Generating heatmaps for Nonnegative Matrix Factorization

Package *NMF* - Version 0.23.0

Renaud Gaujoux

July 31, 2020

Abstract

This vignette describes how to produce different informative heatmaps from NMF objects, such as returned by the function `nmf` in the *NMF* package[1] (**Rpackage:NMF**). The main drawing engine is implemented by the function `aheatmap`, which is a highly enhanced modification of the function `pheatmap` from the *pheatmap* package[2], and provides convenient and quick ways of producing high quality and customizable annotated heatmaps. Currently this function is part of the package *NMF*, but may eventually compose a separate package on its own.

Contents

1 Preliminaries

1.1 Quick reminder on NMF models

Given a nonnegative target matrix X of dimension $n \times p$, NMF algorithms aim at finding a rank k approximation of the form:

$$X \approx WH,$$

where W and H are nonnegative matrices of dimensions $n \times k$ and $k \times p$ respectively.

The matrix W is the basis matrix, whose columns are the basis components. The matrix H is the mixture coefficient or weight matrix, whose columns contain the contribution of each basis component to the corresponding column of X. We call the rows of H the basis profiles.

1.2 Heatmaps for NMF

Because NMF objects essentially wrap up a pair of matrices, heatmaps are convenient to visualise the results of NMF runs. The package *NMF* provides several specialised heatmap functions, designed to produce heatmaps with sensible default configurations according to the data being drawn. Being all based on a common drawing engine, they share almost identical interfaces and capabilities. The following specialised functions are currently implemented:

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

`basismap` draws heatmaps of the basis matrix

`coefmap` draws heatmaps of the mixture coefficient matrix

`consensusmap` draws heatmaps of the consensus matrix, for results of multiple NMF runs.

1.3 Heatmap engine

All the above functions eventually call a common heatmap engine, with different default parameters, chosen to be relevant for the given underlying data. The engine is implemented by the function `aheatmap`. Its development started as modification of the function `pheatmap` from the *pheatmap* package. The initial objective was to improve and increase its capabilities, as well as defining a simplified interface, more consistent with the R core function `heatmap`. We eventually aim at providing 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 even `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. This is a feature many R users were looking for, and that was strictly impossible with base heatmaps.
- Legends: default automatic legend and colouring;
- 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 developping/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.

1.4 Data and model

For the purpose of illustrating the use of each heatmap function, we generate a random target matrix, as well as some annotations or covariates:

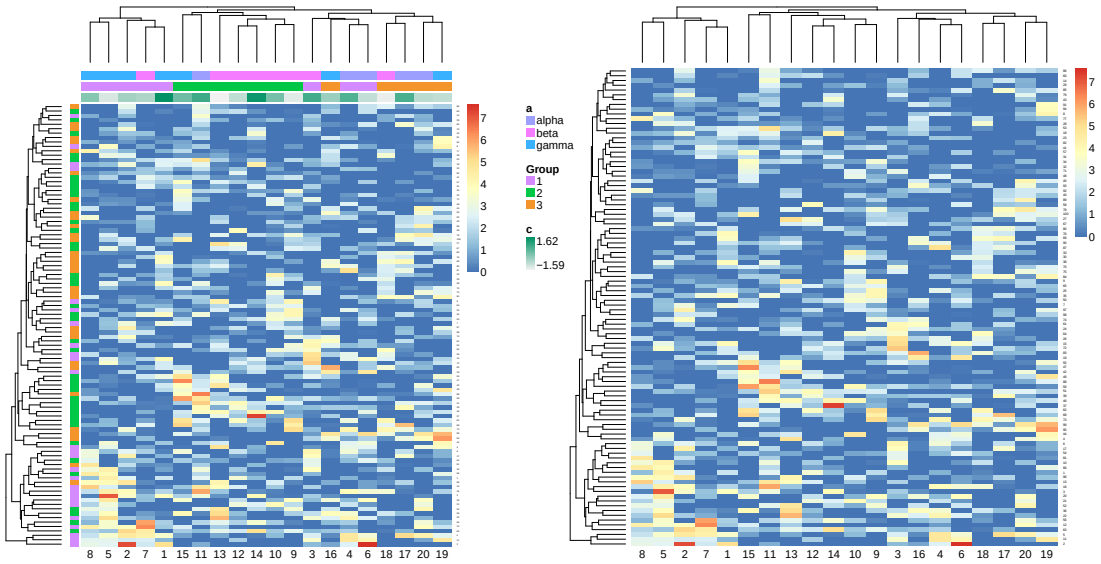
```
# random data that follow an 3-rank NMF model (with quite some noise: sd=2)
X <- syntheticNMF(100, 3, 20, noise=2)

# row annotations and covariates
n <- nrow(X)
d <- rnorm(n)
e <- unlist(mapply(rep, c('X', 'Y', 'Z'), 10))
e <- c(e, rep(NA, n-length(e)))
rdata <- data.frame(Var=d, Type=e)

# column annotations and covariates
p <- ncol(X)
a <- sample(c('alpha', 'beta', 'gamma'), p, replace=TRUE)
# define covariates: true groups and some numeric variable
```

```
c <- rnorm(p)
# gather them in a data.frame
covariates <- data.frame(a, X$pData, c)
```

Note that in the code above, the object `X` returned by `syntheticNMF` *really is* a matrix object, but wrapped through the function `ExposedAttribute` object, which exposes its attributes via a more friendly and access controlled interface `$`. Of particular interests are attributes `'pData'` and `'fData'`, which are lists that contain a factor named `'Group'` that indicates the true underlying clusters. These are respectively defined as each sample's most contrbuting basis component and the basis component to which each feature contributes the most. They are useful to annotate heatmaps and assess the ability of NMF methods to recover the true clusters.

As an example, one can conveniently visualize the target matrix as a heatmap, with or without the relevant sample and feature annotations, using simple calls to the `aheatmap` function:

```
par(mfrow=c(1,2))
aheatmap(X, annCol=covariates, annRow=X$fData)
aheatmap(X)
```



Then, we fit an NMF model using multiple runs, that will be used throughtout this vignette to illustrate the use of NMF heatmaps:

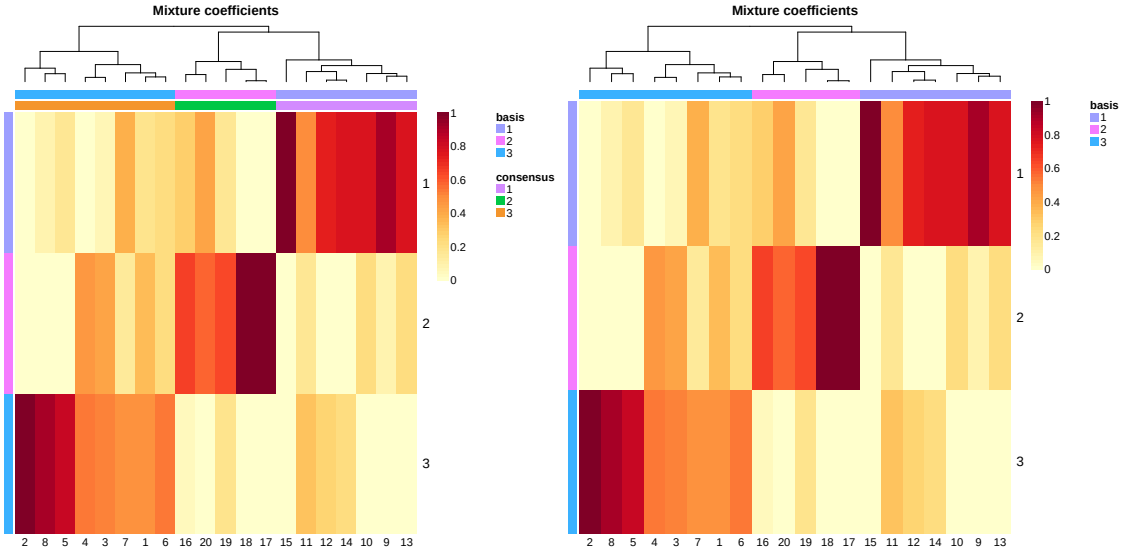
```
res <- nmf(X, 3, nrun=10)
res

## <Object of class: NMFfitX1 >
##   Method: brunet
##   Runs:  10
##   RNG:
##    10407L, -72176537L, 1295796604L, 847127629L, -81266006L, 560487459L, 1768066824L
##   Total timing:
##    user  system elapsed
##   1.303   0.098   3.368
```

NB: To keep the vignette simple, we always use the default NMF method (i.e. `'brunet'`), but all steps could be performed using a different method, or multiple methods in order to compare their perfromances.

2 Mixture Coefficient matrix: `coefmap`

The coefficient matrix of the result can be plotted using the function `coefmap`. The default behaviour for multiple NMF runs is to add two annotation tracks that show the clusters obtained by the best fit and the hierarchical clustering of the consensus matrix[3]. In the legend, these tracks are named *basis* and *consensus* respectively. For single NMF run or NMF model objects, no consensus data are available, and only the clusters from the fit are displayed.

```
opar <- par(mfrow=c(1,2))
# coefmap from multiple run fit: includes a consensus track
coefmap(res)
# coefmap of a single run fit: no consensus track
coefmap(minfit(res))
```




```
par(opar)
```

NB: Note how both heatmaps were drawn on the same plot, simply using the standard call to `par(mfrow=c(1,2))`. This is impossible to achieve with the R core function `heatmap`. See Section 5 for more details about compatibility with base and grid graphics.

By default:

- the rows are not ordered;
- the columns use the default ordering of `aheatmap`, but may easily be ordered according to the clusters defined by the dominant basis component for each column with `Colv="basis"`, or according to those implied by the consensus matrix, i.e. as in `consensusmap`, with `Colv="consensus"`;

[3]The hierarchical clustering is computed using the consensus matrix itself as a similarity measure, and average linkage. See `?consensushc`.

- each column is scaled to sum up to one;
- the color palette used is `'YlOrRd'` from the *RColorBrewer* package[4] (**Rpackage:RColorBrewer**), with 50 breaks.

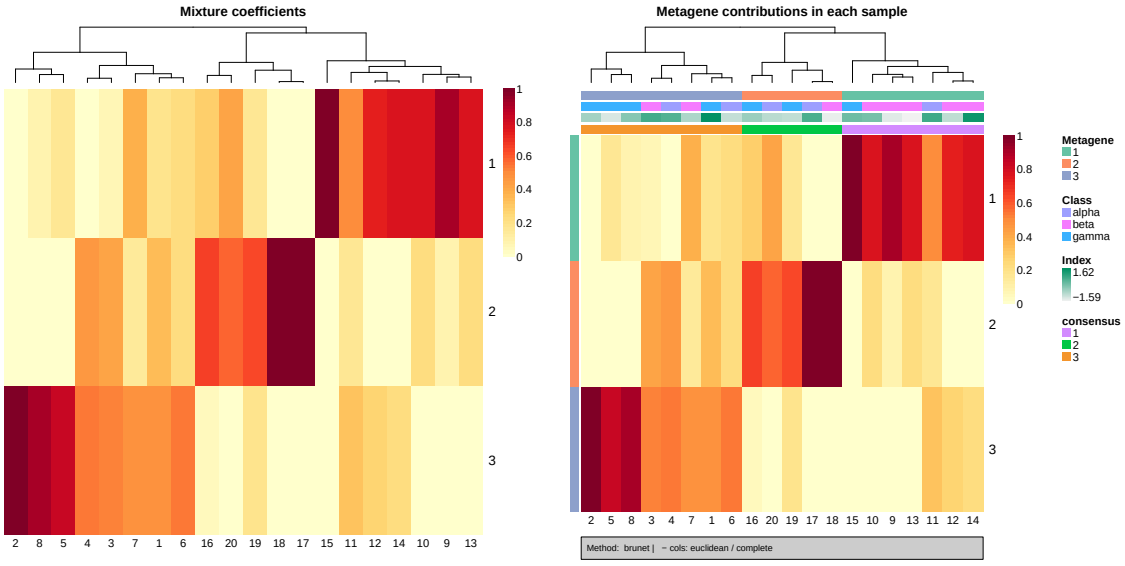
In term of arguments passed to the heatmap engine `aheatmap`, these default settings translate as:

```
Rowv = NA
Colv = TRUE
scale = 'c1'
color = 'YlOrRd:50'
annCol = predict(object) + predict(object, 'consensus')
```

If the ordering does not come from a hierarchical clustering (e.g., if `Colv='basis'`), then no dendrogram is displayed. The default behaviour of `aheatmap` can be obtained by setting arguments `Rowv=TRUE, Colv=TRUE, scale='none'`.

The automatic annotation tracks can be hidden all together by setting argument `tracks=NA`, displayed separately by passing only one of the given names (e.g. `tracks=':basis'` or `tracks='basis:'` for the row or column respectively), and their legend names may be changed by specifying e.g. `tracks=c(Metagene=':basis', 'consensus')`. Beside this, they are handled by the heatmap engine function `aheatmap` and can be customised as any other annotation tracks – that can be added via the same argument `annCol` (see Section 5 or `?aheatmap` for more details).

```
opar <- par(mfrow=c(1,2))
# removing all automatic annotation tracks
coefmap(res, tracks=NA)
# customized plot
coefmap(res, Colv = 'euclidean'
        , main = "Metagene contributions in each sample", labCol = NULL
        , annRow = list(Metagene=':basis'), annCol = list(':basis', Class=a, Index=c)
        , annColors = list(Metagene='Set2')
        , info = TRUE)
```



[4] https://cran.r-project.org/package=RColorBrewer

```
par(opar)
```

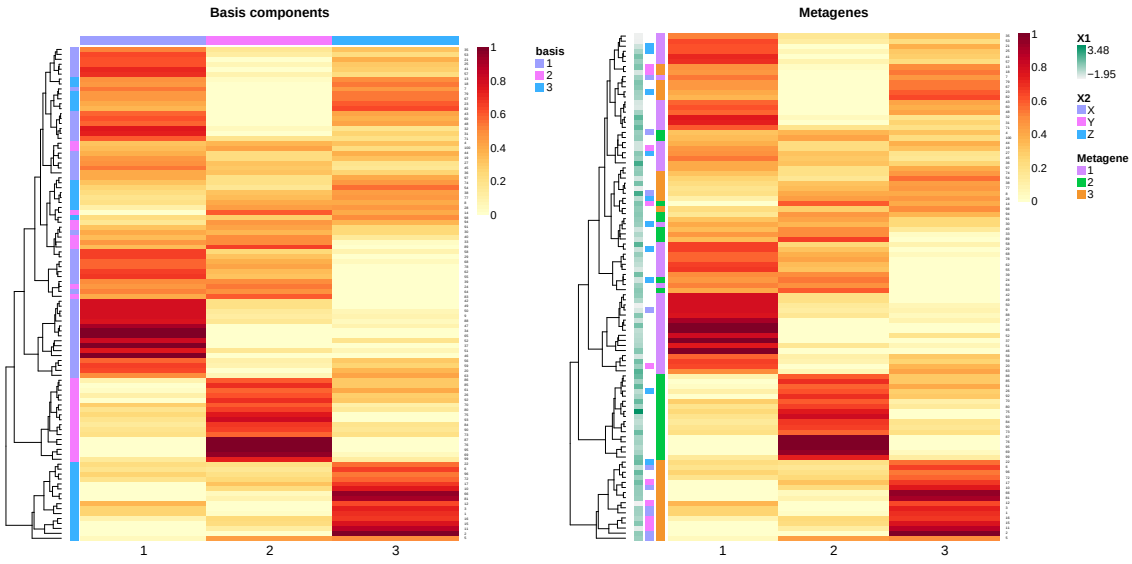
NB: The feature that allows to display some information about the fit at the bottom of the plot via argument `info=TRUE` is still experimental. It is helpful mostly when developing algorithms or doing an analysis, but would seldom be used in publications.

3 Basis matrix: `basismap`

The basis matrix can be plotted using the function **`basismap`**. The default behaviour is to add an annotation track that shows for each row the dominant basis component. That is, for each row, the index of the basis component with the highest loading.

This track can be disabled by setting **`tracks=NA`**, and extra row annotations can be added using the same argument **`annRow`**.

```
opar <- par(mfrow=c(1,2))
# default plot
basismap(res)
# customized plot: only use row special annotation track.
basismap(res, main="Metagenes", annRow=list(d, e), tracks=c(Metagene=':basis'))
```




```
par(opar)
```

By default:

- the columns are not ordered;
- the rows are ordered by hierarchical clustering using default distance and linkage methods (`'eculidean'` and `'complete'`);
- each row is scaled to sum up to one;

- the color palette used is `'YlOrRd'` from the *RColorBrewer* package[5] (**Rpackage:RColorBrewer**), with 50 breaks.

In term of arguments passed to the heatmap engine `aheatmap`, these default settings translate as:

```
Colv = NA
scale = 'r1'
color = 'YlOrRd:50'
annRow = predict(object, 'features')
```

4 Consensus matrix: `consensusmap`

When doing clustering with NMF, a common way of assessing the stability of the clusters obtained for a given rank is to consider the consensus matrix computed over multiple independent NMF runs, which is the average of the connectivity matrices of each separate run [6]. This procedure is usually repeated over a certain range of factorization ranks, and the results are compared to identify which rank gives the best clusters, possibly in the light of some extra knowledge one could have about the samples (e.g. covariates). The functions `nmf` and `consensusmap` make it easy to implement this whole process.

NB: The consensus plots can also be generated for fits obtained from single NMF runs, in which case the consensus matrix simply reduces to a single connectivity matrix. This is a binary matrix (i.e. entries are either 0 or 1), that will always produce a bi-colour heatmap, and by default clear blocks for each cluster.

4.1 Single fit

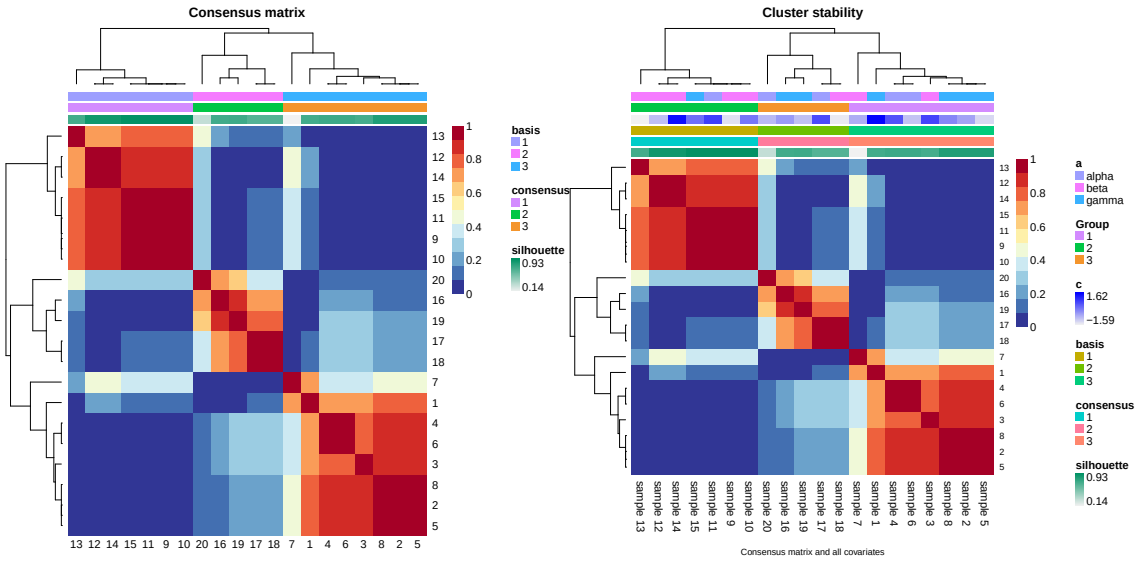
In section Section 1.4, the NMF fit `res` was computed with argument `nrun=10`, and therefore contains the best fit over 10 runs, as well as the consensus matrix computed over all the runs [7]. This can be ploted using the function `consensusmap`, which allows for the same kind of customization as the other NMF heatmap functions:

```
opar <- par(mfrow=c(1,2))
# default plot
consensusmap(res)
# customized plot
consensusmap(res, annCol=covariates, annColors=list(c='blue')
                , labCol='sample ', main='Cluster stability'
                , sub='Consensus matrix and all covariates')
```

[5] https://cran.r-project.org/package=RColorBrewer

[6] Hence, stability here means robustness with regards to the initial starting point, and shall not be interpreted as in e.g. cross-validation/bootstrap analysis. However, one can argue that having very consistent clusters across runs somehow supports for a certain regularity or the presence of an underlying pattern in the data.

[7] If one were interested in keeping the fits from all the runs, the function `nmf` should have been called with argument `.options='k'`. See section *Options* in `?nmf`. The downstream hanlding of the result would remain identical.




```
par(opar)
```

By default:

- the rows and columns of the consensus heatmap are symmetrically ordered by hierarchical clustering using the consensus matrix as a similarity measure and average linkage, and the associated dendrogram is displayed;
- the color palette used is the reverse of `'RdYlBu'` from the *RColorBrewer* package[8] (**Rpackage:RColorBrewer**).

In term of arguments passed to the heatmap engine `aheatmap`, these default settings translate as:

```
distfun = function(x) as.dist(1-x) # x being the consensus matrix
hclustfun = 'average'
Rowv = TRUE
Colv = "Rowv"
color = '-RdYlBu'
```

4.2 Single method over a range of ranks

The function `nmf` accepts a range of value for the rank (argument `rank`), making it fit NMF models for each value in the given range [9]:

```
res2_7 <- nmf(X, 2:7, nrun=10, .options='v')

## Compute NMF rank= 2  ... + measures ... OK
## Compute NMF rank= 3  ... + measures ... OK
## Compute NMF rank= 4  ... + measures ... OK
## Compute NMF rank= 5  ... + measures ... OK
## Compute NMF rank= 6  ... + measures ... OK
## Compute NMF rank= 7  ... + measures ... OK
```

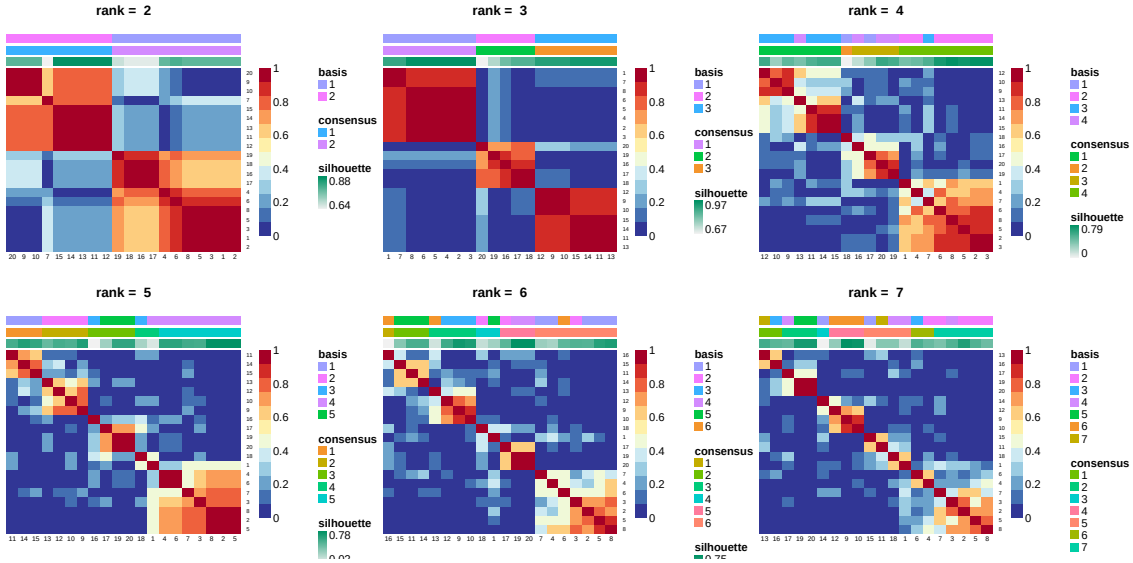
[8] https://cran.r-project.org/package=RColorBrewer

[9] Before version 0.6, this feature was provided by the function `nmfEstimateRank`. From version 0.6, the function `nmf` accepts ranges of ranks, and internally calls the function `nmfEstimateRank` – that remains exported and can still be called directly. See documentation `?nmfEstimateRank` for more details on the returned value.

```
class(res2_7)

## [1] "NMF.rank"
```

The result `res2_7` is an S3 object of class `'NMF.rank'`, that contains – amongst other data – a list of the best fits obtained for each value of the rank in range $[\![2, 7]\!]$. The method of `consensusmap` defined for class `'NMF.rank'`, which plots all the consensus matrices on the same plot:

```
consensusmap(res2_7)
```



NB: The main title of each consensus heatmap can be customized by passing to argument `main` a character vector or a list whose elements specify each title. All other arguments are used in each internal call to consensusmap, and will therefore affect all the plots simultaneously. The layout can be specified via argument `layout` as a numeric vector giving the number of rows and columns in a `mfrow`-like way, or as a matrix that will be passed to R core function `layout`. See `?consensusmap` for more details and example code.

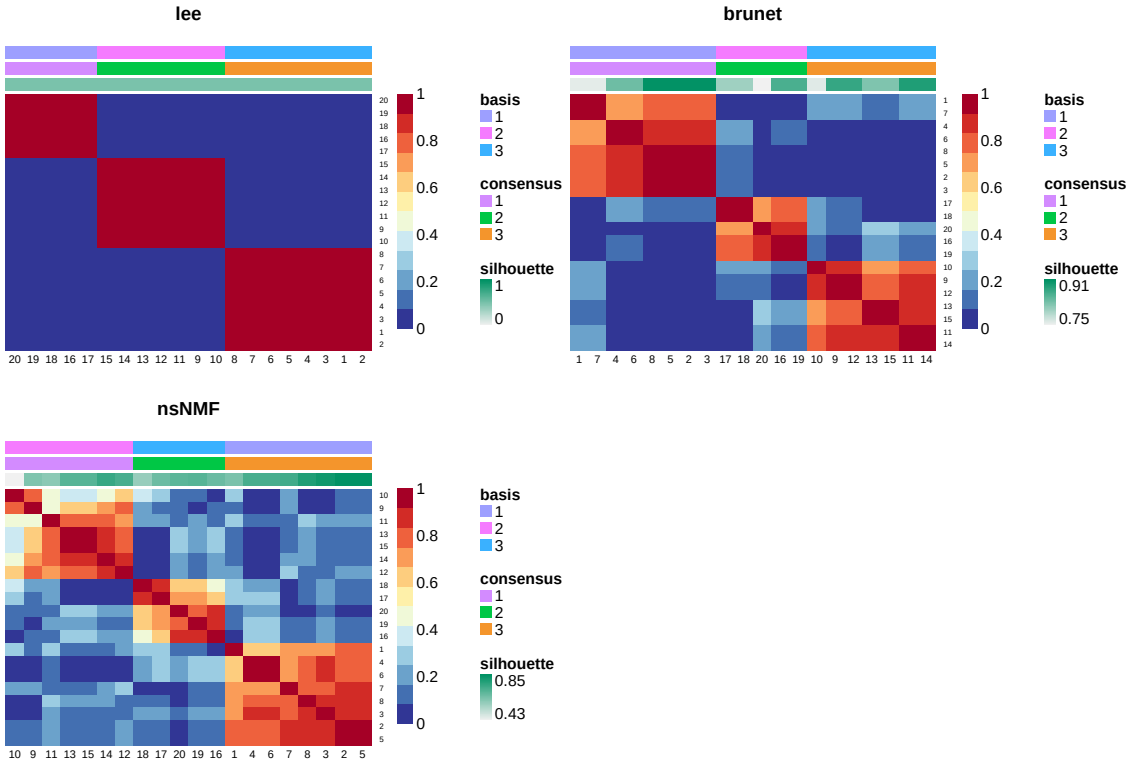
4.3 Single rank over a range of methods

If one is interested in comparing methods, for a given factorization rank, then on can fit an NMF model for each method by providing the function `nmf` with a `list` in argument `method`:

```
res_methods <- nmf(X, 3, list('lee', 'brunet', 'nsNMF'), nrun=10)

## Compute NMF method 'lee' [1/3] ... OK
## Compute NMF method 'brunet' [2/3] ... OK
## Compute NMF method 'nsNMF' [3/3] ... OK

class(res_methods)

## [1] "NMFList"
## attr(,"package")
## [1] "NMF"
```

The result `res_methods` is an S4 object of class `NMFList`, which is essentially a named list, that contains each fits and the CPU time required by the whole computation. As previously, the sequence of consensus matrices is plotted with `consensusmap`:

```
consensusmap(res_methods)
```



5 Generic heatmap engine: `aheatmap`

This section still needs to be written, but many examples of annotated heatmaps can be found in the demos `'aheatmap'` and `'heatmaps'`:

```
demo('aheatmap')
# or
demo('heatmaps')
```

These demos and the plots they generate can also be browsed online at http://nmf.r-forge.r-project.org/_DEMOS.html.

6 Session Info

- R version 3.6.1 (2019-07-05), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_IE.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_IE.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_IE.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_IE.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 18.04.4 LTS`

- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/openblas/libblas.so.3`
- LAPACK: `/usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.44.0, BiocGenerics 0.30.0, NMF 0.23.0, RColorBrewer 1.1-2, bigmemory 4.5.36, cluster 2.1.0, doParallel 1.0.15, foreach 1.4.7, iterators 1.0.12, knitr 1.24, pkgmaker 0.27, registry 0.5-1, rngtools 1.4, synchronicity 1.3.5, xtable 1.8-4
- Loaded via a namespace (and not attached): R6 2.4.0, Rcpp 1.0.2, bibtex 0.4.2, bigmemory.sri 0.1.3, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.1, crayon 1.3.4, digest 0.6.25, dplyr 1.0.0, evaluate 0.14, farver 2.0.3, generics 0.0.2, ggplot2 3.3.2, glue 1.4.1, grid 3.6.1, gridBase 0.4-7, gtable 0.3.0, highr 0.8, labeling 0.3, lifecycle 0.2.0, magrittr 1.5, munsell 0.5.0, pillar 1.4.2, pkgconfig 2.0.2, plyr 1.8.4, purrr 0.3.3, reshape2 1.4.4, rlang 0.4.6, scales 1.1.0, stringi 1.4.3, stringr 1.4.0, tibble 2.1.3, tidyselect 1.1.0, tools 3.6.1, uuid 0.1-2, vctrs 0.3.0, withr 2.1.2, xfun 0.9