MetaOMineR

Edi Prifti & Emmanuelle Le Chatelier

26.06.15

Contents

Context

`momr`, is the base package of a larger suite of R packages named `MetaOMineR`, which stands for *Mining MetaOmics data in R.* It encompasses many useful functions and modules needed for the analyses of shotgun Quantitative Metagenomics (QM) data. It can be also used for 16S or other types of omics data. Developed since the beginning of the field, `momr` has evolved and is structured around different modules such as preprocessing, analysis, visualization, map-reduce parallel computing, etc. The package comes with a small subset of a real metagenomics data-set of human gut microbiome from the MetaHIT project (Le Chatelier et al, Nature, 2013).

MetaOMineR works with data that can be structured as standalone packages or not. They should contain the needed information to describe a given gene catalog, such as for instance the gene length, annotations, clustering information, etc. In this tutorial we demonstrate some of the functionalities of `momr` with simple examples.

Data processing

In this section we will see how to load the test dataset that comes with the package and how to pre-process it for analysis in a second step. Let us start by loading the momr library.

```
library(momr)
library(knitr) # for printing tables
```

To see what data objects are contained in the package we type:

```
data(package="momr")
```

We will see four objects
- `hs_3.3_metahit_genesize` - `hs_3.3_metahit_sample_dat_freq` - `hs_3.3_metahit_sample_dat_raw` - `mgs_hs_3.3_metahit_sup500`

Loading the data

The files are named following these criteria *(hs = homo sapiens; 3.3_metahit = the gene catalog from metahit with 3.3M genes)*. Let us load the raw count data-set after mapping and counting against the *(Qin et al, Nature, 2010)* gene catalog.

```
# Loading the raw count dataset
data("hs_3.3_metahit_sample_dat_raw")
str(hs_3.3_metahit_sample_dat_raw[,10:14])
```

```
##  int [1:5000, 1:5] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:5000] "4955" "4956" "4957" "4958" ...
##   ..$ : chr [1:5] "MH0109" "MH0110" "MH0186" "MH0108" ...
```

As we can see `hs_3.3_metahit_sample_dat_raw` is a data frame containing 5000 features (rows) and 292 samples (columns). This dataset is a subset of the whole 3.3M feature data-frame and as we can see is very sparse.

```
kable(tail(hs_3.3_metahit_sample_dat_raw[,10:14]))
```

	MH0109	MH0110	MH0186	MH0108	MH0236
9904	0	0	0	0	0
9905	0	0	0	0	0
9906	0	0	0	0	0
9907	0	0	0	0	0
9908	38	58	12	3	14
9909	0	0	0	0	0

```
zeroperc <- round(sum(hs_3.3_metahit_sample_dat_raw==0)/
                    length(hs_3.3_metahit_sample_dat_raw)*100)
paste("There are ", zeroperc, "% zeros in this data frame.",sep="")
```

```
## [1] "There are 81% zeros in this data frame."
```

Normalization

This processing step is necessary to be able to compare abundance among genes and samples. For this reason different normalization procedures are implemented in the package. Note that even from an identical number of reads, the total number of counts can vary when filtering the reads for quality or according to the reference exhaustivity. For the experiment to work well we need to select a gene catalog reference that is representative enough for the different microbial ecosystems sampled in the study. There is not yet a gold standard for normalizing data in quantitative metagenomics and the RPKM method has proven to be good enough in different QM projects. We aim to enrich the package with other normalization approached shortly in the future.

1. RPKM **(Reads Per Kilobase per Million reads mapped)** is one of the first methods used in QM and was inspired by the RNA-Seq field (Mortazavi et al., Nature Methods, 2008). This approach was initially introduced to facilitate comparisons between genes within a sample and combines between- and within-sample normalization, as it re-scales gene counts to correct for differences in both library sizes and gene length. Let assume that two genes form a given species have different lengths. The longer gene has a higher probability of having more reads mapped to it compared to the shorter one especially when the abundance is low. For this reason we compute a scaling factor which is dependent on the gene length in the normalization process. A second scaling factor applied is the sequencing depth.

2. TC **(Total count)** is a simpler method also used in the 16S datasets. The high variability of sequencing depth among the different samples id inherent of the NGS technology. For this reason it is important to scale the abundance of reads for each sample by the sequencing depth. Technically we can scale each sample by the total number of counts.

```
# Normalization should be performed with the whole dataset (3.3M)
# Loading the gene length information
data(hs_3.3_metahit_genesize)
str(hs_3.3_metahit_genesize)
```

```
##  Named int [1:5000] 1083 1746 813 504 162 1356 150 1263 1794 273 ...
##  - attr(*, "names")= chr [1:5000] "4955" "4956" "4957" "4958" ...
```

```
norm.data <- normFreqRPKM(dat=hs_3.3_metahit_sample_dat_raw,
                          cat=hs_3.3_metahit_genesize)
```

```
## [1] "The dataset is a matrix"
```

```
kable(tail(norm.data[,10:14]))
```

	MH0109	MH0110	MH0186	MH0108	MH0236
9904	0.0000000	0.0000000	0.0000000	0.00e+00	0.0000000
9905	0.0000000	0.0000000	0.0000000	0.00e+00	0.0000000

	MH0109	MH0110	MH0186	MH0108	MH0236
9906	0.0000000	0.0000000	0.0000000	0.00e+00	0.0000000
9907	0.0000000	0.0000000	0.0000000	0.00e+00	0.0000000
9908	0.0004564	0.0004709	0.0002019	9.99e-05	0.0001233
9909	0.0000000	0.0000000	0.0000000	0.00e+00	0.0000000

Hereafter we will use a subset of the complete dataset normalized using the 3.3M genes. Note that the scaling factor is lower in the extracted dataset compared to the full dataset due to the lower number of reads sampled for this subset of genes.

```
# Loading the frequency dataset
data("hs_3.3_metahit_sample_dat_freq")
kable(tail(hs_3.3_metahit_sample_dat_freq[,10:14]))
```

	MH0109	MH0110	MH0186	MH0108	MH0236
9904	0e+00	0e+00	0e+00	0e+00	0e+00
9905	0e+00	0e+00	0e+00	0e+00	0e+00
9906	0e+00	0e+00	0e+00	0e+00	0e+00
9907	0e+00	0e+00	0e+00	0e+00	0e+00
9908	6e-07	1e-06	2e-07	1e-07	2e-07
9909	0e+00	0e+00	0e+00	0e+00	0e+00

Downsizing

Another method to reduce the variability that is generated by the sequencing depths is the ***downsizing*** also known as ***rarefaction***. It consists of drawing randomly the same number of reads for each sample and mapping those to the catalog. For this we need to determine a common level of reads to be drawn (sequencing depth).

```
# Determining the minimal common number of reads
min_nb_reads <- summary(colSums(hs_3.3_metahit_sample_dat_raw))
(min_nb_reads["Min."]); (min_nb_reads["Max."])
```

```
## Min.
##  661
```

```
##   Max.
## 274400
```

```
min_nb_reads <- min_nb_reads["Min."]
```

We can notice that the sequencing depth varies greatly in this dataset and this is probably because this is an incomplete dataset. We can perform this for the whole dataset only one time. Next the dataset needs to be normalized as shown above.

```
# Downsizing the whole matrix
data.downsized <- downsizeMatrix(data=hs_3.3_metahit_sample_dat_raw[,1:5],
                                 level=min_nb_reads, repetitions=1, silent=FALSE)
```

```
## [1] "1 Sample MH0277 with 661 reads and 54 genes"
## [1] "         step 1 with 54 genes"
## [1] "2 Sample MH0087 with 7346 reads and 160 genes"
## [1] "         step 1 with 74 genes"
## [1] "3 Sample MH0444 with 37988 reads and 384 genes"
## [1] "         step 1 with 118 genes"
## [1] "4 Sample MH0156 with 20868 reads and 333 genes"
## [1] "         step 1 with 101 genes"
## [1] "5 Sample MH0333 with 5671 reads and 182 genes"
## [1] "         step 1 with 98 genes"
```

```
kable(tail(data.downsized))
```

	MH0277	MH0087	MH0444	MH0156	MH0333
9904	0	0	0	0	0
9905	0	0	0	0	0
9906	0	0	2	0	0
9907	0	0	0	0	0
9908	0	2	0	1	2
9909	0	0	0	0	0

```
colSums(data.downsized, na.rm=TRUE)
```

```
## MH0277 MH0087 MH0444 MH0156 MH0333
##    661    661    661    661    661
```

Important note: Let assume that most of the samples are sequenced nicely above a sequencing depth we set, but a few samples have a low number of reads for various reasons. Should we still downsize very low (in order to include them) and lose most of the data? The answer is *No*! We recommend setting up downsizing level sufficiently high to maintain a high counting depth. Samples with a total number of reads below the level won't be downsized (NA will be generated instead) and may be discarded or replaced as a proxy by original raw counts before generating the frequency matrix using `normFreqRPKM` function.

Gene richness

A simple number can describe the complexity of an ecosystem that we call here richness. That is the number of genes that are found to be present (`gene_abundance > 0`) in a given sample. Indeed, different studies have shown that the richness is associated with different aspects of the ecosystem *(Le Chatelier et al, Nature, 2013)* and correlates strongly with the number of present microbial species *(Nielsen, Almeida et al, Nat Biotech, 2014)*.

```
# Downsizing the genecount
richness <- colSums(hs_3.3_metahit_sample_dat_raw>0, na.rm=TRUE)
summary(richness)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    54.0   720.8   956.0   941.8  1123.0  4909.0
```

Downsizing

Gene richness is very sensitive to the sequencing depth. For this reason we use the downsizing approach to estimate it and perform this multiple times. Finally we compute a mean estimation of the multiple drawings.

```
# Downsizing the matrix multiple times for the computation of gene richness
data.genenb <- downsizeGC(data=hs_3.3_metahit_sample_dat_raw,
                          level=min_nb_reads, repetitions=30, silent=TRUE)
head(apply(data.genenb,2,mean))
```

```
##    MH0277    MH0087    MH0444    MH0156    MH0333    MH0233
##  54.00000  77.93333 112.23333 108.66667 105.63333 150.76667
```

```
head(apply(data.genenb,2,sd))
```

```
##   MH0277   MH0087   MH0444   MH0156   MH0333   MH0233
## 0.000000 4.217642 5.537853 4.497764 4.319030 6.273553
```

Notice that the standard deviation is quite small for 30 random drawings.

```
richness.dwnz <- colMeans(data.genenb, na.rm=TRUE)
par(mfrow=c(1,2))
plot(density(richness), main="gene richness", lwd=2,col="darkred")
plot(density(richness.dwnz), main="downsized gene richness", lwd=2,col="darkred")
```

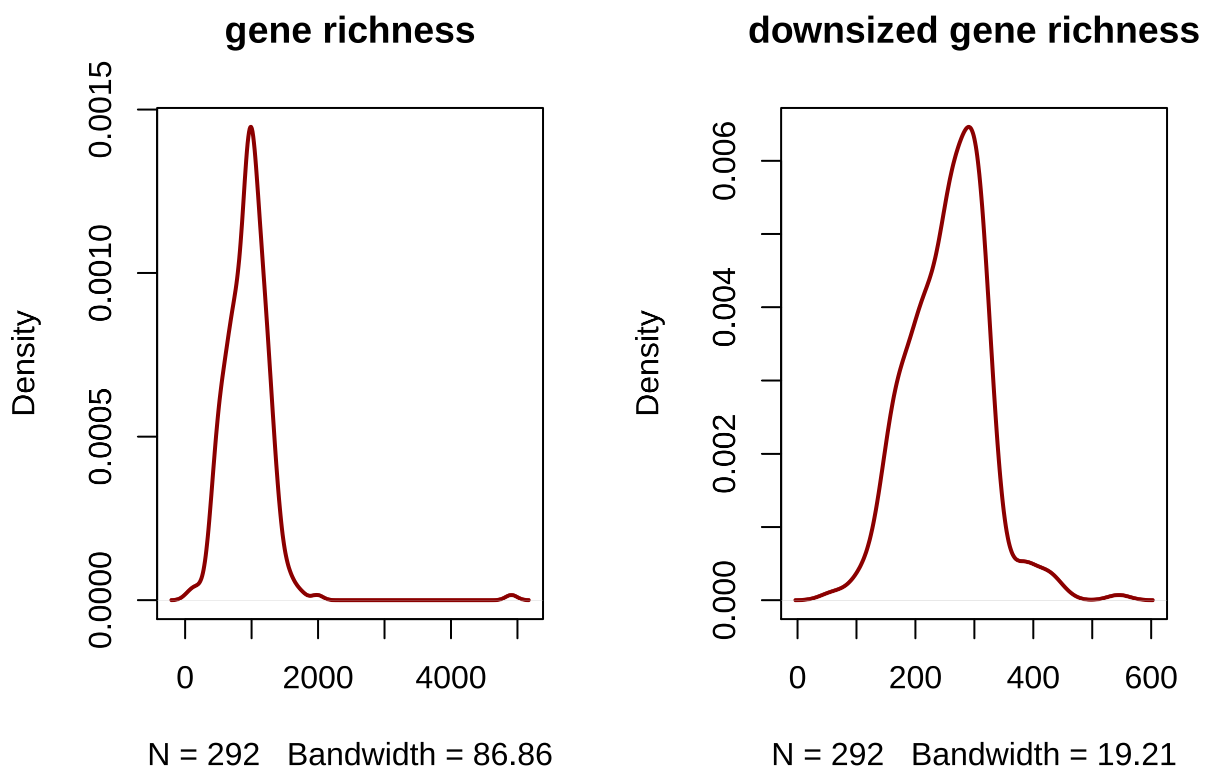

Figure 1: Raw and downsized richness distribution.

In this example (Figure 1) we can see the effect of downsizing on gene richness. For instance one sample had a much higher richness than the rest due to the high variability as mentioned above. After downsizing this sample still remained higher but more comparable with the rest.

```
par(mfrow=c(1,1))
col <-  as.character(cut(colSums(hs_3.3_metahit_sample_dat_raw),
                          c(0,2^seq(0, 9, by=1))*1000,
                          labels=paste("gray",seq(100,10,-10),sep="")))
plot(richness, richness.dwnz, main="downsizing effect on richness",
     pch=20,col=col)
```

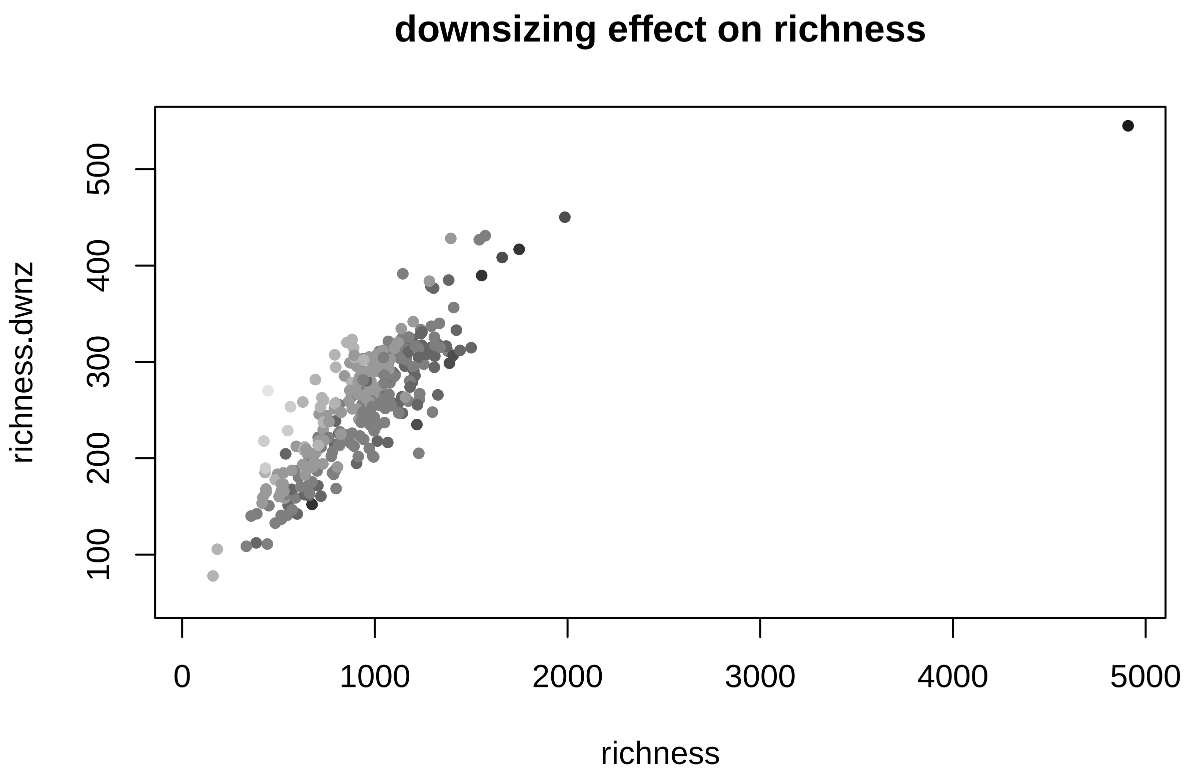

Figure 2: Downsizing effect on gene richness.

This plot (Figure 2) where samples are colored according to read count abundance (the darker the higher) visualize the bias in gene richness estimation due to heterogenous counting depth.

Upsizing

As mentioned above for samples with very low sequencing depth (under the downsizing level) the downsizing process will produce NAs and they will not be exploitable. Based on our observations gene richness downsized at different levels will correlate very strongly among the different levels. This observation led us to propose the *upsizing* approach for gene richness estimation, which allows to estimate a higher level distribution and impute the missing data. In the following example we will use different downsizing levels and show how we can use the up-sizing process to solve this issue.

```
downsize.gc.res <- downsizeGC.all(data = hs_3.3_metahit_sample_dat_raw,
               levels = c(600, 5000, 10000, 15000, 20000),
                  repetitions = 10, silent = TRUE)
kable(downsize.gc.res[[2]])
```

down_6e-04M	down_0.005M	down_0.01M	down_0.015M	down_0.02M
75	141	NA	NA	NA
69	145	NA	NA	NA
74	142	NA	NA	NA
77	143	NA	NA	NA
76	144	NA	NA	NA
74	146	NA	NA	NA
77	139	NA	NA	NA
71	138	NA	NA	NA
78	141	NA	NA	NA
80	137	NA	NA	NA

This function returns a list of samples each containing a matrix of dimension *n=repetitions* x *l=levels* as illustrated above for the second sample. Now let's transform it as a matrix where each column contain the mean-ed downsized values for each repetition.

```
downsize.gc.mat <- downsizedRichnessL2T(richness.list = downsize.gc.res)
kable(head(downsize.gc.mat))
```

	down_6e-04M	down_0.005M	down_0.01M	down_0.015M	down_0.02M
MH0277	49.7	NA	NA	NA	NA
MH0087	75.1	141.6	NA	NA	NA
MH0444	108.6	216.9	262.0	296.2	320.1
MH0156	106.1	231.2	285.2	312.3	330.3
MH0333	100.3	177.8	NA	NA	NA
MH0233	142.4	311.6	369.7	417.2	442.6

Next, we will use the upsizing approach to estimate the missing values as illustrated in Figure 3.

```
upsized <- computeUpsizedGC(richness.table = downsize.gc.mat,
                                    keep.real = TRUE)
kable(head(upsized))
```

	down_6e-04M_una	down_0.005M_una	down_0.01M_una	down_0.015M_una	down_0.02M_una
MH0277	50	78	86	86	89
MH0087	75	142	167	177	187
MH0444	109	217	262	296	320
MH0156	106	231	285	312	330
MH0333	100	178	212	228	241
MH0233	142	312	370	417	443

```
reg <- lm(upsized[,2] ~ upsized[,1])
plot(upsized[,2] ~ upsized[,1], main="Regression of the first two levels",
      xlab=("600 reads"),ylab=("5000 reads"), pch=21)
abline(reg,col="red")
points(upsized[is.na(downsize.gc.mat[,2]),2] ~ upsized[is.na(downsize.gc.mat[,2]),1],
       pch=20, col="red")
```

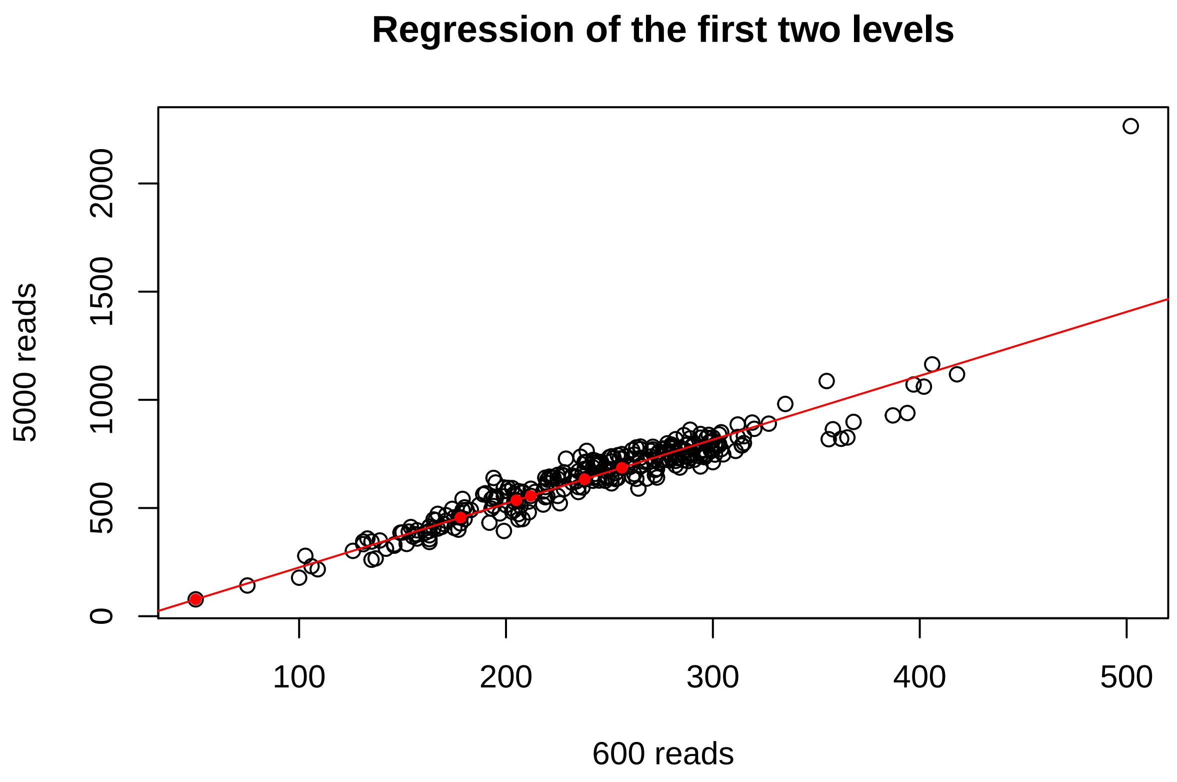

Figure 3: Regression of the first two levels downsizing levels. In red are depicted the points not downsized in the second level.

To compare properly gene richness between samples, we recommend to fix the downsizing/upsizing threshold level in a way that the read counts of most of the samples are above the threshold but also without losing much information with a stringent level.

Sample clustering

Heatmap

Now that the dataset is processed we will relate samples together in order to explore any particular pattern. For this the function `hierClust` will compute the inter-sample distance and use a hierarchical clustering approach cluster samples in a tree. The default distance is computed as `1-cor` where cor is the inter-sample spearman correlation. The hierarchical clustering method is the `ward.D`. This function returns a list containing the correlation matrix, the distance object and the hierarchical clustering object. It also displays a heatmap of the correlation matrix with the ward computed dendrogram (Figure 4). These results can be also used as standalone data to fine-tune the analyses.

```
hc.data <- hierClust(data=hs_3.3_metahit_sample_dat_freq[,1:10], side="col", hclust.method = "ward.D")
```

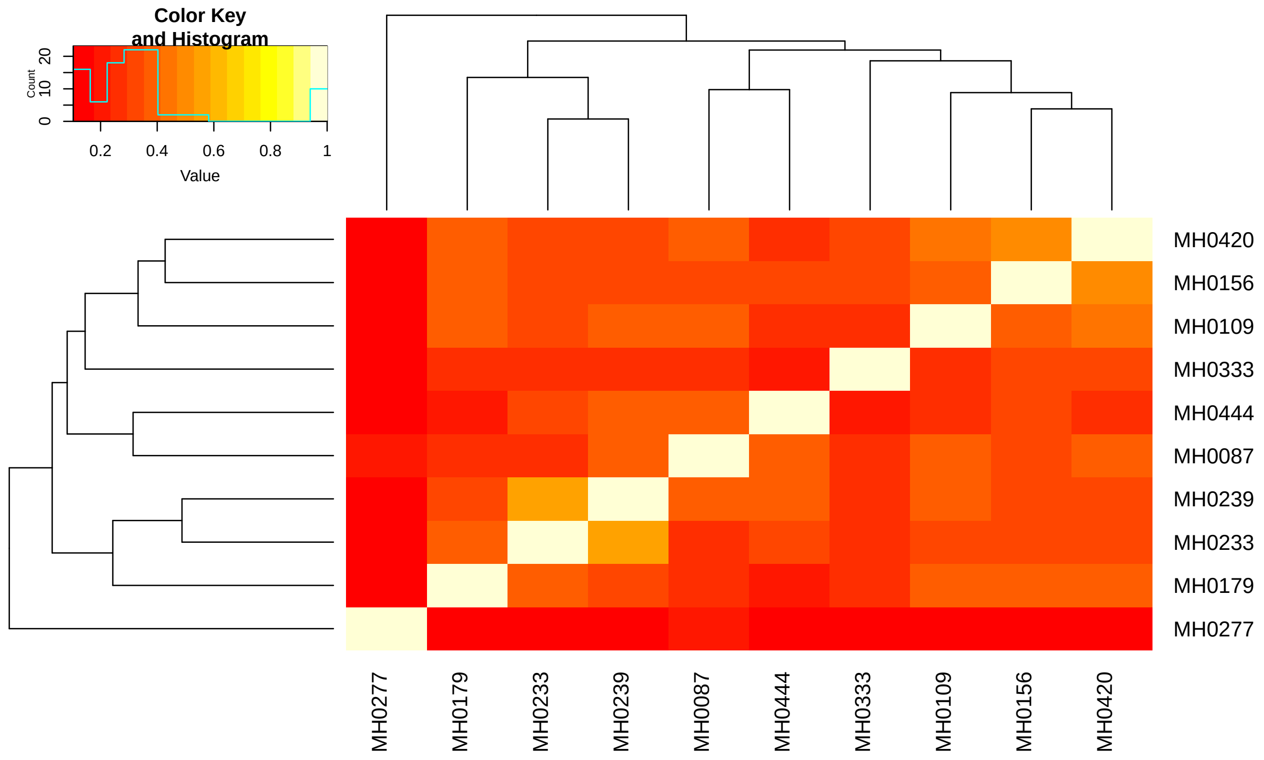

Figure 4: Sample heatmap of the correlation matrix clusterd with the ward approach.

```
str(hc.data)
```

```
## List of 3
##  $ mat.rho   : num [1:10, 1:10] 1 0.182 0.151 0.132 0.104 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:10] "MH0277" "MH0087" "MH0444" "MH0156" ...
##   .. ..$ : chr [1:10] "MH0277" "MH0087" "MH0444" "MH0156" ...
##  $ mat.dist  :Class 'dist'  atomic [1:45] 0.818 0.849 0.868 0.896 0.874 ...
##   .. ..- attr(*, "Labels")= chr [1:10] "MH0277" "MH0087" "MH0444" "MH0156" ...
##   .. ..- attr(*, "Size")= int 10
##   .. ..- attr(*, "call")= language as.dist.default(m = 1 - mat.rho)
##   .. ..- attr(*, "Diag")= logi FALSE
##   .. ..- attr(*, "Upper")= logi FALSE
##  $ mat.hclust:List of 7
##   ..$ merge      : int [1:9, 1:2] -6 -4 -10 -2 -7 -5 4 5 -1 -8 ...
##   ..$ height     : num [1:9] 0.471 0.523 0.608 0.623 0.686 ...
##   ..$ order      : int [1:10] 1 7 6 8 2 3 5 10 4 9
##   ..$ labels     : chr [1:10] "MH0277" "MH0087" "MH0444" "MH0156" ...
##   ..$ method     : chr "ward.D"
##   ..$ call       : language hclust(d = mat.dist, method = hclust.method)
##   ..$ dist.method: NULL
##   ..- attr(*, "class")= chr "hclust"
```

```
clust.order <- hc.data$mat.hclust$order
# order samples followin the hierarchical clustering
ordered.samples <- colnames(hs_3.3_metahit_sample_dat_freq[,1:10])[clust.order]
```

```r
# how close are the two first samples (spearman, rho)
hc.data$mat.rho[ordered.samples[1], ordered.samples[2]]
```

```
## [1] 0.1148695
```

Checking for consistency

When looking for possible contamination or mislabeling in order to make sure that samples in the dataset should not be related ,it is useful to use the `filt.hierClust` function. This routine will extract a subset of the inter-sample correlation matrix and focus on the samples that are closely related (above a given threshold) as illustrated in Figure 5. It also returns a table indicating for each samples the best correlated ones and displays a heatmap of the correlation matrix restricted to the samples correlated above the filter threshold (`plot=TRUE as default`).

```r
# Selecting the most closely related observations
close.samples <- filt.hierClust(hc.data$mat.rho, hclust.method = "ward.D",
                                plot = TRUE, filt = 0.45, size = 4)
```

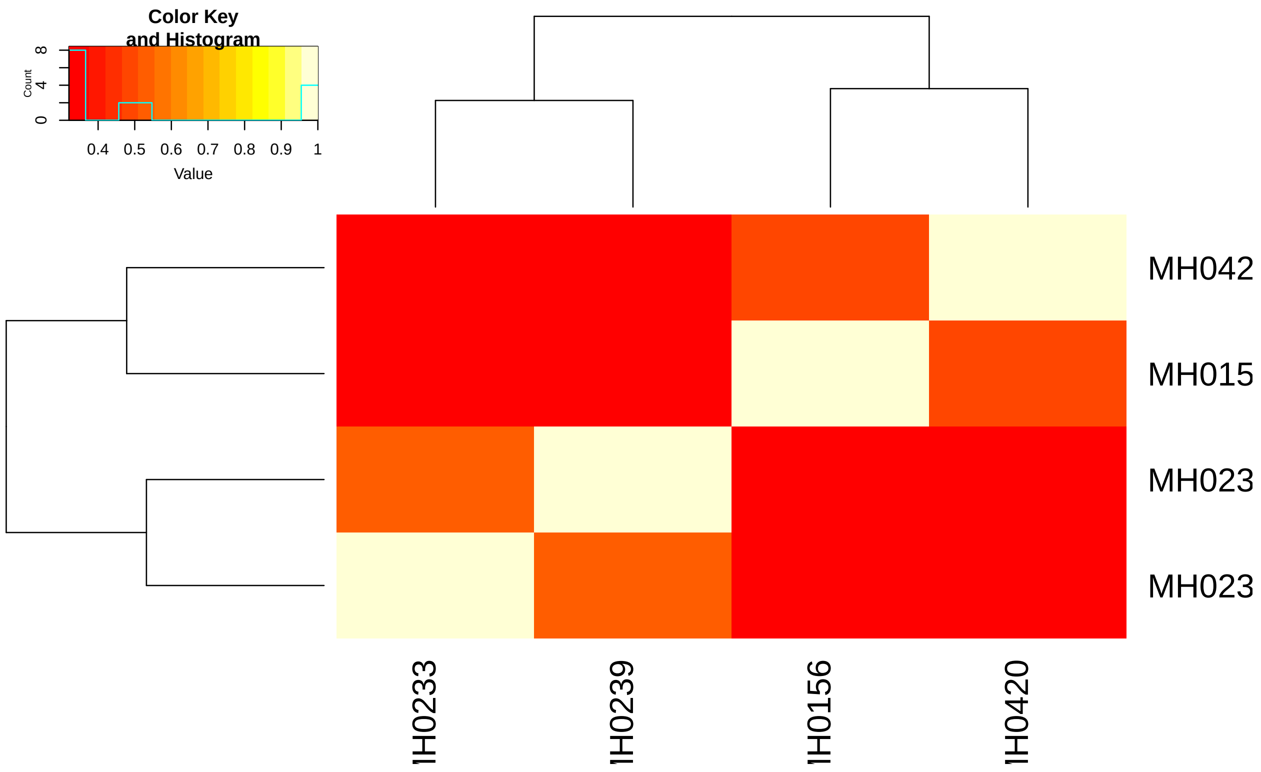

Figure 5: Heatmap of the most correlated observations.

```r
kable(head(close.samples)[,1:6])
```

	Hit_1	Hit_rho_1	Hit_2	Hit_rho_2	Hit_3	Hit_rho_3
MH0277	MH0087	0.182	MH0239	0.159	MH0109	0.156
MH0087	MH0444	0.377	MH0109	0.372	MH0420	0.352

	Hit_1	Hit_rho_1	Hit_2	Hit_rho_2	Hit_3	Hit_rho_3
MH0444	MH0239	0.392	MH0087	0.377	MH0156	0.333
MH0156	MH0420	0.477	MH0109	0.387	MH0179	0.350
MH0333	MH0420	0.332	MH0156	0.302	MH0233	0.259
MH0233	MH0239	0.529	MH0179	0.400	MH0420	0.322

Clustering genes - selecting the most correlated samples

Genes as other features of interest can be clustered using different techniques. In QM it makes sense biologically to cluster genes since they are indeed genetically linked together in the same molecular structure - ***the genome***. Based on this observation the metagenomic species (MGS) were proposed and published in 2014 *(Nielsen, Almeida et al, Nat Biotech, 2014)*. We have build multiple tools that will allow exploring these objects and here is a preview.

The mgs catalog

The MGS catalog can be built using different approaches. We supply in this package a subset of the MGS catalog that was computed in a large dataset in the MetaHIT 3.3M gene catalog. Briefly this is a list of gene (feature) identifiers.

```
# load the curated mgs data for the hs_3.3_metahit catalog
data("mgs_hs_3.3_metahit_sup500")
# the size of each MGS
unlist(lapply(mgs_hs_3.3_metahit_sup500,length))
```

```
## 10763_0_2   10766_2    10770_    10775_    10780_    1_11_2    11747_
##      1708      2249      2778      1436       599       783      4728
##    11752_    11757_      1_20   12719_1   12720_1    12723_       1_3
##      2274       811      2240      1113      2056      2198       827
##    13608_       1_4    1533_1     241_2      273_      319_   4373_16
##      1233       642       694      2931      2241       943      3001
```

Projecting genes onto the MGS catalog

In the following example we will cluster a number of genes in the MGS catalog. We call this: *projecting genes onto the MGS*. The notion of genebag (a bag of genes or features) is recurrent in the architecture of `momr`.

```
# Projecting a list of genes onto the mgs catalogue
genebag <- rownames(hs_3.3_metahit_sample_dat_freq)
mgs <- projectOntoMGS(genebag=genebag, list.mgs=mgs_hs_3.3_metahit_sup500)
length(genebag)
```

```
## [1] 5000
```

```
unlist(lapply(mgs,length))
```

```
##        1533_1          241_2           273_           319_        4373_16
##           131            495             54             80            142
## not_projected
##          4098
```

You can notice that these 5000 genes fall in 5 different MGS and that 4098 genes are not clustered. Indeed only approximately half of the catalog is clustered, due to different stringent criteria for QM purposes. Now that we know which gene is which MGS we can extract their profiles to explore them further.

```
# Extracting the profiles of a list of genes from the whole dataset
mgs.dat <- extractProfiles(mgs, hs_3.3_metahit_sample_dat_freq, silent=FALSE)
```

```
## [1] "Multiple profile extraction"
```

This is a list of data frames where we have a data frame for each MGS.

Visualizing MGS (the barcodes)

The barcode visualization is a very good tool for pattern discovery and recognition (Figure 6). It is a kind of heatmap where white A white color indicates absence and from light blue to dark red an increasing abundance. Each color step is a 4-fold in abundance.

```
# plot the barcodes
par(mfrow=c(length(mgs.dat),1), mar=c(1,0,0,0))
for(i in 1:length(mgs.dat)){
  plotBarcode(mgs.dat[[i]], main=names(mgs.dat)[i])
}
```

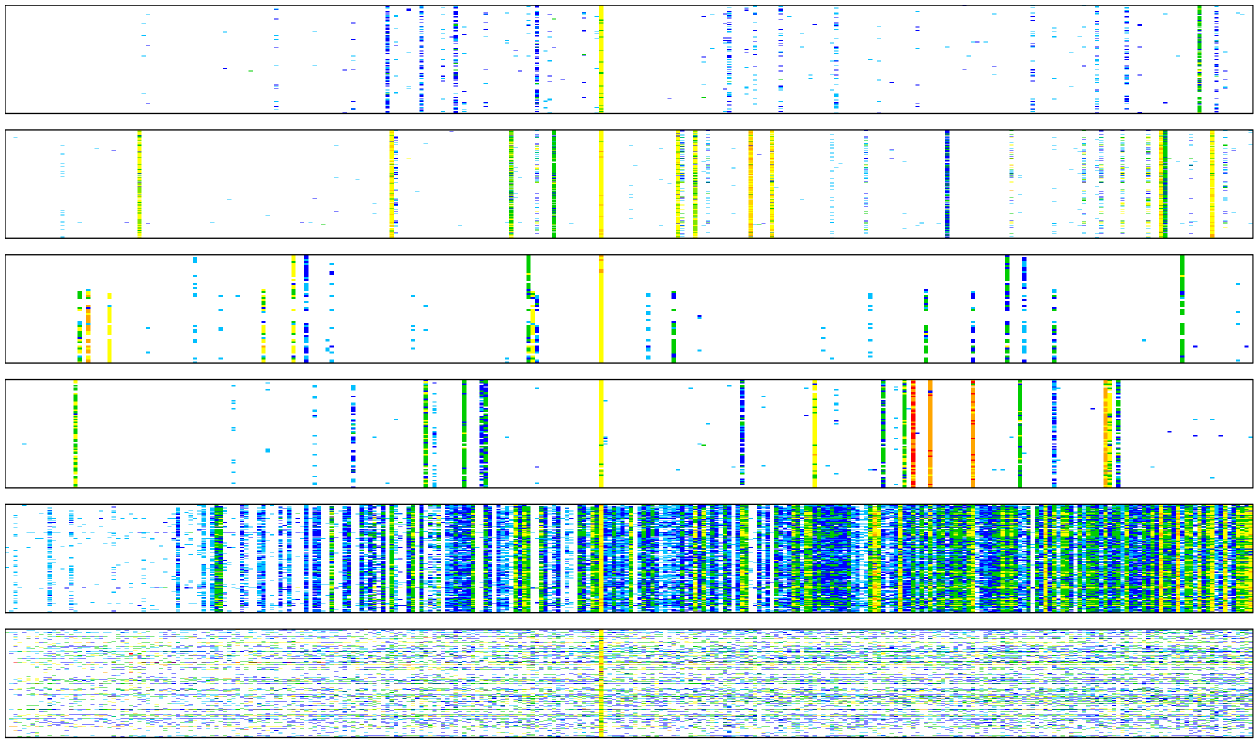

Figure 6: Barcodes of the MGS abundance profiles. Samples are in the columns and genes clustered togetehr in the MGS in the rows.

Reducing dimensions

The MGS can be transformed in simple tracer vectors using `computeFilteredVectors`. This allows to reduce dimensions and apply different statistical learning tools such as clustering (Figure 7). Different metrics can be used to compute this: the mean, the median or the sum are implemeted at the moment. The function returns a table of the calculated MGS signal in each sample.

```
# Computing the filtered vectors
mgs.mean.vect <- computeFilteredVectors(profile=mgs.dat, type="mean")
hierClust(t(mgs.mean.vect))
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

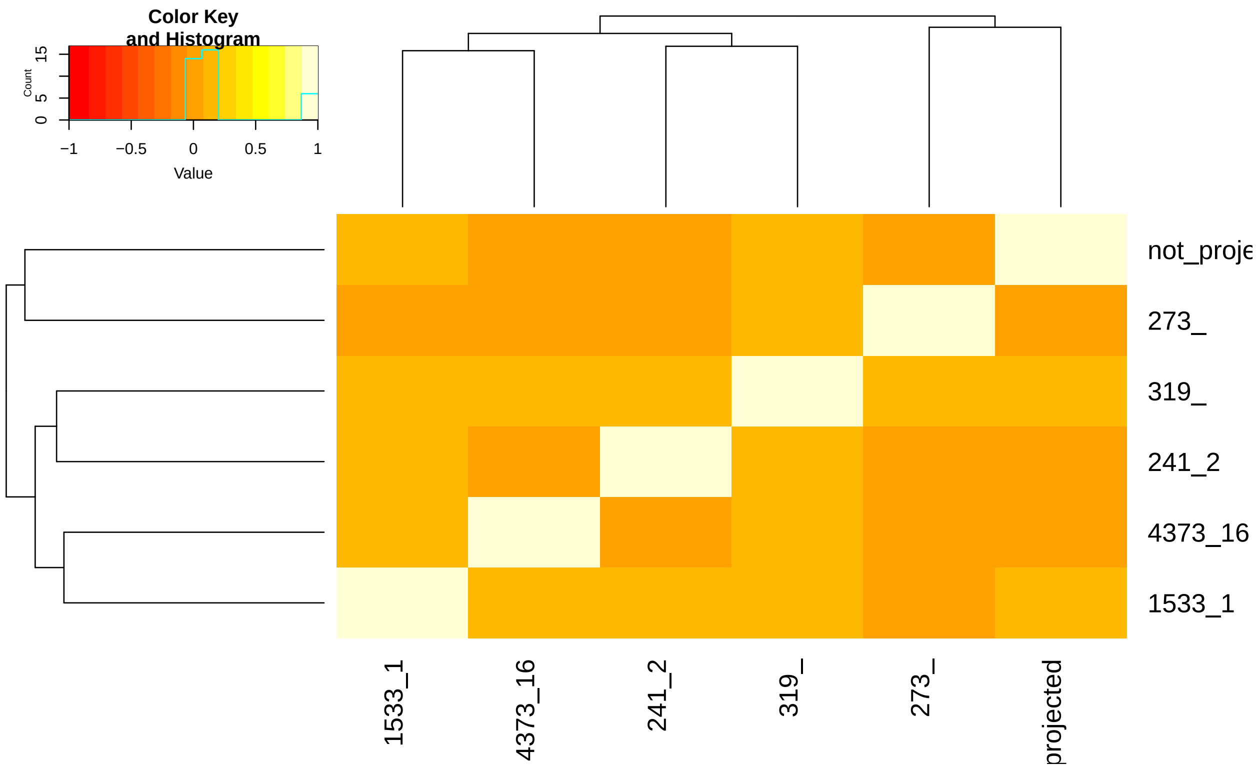

Figure 7: Similarity heatmap and clustering of the MGS.

```
## $mat.rho
##                   1533_1        241_2       273_       319_    4373_16
## 1533_1        1.00000000  0.1336279064 0.03183426 0.10000386 0.16266416
## 241_2         0.13362791  1.0000000000 0.05561149 0.13894849 0.05708167
## 273_          0.03183426  0.0556114912 1.00000000 0.07835247 0.01794230
## 319_          0.10000386  0.1389484853 0.07835247 1.00000000 0.15109205
## 4373_16       0.16266416  0.0570816696 0.01794230 0.15109205 1.00000000
## not_projected 0.07064679 -0.0005112535 0.03702869 0.08821160 0.03419235
##               not_projected
## 1533_1         0.0706467852
## 241_2         -0.0005112535
## 273_           0.0370286889
## 319_           0.0882116035
## 4373_16        0.0341923498
## not_projected  1.0000000000
##
## $mat.dist
##                 1533_1    241_2     273_      319_   4373_16
## 241_2         0.8663721
## 273_          0.9681657 0.9443885
## 319_          0.8999961 0.8610515 0.9216475
## 4373_16       0.8373358 0.9429183 0.9820577 0.8489079
## not_projected 0.9293532 1.0005113 0.9629713 0.9117884 0.9658077
##
## $mat.hclust
##
## Call:
## hclust(d = mat.dist, method = hclust.method)
##
## Cluster method   : ward.D
## Number of objects: 6
```

Identifying differentially abundant features

Another interesting function is `testRelations`, which allows to identify features (genes, MGS, etc) that are differentially abundant between two groups of samples or correlate with some quantitative variable.

```
# for the first 500 genes
class <- c(rep(1,150),rep(2,142))
res.test <- testRelations(data=hs_3.3_metahit_sample_dat_freq[1:500,],
                          trait=class,type="wilcoxon")
print(paste("There are",sum(res.test$p<0.05, na.rm=TRUE),"significant genes and",
            sum(res.test$q<0.05, na.rm=TRUE), "after adjustment for multiple testing"))
```

```
## [1] "There are 135 significant genes and 93 after adjustment for multiple testing"
```

```
# keep the significant genes
res.test <- res.test[res.test$q < 0.05 & !is.na(res.test$q),]
# sort tham by status and q-value
res.test <- res.test[order(res.test$status,res.test$q),]
kable(head(res.test))
```

	rho	rho2	p	q	status
5300	NA	NA	0.0010278	0.0073287	1
5172	NA	NA	0.0010929	0.0076815	1
5272	NA	NA	0.0011985	0.0081897	1
5190	NA	NA	0.0021005	0.0130817	1
5385	NA	NA	0.0033330	0.0202449	1
5191	NA	NA	0.0052412	0.0306984	1

```r
table(res.test$status)
```

```
##
##  1  2
##  9 84
```

```r
# test weather the MGS are also differentially abundant with the class
res.test.mgs <- testRelations(data=mgs.mean.vect, trait=class,type="wilcoxon")
kable(res.test.mgs[res.test.mgs$q<0.05,])
```

	rho	rho2	p	q	status
319_	NA	NA	0.0007663	0.0022988	2
4373_16	NA	NA	0.0000000	0.0000000	2

In the example above we tested whether the first 500 genes of this test dataset are differentially abundant between the groups 1 and 2. Indeed there are 9 genes enriched in 1 and 84 in the second group after multiple testing adjustment. We performed this test also on the vectors of the MGS and two of them are also differentially abundant.

Conclusion

`momr` is a very useful package for quantitative metagenomics and only the main functionalities are described here. The package also allows to perform // computing using the map-reduce principles. We are constantly optimizing algorithms and adding new tools so that it really becomes easy to explore QM datasets. The authors would like to acknowledge the very exciting and fruitful environment that MetaHIT community created.

References

1. Le Chatelier, Emmanuelle, Trine Nielsen, Junjie Qin, Edi Prifti, Falk Hildebrand, Gwen Falony, Mathieu Almeida, et al "Richness of human gut microbiome correlates with metabolic markers." Nature 500, no. 7464 (April 9, 2014): 541–546.

2. Qin, Junjie, Ruiqiang Li, Jeroen Raes, Manimozhiyan Arumugam, Kristoffer Solvsten Burgdorf, Chaysavanh Manichanh, Trine Nielsen, et al "A human gut microbial gene catalogue established by metagenomic sequencing." Nature 464, no. 7285 (March 4, 2010): 59–65.

3. Mortazavi, Ali, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. "Mapping and quantifying mammalian transcriptomes by RNA-Seq.." Nature Methods 5, no. 7 (July 2008): 621–628.

4. Nielsen, H Bjørn, Mathieu Almeida, Agnieszka Sierakowska Juncker, Simon Rasmussen, Junhua Li, Shinichi Sunagawa, Damian R Plichta, et al "Identification and assembly of genomes and genetic elements in complex metagenomic samples without using reference genomes." Nature biotechnology (July 6, 2014): 1–11.