aheatmap: a Powerful Annotated Heatmap Engine

Package *NMF* - Version 0.23.6

Renaud Gaujoux

March 28, 2020

Abstract

This vignette showcases the main features of the annotated heatmap engine implemented by the function `aheatmap`. This engine is a highly enhanced modification of the function `pheatmap` from the *pheatmap* package[1], and provides convenient and quick ways of producing high quality and customizable annotated heatmaps. Currently this function is part of the package *NMF*, but will most probably eventually constitute a separate package on its own.

Contents

1 Overview

The development of the function `aheatmap` started as modification of the function `pheatmap` from the *pheatmap* package[2]. The initial objective was to improve and increase its capabilities, as well as defining a simplified interface that was more consistent with the R core function `heatmap`. It is evolving into a general, flexible, powerful and easy to use engine for drawing annotated heatmaps.

The function `aheatmap` has many advantages compared to other heatmap functions such as `heatmap`, `gplots::heatmap2`, `heatmap.plus::heatmap.plus` , or `pheatmap`:

- Annotations: unlimited number of annotation tracks can be added to *both* columns and rows, with automated colouring for categorical and numeric variables.
- Compatibility with both base and grid graphics: the function can be directly called in drawing contexts such as grid, mfrow or layout. We believe that this is a feature many R users will enjoy, and that is strictly impossible with base heatmaps.
- Legends: default automatic legend and colouring;

[1] https://cran.r-project.org/package=pheatmap
[2] https://cran.r-project.org/package=pheatmap

- Customisation: clustering methods, annotations, colours and legend can all be customised, even separately for rows and columns;
- Convenient interface: many arguments provide multiple ways of specifying their value(s), which speeds up developing/writing and reduce the amount of code required to generate customised plots (e.g. see **??**).
- Aesthetics: the heatmaps look globally cleaner, the image and text components are by default well proportioned relatively to each other, and all fit within the graphic device – if not set to an unreasonably small size.

2 Preliminaries

2.1 Installation

The `aheatmap` function is currently part of the *NMF* package[3], which can be installed from any CRAN mirror or from the GitHub repository[4], for the development version, with the following commands:

```
# latest stable
install.packages('NMF')
# development version
devtools::install_github('NMF', 'renozao', 'devel')
```

2.2 Sample data

For the purpose of illustrating the capabilities of the function `aheatmap`, we first generate some random data that we will use throughout the vignette:

```
# data matrix
x <- rmatrix(20, 10, .rng = 1234)
ann_col <- list(Groups = gl(2, 5))
```

3 Components

Annotated heatmaps essentially use `grid` graphics[5], composing the global picture by putting together the following components (or viewports in `grid` language):

dendrograms clusters and order columns/rows;

annotations are additional *tracks* that provide extra information about each column/row according to some associated auxiliary data;

data matrix , i.e. the heatmap itself, shown as coloured cells;

labels associates each column/row with some textual information;

legends such as value scales or color code used for the data matrix or annotations;

other information like main title, sub-title, extra information pane.

Figure 1 shows a diagram of two possible grid layout that combined the above listed components into a complete annotated heatmap.

[3] https://cran.r-project.org/package=NMF
[4] http://github.com/renozao/NMF
[5] Except for drawing dendrograms, which are plotted using the proven and well optimised base function `plot.dendrogram`.

```
# default layout
aheatmap_layout()
# alternative layout
aheatmap_layout("amld | dlma")
```

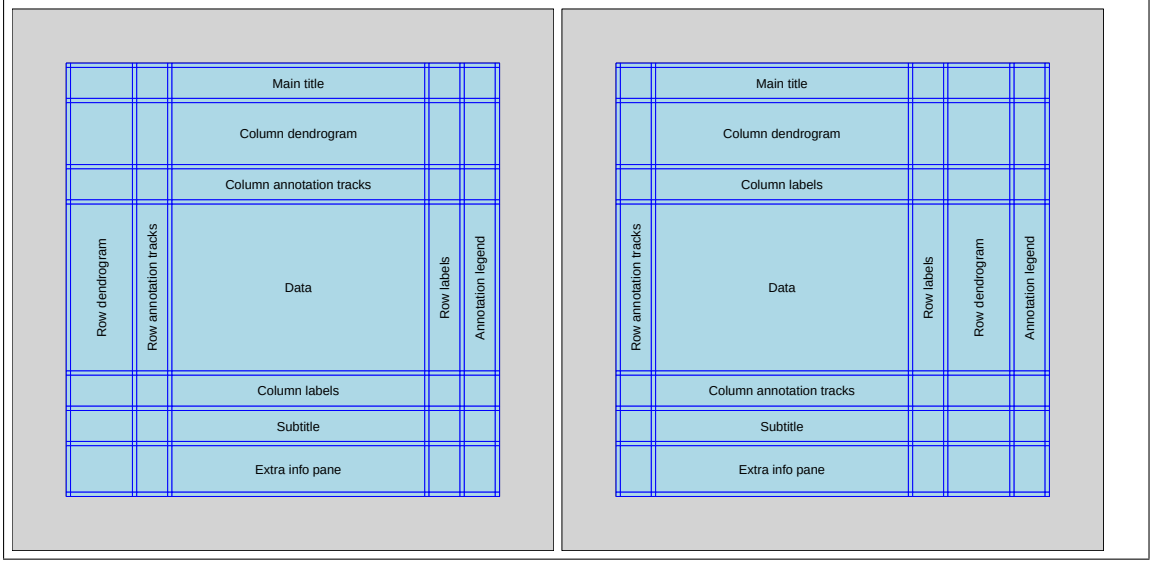


Figure 1: Grid layout diagram of annotated heatmaps: (left) default layout and (right) an alternative layout, with separate specification for rows and columns – passed as a single string.

4 Annotation tracks

5 Column/row ordering

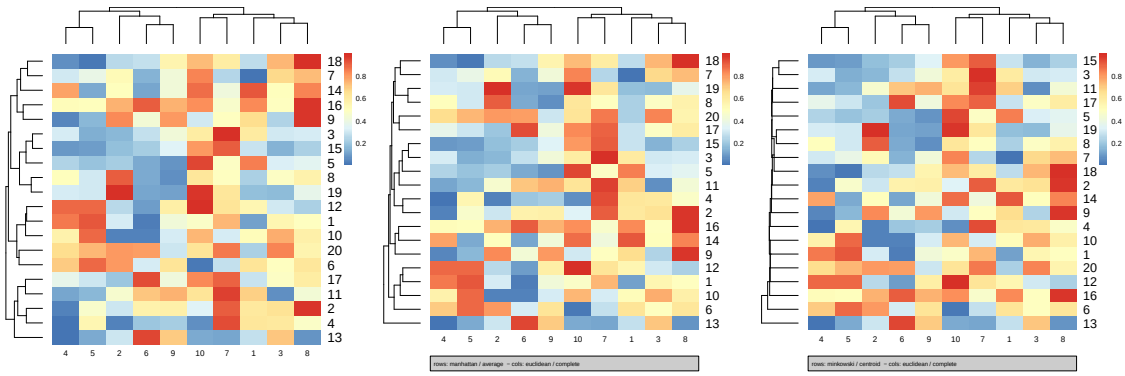
The rows and/or columns of heatmaps are generally ordered in a way that highlight shared value patterns. This ordering can be automatically computed from the data itself, using hierarchical clustering algorithms, or forced to match known groups/order. Arguments `Rowv`, `Colv`, as well as `distfun` and `hclustfun` for automatic clustering, control how the ordering is performed and displayed. They accept the same values as the base function `aheatmap`, mimicking its behaviour, but also supports other convenient ways of specifying ordering and highlighting data patterns, some of which are illustrated in the rest of this section. We refer to the corresponding argument description on the man page `?aheatmap` for a list of all supported values.

5.1 Hierarchical clustering and dendrograms

Dendrograms display result of applying a hierarchical clutering algorithm to the rows or columns, typically using the base function `hclust`.

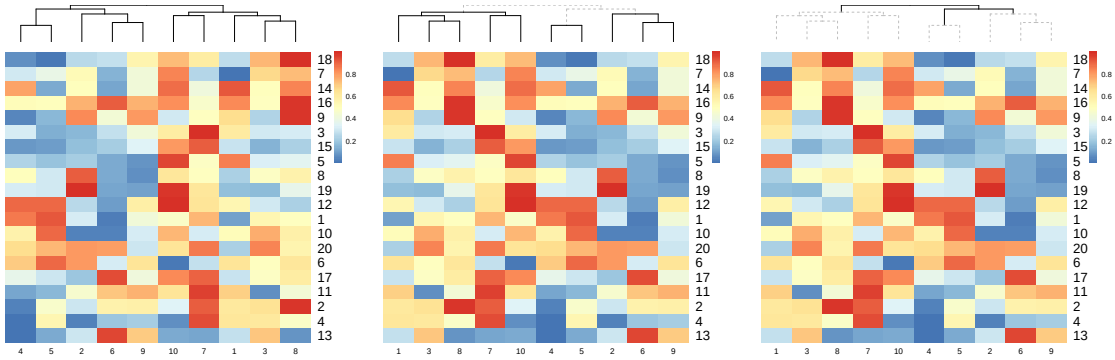
By default `aheatmap` performs hierarchical clustering and show the associated dedrograms of both rows and columns, using the `"euclidean"` distance and the linkage method `"complete"`. However more custom clustering can also be specified:

```
# default
aheatmap(x)
# use different clustering method for rows
aheatmap(x, Rowv = c('manhattan', 'average'), info = TRUE)
# use enternally computed clustering
hc <- hclust(dist(x, method = 'minkowski'), method = 'centroid')
aheatmap(x, Rowv = hc, info = TRUE)
```



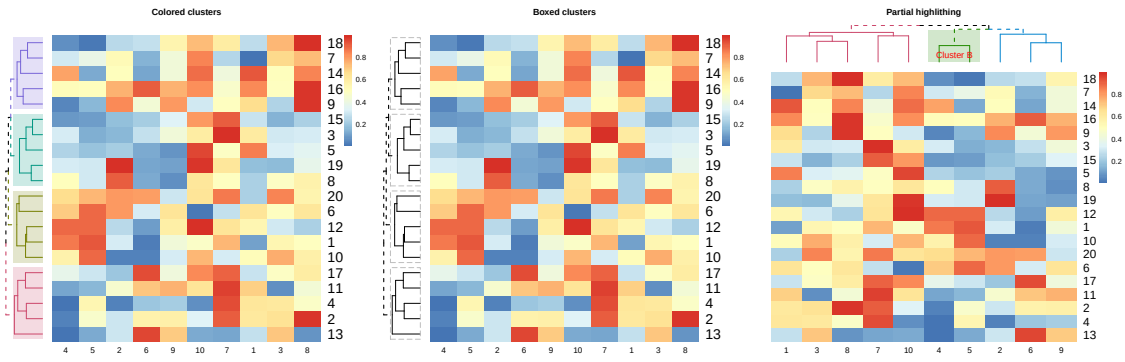
5.1.1 Display

```
# cluster rows but do not show dendrogram
aheatmap(x, Rowv = FALSE)
# cut column dendrogram into 3 clusters
aheatmap(x, Rowv = FALSE, Colv = 3L)
aheatmap(x, Rowv = FALSE, Colv = -3L)
```



`aheatmap` also provides some convenient shortcuts to use the *dendextend* package[6] and produce enhanced cluster highlighting, with colors, text and boxes:

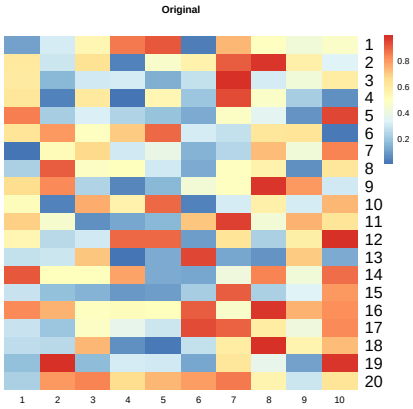
```
# cluster highlighting
aheatmap(x, Rowv = "#4", Colv = FALSE, main = "Colored clusters")
aheatmap(x, Rowv = "#4|!", Colv = FALSE, main = "Boxed clusters")
# highlight cluster #2 only adding some red text
aheatmap(x, Colv = list("#3@2", text_col = 'red', text = 'Cluster B'), Rowv = FALSE
         , main = "Partial highlithing")
```



[6] https://cran.r-project.org/package=dendextend

A completely custom pre-formatted dendrogram can also be build and passed to `Rowv` (or `Colv`), controlling both ordering and display. For example, using the *dendextend* package[7], one can format a dendrogram in a complex way and simply "plug" it into the row or column dendrogram panel:

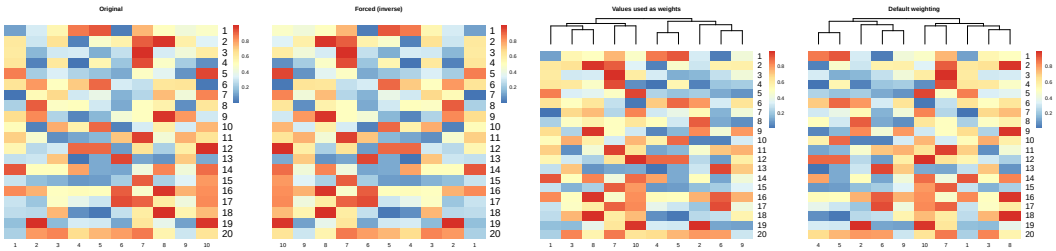
```
# use pre-formatted dedrogram
library(dendextend)
hc <- hclust(dist(t(x)))
hc <- as.dendrogram(hc) %>%
        set("nodes_pch", c(19,1,4)) %>%
        set("nodes_cex", c(2,1,2)) %>%
        set("nodes_col", c(3,4))
hc <- hc %>% set("branches_lwd", c(4,1)) %>%
           set("branches_lty", c(1,1,3)) %>%
           set("branches_col", c(1,2,3))
aheatmap(x, Colv = hc, Rowv = FALSE
            , main = "Custom dendrogram")
```



5.2 Forced order

Column and row order can also be forced to a given order by passing an **integer** indexing vector or `NA` for keeping the original order. It is important that the indexing vector is effectively an integer vector, as passing a numeric vector would only provides weights used when re-ordering the computed dendrogram. When weights are used, the resulting dendrogram is essentially the same, with branches (and leaves) ordered in different ways (see the two last heatmaps below).

```
# orignal order
aheatmap(x, Rowv = NA, Colv = NA, main = 'Original')
# indexing vector
aheatmap(x, Rowv = NA, Colv = seq(ncol(x), 1), main = 'Forced (inverse)')
# not the same as numeric weight vector
aheatmap(x, Rowv = NA, Colv = as.numeric(seq(nrow(x), 1)), main = 'Values used as weights')
# compare with no weights
aheatmap(x, Rowv = NA, main = 'Default weighting')
```



6 Borders

Borders of the different elements can be controlled using argument `border`. Passing a single value draws a border on all relevant elements using the specified color, and a finer control is possible by providing a list of border graphical parameters for specific element separately (Figure 2):

cell border around each cell in the data matrix;

matrix border around the data matrix;

[7] Code borrowed from https://cran.r-project.org/web/packages/dendextend/vignettes/introduction.html

annCol border around each cell in the column annotation;

annRow border around each cell in the row annotation;

annLeg border around each cell in the annotation legend(s);

base default specification to use for each element – if not otherwise defined.

```
# all borders
aheatmap(x, annCol = ann_col, border = TRUE)
# around data matrix
aheatmap(x, annCol = ann_col, border = list(matrix = TRUE))
# finer control on each element
aheatmap(x, annCol = ann_col, border = list(cell = list(lwd=2)
                                            , matrix = list(col='blue', lwd=4)
                                            , annCol='green'
                                            , annLeg = 'red'))
```

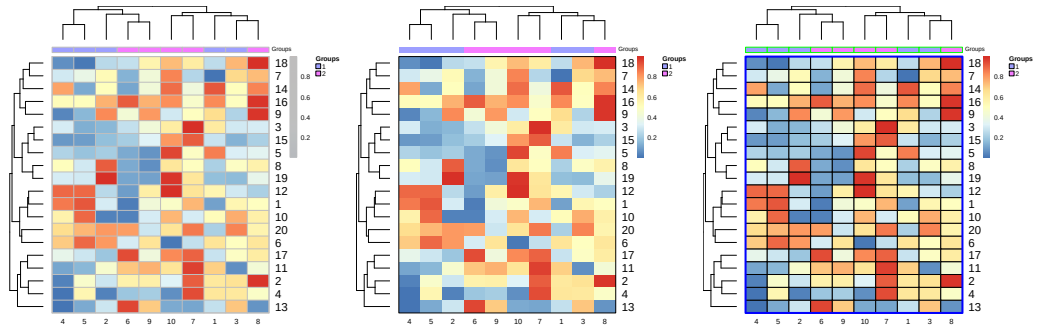


Figure 2: Border control

7 Colours

8 Labels

9 Legends

Annotated heatmaps have two types of legends, one showing the colour-value scale used to visualise the data matrix and another one for the annotation tracks.

9.1 Colour scale

The very principle of a heatmap is to bin data values into a certain number of intervals (or breaks), associating each of these with a given colour. The *colour scale* is the legend that provides details about how to read the resulting colour coded data matrix. As such, it serves multiple purposes:

- provide the mapping between colours and value intervals;
- show the actual range of displayed values;
- optionnaly show the overall distribution of values.

9.1.1 Colours and breaks

9.1.2 Look and position

As for other components in annotated heatmaps, the position of the colour scale is controlled by the argument `layout`, which can also be used to specify if the scale should expand over the full height/width or have a limited fixed size.

By default the scale is placed on the top-right corner of the data matrix, with a limited fixed size. Figure 3 illustrates how to easily obtain some other commonly used positions/look, through the use of special layout shortcuts. More options are available, as detailed in the manual page for `aheatmap_layout`.

```
# vertical on the right expanded over the full height
aheatmap(x, layout = '*')
# horizontal at the bottom-right corner
aheatmap(x, layout = '_')
# horizontal the bottom, expanded over the full width
aheatmap(x, layout = '_*')
# vertical on the left (when not using/showing row dendrogram)
aheatmap(x, Rowv = NA, layout = '^')
```

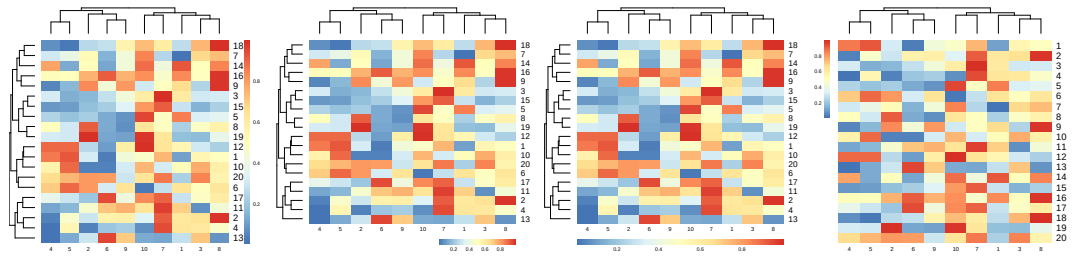


Figure 3: Colour scale alternative layouts: the scale can be placed in different areas around the data matrix and expanded to full height/width.

9.2 Annotations

10 Session Info

- R version 3.6.3 (2020-02-29), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_IL`, `LC_NUMERIC=C`, `LC_TIME=en_IL`, `LC_COLLATE=C`, `LC_MONETARY=en_IL`, `LC_MESSAGES=en_IL`, `LC_PAPER=en_IL`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_IL`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 19.10`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/openblas/libblas.so.3`
- LAPACK: `/usr/lib/x86_64-linux-gnu/libopenblasp-r0.3.7.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.46.0, BiocGenerics 0.32.0, NMF 0.23.6, RColorBrewer 1.1-2, bigmemory 4.5.36, cluster 2.1.0, colorspace 1.4-1, dendextend 1.13.4, doParallel 1.0.15, foreach 1.4.8, iterators 1.0.12, knitr 1.28, pkgmaker 0.31.1, registry 0.5-1, rngtools 1.5.1, synchronicity 1.3.5, xtable 1.8-4

- Loaded via a namespace (and not attached): Matrix 1.2-18, R6 2.4.1, Rcpp 1.0.4, assertthat 0.2.1, bibtex 0.4.2.2, bigmemory.sri 0.1.3, codetools 0.2-16, compiler 3.6.3, crayon 1.3.4, digest 0.6.25, dplyr 0.8.5, evaluate 0.14, farver 2.0.3, ggplot2 3.3.0, glue 1.3.2, grid 3.6.3, gridBase 0.4-7, gridExtra 2.3, gtable 0.3.0, highr 0.8, labeling 0.3, lattice 0.20-40, lifecycle 0.2.0, magrittr 1.5, matrixStats 0.56.0, munsell 0.5.0, pillar 1.4.3, pkgconfig 2.0.3, plyr 1.8.6, purrr 0.3.3, reshape2 1.4.3, rlang 0.4.5, scales 1.1.0, stringi 1.4.6, stringr 1.4.0, tibble 2.1.3, tidyselect 1.0.0, tools 3.6.3, uuid 0.1-4, viridis 0.5.1, viridisLite 0.3.0, withr 2.1.2, xfun 0.12