Package: visualizationQualityControl (via r-universe)

September 13, 2024

Version 0.5.1

Title Development of visualization methods for quality control

Description Provides utilities useful quality control of high-throughput -omics datasets.

Date 2024-02-15

Depends R (>= 3.1.1)

Imports ComplexHeatmap (>= 1.2.1), stats, dendsort, colorspace, dplyr, ggplot2, broom, knitrProgressBar, magrittr, purrr

License MIT + file LICENSE

VignetteBuilder knitr

Suggests testthat, knitr, rmarkdown, circlize, viridis, ICIKendallTau (>= 1.0.0), ggforce

RoxygenNote 7.2.3

Encoding UTF-8

LinkingTo Rcpp

URL https://moseleybioinformaticslab.github.io/visualizationQualityControl https://github.com/moseleybioinformaticslab/visualizationQualityControl

Remotes github::MoseleyBioinformaticsLab/ICIKendallTau

Config/testthat/edition 3

Repository https://moseleybioinformaticslab.r-universe.dev

RemoteUrl https://github.com/MoseleyBioinformaticsLab/visualizationQualityControl

RemoteRef *release

RemoteSha 118c765fd076f08d32cadc7693cc738a36e08c3f

Contents

`calculate_fratio` *calculate F-ratio*

Description

given a data matrix of samples (columns) and features (rows), and a vector of classes (character or factor), calculate an F-ratio for each feature.

Usage

```
calculate_fratio(data, data_classes)
```

Arguments

`data`	the data matrix, with samples (columns) and features (rows)
`data_classes`	what are the classes of the samples (columns)

Value

vector

`calc_sd_rsd` *calculate values from summaries*

Description

given a data.frame of means and variances, calculate mean sd at low end and mean rsd at high end.

Usage

```
calc_sd_rsd(data, low_cut, hi_cut = NULL)
```

Arguments

`data`	data.frame of means and variances
`low_cut`	means <= this value used for average sd
`hi_cut`	means >= this value used for average rsd

Value

vector

`calc_sd_rsd_nls` *calculate values from summaries v2*

Description

given a data.frame of means and variances, use a two step non-linear least squares. The first step is done on the mean vs sd, then the estimates are used in a second that estimates them using the mean vs rsd.

Usage

```
calc_sd_rsd_nls(data, ...)
```

Arguments

`data`	data.frame of means and variances
`...`	other nls parameters

Value

vector

count_matching_features *matching features*

Description

For a given set of feature-sample matrices, calculates how many features are in that sample, as well as in common to all other samples, and if provided, in common within and outside the same sample group.

Usage

```
count_matching_features(feature_matrix, zero_value = NA, groups = NULL)
```

Arguments

feature_matrix	the feature to sample matrix.
zero_value	what is the zero value? Default is NA
groups	what are the groups

Value

data.frame

determine_outliers *determine outliers*

Description

determine outliers

Usage

```
determine_outliers(
  median_correlations = NULL,
  outlier_fraction = NULL,
  cor_weight = 1,
  frac_weight = 1,
  only_high = TRUE
)
```

Arguments

median_correlations	median correlations
outlier_fraction	outlier fractions
cor_weight	how much weight for the correlation score?
frac_weight	how much weight for the outlier fraction?
only_high	should only things at the low end of score be removed?

Details

For outlier sample detection, one should first generate median correlations using 'median_correlations', and outlier fractions using 'outlier_fraction'. If you only have one or the other, than you should use named arguments to only pass the one or the other.

Alternatively, you can change the weighting used for median correlations or outlier fraction, including setting them to 0.

Value

data.frame

---

filter_non_zero_percentage

keep features with percentage of non-zeros

Description

Given a value matrix (features are columns, samples are rows), and sample classes, find those things that are not zero in at least a certain number of one of the classes, and keep them

Usage

```
filter_non_zero_percentage(data_matrix, sample_classes = NULL, keep_num = 0.75)
```

Arguments

data_matrix	the matrix of values to work with
sample_classes	the classes of each sample
keep_num	what number of samples in each class need a non-zero value (see Details)

Details

This function is being deprecated and all code should use the keep_non_zero_percentage function instead.

Value

matrix

See Also

keep_non_zero_percentage

---

generate_group_colors *create set of disjoint colors*

Description

When multiple sample classes need to be visualized on a heatmap, it is useful to be able to distinguish them by color. This function generates a set of colors for sample classes

Usage

```
generate_group_colors(n_group, randomize = NULL)
```

Arguments

n_group	how many groups should there be colors for
randomize	should colors be randomized? (default is NULL). See *details*.

Details

the default for randomize is NULL, so that reordering the colors randomly is decided purely based on the number of colors requested. Currently, that cutoff is *5* colors, less than that the colors will always be in the same order, for *5* colors or more, they will be in a scrambled order, different each time unless set.seed is used. If randomize is TRUE or FALSE, then it overrides the defaults.

---

grp_cor *10 sample to sample correlations*

Description

test data of 10 sample to sample correlations where samples are drawn from two groups. Generated by rmflight from random distributions

Format

matrix with 10 rows and 10 columns, with row and colnames

Source

generated by rmflight

grp_cor_data *grp_cor_data*

Description

Example data used for demonstrating `median correlation`. A list with 2 named entries:

Usage

```
grp_cor_data
```

Format

List with 2 entries, data and class

Details

data a data matrix with 100 rows and 20 columns

class a character vector of 20 entries denoting classes

The data comes from two groups of samples, where there is ~0.85 correlation within each group, and ~0.53 correlation between groups.

Source

Robert M Flight

grp_exp_data *grp_exp_data*

Description

Example data that requires log-transformation before doing PCA or other QC. A list with 2 named entries:

Usage

```
grp_exp_data
```

Format

List with 2 entries, data and class

Details

data a data matrix with 1000 rows and 20 columns

class a character vector of 20 entries denoting classes

The data comes from two groups of samples, where there is ~0.80 correlation within each group, and ~0.38 correlation between groups.

Source

Robert M Flight

`grp_info` *10 sample meta-data*

Description

meta-data for `grp_cor`.

Format

data.frame with 10 rows, and 2 columns, `grp` defining with group and `set`, defining the set.

Source

generated by rmflight

`keep_non_missing_percentage` *keep features with percentage of non-missing*

Description

Given a value matrix (features are rows, samples are columns), and sample classes, find those things that are *not missing* in at least a certain number of samples in one of the classes, and keep those features for further processing.

Usage

```
keep_non_missing_percentage(
  data_matrix,
  sample_classes = NULL,
  keep_num = 0.75,
  missing_value = NA,
  all = FALSE
)
```

Arguments

data_matrix	the matrix of values to work with
sample_classes	the classes of each sample
keep_num	what number of samples in each class need a non-missing value (see Details)
missing_value	what number(s) represents missing values (default NA)
all	is this an either / or OR does it need to be present in all?

Details

The number of samples that must be non-missing can be expressed either as a whole number (that is greater than one), or as a fraction that will be be multiplied by the number of samples in each class to get the lower limits for each of the classes. If there are multiple values that represent missingness, use a vector. For example, to to use both 0 and NA, you can do `missing_value = c(NA, 0)`.

Value

logical

---

keep_non_zero_percentage

keep features with percentage of non-zeros

Description

Given a value matrix (features are rows, samples are columns), and sample classes, find those things that are *not zero* in at least a certain number of samples in one of the classes, and keep those features for further processing.

Usage

```
keep_non_zero_percentage(
  data_matrix,
  sample_classes = NULL,
  keep_num = 0.75,
  zero_value = 0,
  all = FALSE
)
```

Arguments

data_matrix	the matrix of values to work with
sample_classes	the classes of each sample
keep_num	what number of samples in each class need a non-zero value (see Details)
zero_value	what number represents zero values
all	is this an either / or OR does it need to be present in all?

Details

The number of samples that must be non-zero can be expressed either as a whole number (that is greater than one), or as a fraction that will be be multiplied by the number of samples in each class to get the lower limits for each of the classes.

Value

logical

---

median_class_correlations *calculate median class correlations*

Description

Given a correlation matrix the sample class information, calculates the median correlations of the samples within the class and between classes.

Usage

```
median_class_correlations(cor_matrix, sample_classes = NULL)
```

Arguments

cor_matrix	the sample - sample correlations
sample_classes	the sample classes as a character or factor

Value

matrix

---

median_correlations *calculate median correlations*

Description

Given a correlation matrix and optionally the sample class information, calculates the median correlations of each sample to all other samples in the same class. May be useful for determining outliers.

Usage

```
median_correlations(cor_matrix, sample_classes = NULL, between_classes = FALSE)
```

Arguments

cor_matrix	the sample - sample correlations
sample_classes	the sample classes as a character or factor
between_classes	should the between class correlations be evaluated?

Details

The data.frame may have 5 columns, first three are always present, the second two come up if between_classes = TRUE:

med_cor the median correlation with other samples

sample_id the sample id, either the rowname or an index

sample_class the class of the sample. If not provided, set to "C1"

compare_class the class of the other sample

plot_class sample_class::compare_class for easy grouping

Value

data.frame

---

outlier_fraction *fraction of outliers*

Description

Calculates the fraction of entries in each sample that are more than X standard deviations from the trimmed mean. See Details.

Usage

```
outlier_fraction(
  data,
  sample_classes = NULL,
  n_trim = 3,
  n_sd = 5,
  remove_missing = NA
)
```

Arguments

data	the data matrix (samples are columns, rows are features)
sample_classes	the sample classes
n_trim	how many features to trim at each end (default is 3)
n_sd	how many SD before treated as outlier (default is 5)
remove_missing	what missing values be removed before calculating? (default is NA)

Details

Based on the Gerlinski paper link for each feature (in a sample class), take the range across all the samples, remove the `n_trim` lowest and highest values, and calculate the `mean` and `sd`, and the actual upper and lower ranges of `n_sd` from the `mean`. For each sample and feature, determine if *within* or *outside* that limit. Fraction is reported as the number of features outside the range.

Returns a `data.frame` with:

sample_id the sample id, `rownames` are used if available, otherwise this is an index

sample_class the class of the sample if `sample_classes` were provided, otherwise given a default of "C1"

frac the actual outlier fraction calculated for that sample

Value

data.frame

`similarity_reorder` *cluster and reorder*

Description

given a matrix (maybe a distance matrix), cluster and then re-order using dendsort.

Usage

```
similarity_reorder(
  similarity_matrix,
  matrix_indices = NULL,
  transform = "none",
  hclust_method = "complete",
  dendsort_type = "min"
)
```

Arguments

`similarity_matrix`	matrix of similarities
`matrix_indices`	indices to reorder
`transform`	should a transformation be applied to the data first
`hclust_method`	which method for clustering should be used?
`dendsort_type`	how should the reordering be done? (default is "min")

Value

a dendrogram object. To get the order use `order.dendogram`.

`similarity_reorderbyclass` *reorder by sample class*

Description

to avoid spurious visualization problems, it is useful in a heatmap visualization to reorder the samples within each sample class. This function uses hierarchical clustering and dendsort to sort entries in a distance matrix.

Usage

```
similarity_reorderbyclass(
  similarity_matrix,
  sample_classes = NULL,
  transform = "none",
  hclust_method = "complete",
  dendsort_type = "min"
)
```

Arguments

`similarity_matrix`	matrix of similarities between objects
`sample_classes`	data.frame or factor denoting classes
`transform`	a transformation to apply to the data
`hclust_method`	which method for clustering should be used
`dendsort_type`	how should dendsort do reordering?

Details

The `similarity_matrix` should be either a square matrix of similarity values or a distance matrix of class `dist`. If your matrix does not encode a "true" distance, you can use a `transform` to turn it into a true `distance` (for example, if you have correlation, then a distance would be 1 - correlation, use "sub_1" as the transform argument).

The `sample_classes` should be either a data.frame or factor argument. If a data.frame is passed, all columns of the data.frame will be pasted together to create a factor for splitting the data into groups. If the rownames of the data.frame do not correspond to the rownames or colnames of the matrix, then it is assumed that the ordering in the matrix and the data.frame are identical.

Value

a list containing the reordering of the matrix in a:

1. dendrogram
2. numeric vector
3. character vector (will be NULL if rownames are not set on the matrix)

Examples

```
library(visualizationQualityControl)
set.seed(1234)
mat <- matrix(rnorm(100, 2, sd = 0.5), 10, 10)
rownames(mat) <- colnames(mat) <- letters[1:10]
neworder <- similarity_reorderbyclass(mat)
mat[neworder$indices, neworder$indices]

sample_class <- data.frame(grp = rep(c("grp1", "grp2"), each = 5), stringsAsFactors = FALSE)
rownames(sample_class) <- rownames(mat)
neworder2 <- similarity_reorderbyclass(mat, sample_class[, "grp", drop = FALSE])

# if there is a class with only one member, it is dropped, with a warning
sample_class[10, "grp"] = "grp3"
neworder3 <- similarity_reorderbyclass(mat, sample_class[, "grp", drop = FALSE])
neworder3$indices # 10 should be missing

mat[neworder2$indices, neworder2$indices]
cbind(neworder$names, neworder2$names)
```

---

split_groups *split groups*

Description

Given a matrix and a data.frame, character vector or data.frame of groups, splits the indices / names of the matrix into groups appropriately. This function assumes that the matrix and the groups are in the correct order!!

Usage

```
split_groups(in_matrix, groups = NULL)
```

Arguments

in_matrix	the matrix we want to split up
groups	a data.frame, character vector or factor

Value

list of groups

summarize_data *summarize data*

Description

summarizes a matrix or data.frame, where columns are samples and rows are features

Usage

```
summarize_data(
  in_data,
  sample_classes = NULL,
  avg_function = mean,
  log_transform = FALSE,
  remove_missing = NA
)
```

Arguments

in_data	matrix or data.frame
sample_classes	which samples are in which class
avg_function	which function to use for summary
log_transform	apply a log-transform to the mean
remove_missing	remove missing values before summarizing

Value

data.frame

visqc_cor_pca_scores *correlate scores and outcome*

Description

Given a matrix of PCA scores, set of sample attributes to test, goes through and performs an ICI-Kt of the scores versus the attribute.

Usage

```
visqc_cor_pca_scores(pca_scores, sample_info)
```

Arguments

pca_scores	the scores matrix to test
sample_info	data.frame of sample attributes to test Important: All of the attributes must be numeric, or character. If character, they will be transformed to a factor, and the numeric factor levels will be used instead. If missing values are present, that is OK, as long as they are missing-not-at-random (i.e. missing at the low end of the values).

Value

data.frame

---

visqc_heatmap *easier heatmaps*

Description

rolls some of the common Heatmap options into a single function call to make life easier when creating lots of heatmaps. **Note:** clustering of rows and columns is disabled, it is expected that you are reordering the matrix beforehand, or passing in column_order and row_order as arguments to be passed to Heatmap (see example). Matrices can be reordered using similarity_reorderbyclass, and nice class colors generated using generate_group_colors

Usage

```
visqc_heatmap(
  matrix_data,
  color_values,
  title = "",
  row_color_data = NULL,
  row_color_list = NULL,
  col_color_data = NULL,
  col_color_list = NULL,
  ...
)
```

Arguments

matrix_data	the matrix you want to plot as a heatmap
color_values	the color mapping of values to colors (see Details)
title	what do the values represent
row_color_data	data for row annotations
row_color_list	list for row annotations
col_color_data	data for column annotations
col_color_list	list for column annotations
...	other Heatmap parameters

Details

This function uses the ComplexHeatmap package to produce heatmaps with complex row- and column-color annotations. Both row_color_data and col_color_data should be data.frame's where each column describes meta-data about the rows or columns of the matrix. The row_color_list and col_color_list provide the mapping of color to annotation, where each list entry should be a named vector of colors, with the list entry corresponding to a column entry in the data.frame, and the names of the colors corresponding to annotations in that column.

Examples

```
## Not run:
library(circlize)
data(grp_cor)
data(grp_info)
colormap <- colorRamp2(c(0, 1), c("black", "white"))

annotation_color <- c(grp1 = "green", grp2 = "red", set1 = "blue",
                      set2 = "yellow")

row_data <- grp_info[, "grp", drop = FALSE]
col_data <- grp_info[, "set", drop = FALSE]
row_annotation = list(grp = annotation_color[1:2])
col_annotation = list(set = annotation_color[3:4])

visqc_heatmap(grp_cor, colormap, row_color_data = row_data, row_color_list = row_annotation,
                 col_color_data = col_data, col_color_list = col_annotation)

reorder_sim <- similarity_reorderbyclass(grp_cor, transform = "sub_1")
visqc_heatmap(grp_cor, colormap, "reorder1", row_data, row_annotation, col_data, col_annotation,
                 column_order = reorder_sim$indices, row_order = reorder_sim$indices)

sample_classes <- grp_info[, "grp", drop = FALSE]
reorder_sim2 <- similarity_reorderbyclass(grp_cor, sample_classes, "sub_1")
visqc_heatmap(grp_cor, colormap, "reorder2", row_data, row_annotation, col_data, col_annotation,
                 column_order = reorder_sim2$indices, row_order = reorder_sim2$indices)

## End(Not run)
```

---

visqc_score_contributions

calculate pca contributions

Description

Given a set of PCA scores, calculates their variance contributions, cumulative contributions, and generates a percent label that can be used for labeling plots.

Usage

```
visqc_score_contributions(pca_scores)
```

Arguments

pca_scores	matrix of scores, columns are each PC

Value

data.frame

`visqc_test_pca_loadings` *test loadings*

Description

Given a matrix of loadings for principal components, and a set of components to test, for each loading in each component, generates a null distribution from the other loadings in all the other components, and reports a p-value for that loading.

Usage

```
visqc_test_pca_loadings(
  loadings,
  test_columns,
  progress = FALSE,
  direction = FALSE
)
```

Arguments

loadings	matrix of loadings from pca decomposition
test_columns	names of the columns of the loadings to test
progress	should progress be reported
direction	should direction of loading be tested?

Value

named list

`visqc_test_pca_scores` *test scores and outcome*

Description

Given a matrix of PCA scores, set of sample attributes to test, goes through and performs an ANOVA of the scores versus the attribute.

Usage

```
visqc_test_pca_scores(pca_scores, sample_info)
```

Arguments

`pca_scores`	matrix of scores from a PCA decomposition
`sample_info`	data.frame of sample attributes to test

Value

data.frame

Index