Package: scDiagnostics (via r-universe)

May 30, 2026

Type Package

Title Cell type annotation diagnostics

Version 1.6.0

Description The scDiagnostics package provides diagnostic plots to assess the quality of cell type assignments from single cell gene expression profiles. The implemented functionality allows to assess the reliability of cell type annotations, investigate gene expression patterns, and explore relationships between different cell types in query and reference datasets allowing users to detect potential misalignments between reference and query datasets. The package also provides visualization capabilities for diagnostics purposes.

License Artistic-2.0

URL https://github.com/ccb-hms/scDiagnostics

BugReports https://github.com/ccb-hms/scDiagnostics/issues

Depends R (>= 4.4.0)

Imports SingleCellExperiment, methods, isotree, FNN, igraph, ggplot2, GGally, ggridges, SummarizedExperiment, ranger, transport, cramer, rlang, bluster, scales, MASS, stringr, Matrix, grDevices

Suggests AUCell, BiocStyle, knitr, rmarkdown, scran, scRNAseq, SingleR, celldex, scuttle, scater, dplyr, ComplexHeatmap, grid, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews Annotation, Classification, Clustering, GeneExpression, RNASeq, SingleCell, Software, Transcriptomics

Encoding UTF-8

LazyDataCompression xz

RoxygenNote 7.3.2

Config/testthat/edition 3

LazyData true

Config/pak/sysreqs libglpk-dev libicu-dev libxml2-dev libssl-dev zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-04-28 13:03:37 UTC

RemoteUrl https://github.com/bioc/scDiagnostics

RemoteRef RELEASE_3_23

RemoteSha 0f4d33b878da8645b5cb8425408a4e1e06bddcd3

Contents

boxplotPCA *Plot Principal Components for Different Cell Types*

Description

This function generates a ggplot2 visualization of principal components (PCs) for different cell types across two datasets (query and reference), using either boxplots or violin plots.

Usage

```
boxplotPCA(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:5,
  shape = c("box", "violin"),
  assay_name = "logcounts",
  max_cells_query = NULL,
  max_cells_ref = NULL
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
cell_types	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.
pc_subset	A numeric vector specifying which principal components to include in the plot. Default is PC1 to PC5.
shape	Character string indicating the plot type: "box" for boxplots or "violin" for violin plots. Default is "box".
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is NULL.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is NULL.

Details

The function boxplotPCA is designed to provide a visualization of principal component analysis (PCA) results. It projects the query dataset onto the principal components obtained from the reference dataset. The results are then visualized as boxplots or violin plots, grouped by cell types and datasets (query and reference). This allows for a comparative analysis of the distributions of the principal components across different cell types and datasets. The function internally calls projectPCA to perform the PCA projection. It then reshapes the output data into a long format suitable for ggplot2 plotting.

Value

A ggplot object representing the boxplots or violin plots of specified principal components for the given cell types and datasets.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

Examples

```
# Load data
data("reference_data")
data("query_data")

# Plot the PC data with boxplots (default)
pc_plot <- boxplotPCA(query_data = query_data,
                      reference_data = reference_data,
                      cell_types = c("CD4", "CD8", "B_and_plasma", "Myeloid"),
                      query_cell_type_col = "SingleR_annotation",
                      ref_cell_type_col = "expert_annotation",
                      pc_subset = 1:6)
pc_plot

# Plot the PC data with violin plots
pc_violin <- boxplotPCA(query_data = query_data,
                        reference_data = reference_data,
                        cell_types = c("CD4", "CD8", "B_and_plasma", "Myeloid"),
                        query_cell_type_col = "SingleR_annotation",
                        ref_cell_type_col = "expert_annotation",
                        pc_subset = 1:6,
                        shape = "violin")
pc_violin
```

---

calculateCategorizationEntropy

Calculate Categorization Entropy

Description

This function takes a matrix of category scores (cell type by cells) and calculates the entropy of the category probabilities for each cell. This gives a sense of how confident the cell type assignments are. High entropy = lots of plausible category assignments = low confidence. Low entropy = only one or two plausible categories = high confidence. This is confidence in the vernacular sense, not in the "confidence interval" statistical sense. Also note that the entropy tells you nothing about whether or not the assignments are correct – see the other functionality in the package for that. This functionality can be used for assessing how comparatively confident different sets of assignments are (given that the number of categories is the same).

Usage

```
calculateCategorizationEntropy(
  X,
  inverseNormalTransformationform = FALSE,
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

`X`	A matrix of category scores.
`inverseNormalTransformationform`	If TRUE, apply inverse normal transformation to X. Default is FALSE.
`plot`	If TRUE, plot a histogram of the entropies. Default is TRUE.
`verbose`	If TRUE, display messages about the calculations. Default is TRUE.

Details

The function checks if X is already on the probability scale. Otherwise, it applies softmax column-wise.

You can think about entropies on a scale from 0 to a maximum that depends on the number of categories. This is the function for entropy (minus input checking): `entropy(p) = -sum(p*log(p))` . If that input vector p is a uniform distribution over the `length(p)` categories, the entropy will be a high as possible.

Value

A vector of entropy values for each column in X.

Author(s)

Andrew Ghazi, <andrew_ghazi@hms.harvard.edu>

Examples

```
# Simulate 500 cells with scores on 4 possible cell types
X <- rnorm(500 * 4) |> matrix(nrow = 4)
X[1, 1:250] <- X[1, 1:250] + 5 # Make the first category highly scored in the first 250 cells

# The function will issue a message about softmaxing the scores, and the entropy histogram will be
# bimodal since we made half of the cells clearly category 1 while the other half are roughly even.
entropy_scores <- calculateCategorizationEntropy(X)
```

`calculateCellDistancesSimilarity` *Function to Calculate Bhattacharyya Coefficients and Hellinger Distances*

Description

This function computes Bhattacharyya coefficients and Hellinger distances to quantify the similarity of density distributions between query cells and reference data for each cell type.

Usage

```
calculateCellDistancesSimilarity(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  cell_names_query,
  pc_subset = 1:5,
  assay_name = "logcounts",
  max_cells_ref = 5000
)
```

Arguments

`query_data`	A `SingleCellExperiment` object containing numeric expression matrix for the query cells.
`reference_data`	A `SingleCellExperiment` object containing numeric expression matrix for the reference cells.
`query_cell_type_col`	The column name in the `colData` of `query_data` that identifies the cell types.
`ref_cell_type_col`	The column name in the `colData` of `reference_data` that identifies the cell types.
`cell_types`	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.

`cell_names_query`
: A character vector specifying the names of the query cells for which to compute distance measures.

`pc_subset`
: A numeric vector specifying which principal components to include in the plot. Default is 1:5.

`assay_name`
: Name of the assay on which to perform computations. Default is "logcounts".

`max_cells_ref`
: Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

Details

This function first computes distance data using the `calculateCellDistances` function, which calculates pairwise distances between cells within the reference data and between query cells and reference cells in the PCA space. Bhattacharyya coefficients and Hellinger distances are calculated to quantify the similarity of density distributions between query cells and reference data for each cell type. Bhattacharyya coefficient measures the similarity of two probability distributions, while Hellinger distance measures the distance between two probability distributions.

Bhattacharyya coefficients range between 0 and 1. A value closer to 1 indicates higher similarity between distributions, while a value closer to 0 indicates lower similarity

Hellinger distances range between 0 and 1. A value closer to 0 indicates higher similarity between distributions, while a value closer to 1 indicates lower similarity.

Value

A list containing distance data for each cell type. Each entry in the list contains:

ref_distances A vector of all pairwise distances within the reference subset for the cell type.

query_to_ref_distances A matrix of distances from each query cell to all reference cells for the cell type.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

Examples

```
# Load data
data("reference_data")
data("query_data")

# Plot the PC data
distance_data <- calculateCellDistances(query_data = query_data,
                                        reference_data = reference_data,
                                        query_cell_type_col = "SingleR_annotation",
                                        ref_cell_type_col = "expert_annotation",
                                        pc_subset = 1:10)

# Identify outliers for CD4
cd4_anomalies <- detectAnomaly(reference_data = reference_data,
```

```
                                query_data = query_data,
                                query_cell_type_col = "SingleR_annotation",
                                ref_cell_type_col = "expert_annotation",
                                pc_subset = 1:10,
                                n_tree = 500,
                                anomaly_threshold = 0.5)
cd4_top6_anomalies <- names(sort(cd4_anomalies$CD4$query_anomaly_scores, decreasing = TRUE)[1:6])

# Get overlap measures
overlap_measures <- calculateCellDistancesSimilarity(query_data = query_data,
                                                     reference_data = reference_data,
                                                cell_names_query = cd4_top6_anomalies,
                                             query_cell_type_col = "SingleR_annotation",
                                                ref_cell_type_col = "expert_annotation",
                                                     pc_subset = 1:10)
overlap_measures
```

`calculateCramerPValue` *Calculate Cramer Test P-Values for Two-Sample Comparison of Multivariate ECDFs*

Description

This function performs the Cramer test for comparing multivariate empirical cumulative distribution functions (ECDFs) between two samples.

Usage

```
calculateCramerPValue(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:5,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000
)
```

Arguments

`query_data`	A `SingleCellExperiment` object containing numeric expression matrix for the query cells.
`reference_data`	A `SingleCellExperiment` object containing numeric expression matrix for the reference cells.
`query_cell_type_col`	The column name in the `colData` of `query_data` that identifies the cell types.

`ref_cell_type_col`
: The column name in the `colData` of `reference_data` that identifies the cell types.

`cell_types`
: A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.

`pc_subset`
: A numeric vector specifying which principal components to include in the plot. Default is PC1 to PC5.

`assay_name`
: Name of the assay on which to perform computations. Default is "logcounts".

`max_cells_query`
: Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.

`max_cells_ref`
: Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

Details

The function performs the following steps:

1. Projects the data into the PCA space.
2. Subsets the data to the specified cell types and principal components.
3. Performs the Cramer test for each cell type using the `cramer.test` function in the `cramer` package.

Value

A named vector of p-values from the Cramer test for each cell type.

References

Baringhaus, L., & Franz, C. (2004). "On a new multivariate two-sample test". Journal of Multivariate Analysis, 88(1), 190-206.

Examples

```
# Load data
data("reference_data")
data("query_data")

# Plot the PC data (with query data)
cramer_test <- calculateCramerPValue(reference_data = reference_data,
                                     query_data = query_data,
                                     ref_cell_type_col = "expert_annotation",
                                     query_cell_type_col = "SingleR_annotation",
                                cell_types = c("CD4", "CD8", "B_and_plasma", "Myeloid"),
                                     pc_subset = 1:5)
cramer_test
```

`calculateDiscriminantSpace` *Project Query Data onto a Unified Discriminant Space of Reference Data*

Description

This function projects query single-cell RNA-seq data onto a unified discriminant space defined by reference data. The reference data is used to identify important variables across all cell types and compute discriminant vectors, which are then used to project both reference and query data. Similarity between the query and reference projections can be assessed using cosine similarity and Mahalanobis distance.

The S3 plot method visualizes the projected reference and query data on the unified discriminant space.

Usage

```
calculateDiscriminantSpace(
  reference_data,
  query_data = NULL,
  ref_cell_type_col,
  query_cell_type_col = NULL,
  cell_types = NULL,
  n_tree = 500,
  n_top = 20,
  eigen_threshold = 0.1,
  calculate_metrics = FALSE,
  alpha = 0.01,
  assay_name = "logcounts",
  max_cells_ref = NULL,
  max_cells_query = NULL
)

## S3 method for class 'calculateDiscriminantSpaceObject'
plot(
  x,
  cell_types = NULL,
  dv_subset = NULL,
  lower_facet = c("scatter", "contour", "ellipse", "blank"),
  diagonal_facet = c("ridge", "density", "boxplot", "blank"),
  upper_facet = c("blank", "scatter", "contour", "ellipse"),
  max_cells_ref = NULL,
  max_cells_query = NULL,
  ...
)
```

Arguments

reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells. If NULL, only the projected reference data is returned. Default is NULL.
ref_cell_type_col	The column name in reference_data indicating cell type labels.
query_cell_type_col	The column name in query_data indicating cell type labels.
cell_types	A character vector specifying the cell types to plot. If NULL (default), all cell types will be plotted.
n_tree	An integer specifying the number of trees for the random forest used in variable importance calculation.
n_top	An integer specifying the number of top variables to select based on importance scores from each pairwise comparison.
eigen_threshold	A numeric value specifying the threshold for retaining eigenvalues in discriminant analysis.
calculate_metrics	Parameter to determine if cosine similarity and Mahalanobis distance metrics should be computed. Default is FALSE.
alpha	A numeric value specifying the significance level for Mahalanobis distance cutoff.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells_ref	Maximum number of reference cells to include in the plot. If NULL, all available reference cells are plotted. Default is NULL.
max_cells_query	Maximum number of query cells to include in the plot. If NULL, all available query cells are plotted. Default is NULL.
x	An object of class calculateDiscriminantSpaceObject containing the projected data on the discriminant space.
dv_subset	A numeric vector specifying which discriminant vectors to include in the plot. Default is the number of cell types minus 1.
lower_facet	Type of plot to use for the lower panels. Either "scatter" (default), "contour", "ellipse", or "blank".
diagonal_facet	Type of plot to use for the diagonal panels. Either "ridge" (default), "density", "boxplot" or "blank".
upper_facet	Type of plot to use for the upper panels. Either "blank" (default), "scatter", "contour", or "ellipse".
...	Additional arguments to be passed to the plotting functions.

Details

The function performs the following steps:

- Identifies the top important variables to distinguish cell types from the reference data by taking the union of important variables from pairwise comparisons.
- Computes the Ledoit-Wolf shrinkage estimate of the covariance matrix for each cell type using these important genes.
- Constructs within-class and between-class scatter matrices.
- Solves the generalized eigenvalue problem to obtain discriminant vectors.
- Projects both reference and query data onto the unified discriminant space.
- Assesses similarity of the query data projection to the reference data using cosine similarity and Mahalanobis distance.

The S3 plot method generates a pairs plot visualization of discriminant vectors, similar to PCA plot visualization. Each panel shows the relationship between two discriminant vectors with customizable display options for lower, diagonal, and upper panels. The visualization allows for comprehensive examination of the discriminant space structure and cell type separability.

Value

A list with the following components:

`discriminant_eigenvalues`
: Eigenvalues from the discriminant analysis.

`discriminant_eigenvectors`
: Eigenvectors from the discriminant analysis.

`ref_proj`
: Reference data projected onto the discriminant space.

`query_proj`
: Query data projected onto the discriminant space (if query_data is provided).

`query_mahalanobis_dist`
: Mahalanobis distances of query projections (if calculate_metrics is TRUE).

`mahalanobis_crit`
: Cutoff value for Mahalanobis distance significance (if calculate_metrics is TRUE).

`query_cosine_similarity`
: Cosine similarity scores of query projections (if calculate_metrics is TRUE).

The S3 plot method returns a `GGally::ggpairs` object representing the visualization of the projected discriminant space.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

- Fisher, R. A. (1936). "The Use of Multiple Measurements in Taxonomic Problems". *Annals of Eugenics*. 7 (2): 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer. Chapter 4: Linear Methods for Classification.
- Ledoit, O., & Wolf, M. (2004). "A well-conditioned estimator for large-dimensional covariance matrices". *Journal of Multivariate Analysis*. 88 (2): 365–411. doi:10.1016/S0047-259X(03)00096-4.
- De Maesschalck, R., Jouan-Rimbaud, D., & Massart, D. L. (2000). "The Mahalanobis distance". *Chemometrics and Intelligent Laboratory Systems*. 50 (1): 1–18. doi:10.1016/S0169-7439(99)00047-7.
- Breiman, L. (2001). "Random Forests". *Machine Learning*. 45 (1): 5–32. doi:10.1023/A:1010933404324.

See Also

plot.calculateDiscriminantSpaceObject

calculateDiscriminantSpace

Examples

```
# Load data
data("reference_data")
data("query_data")

# Compute discriminant space using unified model across all cell types
disc_output <- calculateDiscriminantSpace(reference_data = reference_data,
                                          query_data = query_data,
                                          query_cell_type_col = "SingleR_annotation",
                                          ref_cell_type_col = "expert_annotation",
                                          n_tree = 500,
                                          n_top = 50,
                                          eigen_threshold  = 1e-1,
                                          calculate_metrics = FALSE,
                                          alpha = 0.01)

# Generate scatter and boxplot
plot(disc_output, plot_type = "scatterplot")
plot(disc_output, cell_types = c("CD4", "CD8"), plot_type = "boxplot")

# Check comparison
table(Expert_Annotation = query_data$expert_annotation,
      SingleR = query_data$SingleR_annotation)
```

`calculateGeneShifts` *Calculate Top Loading Gene Expression Shifts*

Description

This function identifies genes with the highest loadings for specified principal components and performs statistical tests to detect distributional differences between query and reference data. It also calculates the proportion of variance explained by each principal component within specific cell types. Optionally, it can detect anomalous cells using isolation forests.

This function creates visualizations showing expression distributions for top loading genes that exhibit distributional differences between query and reference datasets. Can display results as elegant complex heatmaps, information-rich summary boxplots, or pseudo-bulk fold change barplots. Optionally displays anomaly status when available.

Usage

```
calculateGeneShifts(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:5,
  n_top_loadings = 50,
  genes_to_analyze = NULL,
  p_value_threshold = 0.05,
  adjust_method = "fdr",
  assay_name = "logcounts",
  detect_anomalies = FALSE,
  anomaly_comparison = FALSE,
  anomaly_threshold = 0.6,
  n_tree = 500,
  max_cells_query = 5000,
  max_cells_ref = 5000
)

## S3 method for class 'calculateGeneShiftsObject'
plot(
  x,
  cell_type,
  pc_subset = 1:3,
  plot_type = c("heatmap", "barplot", "boxplot"),
  plot_by = c("p_adjusted", "top_loading"),
  n_genes = 10,
  significance_threshold = 0.05,
  show_anomalies = FALSE,
```

```
  pseudo_bulk = FALSE,
  cluster_cols = FALSE,
  draw_plot = TRUE,
  show_all_query = TRUE,
  max_cells_ref = NULL,
  max_cells_query = NULL,
  ...
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
cell_types	A character vector specifying the cell types to analyze. If NULL, all common cell types are used.
pc_subset	A numeric vector specifying which principal components to plot. Default is 1:3.
n_top_loadings	Number of top loading genes to analyze per PC. Default is 50.
genes_to_analyze	A character vector specifying genes to analyze. If NULL (default), genes are selected based on top loadings from specified principal components (see n_top_loadings). Default is NULL.
p_value_threshold	P-value threshold for statistical significance. Default is 0.05.
adjust_method	Method for multiple testing correction. Default is "fdr".
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
detect_anomalies	Logical indicating whether to perform anomaly detection using isolation forests. Default is FALSE.
anomaly_comparison	Logical indicating whether to perform statistical comparisons between non-anomalous reference cells and anomalous query cells instead of all-vs-all comparisons. When TRUE, only non-anomalous reference cells are compared against only anomalous query cells for each cell type. Requires detect_anomalies = TRUE. Default is FALSE.
anomaly_threshold	A numeric value specifying the threshold for identifying anomalies when detect_anomalies is TRUE. Default is 0.6.
n_tree	An integer specifying the number of trees for the isolation forest when detect_anomalies is TRUE. Default is 500.

`max_cells_query`
: Maximum number of query cells to include in the plot. If NULL, all available query cells are plotted. Default is NULL.

`max_cells_ref`
: Maximum number of reference cells to include in the plot. If NULL, all available reference cells are plotted. Default is NULL.

`x`
: An object of class `calculateGeneShiftsObject`.

`cell_type`
: A character string specifying the cell type to plot (must be exactly one).

`plot_type`
: A character string specifying visualization type. Either "heatmap", "barplot", or "boxplot". Default is "heatmap".

`plot_by`
: A character string specifying gene selection method when ‘n_genes‘ is not NULL. Either "top_loading" or "p_adjusted". Default is "p_adjusted".

`n_genes`
: Number of top genes to show per PC. Can be NULL if ‘significance_threshold‘ is set. Default is 10.

`significance_threshold`
: If not NULL, a numeric value between 0 and 1. Used for gene selection or annotation. Default is 0.05.

`show_anomalies`
: Logical indicating whether to display anomaly status annotations. Default is FALSE. Requires anomaly results to be present in the object.

`pseudo_bulk`
: Logical indicating whether to create pseudo-bulk profiles instead of showing individual cells. When TRUE, expression values are averaged within groups (dataset and optionally anomaly status). Not compatible with boxplot visualization. Required for barplot visualization. Default is FALSE.

`cluster_cols`
: Logical indicating whether to cluster columns in the heatmap when ‘pseudo_bulk = TRUE‘. When TRUE, columns (pseudo-bulk profiles) will be hierarchically clustered. When FALSE, columns maintain their original ordering (Query groups followed by Reference groups). Only applicable when ‘pseudo_bulk = TRUE‘ and ‘plot_type = "heatmap"‘. Default is FALSE.

`draw_plot`
: Logical indicating whether to draw the plot immediately (TRUE) or return the undrawn plot object (FALSE). For heatmaps, FALSE returns a ComplexHeatmap object that can be further customized before drawing. Default is TRUE.

`show_all_query`
: Logical indicating whether to show the yellow bar for all query vs reference comparison. Default is TRUE. When FALSE, only green and red bars are shown.

`...`
: Additional arguments passed to `draw` or not used for other plot types.

Details

This function extracts the top loading genes for each specified principal component from the reference PCA space and performs distributional comparisons between query and reference data. For each gene, it performs statistical tests to identify genes that may be causing PC-specific alignment issues between datasets. A key feature is the calculation of cell-type-specific variance explained by global PCs, providing a more nuanced view of how major biological axes affect individual populations. When anomaly detection is enabled, isolation forests are used to identify anomalous cells based on their PCA projections.

When anomaly_comparison = TRUE, the statistical analysis focuses specifically on comparing non-anomalous reference cells against anomalous query cells. This can help identify genes that are differentially expressed between "normal" reference cells and potentially problematic query cells, providing insights into what makes certain query cells anomalous.

This function visualizes the results from calculateGeneShifts. The "heatmap" option displays a hierarchically clustered set of genes. The "boxplot" option creates a two-panel plot using ‘ggplot2‘: the left panel shows horizontal expression boxplots for up to 5 PCs, while the right panel displays their corresponding PC loadings and adjusted p-values. The "barplot" option creates horizontal barplots showing log2 fold changes between pseudo-bulk expression profiles (query vs reference), with genes ordered identically to the heatmap clustering. Bars show comparisons for query non-anomaly (green), optionally all query cells (yellow), and query anomaly cells (red) versus reference. When anomaly detection results are available and show_anomalies is TRUE, additional annotation bars or visual cues highlight anomalous cells.

Value

A list containing:

- PC results: Named elements for each PC (e.g., "PC1", "PC2") containing data frames with gene-level analysis results.
- expression_data: Matrix of expression values for all analyzed genes (genes × cells).
- cell_metadata: Data frame with columns: cell_id, dataset, cell_type, original_index, and optionally anomaly_status.
- gene_metadata: Data frame with columns: gene, pc, loading for all analyzed genes.
- percent_var: Named numeric vector of global percent variance explained for each analyzed PC.
- cell_type_variance: A data frame detailing the percent of variance a global PC explains within specific cell types for both query and reference datasets.
- anomaly_results: If detect_anomalies is TRUE, contains the full output from detectAnomaly.

The ‘cell_type_variance‘ data frame contains columns: pc, cell_type, dataset, percent_variance. When anomaly detection is enabled, ‘cell_metadata‘ includes an additional ‘anomaly_status‘ column.

A plot object. For heatmaps when draw_plot = FALSE, returns a ComplexHeatmap object. For boxplots and barplots, returns a ggplot2 object.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

See Also

plot.calculateGeneShiftsObject, detectAnomaly

calculateGeneShifts

`calculateGraphIntegration` *Calculate Graph Community Integration Diagnostics*

Description

This function performs graph-based community detection to identify annotation inconsistencies by detecting query-only communities, true cross-cell-type mixing patterns, and local annotation inconsistencies based on immediate neighborhood analysis.

The S3 plot method generates visualizations of annotation consistency diagnostics, including query-only communities, cross-cell-type mixing, and local annotation inconsistencies.

Usage

```
calculateGraphIntegration(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:10,
  k_neighbors = 30,
  assay_name = "logcounts",
  resolution = 0.1,
  min_cells_per_community = 10,
  min_cells_per_celltype = 20,
  high_query_prop_threshold = 0.9,
  cross_type_threshold = 0.15,
  local_consistency_threshold = 0.6,
  local_confidence_threshold = 0.2,
  max_cells_query = 5000,
  max_cells_ref = 5000
)

## S3 method for class 'calculateGraphIntegrationObject'
plot(
  x,
  plot_type = c("community_network", "cell_network", "community_data", "summary",
    "local_issues", "annotation_issues"),
  color_by = c("cell_type", "community_type"),
  max_nodes = 2000,
  point_size = 0.8,
  exclude_reference_only = FALSE,
  ...
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_cell_type_col	A character string specifying the column name in the query dataset containing cell type annotations.
ref_cell_type_col	A character string specifying the column name in the reference dataset containing cell type annotations.
cell_types	A character vector specifying the cell types to include in the analysis. If NULL, all cell types are included.
pc_subset	A vector specifying the subset of principal components to use in the analysis. Default is 1:10.
k_neighbors	An integer specifying the number of nearest neighbors for graph construction. Default is 30.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
resolution	Resolution parameter for Leiden clustering. Default is 0.15 for fewer, larger communities.
min_cells_per_community	Minimum number of cells required for a community to be analyzed. Default is 10.
min_cells_per_celltype	Minimum number of cells required per cell type for inclusion. Default is 20.
high_query_prop_threshold	Minimum proportion of query cells to consider a community "query-only". Default is 0.9.
cross_type_threshold	Minimum proportion needed to flag cross-cell-type mixing. Default is 0.1.
local_consistency_threshold	Minimum proportion of reference neighbors that should support a query cell's annotation. Default is 0.6.
local_confidence_threshold	Minimum confidence difference needed to suggest re-annotation. Default is 0.2.
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.
x	An object of class calculateGraphIntegrationObject containing the diagnostic results.
plot_type	Character string specifying visualization type. Options: "community_network" (default), "cell_network", "community_data", "summary", "local_issues", or "annotation_issues".

`color_by` Character string specifying the variable to use for coloring points/elements if 'plot_type' is "community_network" or "cell_network". Default is "cell_type".

`max_nodes` Maximum number of nodes to display for performance. Default is 2000.

`point_size` Point size for graph nodes. Default is 0.8.

`exclude_reference_only` Logical indicating whether to exclude reference-only communities/cells from visualization. Default is FALSE.

`...` Additional arguments passed to ggplot2 functions.

Details

The function performs three types of analysis: (1) Communities containing only query cells, (2) Communities where query cells are mixed with reference cells of different cell types WITHOUT any reference cells of the same type, and (3) Local analysis of each query cell's immediate neighbors to detect annotation inconsistencies even within mixed communities.

The S3 plot method creates optimized visualizations showing different types of annotation issues including community-level and local neighborhood-level inconsistencies.

Value

A list containing:

`high_query_prop_analysis` Analysis of communities with only query cells

`cross_type_mixing` Analysis of communities with true query-reference cross-cell-type mixing

`local_annotation_inconsistencies` Local neighborhood-based annotation inconsistencies

`local_inconsistency_summary` Summary of local inconsistencies by cell type

`community_composition` Detailed composition of each community

`annotation_consistency` Summary of annotation consistency issues

`overall_metrics` Overall diagnostic metrics

`graph_info` Graph structure information for plotting

`parameters` Analysis parameters used

The list is assigned the class `"calculateGraphIntegration"`.

A `ggplot` object showing integration diagnostics.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

See Also

calculateGraphIntegration

Examples

```
# Load data
data("reference_data")
data("query_data")

# Remove a cell type (Myeloid)
library(scater)
library(SingleR)
reference_data <- reference_data[, reference_data$expert_annotation != "Myeloid"]
reference_data <- runPCA(reference_data, ncomponents = 50)
SingleR_annotation <- SingleR(query_data, reference_data,
                              labels = reference_data$expert_annotation)
query_data$SingleR_annotation <- SingleR_annotation$labels

# Check annotation data
table(Expert = query_data$expert_annotation, SingleR = query_data$SingleR_annotation)

# Run comprehensive annotation consistency diagnostics
graph_diagnostics <- calculateGraphIntegration(
    query_data = query_data,
    reference_data = reference_data,
    query_cell_type_col = "SingleR_annotation",
    ref_cell_type_col = "expert_annotation",
   pc_subset = 1:10,
   k_neighbors = 30,
    resolution = 0.1,
    high_query_prop_threshold = 0.9,
    cross_type_threshold = 0.15,
    local_consistency_threshold = 0.6,
    local_confidence_threshold = 0.2
)

# Look at main output
graph_diagnostics$overall_metrics

# Network graph showing all issue types (color by cell type)
plot(graph_diagnostics, plot_type = "community_network", color_by = "cell_type")

# Network graph showing all issue types
plot(graph_diagnostics, plot_type = "cell_network",
     max_nodes = 2000, color_by = "community_type")

# Network graph showing all issue types
plot(graph_diagnostics, plot_type = "community_data")

# Summary bar chart of all issues by cell type
plot(graph_diagnostics, plot_type = "summary")
```

```
# Focus on local annotation inconsistencies
plot(graph_diagnostics, plot_type = "local_issues")

# Overall annotation issues overview
plot(graph_diagnostics, plot_type = "annotation_issues")
```

`calculateHotellingPValue` *Perform Hotelling's T-squared Test on PCA Scores for Single-cell RNA-seq Data*

Description

Computes Hotelling's T-squared test statistic and p-values for each specified cell type based on PCA-projected data from query and reference datasets.

Usage

```
calculateHotellingPValue(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:5,
  n_permutation = 500,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000
)
```

Arguments

`query_data`	A `SingleCellExperiment` object containing numeric expression matrix for the query cells.
`reference_data`	A `SingleCellExperiment` object containing numeric expression matrix for the reference cells.
`query_cell_type_col`	character. The column name in the `colData` of `query_data` that identifies the cell types.
`ref_cell_type_col`	character. The column name in the `colData` of `reference_data` that identifies the cell types.
`cell_types`	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.

pc_subset	A numeric vector specifying which principal components to include in the plot. Default is PC1 to PC5.
n_permutation	Number of permutations to perform for p-value calculation. Default is 500.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

Details

This function calculates Hotelling's T-squared statistic for comparing multivariate means between reference and query datasets, projected onto a subset of principal components (PCs). It performs a permutation test to obtain p-values for each cell type specified.

Value

A named numeric vector of p-values from Hotelling's T-squared test for each cell type.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

Hotelling, H. (1931). "The generalization of Student's ratio". *Annals of Mathematical Statistics*. 2 (3): 360–378. doi:10.1214/aoms/1177732979.

Examples

```
# Load data
data("reference_data")
data("query_data")

# Get the p-values
p_values <- calculateHotellingPValue(query_data = query_data,
                                     reference_data = reference_data,
                                     query_cell_type_col = "SingleR_annotation",
                                     ref_cell_type_col = "expert_annotation",
                                     pc_subset = 1:10)
round(p_values, 5)
```

`calculateHVGOverlap` *Calculate the Overlap Coefficient for Highly Variable Genes*

Description

Calculates the overlap coefficient between the sets of highly variable genes from a reference dataset and a query dataset.

Usage

```
calculateHVGOverlap(reference_genes, query_genes)
```

Arguments

`reference_genes`	A character vector of highly variable genes from the reference dataset.
`query_genes`	A character vector of highly variable genes from the query dataset.

Details

The overlap coefficient measures the similarity between two gene sets, indicating how well-aligned reference and query datasets are in terms of their highly variable genes. This metric is useful in single-cell genomics to understand the correspondence between different datasets.

The coefficient is calculated using the formula:

$$Coefficient(X, Y) = \frac{|X \cap Y|}{min(|X|, |Y|)}$$

where X and Y are the sets of highly variable genes from the reference and query datasets, respectively, $|X \cap Y|$ is the number of genes common to both X and Y, and $min(|X|, |Y|)$ is the size of the smaller set among X and Y.

Value

Overlap coefficient, a value between 0 and 1, where 0 indicates no overlap and 1 indicates complete overlap of highly variable genes between datasets.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

Luecken et al. Benchmarking atlas-level data integration in single-cell genomics. Nature Methods, 19:41-50, 2022.

Examples

```
# Load data
data("reference_data")
data("query_data")

# Selecting highly variable genes
ref_var <- scran::getTopHVGs(reference_data, n = 500)
query_var <- scran::getTopHVGs(query_data, n = 500)
overlap_coefficient <- calculateHVGOverlap(reference_genes = ref_var,
                                           query_genes = query_var)
overlap_coefficient
```

`calculateMMDPValue` *Calculate Maximum Mean Discrepancy P-Values for Two-Sample Comparison*

Description

This function performs the Maximum Mean Discrepancy (MMD) test for comparing distributions between two samples in PCA space using a custom implementation with permutation testing for better sensitivity.

Usage

```
calculateMMDPValue(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:5,
  n_permutation = 100,
  kernel_type = "gaussian",
  sigma = NULL,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000
)
```

Arguments

`query_data`	A `SingleCellExperiment` object containing numeric expression matrix for the query cells.
`reference_data`	A `SingleCellExperiment` object containing numeric expression matrix for the reference cells.

query_cell_type_col
: The column name in the colData of query_data that identifies the cell types.

ref_cell_type_col
: The column name in the colData of reference_data that identifies the cell types.

cell_types
: A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.

pc_subset
: A numeric vector specifying which principal components to include in the plot. Default is PC1 to PC5.

n_permutation
: Number of permutations for p-value calculation. Default is 100.

kernel_type
: Type of kernel to use. Options are "gaussian" (default) or "linear".

sigma
: Bandwidth parameter for Gaussian kernel. If NULL, uses median heuristic.

assay_name
: Name of the assay on which to perform computations. Default is "logcounts".

max_cells_query
: Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.

max_cells_ref
: Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

Details

The function performs the following steps:

1. Projects the data into the PCA space.
2. Subsets the data to the specified cell types and principal components.
3. Performs a custom MMD test with permutation-based p-values for each cell type.

Value

A named vector of p-values from the MMD test for each cell type.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., & Smola, A. (2012). "A kernel two-sample test". Journal of Machine Learning Research, 13(1), 723-773.

Examples

```
# Load data
data("reference_data")
data("query_data")

# Calculate MMD p-values (with query data)
```

```
mmd_test <- calculateMMDPValue(reference_data = reference_data,
                               query_data = query_data,
                               ref_cell_type_col = "expert_annotation",
                               query_cell_type_col = "SingleR_annotation",
                               cell_types = c("CD4", "CD8"),
                               pc_subset = 1:5,
                               n_permutation = 30)
mmd_test
```

`calculateSIRSpace` *Calculate Sliced Inverse Regression (SIR) Space for Different Cell Types*

Description

This function calculates the SIR space projections for different cell types in the query and reference datasets.

This function plots the Sliced Inverse Regression (SIR) components for different cell types in query and reference datasets.

Usage

```
calculateSIRSpace(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  multiple_cond_means = TRUE,
  cumulative_variance_threshold = 0.7,
  n_neighbor = 1,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000
)

## S3 method for class 'calculateSIRSpaceObject'
plot(
  x,
  plot_type = c("scores", "loadings"),
  cell_types = NULL,
  sir_subset = 1:5,
  lower_facet = c("scatter", "contour", "ellipse", "blank"),
  diagonal_facet = c("ridge", "density", "boxplot", "blank"),
  upper_facet = c("blank", "scatter", "contour", "ellipse"),
  n_top = 10,
```

```
  max_cells_ref = NULL,
  max_cells_query = NULL,
  ...
)
```

Arguments

query_data	A SingleCellExperiment object containing the numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing the numeric expression matrix for the reference cells.
query_cell_type_col	A character string specifying the column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	A character string specifying the column name in the colData of reference_data that identifies the cell types.
cell_types	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included. Only used when plot_type = "scores".
multiple_cond_means	Logical. Whether to compute conditional means for multiple conditions in the reference dataset. Default is TRUE.
cumulative_variance_threshold	A numeric value specifying the cumulative variance threshold for selecting principal components. Default is 0.7.
n_neighbor	A numeric value specifying the number of neighbors for computing the SIR space. Default is 1.
assay_name	A character string specifying the name of the assay on which to perform computations. Default is "logcounts".
max_cells_query	Maximum number of query cells to include in the plot. If NULL, all available query cells are plotted. Default is NULL. Only used when plot_type = "scores".
max_cells_ref	Maximum number of reference cells to include in the plot. If NULL, all available reference cells are plotted. Default is NULL. Only used when plot_type = "scores".
x	An object of class calculateSIRSpaceObject containing SIR projections.
plot_type	A character string specifying the type of plot. Either "scores" (default) for SIR projections or "loadings" for variable loadings.
sir_subset	A numeric vector specifying which SIR components to include in the plot. Default is 1:5.
lower_facet	Type of plot to use for the lower panels. Either "scatter" (default), "contour", "ellipse", or "blank". Only used when plot_type = "scores".
diagonal_facet	Type of plot to use for the diagonal panels. Either "ridge" (default), "density", "boxplot" or "blank". Only used when plot_type = "scores".

upper_facet	Type of plot to use for the upper panels. Either "blank" (default), "scatter", "contour", or "ellipse". Only used when plot_type = "scores".
n_top	A numeric value specifying the number of n_top variables (by absolute loading value) to display. Default is 10 Only used when plot_type = "loadings".
...	Additional arguments passed to the plotting function.

Details

The function projects the query dataset onto the SIR space of the reference dataset based on shared cell types. It computes conditional means for the reference dataset, extracts the SVD components, and performs the projection of both the query and reference data. It uses the ‘projectSIR‘ function to perform the actual projection and allows the user to specify particular cell types for analysis.

This function visualizes the SIR projections for specified cell types, providing a pairs plot of the SIR components. It offers various visualization options for different facets of the plot including scatter plots, contours, ellipses, and density plots. When plot_type is "loadings", it creates horizontal bar plots showing the n_top contributing variables for each SIR component.

Value

A list containing the SIR projections, rotation matrix, and percentage of variance explained for the given cell types.

A ggmatrix object representing a pairs plot of specified SIR components for the given cell types and datasets when plot_type = "scores", or a ggplot object showing loadings when plot_type = "loadings".

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

See Also

plot.calculateSIRSpaceObject

calculateSIRSpace

Examples

```
# Load data
data("reference_data")
data("query_data")

# Compute important variables for all pairwise cell comparisons
sir_output <- calculateSIRSpace(reference_data = reference_data,
                                query_data = query_data,
                                query_cell_type_col = "expert_annotation",
                                ref_cell_type_col = "expert_annotation",
                                multiple_cond_means = TRUE,
                                cumulative_variance_threshold = 0.9,
                                n_neighbor = 1)
```

```
# Generate plots SIR projections
plot(sir_output,
     sir_subset = 1:5,
     cell_types = c("CD4", "CD8", "B_and_plasma", "Myeloid"),
     lower_facet = "scatter",
     diagonal_facet = "boxplot",
     upper_facet = "blank")

# Plot top loadings
plot(sir_output,
     sir_subset = 1:5,
     plot_type = "loadings",
     n_top = 10)
```

`calculateVarImpOverlap` *Compare Gene Importance Across Datasets Using Random Forest*

Description

This function identifies and compares the most important genes for differentiating cell types between a query dataset and a reference dataset using Random Forest.

Usage

```
calculateVarImpOverlap(
  reference_data,
  query_data = NULL,
  ref_cell_type_col,
  query_cell_type_col = NULL,
  cell_types = NULL,
  n_tree = 500,
  n_top = 50,
  assay_name = "logcounts",
  max_cells_ref = 5000,
  max_cells_query = 5000
)
```

Arguments

`reference_data`	A `SingleCellExperiment` object containing numeric expression matrix for the reference cells.
`query_data`	A `SingleCellExperiment` object containing numeric expression matrix for the query cells. If NULL, then the variable importance scores are only computed for the reference data. Default is NULL.

`ref_cell_type_col`
: A character string specifying the column name in the reference dataset containing cell type annotations.

`query_cell_type_col`
: A character string specifying the column name in the query dataset containing cell type annotations.

`cell_types`
: A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.

`n_tree`
: An integer specifying the number of trees to grow in the Random Forest. Default is 500.

`n_top`
: An integer specifying the number of top genes to consider when comparing variable importance scores. Default is 50.

`assay_name`
: Name of the assay on which to perform computations. Defaults to `"logcounts"`.

`max_cells_ref`
: Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

`max_cells_query`
: Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.

Details

This function uses the Random Forest algorithm to calculate the importance of genes in differentiating between cell types within both a reference dataset and a query dataset. The function then compares the top genes identified in both datasets to determine the overlap in their importance scores.

Value

A list containing three elements:

`var_imp_ref`
: A list of data frames containing variable importance scores for each combination of cell types in the reference dataset.

`var_imp_query`
: A list of data frames containing variable importance scores for each combination of cell types in the query dataset.

`var_imp_comparison`
: A named vector indicating the proportion of top genes that overlap between the reference and query datasets for each combination of cell types.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

Breiman, L. (2001). "Random forests". *Machine Learning*, 45(1), 5-32. doi:10.1023/A:1010933404324.

Examples

```
# Load data
data("reference_data")
data("query_data")

# Compute important variables for all pairwise cell comparisons
rf_output <- calculateVarImpOverlap(reference_data = reference_data,
                                    query_data = query_data,
                                    query_cell_type_col = "SingleR_annotation",
                                    ref_cell_type_col = "expert_annotation",
                                    n_tree = 500,
                                    n_top = 50)

# Comparison table
rf_output$var_imp_comparison
```

---

comparePCA *Compare Principal Components Analysis (PCA) Results*

Description

This function compares the principal components (PCs) obtained from separate PCA on reference and query datasets for a single cell type using either cosine similarity or correlation.

The S3 plot method generates a heatmap to visualize the similarities between principal components from the output of the comparePCA function.

Usage

```
comparePCA(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  pc_subset = 1:5,
  n_top_vars = 50,
  metric = c("cosine", "correlation"),
  correlation_method = c("spearman", "pearson"),
  n_permutations = 0
)

## S3 method for class 'comparePCAObject'
plot(
  x,
  show_values = TRUE,
  show_significance = TRUE,
  significance_threshold = 0.05,
```

```
  color_limits = NULL,
  ...
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
pc_subset	A numeric vector specifying the subset of principal components (PCs) to compare. Default is the first five PCs.
n_top_vars	An integer indicating the number of top loading variables to consider for each PC. Default is 50.
metric	The similarity metric to use. It can be either "cosine" or "correlation". Default is "cosine".
correlation_method	The correlation method to use if metric is "correlation". It can be "spearman" or "pearson". Default is "spearman".
n_permutations	Number of permutations for statistical significance testing. If 0, no permutation test is performed. Default is 0.
x	A comparePCAObject output from the comparePCA function.
show_values	Logical, whether to display similarity values on the heatmap. Default is TRUE.
show_significance	Logical, whether to display significance indicators (requires permutation test). Default is TRUE.
significance_threshold	Numeric, p-value threshold for significance. Default is 0.05.
color_limits	Numeric vector of length 2 specifying color scale limits. If NULL, uses data range.
...	Additional arguments passed to the plotting function.

Details

This function compares the PCA results between the reference and query datasets by computing cosine similarities or correlations between the loadings of top variables for each pair of principal components. It first extracts the PCA rotation matrices from both datasets and identifies the top variables with highest loadings for each PC. Then, it computes the cosine similarities or correlations between the loadings of top variables for each pair of PCs using vectorized operations for improved performance. The resulting matrix contains the similarity values, where rows represent reference PCs and columns represent query PCs.

The S3 plot method creates an enhanced heatmap visualization with options to display statistical significance and similarity values. The heatmap uses a blue-white-red color gradient for similarity values, and optionally overlays significance indicators.

Value

A list containing:

`similarity_matrix`	A matrix comparing the principal components of the reference and query datasets.
`top_variables`	A list containing the top loading variables for each PC pair comparison.
`p_values`	A matrix of permutation p-values (if n_permutations > 0).
`metric`	The similarity metric used.
`n_top_vars`	Number of top variables used.

A `ggplot` object representing the heatmap of similarities.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

See Also

`plot.comparePCAObject`

`comparePCA`

Examples

```
# Load libraries
library(scran)
library(scater)

# Load data
data("reference_data")
data("query_data")

# Extract CD4 cells
ref_data_subset <- reference_data[, which(reference_data$expert_annotation == "CD4")]
query_data_subset <- query_data[, which(query_data$expert_annotation == "CD4")]

# Selecting highly variable genes (can be customized by the user)
ref_top_genes <- getTopHVGs(ref_data_subset, n = 500)
query_top_genes <- getTopHVGs(query_data_subset, n = 500)

# Intersect the gene symbols to obtain common genes
common_genes <- intersect(ref_top_genes, query_top_genes)
ref_data_subset <- ref_data_subset[common_genes,]
query_data_subset <- query_data_subset[common_genes,]

# Run PCA on datasets separately
```

```
ref_data_subset <- runPCA(ref_data_subset)
query_data_subset <- runPCA(query_data_subset)

# Call the PCA comparison function
similarity_mat <- comparePCA(query_data = query_data_subset,
                             reference_data = ref_data_subset,
                             query_cell_type_col = "expert_annotation",
                             ref_cell_type_col = "expert_annotation",
                             pc_subset = 1:5,
                             n_top_vars = 50,
                             metric = c("cosine", "correlation")[1],
                             correlation_method = c("spearman", "pearson")[1],
                             n_permutation = 100)

# Create the heatmap
plot(similarity_mat, show_significance = TRUE)
```

`detectAnomaly` *PCA Anomaly Scores via Isolation Forests with Visualization*

Description

This function detects anomalies in single-cell data by projecting the data onto a PCA space and using an isolation forest algorithm to identify anomalies.

This S3 plot method generates faceted scatter plots for specified principal component (PC) combinations within an anomaly detection object. It visualizes the relationship between specified PCs, highlights anomalies detected by the Isolation Forest algorithm, and provides a background gradient representing anomaly scores.

Usage

```
detectAnomaly(
  reference_data,
  query_data = NULL,
  ref_cell_type_col,
  query_cell_type_col = NULL,
  cell_types = NULL,
  pc_subset = 1:5,
  n_tree = 500,
  anomaly_threshold = 0.6,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000,
  ...
)

## S3 method for class 'detectAnomalyObject'
```

```
plot(
  x,
  cell_type = NULL,
  pc_subset = NULL,
  data_type = c("query", "reference"),
  n_tree = 500,
  upper_facet = c("blank", "contour", "ellipse"),
  diagonal_facet = c("density", "ridge", "boxplot", "blank"),
  max_cells_ref = NULL,
  max_cells_query = NULL,
  ...
)
```

Arguments

reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_data	An optional SingleCellExperiment object containing numeric expression matrix for the query cells. If NULL, then the isolation forest anomaly scores are computed for the reference data. Default is NULL.
ref_cell_type_col	A character string specifying the column name in the reference dataset containing cell type annotations.
query_cell_type_col	A character string specifying the column name in the query dataset containing cell type annotations.
cell_types	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.
pc_subset	A numeric vector specifying the indices of the PCs to be included in the plots. If NULL, all PCs in reference_mat_subset will be included.
n_tree	An integer specifying the number of trees for the isolation forest. Default is 500
anomaly_threshold	A numeric value specifying the threshold for identifying anomalies, Default is 0.6.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells_query	Maximum number of query cells to include in the plot. If NULL, all available query cells are plotted. Default is NULL.
max_cells_ref	Maximum number of reference cells to include in the plot. If NULL, all available reference cells are plotted. Default is NULL.
...	Additional arguments passed to the ‘isolation.forest‘ function.
x	A list object containing the anomaly detection results from the detectAnomaly function. Each element of the list should correspond to a cell type and contain reference_mat_subset, query_mat_subset, var_explained, and anomaly.

cell_type	A character string specifying the cell type for which the plots should be generated. This should be a name present in x. If NULL, the "Combined" cell type will be plotted. Default is NULL.
data_type	A character string specifying whether to plot the "query" data or the "reference" data. Default is "query".
upper_facet	Either "blank" (default), "contour", or "ellipse" for the upper facet plots.
diagonal_facet	Either "density" (default), "ridge", "boxplot" or "blank" for the diagonal plots.

Details

This function projects the query data onto the PCA space of the reference data. An isolation forest is then built on the reference data to identify anomalies in the query data based on their PCA projections. If no query dataset is provided by the user, the anomaly scores are computed on the reference data itself. Anomaly scores for the data with all combined cell types are also provided as part of the output.

The function extracts the specified PCs from the given anomaly detection object and generates scatter plots for each pair of PCs. It uses GGally to create a pairs plot where each facet represents a pair of PCs. The plot includes:

1. Lower facets: Scatter plots with a background gradient representing anomaly scores from green (low) to red (high) 2. Diagonal facets: Density, ridge, boxplot or blank visualizations showing the distribution of each PC, separated by anomaly status 3. Upper facets: Blank panels by default, or contour/ellipse plots separated by anomaly status if specified

Value

A list containing the following components for each cell type and the combined data:

anomaly_scores	Anomaly scores for each cell in the query data.
anomaly	Logical vector indicating whether each cell is classified as an anomaly.
reference_mat_subset	PCA projections of the reference data.
query_mat_subset	PCA projections of the query data (if provided).
var_explained	Proportion of variance explained by the retained principal components.

The S3 plot method returns a GGally::ggpairs object representing the PCA plots with anomalies highlighted.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

- Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413-422). IEEE.
- isotree: Isolation-Based Outlier Detection

See Also

plot.detectAnomalyObject

detectAnomaly

Examples

```
# Load data
data("reference_data")
data("query_data")

# Store PCA anomaly data
anomaly_output <- detectAnomaly(reference_data = reference_data,
                                query_data = query_data,
                                ref_cell_type_col = "expert_annotation",
                                query_cell_type_col = "SingleR_annotation",
                                pc_subset = 1:5,
                                n_tree = 500,
                                anomaly_threshold = 0.6)

# Plot the output for a cell type
plot(anomaly_output,
     cell_type = "CD4",
     pc_subset = 1:3,
     data_type = "query")
```

---

histQCvsAnnotation *Histograms: QC Stats and Annotation Scores Visualization*

Description

This function generates histograms for visualizing the distribution of quality control (QC) statistics and annotation scores associated with cell types in single-cell genomic data.

Usage

```
histQCvsAnnotation(
  sce_object,
  cell_type_col,
  cell_types = NULL,
  qc_col,
  score_col,
  max_cells = NULL
)
```

Arguments

sce_object	A SingleCellExperiment containing the single-cell expression data and metadata.
cell_type_col	The column name in the colData of sce_object that contains the cell type labels.
cell_types	A vector of cell types to plot (e.g., c("T-cell", "B-cell")). Defaults to NULL, which will include all the cells.
qc_col	A column name in the colData of sce_object that contains the QC stats of interest.
score_col	The column name in the colData of sce_object that contains the cell type scores.
max_cells	Maximum number of cells to retain. If the object has fewer cells, it is returned unchanged. Default is NULL.

Details

The particularly useful in the analysis of data from single-cell experiments, where understanding the distribution of these metrics is crucial for quality assessment and interpretation of cell type annotations.

Value

A object containing two histograms displayed side by side. The first histogram represents the distribution of QC stats, and the second histogram represents the distribution of annotation scores.

Examples

```
data("query_data")

# Generate histograms
histQCvsAnnotation(sce_object = query_data,
                   cell_type_col = "SingleR_annotation",
                   cell_types = c("CD4", "CD8"),
                   qc_col = "percent_mito",
                   score_col = "annotation_scores")

histQCvsAnnotation(sce_object = query_data,
                   cell_type_col = "SingleR_annotation",
                   cell_types = NULL,
                   qc_col = "percent_mito",
                   score_col = "annotation_scores")
```

`plot.regressPCObject` *Plot Regression Results on Principal Components*

Description

The S3 plot method generates plots to visualize the results of regression analyses performed on principal components (PCs) against cell types, datasets, or their interactions.

This function performs linear regression of a covariate of interest onto one or more principal components, based on the data in a `SingleCellExperiment` object.

Usage

```
## S3 method for class 'regressPCObject'
plot(
  x,
  plot_type = c("r_squared", "variance_contribution", "coefficient_heatmap"),
  alpha = 0.05,
  coefficients_include = NULL,
  ...
)

regressPC(
  query_data,
  reference_data = NULL,
  query_cell_type_col,
  ref_cell_type_col = NULL,
  query_batch_col = NULL,
  cell_types = NULL,
  pc_subset = 1:10,
  adjust_method = c("BH", "holm", "hochberg", "hommel", "bonferroni", "BY", "fdr",
    "none"),
  assay_name = "logcounts",
  max_cells_ref = 5000,
  max_cells_query = 5000
)
```

Arguments

`x`	An object of class `regressPCObject` containing the output of the `regressPC` function.
`plot_type`	Type of plot to generate. Available options: "r_squared", "variance_contribution", "coefficient_heatmap".
`alpha`	Significance threshold for p-values. Default is 0.05.
`coefficients_include`	Character vector specifying which coefficient types to include in the coefficient heatmap. Options are `c("cell_type", "batch", "interaction")`. Default is

	NULL, which includes all available coefficient types. Only applies to plot_type = "coefficient_heatmap".
...	Additional arguments to be passed to the plotting functions.
query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells. If NULL, the PC scores are regressed against the cell types of the query data.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
query_batch_col	The column name in the colData of query_data that identifies the batch or sample. If provided, performs interaction analysis with cell types. Default is NULL.
cell_types	A character vector specifying the cell types to include in the analysis. If NULL, all cell types are included.
pc_subset	A numeric vector specifying which principal components to include in the analysis. Default is PC1 to PC10.
adjust_method	A character string specifying the method to adjust the p-values. Options include "BH", "holm", "hochberg", "hommel", "bonferroni", "BY", "fdr", or "none". Default is "BH" (Benjamini-Hochberg).
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.

Details

Principal component regression, derived from PCA, can be used to quantify the variance explained by a covariate of interest. Applications for single-cell analysis include quantification of batch effects, assessing clustering homogeneity, and evaluating alignment of query and reference datasets in cell type annotation settings.

The function supports multiple regression scenarios:

- Query only, no batch: PC cell_type
- Query only, with batch: PC cell_type * batch
- Query + Reference, no batch: PC cell_type * dataset
- Query + Reference, with batch: PC cell_type * batch (where batch includes Reference)

When batch information is provided with reference data, batches are labeled as "Reference" for reference data and "Query_BatchName" for query batches, with Reference set as the first factor level for interpretation.

Value

The S3 plot method returns a ggplot object representing the specified plot type.

A list containing

- summaries of the linear regression models for each specified principal component,
- the corresponding R-squared (R2) values,
- the variance contributions for each principal component, and
- the total variance explained.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

Luecken et al. Benchmarking atlas-level data integration in single-cell genomics. Nature Methods, 19:41-50, 2022.

See Also

regressPC

plot.regressPCObject

Examples

```
# Load data
data("reference_data")
data("query_data")

# Query only analysis
regress_res <- regressPC(query_data = query_data,
                         query_cell_type_col = "expert_annotation",
                         cell_types = c("CD4", "CD8", "B_and_plasma", "Myeloid"),
                         pc_subset = 1:10)
# Visualize results
plot(regress_res, plot_type = "r_squared")
plot(regress_res, plot_type = "variance_contribution")
plot(regress_res, plot_type = "coefficient_heatmap")

# Query + Reference analysis
regress_res <- regressPC(query_data = query_data,
                         reference_data = reference_data,
                         query_cell_type_col = "SingleR_annotation",
                         ref_cell_type_col = "expert_annotation",
                         cell_types = c("CD4", "CD8", "B_and_plasma", "Myeloid"),
                         pc_subset = 1:10)
# Visualize results
plot(regress_res, plot_type = "r_squared")
plot(regress_res, plot_type = "variance_contribution")
plot(regress_res, plot_type = "coefficient_heatmap")
```

plotCellTypeMDS *Plot Reference and Query Cell Types using MDS*

Description

This function facilitates the assessment of similarity between reference and query datasets through Multidimensional Scaling (MDS) scatter plots. It allows the visualization of cell types, color-coded with user-defined custom colors, based on a dissimilarity matrix computed from a user-selected gene set. If MDS coordinates are precomputed in reducedDims, they will be used; otherwise, MDS will be computed from scratch.

Usage

```
plotCellTypeMDS(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000
)
```

Arguments

query_data	A SingleCellExperiment containing the single-cell expression data and metadata.
reference_data	A SingleCellExperiment object containing the single-cell expression data and metadata.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
cell_types	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

Details

The function first checks if MDS coordinates are available in the reducedDims of both datasets. If precomputed MDS is found, it uses those coordinates directly for visualization.

If MDS is not precomputed, the function selects specific subsets of cells from both reference and query datasets. It then calculates Spearman correlations between gene expression profiles, deriving a dissimilarity matrix. This matrix undergoes Classical Multidimensional Scaling (MDS) for visualization, presenting cell types in a scatter plot, distinguished by colors defined by the user.

Value

A ggplot object representing the MDS scatter plot with cell type coloring.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

References

- Kruskal, J. B. (1964). "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". *Psychometrika*, 29(1), 1-27. doi:10.1007/BF02289565.
- Borg, I., & Groenen, P. J. F. (2005). *Modern multidimensional scaling: Theory and applications* (2nd ed.). Springer Science & Business Media. doi:10.1007/978-0-387-25975-1.

Examples

```
# Load data
data("reference_data")
data("query_data")

# Generate the MDS scatter plot with cell type coloring
mds_plot <- plotCellTypeMDS(query_data = query_data,
                            reference_data = reference_data,
                           cell_types = c("CD4", "CD8", "B_and_plasma", "Myeloid")[1:4],
                            query_cell_type_col = "SingleR_annotation",
                            ref_cell_type_col = "expert_annotation")
mds_plot
```

`plotCellTypePCA` *Plot Principal Components for Different Cell Types*

Description

This function plots the principal components for different cell types in the query and reference datasets.

Usage

```
plotCellTypePCA(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:5,
  assay_name = "logcounts",
  lower_facet = c("scatter", "contour", "ellipse", "blank"),
  diagonal_facet = c("ridge", "density", "boxplot"),
  upper_facet = c("blank", "scatter", "contour", "ellipse"),
  max_cells_query = 2000,
  max_cells_ref = 2000
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
cell_types	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.
pc_subset	A numeric vector specifying which principal components to include in the plot. Default is 1:5.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
lower_facet	Type of plot to use for the lower panels. Either "scatter" (default), "contour", "ellipse", or "blank".
diagonal_facet	Type of plot to use for the diagonal panels. Either "ridge" (default), "density", or "boxplot".
upper_facet	Type of plot to use for the upper panels. Either "blank" (default), "scatter", "contour", or "ellipse".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 2000.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 2000.

Details

This function projects the query dataset onto the principal component space of the reference dataset and then plots the specified principal components for the specified cell types. It uses the 'project-PCA' function to perform the projection and `GGally` to create the pairs plot.

Value

A ggmatrix object representing a pairs plot of specified principal components for the given cell types and datasets.

`plotGeneExpressionDimred` *Visualize gene expression on a dimensional reduction plot*

Description

This function plots gene expression on a dimensional reduction plot using methods like t-SNE, UMAP, or PCA. Each single cell is color-coded based on the expression of a specific gene or feature.

Usage

```
plotGeneExpressionDimred(
  sce_object,
  method = c("TSNE", "UMAP", "PCA"),
  pc_subset = 1:5,
  feature,
  cell_type_col,
  cell_types = NULL,
  assay_name = "logcounts",
  max_cells = 2000
)
```

Arguments

`sce_object`	An object of class `SingleCellExperiment` containing log-transformed expression matrix and other metadata. It can be either a reference or query dataset.
`method`	The reduction method to use for visualization. It should be one of the supported methods: "TSNE", "UMAP", or "PCA".
`pc_subset`	An optional vector specifying the principal components (PCs) to include in the plot if method = "PCA". Default is 1:5.
`feature`	A character string representing the name of the gene or feature to be visualized.
`cell_type_col`	The column name in the `colData` of `sce_object` that identifies the cell types.
`cell_types`	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.

assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells	Maximum number of cells to retain. If the object has fewer cells, it is returned unchanged. Default is 2000.

Value

A ggplot object representing the dimensional reduction plot with gene expression.

Examples

```
# Load data
data("query_data")

# Plot gene expression on PCA plot
plotGeneExpressionDimred(sce_object = query_data,
                         cell_type_col = "SingleR_annotation",
                         method = "PCA",
                         pc_subset = 1:5,
                         feature = "CD8A",
                         cell_types = "CD4")
```

---

plotGeneSetScores *Visualization of gene sets or pathway scores on dimensional reduction plot*

Description

Plot gene sets or pathway scores on PCA, TSNE, or UMAP. Single cells are color-coded by scores of gene sets or pathways.

Usage

```
plotGeneSetScores(
  sce_object,
  cell_type_col,
  method = c("PCA", "TSNE", "UMAP"),
  score_col,
  pc_subset = 1:5,
  cell_types = NULL,
  max_cells = 2000
)
```

Arguments

sce_object	An object of class SingleCellExperiment containing numeric expression matrix and other metadata. It can be either a reference or query dataset.
cell_type_col	The column name in the colData of sce_object that identifies the cell types.

method	A character string indicating the method for visualization ("PCA", "TSNE", or "UMAP").
score_col	A character string representing the name of the score_col (score) in the colData(sce_object) to plot.
pc_subset	An optional vector specifying the principal components (PCs) to include in the plot if method = "PCA". Default is 1:5.
cell_types	A character vector specifying the cell types to include in the plot. If NULL, all cell types are included.
max_cells	Maximum number of cells to retain. If the object has fewer cells, it is returned unchanged. Default is 2000.

Details

This function plots gene set scores on reduced dimensions such as PCA, t-SNE, or UMAP. It extracts the reduced dimensions from the provided SingleCellExperiment object. Gene set scores are visualized as a scatter plot with colors indicating the scores. For PCA, the function automatically includes the percentage of variance explained in the plot's legend.

Value

A ggplot2 object representing the gene set scores plotted on the specified reduced dimensions.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

Examples

```
# Load data
data("query_data")

# Plot gene set scores on PCA
plotGeneSetScores(sce_object = query_data,
                  method = "PCA",
                  score_col = "gene_set_scores",
                  pc_subset = 1:5,
                  cell_types = "CD8",
                  cell_type_col = "SingleR_annotation")
```

---

plotMarkerExpression *Plot gene expression distribution from overall and cell type-specific perspective*

Description

This function generates density plots to visualize the distribution of gene expression values for a specific gene across the overall dataset and within a specified cell type.

Usage

```
plotMarkerExpression(
  query_data,
  reference_data,
  ref_cell_type_col,
  query_cell_type_col,
  cell_type,
  gene_name,
  assay_name = "logcounts",
  normalization = c("z_score", "min_max", "rank", "none"),
  max_cells_query = NULL,
  max_cells_ref = NULL
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
cell_type	A cell type to plot (e.g., c("T-cell", "B-cell")).
gene_name	The gene name for which the distribution is to be visualized.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
normalization	Method for normalizing expression values. Options: "z_score" (default), "min_max", "rank", "none".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is NULL.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is NULL.

Details

This function generates density plots to compare the distribution of a specific marker gene between reference and query datasets. The aim is to inspect the alignment of gene expression levels as a surrogate for dataset similarity. Similar distributions suggest a good alignment, while differences may indicate discrepancies or incompatibilities between the datasets.

Multiple normalization options are available: - "z_score": Standard z-score normalization within each dataset - "min_max": Min-max scaling to [0,1] range within each dataset - "rank": Maps values to quantile ranks (0-100 scale) - "none": No transformation (preserves original scale differences)

Value

A ggplot object containing density plots comparing reference and query distributions.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

Examples

```
# Load data
data("reference_data")
data("query_data")

# Note: Users can use SingleR or any other method to obtain the cell type annotations.
plotMarkerExpression(reference_data = reference_data,
                     query_data = query_data,
                     ref_cell_type_col = "expert_annotation",
                   query_cell_type_col = c("expert_annotation", "SingleR_annotation")[1],
                     gene_name = "CD8A",
                     cell_type = "CD4",
                     normalization = "z_score")
```

---

plotPairwiseDistancesDensity

Ridgeline Plot of Pairwise Distance Analysis

Description

This function calculates pairwise distances or correlations between query and reference cells of a specified cell type and visualizes the results using ridgeline plots, displaying the density distribution for each comparison.

Usage

```
plotPairwiseDistancesDensity(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_type,
  pc_subset = 1:5,
  distance_metric = c("correlation", "euclidean"),
  correlation_method = c("spearman", "pearson"),
  bandwidth = 0.25,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000
)
```

Arguments

query_data	A SingleCellExperiment containing the single-cell expression data and metadata.
reference_data	A SingleCellExperiment object containing the single-cell expression data and metadata.
query_cell_type_col	The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	The column name in the colData of reference_data that identifies the cell types.
cell_type	The cell type for which distances or correlations are calculated.
pc_subset	A numeric vector specifying which principal components to use in the analysis. Default is 1:5. If set to NULL, the assay data is used directly for computations without dimensionality reduction.
distance_metric	The distance metric to use for calculating pairwise distances, such as euclidean, manhattan, etc. Set to "correlation" to calculate correlation coefficients.
correlation_method	The correlation method to use when distance_metric is "correlation". Possible values are "pearson" and "spearman".
bandwidth	Numeric value controlling the smoothness of the density estimate; smaller values create more detailed curves. Default is 0.25.
assay_name	Name of the assay on which to perform computations. Default is "logcounts".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

Details

Designed for SingleCellExperiment objects, this function subsets data for the specified cell type, computes pairwise distances or correlations, and visualizes these measurements through ridgeline plots. The plots help evaluate the consistency and differentiation of annotated cell types within single-cell datasets.

Value

A ggplot2 object showing ridgeline plots of calculated distances or correlations.

See Also

calculateWassersteinDistance

Examples

```
# Load data
data("reference_data")
data("query_data")

# Example usage of the function
plotPairwiseDistancesDensity(query_data = query_data,
                             reference_data = reference_data,
                             query_cell_type_col = "SingleR_annotation",
                             ref_cell_type_col = "expert_annotation",
                             cell_type = "CD8",
                             pc_subset = 1:5,
                             distance_metric = "euclidean",
                             correlation_method = "pearson")
```

`plotQCvsAnnotation` *Scatter plot: QC stats vs Cell Type Annotation Scores*

Description

Creates a scatter plot to visualize the relationship between QC stats (e.g., library size) and cell type annotation scores for one or more cell types.

Usage

```
plotQCvsAnnotation(
  sce_object,
  cell_type_col,
  cell_types = NULL,
  qc_col,
  score_col,
  max_cells = 5000
)
```

Arguments

`sce_object`	A `SingleCellExperiment` containing the single-cell expression data and metadata.
`cell_type_col`	The column name in the `colData` of `sce_object` that contains the cell type labels.
`cell_types`	A vector of cell type labels to plot (e.g., c("T-cell", "B-cell")). Defaults to `NULL`, which will include all the cells.
`qc_col`	A column name in the `colData` of `sce_object` that contains the QC stats of interest.
`score_col`	The column name in the `colData` of `sce_object` that contains the cell type annotation scores.

max_cells Maximum number of cells to retain. If the object has fewer cells, it is returned unchanged. Default is 5000.

Details

This function generates a scatter plot to explore the relationship between various quality control (QC) statistics, such as library size and mitochondrial percentage, and cell type annotation scores. By examining these relationships, users can assess whether specific QC metrics, systematically influence the confidence in cell type annotations, which is essential for ensuring reliable cell type annotation.

Value

A ggplot object displaying a scatter plot of QC stats vs annotation scores, where each point represents a cell, color-coded by its cell type.

Examples

```
# Load data
data("qc_data")

# Remove cell types with very few cells
qc_data_subset <- qc_data[, !(qc_data$SingleR_annotation
                              %in% c("Chondrocytes", "DC",
                                     "Neurons","Platelets"))]

p1 <- plotQCvsAnnotation(sce_object = qc_data_subset,
                         cell_type_col = "SingleR_annotation",
                         cell_types = NULL,
                         qc_col = "total",
                         score_col = "annotation_scores")
p1 + ggplot2::xlab("Library Size")
```

---

processPCA *Process PCA for SingleCellExperiment Objects*

Description

This function ensures that a SingleCellExperiment object has valid PCA computed using highly variable genes when needed. It only performs downsampling when PCA computation is required, preserving existing valid PCA computations without modification.

Usage

```
processPCA(
  sce_object,
  assay_name = "logcounts",
  n_hvgs = 2000,
```

```
  max_cells = NULL
)
```

Arguments

sce_object	A SingleCellExperiment object to process.
assay_name	Name of the assay to use for HVG selection and PCA computation. Should contain log-normalized expression values. Default is "logcounts".
n_hvgs	Number of highly variable genes to select for PCA computation. Default is 2000.
max_cells	Maximum number of cells to retain if downsampling is needed for PCA computation. If NULL, no downsampling is performed. Default is NULL.

Details

The function performs the following operations:

- Checks if PCA exists and is valid in the provided SingleCellExperiment object
- Validates PCA integrity including rotation matrix, percentVar, gene consistency, and dimensions
- If PCA is valid, returns the object unchanged (no downsampling)
- If PCA is missing or invalid and dataset is large, downsamples before computing PCA
- Computes PCA using highly variable genes when PCA is missing or invalid
- Utilizes scran for HVG selection and scater for PCA computation (soft dependencies)

The downsampling strategy uses random sampling without replacement and only occurs when PCA computation is necessary. This preserves expensive pre-computed PCA results while ensuring computational efficiency for new PCA computations.

PCA validation includes checking for:

- Presence of PCA in reducedDims
- Existence of rotation matrix and percentVar attributes
- Gene consistency between rotation matrix and current assay
- Dimension consistency between PCA coordinates and cell count

Value

A SingleCellExperiment object with valid PCA in the reducedDims slot, including rotation matrix and percentVar attributes. Will have original cell count if PCA was valid, or at most max_cells if PCA was computed.

Note

This function requires the scran and scater packages for HVG selection and PCA computation. These packages should be installed via BiocManager::install(c("scran", "scater")).

Objects with existing valid PCA are returned unchanged to preserve expensive pre-computations. Only datasets requiring PCA computation are subject to downsampling.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

Examples

```
library(SingleCellExperiment)

# Load data
data("reference_data")
data("query_data")

# Example 1: Dataset without PCA (will compute PCA)
query_no_pca <- query_data
reducedDims(query_no_pca) <- list()  # Remove existing PCA

processed_query <- processPCA(sce_object = query_no_pca, n_hvgs = 500)
"PCA" %in% reducedDimNames(processed_query)  # Should be TRUE
ncol(processed_query)  # Should be 503 (unchanged)

# Example 2: Dataset with existing valid PCA (will be preserved)
processed_existing <- processPCA(sce_object = query_data, n_hvgs = 500)
ncol(processed_existing)  # Should be 503 (unchanged, no downsampling)

# Example 3: Large dataset requiring downsampling for PCA computation
ref_no_pca <- reference_data
reducedDims(ref_no_pca) <- list()  # Remove existing PCA

processed_large <- processPCA(sce_object = ref_no_pca,
                              n_hvgs = 800,
                              max_cells = 1000)
ncol(processed_large)  # Should be 1000 (downsampled for PCA computation)

# Example 4: Large dataset with existing PCA (no downsampling)
processed_large_existing <- processPCA(sce_object = reference_data,
                                       n_hvgs = 800,
                                       max_cells = 1000)
ncol(processed_large_existing)  # Should be 1500 (preserved, no downsampling)
```

`projectPCA` *Project Query Data Onto PCA Space of Reference Data*

Description

This function projects a query singleCellExperiment object onto the PCA space of a reference singleCellExperiment object. The PCA analysis on the reference data is assumed to be pre-computed and stored within the object. Optionally filters by cell types and downsamples the results.

Usage

```
projectPCA(
  query_data,
  reference_data,
  query_cell_type_col,
  ref_cell_type_col,
  cell_types = NULL,
  pc_subset = 1:10,
  assay_name = "logcounts",
  max_cells_query = NULL,
  max_cells_ref = NULL
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_cell_type_col	character. The column name in the colData of query_data that identifies the cell types.
ref_cell_type_col	character. The column name in the colData of reference_data that identifies the cell types.
cell_types	A character vector specifying which cell types to retain in the output. If NULL, no cell type filtering is performed. Default is NULL.
pc_subset	A numeric vector specifying the subset of principal components (PCs) to compare. Default is 1:10.
assay_name	Name of the assay on which to perform computations. Defaults to "logcounts".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is NULL.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is NULL.

Details

This function assumes that the "PCA" element exists within the reducedDims of the reference data (obtained using reducedDim(reference_data)) and that the genes used for PCA are present in both the reference and query data. It performs centering and scaling of the query data based on the reference data before projection using the FULL datasets to maintain proper mean centering. Cell type filtering and downsampling are performed AFTER projection to preserve the statistical properties of the PCA space. Cell names from the original SCE objects are preserved as rownames in the output.

Value

A `data.frame` containing the projected data in rows (reference and query data combined), optionally filtered by cell types and downsampled. Rownames preserve the original cell names from the SCE objects.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

Examples

```
# Load data
data("reference_data")
data("query_data")

# Project the query data onto PCA space of reference
pca_output <- projectPCA(query_data = query_data,
                         reference_data = reference_data,
                         query_cell_type_col = "SingleR_annotation",
                         ref_cell_type_col = "expert_annotation",
                         pc_subset = 1:10)

# Project with cell type filtering and balanced downsampling
pca_output_filtered <- projectPCA(query_data = query_data,
                                  reference_data = reference_data,
                                  query_cell_type_col = "SingleR_annotation",
                                  ref_cell_type_col = "expert_annotation",
                                  pc_subset = 1:5,
                                  cell_types = c("CD4", "CD8"),
                                  max_cells_ref = 1000,
                                  max_cells_query = 1000)
```

---

projectSIR *Project Query Data Onto SIR Space of Reference Data*

Description

This function projects a query `SingleCellExperiment` object onto the SIR (supervised independent component) space of a reference `SingleCellExperiment` object. The SVD of the reference data is computed on conditional means per cell type, and the query data is projected based on these reference components.

Usage

```
projectSIR(
  query_data,
  reference_data,
  query_cell_type_col,
```

```
  ref_cell_type_col,
  cell_types = NULL,
  multiple_cond_means = TRUE,
  cumulative_variance_threshold = 0.7,
  n_neighbor = 1,
  assay_name = "logcounts",
  max_cells_query = 5000,
  max_cells_ref = 5000
)
```

Arguments

query_data	A SingleCellExperiment object containing numeric expression matrix for the query cells.
reference_data	A SingleCellExperiment object containing numeric expression matrix for the reference cells.
query_cell_type_col	A character string specifying the column in the colData of query_data that identifies the cell types.
ref_cell_type_col	A character string specifying the column in the colData of reference_data that identifies the cell types.
cell_types	A character vector of cell types for which to compute conditional means in the reference data.
multiple_cond_means	A logical value indicating whether to compute multiple conditional means per cell type (through PCA and clustering). Defaults to TRUE.
cumulative_variance_threshold	A numeric value between 0 and 1 specifying the variance threshold for PCA when computing multiple conditional means. Defaults to 0.7.
n_neighbor	An integer specifying the number of nearest neighbors for clustering when computing multiple conditional means. Defaults to 1.
assay_name	A character string specifying the assay name on which to perform computations. Defaults to "logcounts".
max_cells_query	Maximum number of query cells to retain after cell type filtering. If NULL, no downsampling of query cells is performed. Default is 5000.
max_cells_ref	Maximum number of reference cells to retain after cell type filtering. If NULL, no downsampling of reference cells is performed. Default is 5000.

Details

The genes used for the projection (SVD) must be present in both the reference and query datasets. The function first computes conditional means for each cell type in the reference data, then performs SVD on these conditional means to obtain the rotation matrix used for projecting both the reference and query datasets. The query data is centered and scaled based on the reference data.

Value

A list containing:

cond_means	A matrix of the conditional means computed for the reference data.
rotation_mat	The rotation matrix obtained from the SVD of the conditional means.
sir_projections	A data.frame containing the SIR projections for both the reference and query datasets.
percent_var	The percentage of variance explained by each component of the SIR projection.

Author(s)

Anthony Christidis, <anthony-alexander_christidis@hms.harvard.edu>

Examples

```
# Load data
data("reference_data")
data("query_data")

# Project the query data onto SIR space of reference
sir_output <- projectSIR(query_data = query_data,
                         reference_data = reference_data,
                         query_cell_type_col = "SingleR_annotation",
                         ref_cell_type_col = "expert_annotation")
```

Index