the sign of the DE will be extracted.

Value

A vector of the marker gene overlap per cell.

Examples

```
suppressWarnings({
    set.seed(1234)
    library(Seurat)
    data("pbmc_small")

    # cluster with Louvain algorithm
    pbmc_small <- FindClusters(pbmc_small, resolution = 0.8, verbose = FALSE)

    # cluster with k-means
    pbmc.pca <- Embeddings(pbmc_small, "pca")
    pbmc_small@meta.data$kmeans_clusters <- kmeans(pbmc.pca, centers = 3)$cluster

    # compare the markers
    Idents(pbmc_small) <- pbmc_small@meta.data$seurat_clusters
    louvain.markers <- FindAllMarkers(pbmc_small,
        logfc.threshold = 1,
        test.use = "t",
        verbose = FALSE
    )

    Idents(pbmc_small) <- pbmc_small@meta.data$kmeans_clusters
    kmeans.markers <- FindAllMarkers(pbmc_small,
        logfc.threshold = 1,
        test.use = "t",
        verbose = FALSE
    )

    pbmc_small@meta.data$jsi <- marker_overlap(
        louvain.markers, kmeans.markers,
        pbmc_small@meta.data$seurat_clusters, pbmc_small@meta.data$kmeans_clusters
    )
    # which cells have the same markers, regardless of clustering?
    FeaturePlot(pbmc_small, "jsi")
})
```

---

merge_partitions *Merge Partitions*

Description

Merge flat disjoint clusterings whose pairwise ECS score is above a given threshold. The merging is done using a complete linkage approach.

Usage

```
merge_partitions(
  partition_list,
  ecs_thresh = 1,
  order_logic = c("freq", "avg_agreement", "none"),
  return_ecs_matrix = FALSE,
  check_ties = TRUE
)
```

Arguments

`partition_list`	A list of flat disjoint membership vectors.
`ecs_thresh`	A numeric: the ecs threshold.
`order_logic`	Variable indicating the method of ordering the partitions. It can take these three values: • "freq": order the partitions based on their frequencies. The partition with the highest frequency will be the first on the list (default). • "avg_agreement": order the partitions based on their average agreement index. The average agreement index of a partition is calculated as the mean of the ECS scores between that partition and the other partitions from the list. The partition with the highest agreement will be the first on the list. • "none": do not perform any ordering (not recommended). If selected, the average agreement scores will not be calculated.
`return_ecs_matrix`	A logical: if TRUE, the function will add the ECS matrix to the return list. Defaults to FALSE.
`check_ties`	A logical value that indicates whether to check for ties in the highest frequency partitions or not. If TRUE, the function will put at the first position the partition that has the highest similarity with the other partitions. Defaults to `FALSE`.

Value

a list of the merged partitions, together with their associated ECC score. If `return_ecs_matrix` is set to TRUE, the function will also return the ECS matrix.

Examples

```
initial_list <- list(c(1, 1, 2), c(2, 2, 2), c("B", "B", "A"))
merge_partitions(initial_list, 1)
```

`merge_resolutions` *Merge Partitions from different Resolutions*

Description

Merge partitions obtained with different resolution values. The partitions will be grouped based on the number of clusters. The identical partitions will be merged into a single partition by updating the frequency using the `merge_partitions` method.

Usage

```
merge_resolutions(res_obj)
```

Arguments

`res_obj`	A list associated to a configuration field from the object returned by the `assess_clustering_importance` method.

Value

A list having one field assigned to each number of clusters. A number of cluster will contain a list of all merged partitions. To avoid duplicates, `merged_partitions` with threshold 1 is applied.

`pac_convergence` *PAC Convergence Plot*

Description

Plot PAC across iterations for a set of k to assess convergence.

Usage

```
pac_convergence(pac_res, k_plot)
```

Arguments

`pac_res`	The data.frame output by consensus_cluster.
`k_plot`	A vector with values of k to plot.

Value

A ggplot2 object with the convergence plot. Convergence has been reached when the lines flatten out across k_plot values. out across

Examples

```
pac.res <- consensus_cluster(iris[, 1:4], k_max = 20)
pac_convergence(pac.res, k_plot = c(3, 5, 7, 9))
```

`pac_landscape` *PAC Landscape Plot*

Description

Plot final PAC values across range of k to find optimal number of clusters.

Usage

```
pac_landscape(pac_res, n_shade = max(pac_res$iteration)/5)
```

Arguments

`pac_res`	The data.frame output by consensus_cluster.
`n_shade`	The PAC values across the last n_shade iterations will be shaded to illustrate the how stable the PAC score is.

Value

A ggplot2 object with the final PAC vs k plot. A local minimum in the landscape indicates an especially stable value of k.

Examples

```
pac.res <- consensus_cluster(iris[, 1:4], k_max = 20)
pac_landscape(pac.res)
```

`plot_clustering_difference_facet` *Clustering Method Stability Facet Plot*

Description

Display the distribution of the EC consistency for each clustering method and each resolution value on a given embedding The `all` field of the object returned by the `get_clustering_difference_object` method is used.

Usage

```
plot_clustering_difference_facet(
  clust_object,
  embedding,
  low_limit = 0,
  high_limit = 1,
  grid = TRUE
)
```

Arguments

`clust_object`	An object returned by the `assess_clustering_stability` method.
`embedding`	An embedding (only the first two dimensions will be used for visualization).
`low_limit`	The lowest value of ECC that will be displayed on the embedding.
`high_limit`	The highest value of ECC that will be displayed on the embedding.
`grid`	Boolean value indicating whether the facet should be a grid (where each row is associated with a resolution value and each column with a clustering method) or a wrap.

Value

A ggplot2 object. #TODO should export

Examples

```
# FIXME fix the examples
# set.seed(2021)
# # create an artificial PCA embedding
# pca_embedding <- matrix(runif(100 * 30), nrow = 100)
# rownames(pca_embedding) <- as.character(1:100)
# colnames(pca_embedding) <- paste0("PCA_", 1:30)

# adj_matrix <- Seurat::FindNeighbors(pca_embedding,
#     k.param = 10,
#     nn.method = "rann",
#     verbose = FALSE,
#     compute.SNN = FALSE
# )$nn
# clust_diff_obj <- assess_clustering_stability(
#     graph_adjacency_matrix = adj_matrix,
#     resolution = c(0.5, 1),
#     n_repetitions = 10,
#     algorithm = 1:2,
#     verbose = FALSE
# )
# plot_clustering_difference_facet(clust_diff_obj, pca_embedding)
```

plot_clustering_overall_stability *Clustering Method Overall Stability Boxplot*

Description

Display EC consistency across clustering methods by summarising the distribution of the EC consistency for each number of clusters.

Usage

```
plot_clustering_overall_stability(
  clust_object,
  value_type = c("k", "resolution"),
  summary_function = stats::median
)
```

Arguments

`clust_object`	An object returned by the `assess_clustering_stability` method.
`value_type`	A string that specifies the type of value that was used for grouping the partitions and calculating the ECC score. It can be either `k` or `resolution`. Defaults to `k`.
`summary_function`	The function that will be used to summarize the distribution of the ECC values obtained for each number of clusters. Defaults to `median`.

Value

A ggplot2 object with the EC consistency distributions grouped by the clustering methods. Higher consistency indicates a more stable clustering.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_embedding <- matrix(runif(100 * 30), nrow = 100)
rownames(pca_embedding) <- paste0("cell_", seq_len(nrow(pca_embedding)))
colnames(pca_embedding) <- paste0("PC_", 1:30)


adj_matrix <- getNNmatrix(
    RANN::nn2(pca_embedding, k = 10)$nn.idx,
    10,
    0,
    -1
)$nn
rownames(adj_matrix) <- paste0("cell_", seq_len(nrow(adj_matrix)))
colnames(adj_matrix) <- paste0("cell_", seq_len(ncol(adj_matrix)))
```

```
# alternatively, the adj_matrix can be calculated
# using the `Seurat::FindNeighbors` function.

clust_diff_obj <- assess_clustering_stability(
    graph_adjacency_matrix = adj_matrix,
    resolution = c(0.5, 1),
    n_repetitions = 10,
    clustering_algorithm = 1:2,
    verbose = FALSE
)
plot_clustering_overall_stability(clust_diff_obj)
```

---

plot_clustering_per_value_stability
Clustering Method per value Stability Boxplot

Description

Display EC consistency across clustering methods, calculated for each value of the resolution parameter or the number of clusters.

Usage

```
plot_clustering_per_value_stability(
  clust_object,
  value_type = c("k", "resolution")
)
```

Arguments

clust_object	An object returned by the `assess_clustering_stability` method.
value_type	A string that specifies the type of value that was used for grouping the partitions and calculating the ECC score. It can be either `k` or `resolution`. Defaults to `k`.

Value

A ggplot2 object with the EC consistency distributions grouped by the clustering methods. Higher consistency indicates a more stable clustering. The X axis is decided by the `value_type` parameter.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_embedding <- matrix(runif(100 * 30), nrow = 100)
rownames(pca_embedding) <- paste0("cell_", seq_len(nrow(pca_embedding)))
colnames(pca_embedding) <- paste0("PC_", 1:30)
```

```
adj_matrix <- getNNmatrix(
    RANN::nn2(pca_embedding, k = 10)$nn.idx,
    10,
    0,
    -1
)$nn
rownames(adj_matrix) <- paste0("cell_", seq_len(nrow(adj_matrix)))
colnames(adj_matrix) <- paste0("cell_", seq_len(ncol(adj_matrix)))

# alternatively, the adj_matrix can be calculated
# using the `Seurat::FindNeighbors` function.

clust_diff_obj <- assess_clustering_stability(
    graph_adjacency_matrix = adj_matrix,
    resolution = c(0.5, 1),
    n_repetitions = 10,
    clustering_algorithm = 1:2,
    verbose = FALSE
)
plot_clustering_per_value_stability(clust_diff_obj)
```

`plot_connected_comps_evolution`
Relationship Between Number of Nearest Neighbours and Graph Connectivity

---

Description

Display the distribution of the number connected components obtained for each number of neighbours across random seeds.

Usage

```
plot_connected_comps_evolution(nn_conn_comps_object)
```

Arguments

`nn_conn_comps_object`	An object or a concatenation of objects returned by the `get_nn_conn_comps` method.

Value

A ggplot2 object with boxplots for the connected component distributions.

Note

The number of connected components is displayed on a logarithmic scale.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_emb <- matrix(runif(100 * 30), nrow = 100, byrow = TRUE)
rownames(pca_emb) <- as.character(1:100)
colnames(pca_emb) <- paste0("PCA_", 1:30)

nn_conn_comps_obj <- get_nn_conn_comps(
    embedding = pca_emb,
    n_neigh_sequence = c(2, 5),
    n_repetitions = 3,
    # arguments that are passed to the uwot function
    umap_arguments = list(
        min_dist = 0.3,
        metric = "cosine"
    )
)
plot_connected_comps_evolution(nn_conn_comps_obj)
```

---

plot_feature_overall_stability_boxplot
Overall Feature Stability Boxplot

Description

Display EC consistency for each feature set and for each step. Above each boxplot there is a number representing the step (or the size of the subset). The ECC values are extracted for each resolution value and summarized using the `summary_function` parameter.

Usage

```
plot_feature_overall_stability_boxplot(
  feature_object_list,
  summary_function = stats::median,
  text_size = 4,
  boxplot_width = 0.4,
  dodge_width = 0.7,
  return_df = FALSE
)
```

Arguments

`feature_object_list`	An object or a concatenation of objects returned by the `assess_feature_stability` method
`summary_function`	The function that will be used to summarize the ECC values. Defaults to `median`.
`text_size`	The size of the labels above boxplots

`boxplot_width`	Used for adjusting the width of the boxplots; the value will be passed to the `width` argument of the `ggplot2::geom_boxplot` method.
`dodge_width`	Used for adjusting the horizontal position of the boxplot; the value will be passed to the `width` argument of the `ggplot2::position_dodge` method.
`return_df`	If TRUE, the function will return the ECS values as a dataframe. Default is `FALSE`.

Value

A ggplot2 object.

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(100 * 10), runif(100 * 10, min = 3, max = 4)),
    nrow = 200, byrow = TRUE
)
rownames(expr_matrix) <- as.character(1:200)
colnames(expr_matrix) <- paste("feature", 1:10)

feature_stability_result <- assess_feature_stability(
    data_matrix = t(expr_matrix),
    feature_set = colnames(expr_matrix),
    steps = 5,
    feature_type = "feature_name",
    resolution = c(0.1, 0.5, 1),
    n_repetitions = 10,
    umap_arguments = list(
        # the following parameters are used by the umap function
        # and are not mandatory
        n_neighbors = 3,
        approx_pow = TRUE,
        n_epochs = 0,
        init = "random",
        min_dist = 0.3
    ),
    clustering_algorithm = 1
)
plot_feature_overall_stability_boxplot(feature_stability_result)
```

`plot_feature_overall_stability_incremental`
Overall Feature Stability Incremental Boxplot

Description

Perform an incremental ECS between two consecutive feature steps. The ECS values are extracted for every resolution value and summarized using a function (e.g. median, mean, etc.).

Usage

```
plot_feature_overall_stability_incremental(
  feature_object_list,
  summary_function = stats::median,
  dodge_width = 0.7,
  text_size = 4,
  boxplot_width = 0.4,
  return_df = FALSE
)
```

Arguments

`feature_object_list`	An object or a concatenation of objects returned by the `assess_feature_stability` method.
`summary_function`	The function used to summarize the ECS values. Default is `median`.
`dodge_width`	Used for adjusting the horizontal position of the boxplot; the value will be passed to the `width` argument of the `ggplot2::position_dodge` method.
`text_size`	The size of the labels above boxplots.
`boxplot_width`	Used for adjusting the width of the boxplots; the value will be passed to the `width` argument of the `ggplot2::geom_boxplot` method.
`return_df`	If TRUE, the function will return the ECS values as a dataframe. Default is `FALSE`.

Value

A ggplot2 object with ECS distribution will be displayed as a boxplot. Above each boxplot there will be a pair of numbers representing the two steps that are compared.

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(50 * 10), runif(50 * 10, min = 3, max = 4)),
    nrow = 100, byrow = TRUE
)
rownames(expr_matrix) <- as.character(1:100)
colnames(expr_matrix) <- paste("feature", 1:10)

feature_stability_result <- assess_feature_stability(
    data_matrix = t(expr_matrix),
    feature_set = colnames(expr_matrix),
    steps = c(5, 10),
    feature_type = "feature_name",
    resolution = c(0.1, 0.5),
    n_repetitions = 3,
    umap_arguments = list(
        # the following parameters are used by the umap function
        # and are not mandatory
        n_neighbors = 3,
        approx_pow = TRUE,
        n_epochs = 0,
        init = "random",
        min_dist = 0.3
    ),
    clustering_algorithm = 1
)
plot_feature_overall_stability_incremental(feature_stability_result)
```

plot_feature_per_resolution_stability_boxplot

Per resolution Feature Stability Boxplot

Description

Display EC consistency for each feature set and for each step. Above each boxplot there is a number representing the step (or the size of the subset). The ECC values are extracted depdening on the resolution value provided by the user.

Usage

```
plot_feature_per_resolution_stability_boxplot(
  feature_object_list,
  resolution,
  violin_plot = FALSE,
  text_size = 4,
  boxplot_width = 0.4,
  dodge_width = 0.7,
  return_df = FALSE
)
```

Arguments

`feature_object_list`	An object or a concatenation of objects returned by the `assess_feature_stability` method
`resolution`	The resolution value for which the ECC will be extracted.
`violin_plot`	If TRUE, the function will return a violin plot instead of a boxplot. Default is `FALSE`.
`text_size`	The size of the labels above boxplots
`boxplot_width`	Used for adjusting the width of the boxplots; the value will be passed to the `width` argument of the `ggplot2::geom_boxplot` method.
`dodge_width`	Used for adjusting the horizontal position of the boxplot; the value will be passed to the `width` argument of the `ggplot2::position_dodge` method.

`return_df`	If TRUE, the function will return the ECS values as a dataframe. Default is `FALSE`.

Value

A ggplot2 object.

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(100 * 10), runif(100 * 10, min = 3, max = 4)),
    nrow = 200, byrow = TRUE
)
rownames(expr_matrix) <- as.character(1:200)
colnames(expr_matrix) <- paste("feature", 1:10)

feature_stability_result <- assess_feature_stability(
    data_matrix = t(expr_matrix),
    feature_set = colnames(expr_matrix),
    steps = 5,
    feature_type = "feature_name",
    resolution = c(0.1, 0.5, 1),
    n_repetitions = 10,
    umap_arguments = list(
        # the following parameters are used by the umap function
        # and are not mandatory
        n_neighbors = 3,
        approx_pow = TRUE,
        n_epochs = 0,
        init = "random",
        min_dist = 0.3
    ),
    clustering_algorithm = 1
)
plot_feature_per_resolution_stability_boxplot(feature_stability_result, 0.5)
```

`plot_feature_per_resolution_stability_incremental`
Per resolution - Feature Stability Incremental Boxplot

Description

Perform an incremental ECS between two consecutive feature steps. The ECS values are extracted only for a specified resolution value.

Usage

```
plot_feature_per_resolution_stability_incremental(
  feature_object_list,
  resolution,
  dodge_width = 0.7,
  text_size = 4,
  boxplot_width = 0.4,
  return_df = FALSE
)
```

Arguments

`feature_object_list`	An object or a concatenation of objects returned by the `assess_feature_stability` method.
`resolution`	The resolution value for which the ECS will be extracted.
`dodge_width`	Used for adjusting the horizontal position of the boxplot; the value will be passed to the `width` argument of the `ggplot2::position_dodge` method.
`text_size`	The size of the labels above boxplots.
`boxplot_width`	Used for adjusting the width of the boxplots; the value will be passed to the `width` argument of the `ggplot2::geom_boxplot` method.
`return_df`	If TRUE, the function will return the ECS values as a dataframe. Default is `FALSE`.

Value

A ggplot2 object with ECS distribution will be displayed as a boxplot. Above each boxplot there will be a pair of numbers representing the two steps that are compared.

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(50 * 10), runif(50 * 10, min = 3, max = 4)),
    nrow = 100, byrow = TRUE
)
rownames(expr_matrix) <- as.character(1:100)
colnames(expr_matrix) <- paste("feature", 1:10)

feature_stability_result <- assess_feature_stability(
    data_matrix = t(expr_matrix),
    feature_set = colnames(expr_matrix),
    steps = c(5, 10),
    feature_type = "feature_name",
    resolution = c(0.1, 0.5),
    n_repetitions = 3,
    umap_arguments = list(
        # the following parameters are used by the umap function
        # and are not mandatory
        n_neighbors = 3,
        approx_pow = TRUE,
        n_epochs = 0,
        init = "random",
        min_dist = 0.3
    ),
    clustering_algorithm = 1
)
plot_feature_per_resolution_stability_incremental(feature_stability_result, 0.1)
```

---

plot_feature_stability_ecs_facet
Feature Stability - EC Consistency Facet Plot

Description

Display a facet of plots where each subpanel is associated with a feature set and illustrates the distribution of the EC consistency score over the UMAP embedding.

Usage

```
plot_feature_stability_ecs_facet(
  feature_object_list,
  resolution,
  n_facet_cols = 3,
  point_size = 0.3
)
```

Arguments

`feature_object_list`	An object or a concatenation of objects returned by the `assess_feature_stability` method
`resolution`	The resolution value for which the ECS will be extracted.
`n_facet_cols`	The number of facet's columns.
`point_size`	The size of the points displayed on the plot.

Value

A ggplot2 object

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(100 * 10), runif(50 * 10, min = 3, max = 4)),
    nrow = 150, byrow = TRUE
)
rownames(expr_matrix) <- as.character(1:150)
colnames(expr_matrix) <- paste("feature", 1:10)

feature_stability_result <- assess_feature_stability(
    data_matrix = t(expr_matrix),
    feature_set = colnames(expr_matrix),
    steps = 5,
    feature_type = "feature_name",
    resolution = c(0.1, 0.5, 1),
    n_repetitions = 10,
    clustering_algorithm = 1
)
plot_feature_stability_ecs_facet(
    feature_stability_result,
    0.5,
    point_size = 2
)
```

`plot_feature_stability_mb_facet`
Feature Stability - Cluster Membership Facet Plot

Description

Display a facet of plots where each subpanel is associated with a feature set and illustrates the distribution of the most frequent partition over the UMAP embedding.

Usage

```
plot_feature_stability_mb_facet(
  feature_object_list,
  resolution,
  text_size = 5,
  n_facet_cols = 3,
  point_size = 0.3
)
```

Arguments

`feature_object_list`	An object or a concatenation of objects returned by the `assess_feature_stability` method

`resolution`	The resolution value for which the ECS will be extracted.
`text_size`	The size of the cluster label
`n_facet_cols`	The number of facet's columns.
`point_size`	The size of the points displayed on the plot.

Value

A ggplot2 object.

Examples

```
set.seed(2024)
# create an artificial expression matrix
expr_matrix <- matrix(
    c(runif(100 * 10), runif(50 * 10, min = 3, max = 4)),
    nrow = 150, byrow = TRUE
)
rownames(expr_matrix) <- as.character(1:150)
colnames(expr_matrix) <- paste("feature", 1:10)

feature_stability_result <- assess_feature_stability(
    data_matrix = t(expr_matrix),
    feature_set = colnames(expr_matrix),
    steps = 5,
    feature_type = "feature_name",
    resolution = c(0.1, 0.5, 1),
    n_repetitions = 10,
    clustering_algorithm = 1
)
plot_feature_stability_mb_facet(
    feature_stability_result,
    0.5,
    point_size = 2
)
```

`plot_k_n_partitions` *Relationship Between the Number of Clusters and the Number of Unique Partitions*

Description

For each configuration provided in clust_object, display how many different partitions with the same number of clusters can be obtained by changing the seed.

Usage

```
plot_k_n_partitions(
  clust_object,
  colour_information = c("ecc", "freq_part"),
  dodge_width = 0.3,
  pt_size_range = c(1.5, 4),
  summary_function = stats::median,
  y_step = 5
)
```

Arguments

`clust_object`	An object returned by the `assess_clustering_stability` method.
`colour_information`	String that specifies the information type that will be illustrated using gradient colour: either `freq_part` for the frequency of the most common partition or `ecc` for the Element-Centric Consistency of the partitions obtained when the the number of clusters is fixed. Defaults to `ecc`.
`dodge_width`	Used for adjusting the distance between the boxplots representing a clustering method. Defaults to `0.3`.
`pt_size_range`	Indicates the minimum and the maximum size a point on the plot can have. Defaults to `c(1.5, 4)`.
`summary_function`	The function that will be used to summarize the distribution of the ECC values obtained for each number of clusters. Defaults to `median`.
`y_step`	The step used for the y-axis. Defaults to `5`.

Value

A ggplot2 object. The color gradient suggests the frequency of the most common partition relative to the total number of appearances of that specific number of clusters or the Element-Centric Consistency of the partitions. The size illustrates the frequency of the partitions with *k* clusters relative to the total number of partitions. The shape of the points indicates the clustering method.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_embedding <- matrix(runif(100 * 30), nrow = 100)
rownames(pca_embedding) <- paste0("cell_", seq_len(nrow(pca_embedding)))
colnames(pca_embedding) <- paste0("PC_", 1:30)


adj_matrix <- getNNmatrix(
    RANN::nn2(pca_embedding, k = 10)$nn.idx,
    10,
    0,
    -1
)$nn
rownames(adj_matrix) <- paste0("cell_", seq_len(nrow(adj_matrix)))
colnames(adj_matrix) <- paste0("cell_", seq_len(ncol(adj_matrix)))

# alternatively, the adj_matrix can be calculated
# using the `Seurat::FindNeighbors` function.

clust_diff_obj <- assess_clustering_stability(
    graph_adjacency_matrix = adj_matrix,
    resolution = c(0.5, 1),
    n_repetitions = 10,
    clustering_algorithm = 1:2,
    verbose = FALSE
)
plot_k_n_partitions(clust_diff_obj)
```

plot_k_resolution_corresp

Correspondence Between Resolution and the Number of Clusters

Description

For each configuration provided in the clust_object, display what number of clusters appear for different values of the resolution parameters.

Usage

```
plot_k_resolution_corresp(
  clust_object,
  colour_information = c("ecc", "freq_k"),
  dodge_width = 0.3,
  pt_size_range = c(1.5, 4),
  summary_function = stats::median
)
```

Arguments

`clust_object`	An object returned by the `assess_clustering_stability` method.
`colour_information`	String that specifies the information type that will be illustrated using gradient colour: either `freq_part` for the frequency of the most common partition or `ecc` for the Element-Centric Consistency of the partitions obtained when the the number of clusters is fixed. Defaults to `ecc`.
`dodge_width`	Used for adjusting the distance between the boxplots representing a clustering method. Defaults to `0.3`.
`pt_size_range`	Indicates the minimum and the maximum size a point on the plot can have. Defaults to `c(1.5, 4)`.

`summary_function`
: The function that will be used to summarize the distribution of the ECC values obtained for each number of clusters. Defaults to `median`.

Value

A ggplot2 object. Different shapes of points indicate different parameter configuration, while the color illustrates the frequency of the most common partition or the Element-Centric Consistency of the partitions. The frequency is calculated as the fraction between the number of total appearances of partitions with a specific number of clusters and resolution value and the number of runs. The size illustrates the frequency of the most common partition with *k* clusters relative to the partitions obtained with the same resolution value and have *k* clusters.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_embedding <- matrix(runif(100 * 30), nrow = 100)
rownames(pca_embedding) <- paste0("cell_", seq_len(nrow(pca_embedding)))
colnames(pca_embedding) <- paste0("PC_", 1:30)


adj_matrix <- getNNmatrix(
    RANN::nn2(pca_embedding, k = 10)$nn.idx,
    10,
    0,
    -1
)$nn
rownames(adj_matrix) <- paste0("cell_", seq_len(nrow(adj_matrix)))
colnames(adj_matrix) <- paste0("cell_", seq_len(ncol(adj_matrix)))

# alternatively, the adj_matrix can be calculated
# using the `Seurat::FindNeighbors` function.

clust_diff_obj <- assess_clustering_stability(
    graph_adjacency_matrix = adj_matrix,
    resolution = c(0.5, 1),
    n_repetitions = 10,
    clustering_algorithm = 1:2,
    verbose = FALSE
)
plot_k_resolution_corresp(clust_diff_obj)
```

`plot_n_neigh_ecs` *Graph construction parameters - ECC facet*

Description

Display, for all configurations consisting in different number of neighbours, graph types and base embeddings, the EC Consistency of the partitions obtained over multiple runs on an UMAP embedding.

Usage

```
plot_n_neigh_ecs(nn_ecs_object, boxplot_width = 0.5)
```

Arguments

`nn_ecs_object`	An object or a concatenation of objects returned by the `get_nn_importance` method.
`boxplot_width`	Used for adjusting the width of the boxplots; the value will be passed to the `width` argument of the `ggplot2::geom_boxplot` method.

Value

A ggplot2 object.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_emb <- matrix(runif(100 * 30), nrow = 100, byrow = TRUE)
rownames(pca_emb) <- as.character(1:100)
colnames(pca_emb) <- paste0("PC_", 1:30)

nn_stability_obj <- assess_nn_stability(
    embedding = pca_emb,
    n_neigh_sequence = c(10, 15, 20),
    n_repetitions = 10,
    graph_reduction_type = "PCA",
    clustering_algorithm = 1
)
plot_n_neigh_ecs(nn_stability_obj)
```

`plot_n_neigh_k_correspondence` *Relationship Between Number of Nearest Neighbours and Number of Clusters*

Description

Display the distribution of the number of clusters obtained for each number of neighbours across random seeds.

Usage

```
plot_n_neigh_k_correspondence(nn_object_n_clusters)
```

Arguments

`nn_object_n_clusters`	An object or a concatenation of objects returned by the `get_nn_importance` method.

Value

A ggplot2 object with the distributions displayed as boxplots.

Note

The number of clusters is displayed on a logarithmic scale.

Examples

```
set.seed(2024)
# create an artificial PCA embedding
pca_emb <- matrix(runif(100 * 30), nrow = 100, byrow = TRUE)
rownames(pca_emb) <- as.character(1:100)
colnames(pca_emb) <- paste0("PC_", 1:30)

nn_stability_obj <- assess_nn_stability(
    embedding = pca_emb,
    n_neigh_sequence = c(10, 15, 20),
    n_repetitions = 10,
    graph_reduction_type = "PCA",
    clustering_algorithm = 1
)
plot_n_neigh_k_correspondence(nn_stability_obj)
```

`server_comparisons` *Server - Comparison module*

Description

Creates the backend interface for the comparison module inside the ClustAssess Shiny application.

Usage

```
server_comparisons(id, chosen_config, chosen_method)
```

Arguments

`id`	The id of the module, used to acess the UI elements.
`chosen_config`	A reactive object that contains the chosen configuration from the Dimensionality Reduction tab.
`chosen_method`	A reactive object that contains the chosen method from the Clustering tab.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

server_dimensionality_reduction

Server - Dimensionality reduction module

Description

Creates the backend interface for the dimensionality reduction module inside the ClustAssess Shiny application.

Usage

```
server_dimensionality_reduction(id, parent_session)
```

Arguments

`id`	The id of the module, used to acess the UI elements.
`parent_session`	The session of the parent module, used to control the tabs of the application.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

server_graph_clustering

Server - Graph clustering module

Description

Creates the backend interface for the graph clustering module inside the ClustAssess Shiny application.

Usage

```
server_graph_clustering(id, feature_choice, parent_session)
```

Arguments

`id`	The id of the module, used to acess the UI elements.
`feature_choice`	A reactive object that contains the chosen configuration from the Dimensionality Reduction tab.
`parent_session`	The session of the parent module, used to control the tabs of the application.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

server_graph_construction

Server - Graph construction module

Description

Creates the backend interface for the graph construction module inside the ClustAssess Shiny application.

Usage

```
server_graph_construction(id, chosen_config)
```

Arguments

`id`	The id of the module, used to acess the UI elements.
`chosen_config`	A reactive object that contains the chosen configuration from the Dimensionality Reduction tab.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

server_landing_page

Server - Landing page module

Description

Creates the backend interface for the landing page module inside the ClustAssess Shiny application.

Usage

```
server_landing_page(
  id,
  height_ratio,
  dimension,
  parent_session,
  organism = "hsapiens"
)
```

Arguments

`id`	The id of the module, used to acess the UI elements.
`height_ratio`	A reactive object that contains the height ratio of the plots in the application (the height of the plot is calculated using the height ratio and the height of the webpage).
`dimension`	A reactive object that contains the dimensions of the webpage.
`parent_session`	The session of the parent module, used to control the tabs of the application.
`organism`	The organism of the dataset, which will be used in the enrichment analysis.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

`server_sandbox` *Server - Sandbox module*

Description

Creates the backend interface for the sandbox module inside the ClustAssess Shiny application.

Usage

```
server_sandbox(id)
```

Arguments

`id`	The id of the module, used to acess the UI elements.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

`ui_comparisons` *UI - Comparison module*

Description

Creates the UI interface for the comparison module inside the ClustAssess Shiny application.

Usage

```
ui_comparisons(id)
```

Arguments

`id`	The id of the module, used to identify the UI elements.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

`ui_dimensionality_reduction` *UI - Dimensionality reduction module*

Description

Creates the UI interface for the dimensionality reduction module inside the ClustAssess Shiny application.

Usage

```
ui_dimensionality_reduction(id)
```

Arguments

`id`	The id of the module, used to identify the UI elements.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

`ui_graph_clustering` *UI - Graph clustering module*

Description

Creates the UI interface for the graph clustering module inside the ClustAssess Shiny application.

Usage

```
ui_graph_clustering(id)
```

Arguments

`id`	The id of the module, used to identify the UI elements.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

`ui_graph_construction` *UI - Graph construction module*

Description

Creates the UI interface for the graph construction module inside the ClustAssess Shiny application.

Usage

```
ui_graph_construction(id)
```

Arguments

`id`	The id of the module, used to identify the UI elements.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

`ui_landing_page` *UI - Landing page module*

Description

Creates the UI interface for the landing page module inside the ClustAssess Shiny application.

Usage

```
ui_landing_page(id)
```

Arguments

`id`	The id of the module, used to identify the UI elements.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

`ui_sandbox` *UI - Sandbox module*

Description

Creates the UI interface for the sandbox module inside the ClustAssess Shiny application.

Usage

```
ui_sandbox(id)
```

Arguments

`id`	The id of the module, used to identify the UI elements.

Note

This function should not be called directly, but in the context of the app that is created using the `write_shiny_app` function.

weighted_element_consistency *Weighted Element-Centric Consistency*

Description

Calculate the weighted element-centric consistency of a set of clusterings. The weights are used to give more importance to some clusterings over others.

Usage

```
weighted_element_consistency(
  clustering_list,
  weights = NULL,
  calculate_sim_matrix = FALSE
)
```

Arguments

`clustering_list`
: The list of clustering results, each of which is either:
  - A numeric/character/factor vector of cluster labels for each element.
  - A samples x clusters matrix/Matrix::Matrix of nonzero membership values.
  - An hclust object.

`weights`
: A numeric vector of weights for each clustering in `clustering_list`. If `NULL`, then all weights will be equal to 1. Defaults to `NULL`.

`calculate_sim_matrix`
: A logical value that indicates whether to calculate the similarity matrix or not along with the consistency score. Defaults to `FALSE`.

Value

A vector containing the weighted element-wise consistency. If `calculate_sim_matrix` is set to `TRUE`, the element similarity matrix will be returned as well.

Note

The weighted ECC will be calculated as $\frac{\sum_i \sum_j w_i w_j ECS(i,j)}{\sum_i w_i}$

Examples

```
# cluster across 20 random seeds
clustering_list <- lapply(1:20, function(x) kmeans(mtcars, centers = 3)$cluster)
weights <- sample(1:10, 20, replace = TRUE)
weighted_element_consistency(clustering_list, weights = weights)
```

write_objects *Write the objects for the ClustAssess ShinyApp*

Description

Given the output of the ClustAssess pipeline, the expression matrix and the metadata, this function creates the files needed for the ClustAssess ShinyApp. The files are written in the project_folder and are the following:

- metadata.rds: the metadata file
- stability.h5: contains the stability results
- expression.h5: contains the expression matrix and the rank matrix

Usage

```
write_objects(
  clustassess_object,
  expression_matrix,
  metadata,
  project_folder = ".",
  compression_level = 6,
  chunk_size = 100,
  gene_variance_threshold = 0,
  summary_function = stats::median,
  qualpalr_colorspace = "pretty"
)
```

Arguments

`clustassess_object`	The output of the ClustAssess automatic pipeline
`expression_matrix`	The expression matrix
`metadata`	The metadata
`project_folder`	The folder where the files will be written
`compression_level`	The compression level for the h5 files (See ‘rhdf5::h5createFile“ for more details)
`chunk_size`	The chunk size for the rank matrix (See `rhdf5::h5createDataset` for more details)
`gene_variance_threshold`	The threshold for the gene variance; genes with variance below this threshold will be removed
`summary_function`	The function used for summarizing the stability values; the default is `median`
`qualpalr_colorspace`	The colorspace used for generating the colors; the default is `pretty`

Value

NULL (the files are written in the project_folder)

`write_shiny_app` *Create the ClustAssess ShinyApp*

Description

Creates the ClustAssess ShinyApp based on the output of the automatic ClustAssess pipeline. In addition to that, the expression matrix and the metadata dataframe are provided as input to the ShinyApp.

Usage

```
write_shiny_app(
  object,
  metadata = NULL,
  assay_name = NULL,
  clustassess_object,
  project_folder,
  compression_level = 6,
  summary_function = stats::median,
  shiny_app_title = "",
  organism_enrichment = "hsapiens",
  height_ratio = 0.6,
  qualpalr_colorspace = "pretty"
)

## S3 method for class 'Seurat'
write_shiny_app(
  object,
  metadata = NULL,
  assay_name,
  clustassess_object,
  project_folder,
  compression_level = 6,
  summary_function = stats::median,
  shiny_app_title = "",
  organism_enrichment = "hsapiens",
  height_ratio = 0.6,
  qualpalr_colorspace = "pretty"
)

## Default S3 method:
write_shiny_app(
  object,
  metadata = NULL,
  assay_name = NULL,
  clustassess_object,
  project_folder,
  compression_level = 6,
  summary_function = stats::median,
  shiny_app_title = "",
  organism_enrichment = "hsapiens",
  height_ratio = 0.6,
  qualpalr_colorspace = "pretty"
)
```

Arguments

`object`	A Seurat object or an expression matrix
`metadata`	The metadata dataframe. This parameter will be ignored if the object is a Seurat object.
`assay_name`	The name of the assay to be used to extract the expression matrix from the Seurat object. This parameter will be ignored if the object is not a Seurat object.
`clustassess_object`	The output of the ClustAssess automatic pipeline
`project_folder`	The folder where the files will be written
`compression_level`	The compression level for the h5 files (See ‘rhdf5::h5createFile“ for more details)
`summary_function`	The function used for summarizing the stability values; the default is `median`
`shiny_app_title`	The title of the shiny app
`organism_enrichment`	The organism used for the enrichment analysis; the default is `hsapiens`
`height_ratio`	The ratio of the height of the plot to the height of the browser; the default is `0.6`
`qualpalr_colorspace`	The colorspace used for generating the colors; the default is `pretty`

Index