Package: slanter (via r-universe)

September 2, 2024

Version 0.2-0

Date 2021-05-09

Title Slanted Matrices and Ordered Clustering

Description Slanted matrices and ordered clustering for better visualization of similarity data.

RoxygenNote 7.0.2

Encoding UTF-8

License MIT + file LICENSE

Imports Matrix, pheatmap, pracma, stats

LazyData true

Suggests knitr, rmarkdown

VignetteBuilder knitr

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

RemoteUrl https://github.com/tanaylab/slanter

RemoteRef HEAD

RemoteSha d4324044093651a8613200c9f1a46dd9ead840e1

Contents

meristems — *Sample RNA data of similarity between batches of 1000 cells of tomato meristem cells.*

Description

This is a simple matrix where each entry is the similarity (correlation) between a pair of batches. Negative correlations were changed to zero to simplify the analysis.

Usage

```
data(meristems)
```

Format

A simple square matrix.

Examples

```
data(meristems)
similarity <- meristems
similarity[similarity < 0] = 0
slanter::sheatmap(meristems, order_data=similarity, show_rownames=FALSE, show_colnames=FALSE)
```

oclust — *Hierarchically cluster ordered data.*

Description

Given a distance matrix for sorted objects, compute a hierarchical clustering preserving this order. That is, this is similar to `hclust` with the constraint that the result's order is always `1:N`.

Usage

```
oclust(distances, method = "ward.D2", order = NULL, members = NULL)
```

Arguments

`distances`	A distances object (as created by `stats::dist`).
`method`	The clustering method to use (only `ward.D` and `ward.D2` are supported).
`order`	If specified, assume the data will be re-ordered by this order.
`members`	Optionally, the number of members for each row/column of the distances (by default, one each).

Details

If an `order` is specified, assumes that the data will be re-ordered by this order. That is, the indices in the returned `hclust` object will refer to the post-reorder data locations, **not** to the current data locations.

This can be applied to the results of `slanted_reorder`, to give a "plausible" clustering for the data.

Value

A clustering object (as created by `hclust`).

Examples

```
clusters <- slanter::oclust(dist(mtcars), order=1:dim(mtcars)[1])
clusters$order
```

`reorder_frame` *Reorder the rows of a frame.*

Description

You'd expect `data[order,]` to "just work". It doesn't for data frames with a single column, which happens for annotation data, hence the need for this function. Sigh.

Usage

```
reorder_frame(frame, order)
```

Arguments

`frame`	A data frame to reorder the rows of.
`order`	An array containing indices permutation to apply to the rows.

Value

The data frame with the new row orders.

Examples

```
df <- data.frame(foo=c(1, 2, 3))
df[c(1,3,2),]
slanter::reorder_frame(df, c(1,3,2))
```

`reorder_hclust` *Given a clustering of some data, and some ideal order we'd like to use to visualize it, reorder (but do not modify) the clustering to be as consistent as possible with this ideal order.*

Description

Given a clustering of some data, and some ideal order we'd like to use to visualize it, reorder (but do not modify) the clustering to be as consistent as possible with this ideal order.

Usage

```
reorder_hclust(clusters, order)
```

Arguments

`clusters`	The existing clustering of the data.
`order`	The ideal order we'd like to see the data in.

Value

A reordered clustering which is consistent, wherever possible, the ideal order.

Examples

```
clusters <- hclust(dist(mtcars))
clusters$order
clusters <- slanter::reorder_hclust(clusters, 1:length(clusters$order))
clusters$order
```

`sheatmap` *Plot a heatmap with values as close to the diagonal as possible.*

Description

Given a matrix expressing the cross-similarity between two (possibly different) sets of entities, this will reorder it to move the high values close to the diagonal, for a better visualization.

Usage

```
sheatmap(
  data,
  ...,
  order_data = NULL,
  annotation_col = NULL,
  annotation_row = NULL,
  order_rows = TRUE,
  order_cols = TRUE,
  squared_order = TRUE,
  same_order = FALSE,
  patch_cols_order = NULL,
  patch_rows_order = NULL,
  discount_outliers = TRUE,
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  oclust_rows = TRUE,
  oclust_cols = TRUE,
  clustering_distance_rows = "euclidian",
  clustering_distance_cols = "euclidian",
  clustering_method = "ward.D2",
  clustering_callback = NA
)
```

Arguments

data	A rectangular matrix to plot, of non-negative values (unless order_data is specified).
...	Additional flags to pass to pheatmap.
order_data	An optional matrix of non-negative values of the same size to use for computing the orders.
annotation_col	Optional data frame describing each column.
annotation_row	Optional data frame describing each row.
order_rows	Whether to reorder the rows. Otherwise, use the current order.
order_cols	Whether to reorder the columns. Otherwise, use the current order.
squared_order	Whether to reorder to minimize the l2 norm (otherwise minimizes the l1 norm).
same_order	Whether to apply the same order to both rows and columns (if reordering both). For a square matrix, may also contain 'row' or 'column' to force the order of one axis to apply to both.
patch_cols_order	Optional function that may be applied to the columns order, returning a better order.
patch_rows_order	Optional function that may be applied to the rows order, returning a better order.

`discount_outliers`
: Whether to do a final order phase discounting outlier values far from the diagonal.

`cluster_rows`
: Whether to cluster the rows, or the clustering to use.

`cluster_cols`
: Whether to cluster the columns, or the clustering to use.

`oclust_rows`
: Whether to use `oclust` instead of `hclust` for the rows (if clustering them).

`oclust_cols`
: Whether to use `oclust` instead of `hclust` for the columns (if clustering them).

`clustering_distance_rows`
: The default method for computing row distances (by default, `euclidian`).

`clustering_distance_cols`
: The default method for computing column distances (by default, `euclidian`).

`clustering_method`
: The default method to use for hierarchical clustering (by default, `ward.D2` and *not* `complete`).

`clustering_callback`
: Is not supported.

Details

If you have an a-priori order for the rows and/or columns, you can prevent reordering either or both by specifying `order_rows=FALSE` and/or `order_cols=FALSE`. Otherwise, `slanted_orders` is used to compute the "ideal" slanted order for the data.

By default, the rows and columns are ordered independently from each other. If the matrix is asymmetric but square (e.g., a matrix of weights of a directed graph such as a K-nearest-neighbors graph), then you can can specify `same_order=TRUE` to force both rows and columns to the same order. You can also specify `same_order='row'` to force the columns to use the same order as the rows, or `same_order='column'` to force the rows to use the same order as the columns.

You can also specify a `patch_cols_order` and/or a ‘`patch_rows_order`‘ function that takes the computed "ideal" order and returns a patched order. For example, this can be used to force special values (such as "outliers") to the side of the heatmap.

There are four options for controlling clustering:

* By default, `sheatmap` will generate a clustering tree using `oclust`, to generate the "best" clustering that is also compatible with the slanted order.

* Request that `sheatmap` will use the same `hclust` as `pheatmap` (e.g., `oclust_rows=FALSE`). In this case, the tree is reordered to be the "most compatible" with the target slanted order. That is, `sheatmap` will invoke `reorder_hclust` so that, for each node of the tree, the order of the two subtrees will be chosen to best match the target slanted order. The end result need not be identical to the slanted order, but is as close as possible given the `hclust` clustering tree.

* Specify an explicit clustering (e.g., `cluster_rows=hclust(...)`). In this case, `sheatmap` will again merely reorder the tree but will not modify it.

In addition, you can give this function any of the `pheatmap` flags, and it will just pass them on. This allows full control over the diagram’s features.

Note that `clustering_callback` is not supported. In addition, the default `clustering_method` here is `ward.D2` instead of `complete`, since the only methods supported by `oclust` are `ward.D` and `ward.D2`.

Value

Whatever pheatmap returns.

Examples

```
slanter::sheatmap(cor(t(mtcars)))
slanter::sheatmap(cor(t(mtcars)), oclust_rows=FALSE, oclust_cols=FALSE)
pheatmap::pheatmap(cor(t(mtcars)))
```

---

slanted_orders *Compute rows and columns orders which move high values close to the diagonal.*

Description

For a matrix expressing the cross-similarity between two (possibly different) sets of entities, this produces better results than clustering (e.g. as done by pheatmap). This is because clustering does not care about the order of each two sub-partitions. That is, clustering is as happy with `((2, 1), (4, 3))` as it is with the more sensible `((1, 2), (3, 4))`. As a result, visualizations of similarities using naive clustering can be misleading.

Usage

```
slanted_orders(
  data,
  order_rows = TRUE,
  order_cols = TRUE,
  squared_order = TRUE,
  same_order = FALSE,
  discount_outliers = TRUE,
  max_spin_count = 10
)
```

Arguments

data	A rectangular matrix containing non-negative values.
order_rows	Whether to reorder the rows.
order_cols	Whether to reorder the columns.
squared_order	Whether to reorder to minimize the l2 norm (otherwise minimizes the l1 norm).
same_order	Whether to apply the same order to both rows and columns.
discount_outliers	Whether to do a final order phase discounting outlier values far from the diagonal.
max_spin_count	How many times to retry improving the solution before giving up.

Value

A list with two keys, `rows` and `cols`, which contain the order.

Examples

```
slanter::slanted_orders(cor(t(mtcars)))
```

`slanted_reorder` *Reorder data rows and columns to move high values close to the diagonal.*

Description

Given a matrix expressing the cross-similarity between two (possibly different) sets of entities, this uses `slanted_orders` to compute the "best" order for visualizing the matrix, then returns the reordered data. Commonly used in `pheatmap(slanted_reorder(data), ...)`, and of course `sheatmap` does this internally for you.

Usage

```
slanted_reorder(
  data,
  order_data = NULL,
  order_rows = TRUE,
  order_cols = TRUE,
  squared_order = TRUE,
  same_order = FALSE,
  discount_outliers = TRUE
)
```

Arguments

`data`	A rectangular matrix to reorder, of non-negative values (unless `order_data` is specified).
`order_data`	An optional matrix of non-negative values of the same size to use for computing the orders.
`order_rows`	Whether to reorder the rows.
`order_cols`	Whether to reorder the columns.
`squared_order`	Whether to reorder to minimize the l2 norm (otherwise minimizes the l1 norm).
`same_order`	Whether to apply the same order to both rows and columns.
`discount_outliers`	Whether to do a final order phase discounting outlier values far from the diagonal.

Value

A matrix of the same shape whose rows and columns are a permutation of the input.

Examples

```
slanter::slanted_reorder(cor(t(mtcars)))
```

Index