Package: quollr (via r-universe)

September 18, 2024

Type Package

Title Visualising How Nonlinear Dimension Reduction Warps Your Data

Version 0.1.8

Description To construct a model in 2D space from 2D embedding data
and then lift it to the high-dimensional space. Additionally,
it provides tools to visualize the model in 2D space and to
overlay the fitted model on data using the tour technique.
Furthermore, it facilitates the generation of summaries of
high-dimensional distributions.

License MIT + file LICENSE

URL `https://github.com/JayaniLakshika/quollr`

BugReports `https://github.com/JayaniLakshika/quollr/issues`

Depends R (>= 3.5.0)

Imports dplyr, ggplot2, grid, interp (>= 1.1-6), langevitour, proxy,
rlang, rsample, stats, tibble, tidyselect

Suggests knitr, rmarkdown, testthat (>= 3.0.0), vdiffr, umap

VignetteBuilder knitr

Config/testthat/edition 3

Language en-GB

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

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

RemoteUrl https://github.com/jayanilakshika/quollr

RemoteRef HEAD

RemoteSha 14e3c59bcaf97406f3aa0951ed3f582362091bcc

Contents

`assign_data` *Assign data to hexagons*

Description

This function assigns the data to hexagons.

Usage

```
assign_data(data, centroid_df)
```

Arguments

`data`	data A tibble or data frame.
`centroid_df`	The dataset with centroid coordinates.

Value

A tibble contains embedding components and corresponding hexagon ID.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
all_centroids_df <- gen_centroids(bin1 = 4, r2 = r2, q = 0.1)
assign_data(data = s_curve_noise_umap_scaled, centroid_df = all_centroids_df)
```

`augment` *Augment Data with Predictions and Error Metrics*

Description

This function augments a dataset with predictions and error metrics obtained from a nonlinear dimension reduction (NLDR) model.

Usage

```
augment(
  df_bin_centroids,
  df_bin,
  training_data,
  newdata = NULL,
  type_NLDR,
  col_start = "x"
)
```

Arguments

`df_bin_centroids`	Centroid coordinates of hexagonal bins in 2D space.
`df_bin`	Centroid coordinates of hexagonal bins in high dimensions.
`training_data`	Training data used to fit the model.
`newdata`	Data to be augmented with predictions and error metrics. If NULL, the training data is used (default is NULL).
`type_NLDR`	The type of non-linear dimensionality reduction (NLDR) used.
`col_start`	The text that begin the column name of the high-dimensional data.

Value

A tibble containing the augmented data with predictions, error metrics, and absolute error metrics.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
model <- fit_highd_model(training_data = s_curve_noise_training,
emb_df = s_curve_noise_umap_scaled, bin1 = 4, r2 = r2, col_start_highd = "x")
df_bin_centroids <- model$df_bin_centroids
df_bin <- model$df_bin
augment(df_bin_centroids = df_bin_centroids, df_bin = df_bin,
training_data = s_curve_noise_training, newdata = NULL, type_NLDR = "UMAP",
col_start = "x")
```

avg_highd_data *Create a dataframe with averaged high-dimensional data*

Description

This function calculates the average values of high-dimensional data within each hexagonal bin.

Usage

```
avg_highd_data(data, col_start = "x")
```

Arguments

`data`	A tibble that contains the high-dimensional data and embedding with hexagonal bin IDs.
`col_start`	The text that begin the column name of the high-dimensional data

Value

A tibble with the average values of the high-dimensional data within each hexagonal bin.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
umap_data_with_hb_id <- hb_obj$data_hb_id
df_all <- dplyr::bind_cols(s_curve_noise_training, umap_data_with_hb_id)
avg_highd_data(data = df_all, col_start = "x")
```

---

calc_bins_y *Calculate the effective number of bins along x-axis and y-axis*

Description

This function calculates the effective number of bins along the x and y axes of a hexagonal grid.

Usage

```
calc_bins_y(bin1 = 4, r2, q = 0.1)
```

Arguments

bin1	Number of bins along the x axis.
r2	The ratio of the ranges of the original embedding components.
q	The buffer amount as proportion of data range.

Value

A list of numeric values that represents the effective number of bins along the y axis and width of the hexagon.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
calc_bins_y(bin1 = 4, r2 = r2, q = 0.1)
```

`cal_2d_dist` *Calculate 2D Euclidean distances between vertices*

Description

This function calculates the 2D distances between pairs of points in a data frame.

Usage

```
cal_2d_dist(tr_coord_df, start_x, start_y, end_x, end_y, select_vars)
```

Arguments

`tr_coord_df`	A tibble that contains the x and y coordinates of start and end points.
`start_x`	Column name for the x-coordinate of the starting point.
`start_y`	Column name for the y-coordinate of the starting point.
`end_x`	Column name for the x-coordinate of the ending point.
`end_y`	Column name for the y-coordinate of the ending point.
`select_vars`	A character vector specifying the columns to be selected in the resulting data frame.

Value

A tibble with columns for the starting point, ending point, and calculated distances.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
tr1_object <- tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
tr_from_to_df <- gen_edges(tri_object = tr1_object)
cal_2d_dist(tr_coord_df = tr_from_to_df, start_x = "x_from", start_y = "y_from",
end_x = "x_to", end_y = "y_to", select_vars = c("from", "to", "distance"))
```

compute_mean_density_hex *Compute mean density of hexagonal bins*

Description

This function calculates the mean density of hexagonal bins based on their neighboring bins.

Usage

```
compute_mean_density_hex(df_bin_centroids, bin1)
```

Arguments

`df_bin_centroids`	A tibble that contains information about hexagonal bin centroids, including the hexagon ID and the standard normalized counts (`std_counts`).
`bin1`	The number of bins along the x-axis for the hexagonal grid.

Value

A tibble contains hexagonal IDs and the mean density of each hexagonal bin based on its neighboring bins.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
compute_mean_density_hex(df_bin_centroids, bin1 = num_bins_x)
```

compute_std_counts *Compute standardize counts in hexagons*

Description

This function computes the standardize number of points within each hexagon.

Usage

```
compute_std_counts(data_hb)
```

Arguments

`data_hb`	A tibble with embedding and hexagonal bin IDs.

Value

A tibble that contains hexagon IDs and the corresponding standardize counts.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
all_centroids_df <- gen_centroids(bin1 = 4, r2 = r2, q = 0.1)
umap_with_hb_id <- assign_data(data = s_curve_noise_umap_scaled,
centroid_df = all_centroids_df)
compute_std_counts(data_hb = umap_with_hb_id)
```

---

extract_hexbin_centroids

Extract hexagonal bin centroids coordinates and the corresponding standardise counts.

Description

Extract hexagonal bin centroids coordinates and the corresponding standardise counts.

Usage

```
extract_hexbin_centroids(centroids_df, counts_df)
```

Arguments

`centroids_df`	A tibble that contains all hexagonal bin centroid coordinates with hexagon IDs.
`counts_df`	A tibble that contains hexagon IDs with the standardise number of points within each hexagon.

Value

A tibble contains hexagon ID, centroid coordinates, and standardise counts.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2, q = 0.1)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df)
```

`extract_hexbin_mean` *Extract hexagonal bin mean coordinates and the corresponding standardize counts.*

Description

Extract hexagonal bin mean coordinates and the corresponding standardize counts.

Usage

```
extract_hexbin_mean(data_hb, counts_df, centroids_df)
```

Arguments

`data_hb`	A tibble with embedding components and hexagonal bin IDs.
`counts_df`	A tibble that contains hexagon IDs with the standardise number of points within each hexagon.
`centroids_df`	A tibble that contains all hexagonal bin centroid coordinates with hexagon IDs.

Value

A tibble contains hexagon ID, bin mean coordinates, and standardize counts.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2, q = 0.1)
all_centroids_df <- hb_obj$centroids
umap_with_hb_id <- hb_obj$data_hb_id
counts_df <- hb_obj$std_cts
extract_hexbin_mean(data_hb = umap_with_hb_id, counts_df = counts_df,
centroids_df = all_centroids_df)
```

`find_lg_benchmark` *Compute a benchmark value to remove long edges*

Description

This function finds the benchmark value to remove long edges based on the differences in a distance column.

Usage

```
find_lg_benchmark(distance_edges, distance_col)
```

Arguments

`distance_edges`	The tibble contains the distances.
`distance_col`	The name of the column containing the distances.

Value

The benchmark value, which is the first largest difference in the distance column.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
tr1_object <- tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
tr_from_to_df <- gen_edges(tri_object = tr1_object)
distance_df <- cal_2d_dist(tr_coord_df = tr_from_to_df, start_x = "x_from",
start_y = "y_from", end_x = "x_to", end_y = "y_to",
select_vars = c("from", "to", "distance"))
find_lg_benchmark(distance_edges = distance_df, distance_col = "distance")
```

`find_low_dens_hex` *Find low-density Hexagons*

Description

This function identifies hexagons with low density based on the mean density of their neighboring hexagons.

Usage

```
find_low_dens_hex(df_bin_centroids_all, bin1, df_bin_centroids_low)
```

Arguments

`df_bin_centroids_all`	The tibble that contains all hexagonal bin centroids.
`bin1`	Number of bins along the x-axis for hexagon binning.
`df_bin_centroids_low`	The tibble that contains identified low-density hexagonal bin centroids.

Value

A vector containing the IDs of hexagons to be removed after investigating their neighboring bins.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
df_bin_centroids_low <- df_bin_centroids |>
dplyr::filter(std_counts <= 0.43)
find_low_dens_hex(df_bin_centroids_all = df_bin_centroids, bin1 = num_bins_x,
df_bin_centroids_low = df_bin_centroids_low)
```

`find_non_empty_bins` *Find the number of bins required to achieve required number of non-empty bins.*

Description

This function determines the number of bins along the x and y axes to obtain a specific number of non-empty bins.

Usage

```
find_non_empty_bins(data, non_empty_bins, r2, q = 0.1)
```

Arguments

`data`	A tibble that contains the embedding.
`non_empty_bins`	The desired number of non-empty bins.
`r2`	The range of the original second embedding component.
`q`	The buffer amount as proportion of data range.

Value

The number of bins along the x and y axes needed to achieve a specific number of non-empty bins.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
find_non_empty_bins(data = s_curve_noise_umap_scaled, non_empty_bins = 5,
r2 = r2)
```

`find_pts` *Find points in hexagonal bins*

Description

This function maps points to their corresponding hexagonal bins.

Usage

```
find_pts(data_hb)
```

Arguments

`data_hb`	A data frame with data, ID and hexagonal bin IDs.

Value

A tibble with hexagonal bin IDs and the corresponding points.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
all_centroids_df <- gen_centroids(bin1 = 4, r2 = r2, q = 0.1)
umap_with_hb_id <- assign_data(data = s_curve_noise_umap_scaled,
centroid_df = all_centroids_df)
find_pts(data_hb = umap_with_hb_id)
```

`fit_highd_model` *Construct the 2D model and lift into high-D*

Description

This function fits a high-dimensional model using hexagonal bins and provides options to customize the modeling process, including the choice of bin centroids or bin means, removal of low-density hexagons, and averaging of high-dimensional data.

Usage

```
fit_highd_model(
  training_data,
  emb_df,
  bin1 = 4,
  r2,
  q = 0.1,
  is_bin_centroid = TRUE,
  is_rm_lwd_hex = FALSE,
  benchmark_to_rm_lwd_hex = NULL,
  col_start_highd = "x"
)
```

Arguments

`training_data`	A tibble that contains the training high-dimensional data.
`emb_df`	A tibble that contains embedding with a unique identifier.
`bin1`	Number of bins along the x axis.
`r2`	The ratio of the ranges of the original embedding components.
`q`	The buffer amount as proportion of data range.
`is_bin_centroid`	Logical, indicating whether to use bin centroids (default is TRUE).
`is_rm_lwd_hex`	Logical, indicating whether to remove low-density hexagons (default is FALSE).

`benchmark_to_rm_lwd_hex`	The benchmark value to remove low-density hexagons.
`col_start_highd`	The text prefix for columns in the high-dimensional data.

Value

A list containing the data frame with high-dimensional coordinates for 2D bin centroids (`df_bin`) and the data frame containing information about hexagonal bin centroids (`df_bin_centroids`) in 2D.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
fit_highd_model(training_data = s_curve_noise_training,
emb_df = s_curve_noise_umap_scaled, bin1 = 4, r2 = r2,
col_start_highd = "x")
```

`gen_centroids` *Generate centroid coordinate*

Description

This function generates all possible centroids in the hexagonal grid.

Usage

```
gen_centroids(bin1 = 2, r2, q = 0.1)
```

Arguments

`bin1`	Number of bins along the x axis.
`r2`	The ratio of the ranges of the original embedding components.
`q`	The buffer amount as proportion of data range.

Value

A tibble contains hexIDs, x and y coordinates (hexID, c_x, c_y respectively) of all hexagon bin centroids.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
gen_centroids(bin1 = 4, r2 = r2, q = 0.1)
```

gen_edges *Generate edge information*

Description

This function generates edge information from a given triangular object, including the coordinates of the vertices and the from-to relationships between the vertices.

Usage

```
gen_edges(tri_object)
```

Arguments

`tri_object`	The triangular object from which to generate edge information.

Value

A tibble that contains the edge information, including the from-to relationships and the corresponding x and y coordinates.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
tr1_object <- tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
gen_edges(tri_object = tr1_object)
```

gen_hex_coord *Generate hexagonal polygon coordinates*

Description

This function generates the coordinates of hexagons after passing hexagonal centroids.

Usage

```
gen_hex_coord(centroids_df, a1)
```

Arguments

`centroids_df`	The dataset with all hexagonal bin IDs and centroid coordinates.
`a1`	The width of the hexagon.

Value

A tibble contains polygon id, x and y coordinates (hex_poly_id, x, and y respectively) of hexagons.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_list <- calc_bins_y(bin1 = 4, r2 = r2, q = 0.1)
width <- num_bins_list$a1
all_centroids_df <- gen_centroids(bin1 = 4, r2 = r2, q = 0.1)
gen_hex_coord(centroids_df = all_centroids_df, a1 = width)
```

`gen_proj_langevitour` *Visualize a specific projection of langevitour*

Description

This function visualize a specific projection of langevitour.

Usage

```
gen_proj_langevitour(points_df, projection, edge_df)
```

Arguments

`points_df`	The tibble that contains the model and data.
`projection`	The tibble of the projection.
`edge_df`	The tibble that contains the edge information (from, to).

Value

A ggplot object with the specific projection of langevitour.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
```

```
dplyr::filter(drop_empty == FALSE)
tr1_object <- tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
tr_from_to_df <- gen_edges(tri_object = tr1_object)
distance_df <- cal_2d_dist(tr_coord_df = tr_from_to_df, start_x = "x_from",
start_y = "y_from", end_x = "x_to", end_y = "y_to",
select_vars = c("from", "to", "distance"))
umap_data_with_hb_id <- hb_obj$data_hb_id
df_all <- dplyr::bind_cols(s_curve_noise_training |> dplyr::select(-ID),
umap_data_with_hb_id)
df_bin <- avg_highd_data(data = df_all, col_start = "x")
### Define type column
df <- df_all |>
  dplyr::select(tidyselect::starts_with("x")) |>
  dplyr::mutate(type = "data") ## original dataset

df_b <- df_bin |>
  dplyr::filter(hb_id %in% df_bin_centroids$hexID) |>
  dplyr::mutate(type = "model") ## Data with summarized mean

## Reorder the rows of df_b according to the hexID order in df_b_with_center_data
df_b <- df_b[match(df_bin_centroids$hexID, df_b$hb_id),] |>
  dplyr::select(-hb_id)

df_exe <- dplyr::bind_rows(df_b, df)
benchmark <- 0.663

## Set the maximum difference as the criteria
distance_df_small_edges <- distance_df |>
  dplyr::filter(distance < benchmark) |>
  dplyr::select(-distance)

projection_df <- cbind(
c(-0.17353,-0.02906,0.19857,0.00037,0.00131,-0.05019,0.03371),
c(-0.10551,0.14829,-0.02063,0.02658,-0.03150,0.19698,0.00044))


gen_proj_langevitour(points_df = df_exe, projection = projection_df,
edge_df = distance_df_small_edges)
```

---

gen_scaled_data *Scaling the NLDR data*

Description

This function scales first and second columns.

Usage

```
gen_scaled_data(data)
```

Arguments

`data`	A tibble that contains embedding components in the first and second columns.

Value

A list of a tibble contains scaled first and second columns, and numeric vectors representing the limits of the original data.

Examples

```
gen_scaled_data(data = s_curve_noise_umap)
```

`GeomHexgrid` *GeomHexgrid: A Custom ggplot2 Geom for Hexagonal Grid*

Description

This function defines a custom ggplot2 Geom, GeomHexgrid, for rendering hexagonal grid.

Usage

```
GeomHexgrid
```

Format

A ggproto object

`GeomTrimesh` *GeomTrimesh: A Custom ggplot2 Geom for Triangular Meshes*

Description

This function defines a custom ggplot2 Geom, GeomTrimesh, for rendering triangular meshes.

Usage

```
GeomTrimesh
```

Format

A ggproto object

Details

- `required_aes`: The required aesthetics for this geometry are `"x"`, `"y"`, `"xend"`, and `"yend"`. - `default_aes`: The default aesthetics for this geometry include `shape = 19`, `linetype = 1`, `linewidth = 0.5`, `size = 0.5`, `alpha = NA`, and `colour = "black"`. - `draw_key`: The function describing how to draw the key glyph is `ggplot2::draw_key_point`. - `draw_panel`: The function describing how to draw the panel takes `data`, `panel_scales`, and `coord`. It creates a tibble of `vertices` and a tibble of `trimesh`. The final plot is constructed using `ggplot2::GeomPoint$draw_panel` for vertices and `ggplot2::GeomSegment$draw_panel` for trimesh.

`geom_hexgrid` *Create a hexgrid plot*

Description

Create a hexgrid plot

Usage

```
geom_hexgrid(
  mapping = NULL,
  data = NULL,
  stat = "hexgrid",
  position = "identity",
  show.legend = NA,
  na.rm = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Aesthetic mappings for the plot.
`data`	The data to be plotted.
`stat`	The statistical transformation to be applied.
`position`	The position adjustment to be applied.
`show.legend`	Whether to show the legend for this layer.
`na.rm`	Whether to remove missing values.
`inherit.aes`	Whether to inherit aesthetics from the plot or the layer.
`...`	Additional arguments to be passed to the ‘layer‘ function.

Value

A ‘ggplot2‘ layer object.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
ggplot2::ggplot() +
geom_hexgrid(data = all_centroids_df, mapping = ggplot2::aes(x = c_x, y = c_y))
```

`geom_trimesh` *Create a trimesh plot*

Description

Create a trimesh plot

Usage

```
geom_trimesh(
  mapping = NULL,
  data = NULL,
  stat = "trimesh",
  position = "identity",
  show.legend = NA,
  na.rm = FALSE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Aesthetic mappings for the plot.
`data`	The data to be plotted.
`stat`	The statistical transformation to be applied.
`position`	The position adjustment to be applied.
`show.legend`	Whether to show the legend for this layer.
`na.rm`	Whether to remove missing values.
`inherit.aes`	Whether to inherit aesthetics from the plot or the layer.
`...`	Additional arguments to be passed to the ‘layer‘ function.

Value

A ‘ggplot2‘ layer object.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
ggplot2::ggplot() +
geom_trimesh(data = df_bin_centroids, mapping = ggplot2::aes(x = c_x, y = c_y))
```

`get_min_indices` *Get indices of all minimum distances*

Description

This function returns the indices of all minimum distances.

Usage

```
get_min_indices(x)
```

Arguments

`x`	A numeric vector.

Value

A numeric vector containing the indices of all minimum distances.

Examples

```
x <- c(1, 2, 1, 3)
get_min_indices(x)
```

`glance` *Generate evaluation metrics*

Description

This function generates an evaluation data frame based on the provided data and predictions.

Usage

```
glance(
  df_bin_centroids,
  df_bin,
  training_data,
  newdata = NULL,
  type_NLDR,
  col_start = "x"
)
```

Arguments

`df_bin_centroids`	Centroid coordinates of hexagonal bins in 2D space.
`df_bin`	Centroid coordinates of hexagonal bins in high dimensions.
`training_data`	Training data used to fit the model.
`newdata`	Data to be augmented with predictions and error metrics. If NULL, the training data is used (default is NULL).
`type_NLDR`	The type of non-linear dimensionality reduction (NLDR) used.
`col_start`	The text that begin the column name of the high-dimensional data.

Value

A tibble contains Error, and MSE values.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
model <- fit_highd_model(training_data = s_curve_noise_training,
emb_df = s_curve_noise_umap_scaled, bin1 = 4, r2 = r2, col_start_highd = "x")
df_bin_centroids <- model$df_bin_centroids
df_bin <- model$df_bin
glance(df_bin_centroids = df_bin_centroids, df_bin = df_bin,
training_data = s_curve_noise_training, newdata = NULL, type_NLDR = "UMAP",
col_start = "x")
```

`hex_binning` *Hexagonal binning*

Description

This function generates the hexagonal object.

Usage

```
hex_binning(data, bin1 = 4, r2, q = 0.1)
```

Arguments

`data`	A tibble that contains embedding components.
`bin1`	Number of bins along the x axis.
`r2`	The ratio of the ranges of the original embedding components.
`q`	The buffer amount as proportion of data range.

Value

A object that contains numeric vector that contains binwidths (a1), vertical distance (a2), bins along the x and y axes respectively (bins), numeric vector that contains hexagonal starting point coordinates all hexagonal bin centroids (centroids), hexagonal coordinates of the full grid(hex_poly), embedding components with their corresponding hexagon IDs (data_hb_id), hex bins with their corresponding standardise counts (std_cts), total number of hex bins(tot_bins), number of non-empty hex bins (non_bins) and points within each hexagon (pts_bins).

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2, q = 0.1)
```

`predict_emb` *Predict 2D embeddings*

Description

Given a test dataset, the centroid coordinates of hexagonal bins in 2D and high-dimensional space, predict the 2D embeddings for each data point in the test dataset.

Usage

```
predict_emb(test_data, df_bin_centroids, df_bin, type_NLDR)
```

Arguments

`test_data`	The test dataset containing high-dimensional coordinates and an unique identifier.
`df_bin_centroids`	Centroid coordinates of hexagonal bins in 2D space.
`df_bin`	Centroid coordinates of hexagonal bins in high dimensions.
`type_NLDR`	The type of non-linear dimensionality reduction (NLDR) used.

Value

A tibble contains predicted 2D embeddings, ID in the test data, and predicted hexagonal IDs.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
model <- fit_highd_model(training_data = s_curve_noise_training,
emb_df = s_curve_noise_umap_scaled, bin1 = 4, r2 = r2, col_start_highd = "x")
df_bin_centroids <- model$df_bin_centroids
df_bin <- model$df_bin
predict_emb(test_data = s_curve_noise_training, df_bin_centroids = df_bin_centroids,
df_bin = df_bin, type_NLDR = "UMAP")
```

`show_langevitour` *Visualize the model overlaid on high-dimensional data*

Description

This function generates a LangeviTour visualization based on different conditions and input parameters.

Usage

```
show_langevitour(
  df,
  df_b,
  df_b_with_center_data,
  benchmark_value,
  distance_df,
  distance_col,
  use_default_benchmark_val = FALSE,
  col_start
)
```

Arguments

`df`	A tibble that contains the high-dimensional data.
`df_b`	A tibble that contains the high-dimensional coordinates of bin centroids.
`df_b_with_center_data`	The dataset with hexagonal bin centroids.
`benchmark_value`	The benchmark value used to remove long edges (optional).
`distance_df`	The tibble with distance.
`distance_col`	The name of the distance column.
`use_default_benchmark_val`	Logical, indicating whether to use default benchmark value to remove long edges (default is FALSE).
`col_start`	The text that begin the column name of the high-dimensional data.

Value

A langevitour object with the model and the high-dimensional data.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
tr1_object <- tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
tr_from_to_df <- gen_edges(tri_object = tr1_object)
distance_df <- cal_2d_dist(tr_coord_df = tr_from_to_df, start_x = "x_from",
start_y = "y_from", end_x = "x_to", end_y = "y_to",
select_vars = c("from", "to", "distance"))
umap_data_with_hb_id <- hb_obj$data_hb_id
df_all <- dplyr::bind_cols(s_curve_noise_training |> dplyr::select(-ID),
umap_data_with_hb_id)
df_bin <- avg_highd_data(data = df_all, col_start = "x")
show_langevitour(df = df_all, df_b = df_bin, df_b_with_center_data = df_bin_centroids,
benchmark_value = 1.16, distance = distance_df, distance_col = "distance",
use_default_benchmark_val = FALSE, col_start = "x")
```

stat_hexgrid *stat_hexgrid Custom Stat for hexagonal grid plot*

Description

stat_hexgrid Custom Stat for hexagonal grid plot

Usage

```
stat_hexgrid(
  mapping = NULL,
  data = NULL,
  geom = GeomHexgrid$default_aes(),
  position = "identity",
  show.legend = NA,
  outliers = TRUE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Aesthetic mappings for the plot.
`data`	The data to be plotted.
`geom`	The geometry to be used in the plot.
`position`	The position adjustment to be applied.
`show.legend`	Whether to show the legend for this layer.
`outliers`	Whether to include outliers.
`inherit.aes`	Whether to inherit aesthetics from the plot or the layer.
`...`	Additional arguments to be passed to the ‘layer‘ function.

Value

A ‘ggplot2‘ layer object.

`stat_trimesh` *stat_trimesh Custom Stat for trimesh plot*

Description

stat_trimesh Custom Stat for trimesh plot

Usage

```
stat_trimesh(
  mapping = NULL,
  data = NULL,
  geom = GeomTrimesh$default_aes(),
  position = "identity",
  show.legend = NA,
  outliers = TRUE,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Aesthetic mappings for the plot.
`data`	The data to be plotted.
`geom`	The geometry to be used in the plot.
`position`	The position adjustment to be applied.
`show.legend`	Whether to show the legend for this layer.
`outliers`	Whether to include outliers.
`inherit.aes`	Whether to inherit aesthetics from the plot or the layer.
`...`	Additional arguments to be passed to the ‘layer‘ function.

Value

A ‘ggplot2‘ layer object.

`s_curve_noise` *S-curve dataset with noise dimensions*

Description

The ‘s_curve_noise‘ dataset contains a 3-dimensional S-curve with added noise dimensions. Each data point is represented by seven dimensions (x1 to x7) and an ID.

Usage

```
data(s_curve_noise)
```

Format

A data frame with 100 rows and 8 columns:

ID Identification number

x1, x2, x3, x4, x5, x6, x7 High-dimensional coordinates

Source

This dataset is generated for illustrative purposes.

Examples

```
# Load the s_curve_noise dataset
data(s_curve_noise)

# Display the first few rows of the dataset
head(s_curve_noise)
```

`s_curve_noise_test` *S-curve dataset with noise dimensions for test*

Description

The ‘s_curve_noise_test‘ dataset contains test data with dimensions x1, x2, x3, x4, x5, x6, and x7. Each data point is identified by an ID.

Usage

```
data(s_curve_noise_test)
```

Format

A data frame with 25 rows and 8 columns:

ID Identification number

x1, x2, x3, x4, x5, x6, x7 High-dimensional coordinates

Source

This dataset is generated for training purposes.

Examples

```
# Load the s_curve_noise_test dataset
data(s_curve_noise_test)

# Display the first few rows of the dataset
head(s_curve_noise_test)
```

---

s_curve_noise_training *S-curve dataset with noise dimensions for training*

Description

The ‘s_curve_noise_training‘ dataset contains training data with dimensions x1, x2, x3, x4, x5, x6, and x7. Each data point is identified by an ID.

Usage

```
data(s_curve_noise_training)
```

Format

A data frame with 75 rows and 8 columns:

ID Identification number

x1, x2, x3, x4, x5, x6, x7 High-dimensional coordinates

Source

This dataset is generated for training purposes.

Examples

```
# Load the s_curve_noise_training dataset
data(s_curve_noise_training)

# Display the first few rows of the dataset
head(s_curve_noise_training)
```

---

s_curve_noise_umap *UMAP embedding for S-curve dataset which with noise dimensions*

Description

The ‘s_curve_noise_umap‘ dataset contains the UMAP (Uniform Manifold Approximation and Projection) embeddings of a three-dimensional S-curve with added noise. Each data point is represented by two UMAP coordinates (UMAP1 and UMAP2) and an ID.

Usage

```
data(s_curve_noise_umap)
```

Format

## ‘s_curve_noise_umap‘ A data frame with 75 rows and 3 columns:

UMAP1 Numeric, first UMAP 2D embeddings.

UMAP2 Numeric, second UMAP 2D embeddings.

ID Numeric, identifier for each data point.

Source

This dataset is generated for illustrative purposes.

Examples

```
# Load the s_curve_noise_umap dataset
data(s_curve_noise_umap)

# Display the first few rows of the dataset
head(s_curve_noise_umap)
```

## `s_curve_noise_umap_predict` *Predicted UMAP embedding for S-curve dataset which with noise dimensions*

Description

The ‘s_curve_noise_umap_predict‘ dataset contains the predicted UMAP (Uniform Manifold Approximation and Projection) embeddings of a three-dimensional S-curve with added noise. Each data point is represented by two UMAP coordinates (UMAP1 and UMAP2) and an ID.

Usage

```
data(s_curve_noise_umap_predict)
```

Format

## ‘s_curve_noise_umap_predict‘ A data frame with 75 rows and 3 columns:

UMAP1 Numeric, predicted first UMAP 2D embeddings.

UMAP2 Numeric, predicted second UMAP 2D embeddings.

ID Numeric, identifier for each data point.

Source

This dataset is generated for illustrative purposes.

Examples

```
# Load the s_curve_noise_umap_predict dataset
data(s_curve_noise_umap_predict)

# Display the first few rows of the dataset
head(s_curve_noise_umap_predict)
```

`s_curve_noise_umap_scaled` *Scaled UMAP embedding for S-curve dataset which with noise dimensions*

Description

The ‘s_curve_noise_umap_scaled‘ dataset contains the scaled UMAP (Uniform Manifold Approximation and Projection) embeddings.

Usage

```
data(s_curve_noise_umap_scaled)
```

Format

## ‘s_curve_noise_umap_scaled‘ A data frame with 25 rows and 3 columns:

UMAP1 Numeric, Scaled first UMAP 2D embeddings.

UMAP2 Numeric, Scaled second UMAP 2D embedding.

ID Numeric, identifier for each data point.

Source

This dataset is generated for illustrative purposes.

Examples

```
# Load the s_curve_noise_umap_scaled dataset
data(s_curve_noise_umap_scaled)

# Display the first few rows of the dataset
head(s_curve_noise_umap_scaled)
```

`tri_bin_centroids` *Triangulate bin centroids*

Description

This function triangulates the bin centroids using the x and y coordinates provided in the input data frame and returns the triangular object.

Usage

```
tri_bin_centroids(hex_df, x, y)
```

Arguments

`hex_df`	The tibble containing the bin centroids.
`x`	The name of the column that contains x coordinates of bin centroids.
`y`	The name of the column that contains y coordinates of bin centroids.

Value

A triangular object representing the triangulated bin centroids.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df)
tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
```

`vis_lg_mesh` *Visualize triangular mesh with coloured long edges*

Description

This function visualize triangular mesh with coloured long edges.

Usage

```
vis_lg_mesh(distance_edges, benchmark_value, tr_coord_df, distance_col)
```

Arguments

`distance_edges`	The tibble that contains the edge information.
`benchmark_value`	The threshold value to determine long edges.
`tr_coord_df`	A tibble that contains the x and y coordinates of start and end points.
`distance_col`	The column name in ‘distance_edges‘ representing the distances.

Value

A ggplot object with the triangular mesh plot where long edges are coloured differently.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
tr1_object <- tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
tr_from_to_df <- gen_edges(tri_object = tr1_object)
```

```
distance_df <- cal_2d_dist(tr_coord_df = tr_from_to_df, start_x = "x_from",
start_y = "y_from", end_x = "x_to", end_y = "y_to",
select_vars = c("from", "to", "distance"))
vis_lg_mesh(distance_edges = distance_df, benchmark_value = 0.75,
tr_coord_df = tr_from_to_df, distance_col = "distance")
```

vis_rmlg_mesh *Visualize triangular mesh after removing the long edges*

Description

This function visualize the triangular mesh after removing the long edges.

Usage

```
vis_rmlg_mesh(distance_edges, benchmark_value, tr_coord_df, distance_col)
```

Arguments

distance_edges	The tibble that contains the edge information.
benchmark_value	The threshold value to determine long edges.
tr_coord_df	A tibble that contains the x and y coordinates of start and end points.
distance_col	The column name in ‘distance_edges‘ representing the distances.

Value

A ggplot object with the triangular mesh plot where long edges are removed.

Examples

```
r2 <- diff(range(s_curve_noise_umap$UMAP2))/diff(range(s_curve_noise_umap$UMAP1))
num_bins_x <- 4
hb_obj <- hex_binning(data = s_curve_noise_umap_scaled, bin1 = num_bins_x,
r2 = r2)
all_centroids_df <- hb_obj$centroids
counts_df <- hb_obj$std_cts
df_bin_centroids <- extract_hexbin_centroids(centroids_df = all_centroids_df,
counts_df = counts_df) |>
dplyr::filter(drop_empty == FALSE)
tr1_object <- tri_bin_centroids(hex_df = df_bin_centroids, x = "c_x", y = "c_y")
tr_from_to_df <- gen_edges(tri_object = tr1_object)
distance_df <- cal_2d_dist(tr_coord_df = tr_from_to_df, start_x = "x_from",
start_y = "y_from", end_x = "x_to", end_y = "y_to",
select_vars = c("from", "to", "distance"))
vis_rmlg_mesh(distance_edges = distance_df, benchmark_value = 0.75,
tr_coord_df = tr_from_to_df, distance_col = "distance")
```

Index