CluMSID — Clustering of MS^2 Spectra for Metabolite Identification

A General Tutorial.

Tobias Depke

December 31, 2018

Contents

Introduction

This tutorial shows how to use the `CluMSID` package to help annotate MS^2 spectra from untargeted LC-MS/MS data. `CluMSID` works with MS^2 data generated by data-dependent acquisition and requires an mzXML file (like in this example) or any other file that can be parsed by `mzR`, like mzML, mzTab or netCDF, as input. It can be used both stand-alone and together with the XCMS suite of preprocessing tools.

`CluMSID` extracts and merges MS^2 spectra and generates neutral loss patterns for each feature. Additionally, it can make use of information from the `CAMERA` package to generate pseudospectra from MS^1 level data. The tool uses cosine similarity to generate distance matrices from MS^2 spectra, neutral loss patterns and pseudospectra.

These distance matrices are the basis for multivariate statistics methods such as multidimensional scaling, density-based clustering, hierarchical clustering and correlation networks. The `CluMSID` package provides functions for these methods including (interactive) visualisation but the distance/similarity data can also be analysed with other `R` functions.

For the demonstrations in this tutorial, we will mainly use data from pooled *Pseudomonas aeruginosa* cell extracts, measured in ESI-(+) mode with auto-MS/MS on a Bruker maxis^{HD} qTOF after reversed phase separation by UPLC. For details, please refer to the Depke *et al.* 2017 publication (doi: 10.1016/j.jchromb.2017.06.002.).

To be able to access the example data, we also need the related package `CluMSIDdata`. Both packages are available from Bioconductor, starting from version 3.9, and can be installed as follows:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(c("CluMSIDdata", "CluMSID"))
```

Before the release of R 3.6 in April/May 2019, the installation from Bioconductor requires the user to install the development versions of R and Bioconductor. For those who wish to avoid working with devel version, R 3.5 compatible versions of `CluMSID` and `CluMSIDdata` are available from GitHub and can be installed as follows:

```
if (!requireNamespace("devtools", quietly = TRUE)) install.packages("devtools")
devtools::install_github("tdepke/CluMSIDdata", ref = "pkg")
devtools::install_github("tdepke/CluMSID", ref = "pkg")
```

Once installed, both packages are loaded, along with `tidyverse` which we will use later.

```
library(CluMSID)
library(CluMSIDdata)
library(tidyverse)
```

`MS2spectrum` and `pseudospectrum` classes

`CluMSID` uses a custom S4 class named `MS2spectrum` to store spectral information in the following slots:

- `id`: a character string similar to the ID used by XCMSonline or the ID given in a predefined peak list
- `annotation`: a character string containing a user-defined annotation, defaults to empty
- `precursor`: (median) *m/z* of the spectrum's precursor ion
- `rt`: (median) retention time of the spectrum's precursor ion
- `polarity`: the polarity with which the spectrum was recorded, either `positive` or `negative`
- `spectrum`: the actual MS^2 spectrum as two-column matrix (column 1 is (median) *m/z*, column 2 is (median) intensity of the product ions)
- `neutral_losses`: a neutral loss pattern generated by subtracting the product ion mass-to-charge ratios from the precursor *m/z* in a matrix format analogous to the `spectrum` slot

The `pseudospectrum` class is very similar but it contains no information on precursor *m/z* and therefore no neutral loss pattern, either. By default, the `id` slot contains the "pcgroup" number assigned by `CAMERA`.

The individual slots of MS2spectrum and pseudospectrum objects can be accessed via the standard S4 way using object@slot, e.g. object@annotation or by using an accessor function. These exist for all slots and are called accessFoo(), where Foo is the slot name (not exactly, though, because Bioconductor does not allow to mix snake_case and camelCase in function names):

- accessID(object)
- accessAnnotation(object)
- accessPrecursor(object)
- accessRT(object)
- accessPolarity(object)
- accessSpectrum(object)
- accessNeutralLosses(object).

Extract MS^2 spectra from *.mzXML file

The first step in the CluMSID workflow is to extract MS^2 spectra from the raw data file (in mzXML format). This is done by the extractMS2spectra function which internally uses several functions from the mzR package. The function offers the possibility to filter spectra that contain less a defined number of peaks and/or do not fall in a defined retention time window. Setting the recalibrate_precursor argument to TRUE activates a correction process for uncalibrated precursor *m/z* data that existed in older version of Bruker's Compass Xport (*cf.* Depke *et al.* 2017). It is not necessary to use it with files generated by other software but does not corrupt the data, either.

Please be aware that mzR often throws warnings concerning the Rcpp version that can usually be ignored.

```
ms2list <- extractMS2spectra(system.file("extdata",
                                         "PoolA_R_SE.mzXML",
                                         package = "CluMSIDdata"),
                             min_peaks = 2,
                             recalibrate_precursor = TRUE,
                             RTlims = c(0,25))
```

This operation has now extracted all the MS^2 spectra from the raw data file and stored them in a list. Each list entry is an object of class MS2spectrum. The list is quite long because it still contains a lot of spectra that derive from the same chromatographic peak.

```
length(ms2list)
#> [1] 2290
```

In our example, the first two spectra in the list derive from the same peak and thus have the same precursor ion and almost the same retention time.

```
head(ms2list, 4)
#> [[1]]
#> An object of class "MS2spectrum"
#>  id:
#>  annotation:
#>  precursor: 146.1652
#>  retention time: 56.266
#>  polarity: positive
#>  MS2 spectrum with 2 fragment peaks
#>  neutral loss pattern with 0 neutral losses
#> [[2]]
#> An object of class "MS2spectrum"
#>  id:
#>  annotation:
```

```
#>  precursor: 146.1653
#>  retention time: 57.292
#>  polarity: positive
#>  MS2 spectrum with 3 fragment peaks
#>  neutral loss pattern with 0 neutral losses
#> [[3]]
#> An object of class "MS2spectrum"
#>  id:
#>  annotation:
#>  precursor: 129.1387
#>  retention time: 57.545
#>  polarity: positive
#>  MS2 spectrum with 2 fragment peaks
#>  neutral loss pattern with 0 neutral losses
#> [[4]]
#> An object of class "MS2spectrum"
#>  id:
#>  annotation:
#>  precursor: 112.1119
#>  retention time: 57.797
#>  polarity: positive
#>  MS2 spectrum with 2 fragment peaks
#>  neutral loss pattern with 0 neutral losses
```

From the output above, you also see that the `MS2spectrum` class has a `show()` generic that summarises the MS^2 spectrum and neutral loss pattern data. To show the default output, use `showDefault()`. Be aware that neutral loss patterns have not been calculated in this step.

```
showDefault(ms2list[[2]])
#> An object of class "MS2spectrum"
#> Slot "id":
#> character(0)
#>
#> Slot "annotation":
#> character(0)
#>
#> Slot "precursor":
#> [1] 146.1653
#>
#> Slot "rt":
#> [1] 57.292
#>
#> Slot "polarity":
#> [1] "positive"
#>
#> Slot "spectrum":
#>           [,1] [,2]
#> [1,]  72.08064 2448
#> [2,]  84.08077  328
#> [3,] 112.11228  843
#>
#> Slot "neutral_losses":
#> <0 x 0 matrix>
```

Merge MS^2 spectra that derive from the same peak/feature

To reduce the amount of redundant MS^2 spectra, the `mergeMS2spectra()` function is used to generate consensus spectra from the MS^2 spectra that derive from the same precursor. `CluMSID` offers two possibilities to do so:

Merge spectra without external peaktable

This possibility is the standard method for stand-alone use of `CluMSID` and is equivalent to what has been described in Depke *et al.* 2017. It does not need additional input and summarises consecutive spectra that have the same precursor *m/z* if their retention time fall within a defined threshold (`rt_tolerance`, defaults to 30s). A retention time difference between consecutive spectra larger than `rt_tolerance` is interpreted as chromatographic separation and respective spectra will be assigned to a new feature. The `mz_tolerance` argument should be set according to your instruments *m/z* precision, the default is $1 * 10^{-5}$ (10ppm, equivalent to $\pm$5ppm instrument precision). The `peaktable` and `exclude_unmatched` arguments are not used in this method and are to be left at their default.

```
featlist <- mergeMS2spectra(ms2list)
```

```
length(featlist)
#> [1] 518
```

```
head(featlist, 4)
#> [[1]]
#> An object of class "MS2spectrum"
#>  id: M146.17T59.35
#>  annotation:
#>  precursor: 146.1653
#>  retention time: 59.35
#>  polarity: positive
#>  MS2 spectrum with 8 fragment peaks
#>  neutral loss pattern with 7 neutral losses
#> [[2]]
#> An object of class "MS2spectrum"
#>  id: M129.14T58.57
#>  annotation:
#>  precursor: 129.1387
#>  retention time: 58.57
#>  polarity: positive
#>  MS2 spectrum with 4 fragment peaks
#>  neutral loss pattern with 3 neutral losses
#> [[3]]
#> An object of class "MS2spectrum"
#>  id: M112.11T57.8
#>  annotation:
#>  precursor: 112.1119
#>  retention time: 57.8
#>  polarity: positive
#>  MS2 spectrum with 2 fragment peaks
#>  neutral loss pattern with 1 neutral losses
#> [[4]]
#> An object of class "MS2spectrum"
#>  id: M251.16T60.64
#>  annotation:
```

```
#>  precursor: 251.1603
#>  retention time: 60.64
#>  polarity: positive
#>  MS2 spectrum with 9 fragment peaks
#>  neutral loss pattern with 8 neutral losses
```

The total amount of spectra was reduced from 2290 to 518 and as many other, the redundant spectra #1 and #2 in the raw list are now merged to one consensus spectrum (#1 in the merged list).

In this step, neutral loss patterns have been generated that look like this:

```
accessNeutralLosses(featlist[[1]])
#>          [,1] [,2]
#> [1,] 74.08475 6429
#> [2,] 73.08163  262
#> [3,] 71.07394  239
#> [4,] 62.08476 1044
#> [5,] 34.05341 2363
#> [6,] 33.05024  144
#> [7,] 17.02688  852
```

Merge spectra with external peaktable, e.g. from XCMS

The second possibility is to supply a peaktable, i.e. a list of picked peaks with their mass-to-charge ratios and retention times. This is particularly useful if you want to annotate a complete metabolomics data set. In our example, we have a metabolomics dataset called "TD035" in which we have measured a range of samples in MS^1 mode for relative quantification. Additionally, we have measured a pooled QC sample in MS^2 mode for annotation. The MS^1 data were analysed using XCMSonline and we want to group the MS^2 spectra so that they match the XCMSonline peak picking.

The spectra are extracted as shown above:

```
ms2list2 <- extractMS2spectra(system.file("extdata",
                                           "TD035-PoolMSMS2.mzXML",
                                           package = "CluMSIDdata"),
                              min_peaks = 2,
                              recalibrate_precursor = TRUE,
                              RTlims = c(0,25))
```

The peaklist is imported from the XCMSonline output. The list has to contain at least 3 columns:

- column 1: name/identifier of the feature
- column 2: m/z
- column 3: retention time

Shown below is an easy way of getting from an XCMSonline annotated diffreport to a suitable peaktable using `tidyverse` functions. Of course, you can achieve the same goal with base R functions or even in Excel. Depending on the retention time format in your *.mzXML file, you might have to convert from minutes to seconds or vice versa. Here, we have minutes in the XCMSonline output but seconds in the MS^2 file, so we multiply by 60.

```
ptable <- read_delim(file = system.file("extdata",
                                        "TD035_XCMS.annotated.diffreport.tsv",
                                        package = "CluMSIDdata"),
                     delim = "\t") %>%
    select(c(name, mzmed, rtmed)) %>%
    mutate(rtmed = rtmed * 60)
```

```
head(ptable)
#> # A tibble: 6 x 3
#>   name     mzmed  rtmed
#>   <chr>    <dbl>  <dbl>
#> 1 M245T2   245.   100.
#> 2 M440T2_1 440.   107.
#> 3 M578T2   578.   104.
#> 4 M85T1     85.0   60.8
#> 5 M126T1_1 126.    61.0
#> 6 M688T24  688.  1468.
```

We can now use this peaktable as an argument for `mergeMS2spectra()`. You can choose whether you want to keep or exclude MS^2 spectra that do not match any peak in the peaktable. These can occur in regions of the chromatogramm where there are no clear peaks but the auto-MS/MS still fragments the most abundant ions. These unmatched spectra are merged following the same rules as described above (method without peaktable). In this example, we keep the unmatched spectra. We use the default values for *m/z* and retention time tolerance and thus do not need to specify them.

```
featlist2 <- mergeMS2spectra(ms2list2,
                             peaktable = ptable,
                             exclude_unmatched = FALSE)

head(featlist2, 4)
#> [[1]]
#> An object of class "MS2spectrum"
#>  id: M213T0
#>  annotation:
#>  precursor: 213.1462
#>  retention time: 6.04
#>  polarity: positive
#>  MS2 spectrum with 5 fragment peaks
#>  neutral loss pattern with 3 neutral losses
#> [[2]]
#> An object of class "MS2spectrum"
#>  id: xM158T31.17
#>  annotation:
#>  precursor: 158.0027
#>  retention time: 31.17
#>  polarity: positive
#>  MS2 spectrum with 3 fragment peaks
#>  neutral loss pattern with 3 neutral losses
#> [[3]]
#> An object of class "MS2spectrum"
#>  id: M146T1_3
#>  annotation:
#>  precursor: 146.1650
#>  retention time: 61.15
#>  polarity: positive
#>  MS2 spectrum with 7 fragment peaks
#>  neutral loss pattern with 6 neutral losses
#> [[4]]
#> An object of class "MS2spectrum"
#>  id: M129T1_4
#>  annotation:
```

```
#>  precursor: 129.1384
#>  retention time: 60.74
#>  polarity: positive
#>  MS2 spectrum with 2 fragment peaks
#>  neutral loss pattern with 2 neutral losses
```

Note that the 2nd entry in `featlist2` is marked with an 'x' which means that it could not be assigned to a feature in the peaktable.

For the sake of simplicity, only the data generated from the stand-alone procedure will be used for the following examples. Be assured that all of them would also work with the data generated with the help of an external peaktable (`featlist2`).

Add annotations

The next step is to add (external) annotations to the list of features, e.g. from a spectral library that you curate in-house or one that has been supplied by your instrument manufacturer. If you do not (want to) annotate your features at all, this step can be skipped completely, leaving the `annotation` slot of the `MS2spectrum` objects empty.

Manual procedure

`CluMSID` offers several possibilities to add annotations to your feature list. The most basic one first generates a list of features and saves it as *.csv file. For that you use the `writeFeaturelist()` function and only have to specify your list of spectra and a file name for the output file (here: `pre_anno.csv`). You can then manually fill in your annotations in a new column in the table, save it (in this example under the name `post_anno.csv`) and reload it to `R`:

```
writeFeaturelist(featlist, "pre_anno.csv")
```

```
annotatedSpeclist <- addAnnotations(featlist,
                                    system.file("extdata",
                                                "post_anno.csv",
                                                package = "CluMSIDdata"))
```

`annotatedSpeclist` will then be equivalent to `featlist` with annotations added to the `annotation` slot of the list entries.

Alternative procedures

You can add annotations without leaving the `R` environment, too. `addAnnotations()` also accepts objects of class `data.frame` as `annolist` argument. Be aware that `addAnnotations()` assigns the annotation based on the position in the feature list. I.e., if the order of the features in your list of features (`featlist`) and your list of annotations (`annolist`) is different, you will get nonsense results.

The savest ways to `addAnnotations()` with a `data.frame` is to use `Featurelist()` to generate a `data.frame` that is formatted in the same way as the file output from `writeFeaturelist()` and then match your identifications against this `data.frame` and use the result as argument for `addAnnotations()`.

Say you have an object called `annos` that contains feature IDs (the same as in `featlist`) and annotations in a two-column `data.frame` with `"id"` and `"annotation"` as column names. It could look like this:

```
str(annos)
#> 'data.frame':    154 obs. of  2 variables:
#>  $ id        : chr  "M146.17T59.35" "M129.14T58.57" "M112.11T57.8" "M148.06T69.65" ...
#>  $ annotation: chr  "spermidine" "spermidine (fragment)" "spermidine (fragment)" "glutamate" ...
head(annos)
#>              id            annotation
#> 1 M146.17T59.35            spermidine
#> 2 M129.14T58.57 spermidine (fragment)
#> 3  M112.11T57.8 spermidine (fragment)
#> 4 M148.06T69.65             glutamate
#> 5 M130.05T69.64  glutamate (fragment)
#> 6 M179.06T71.32        gluconolactone
```

`addAnnotations(featlist, annos, annotationColumn = 2)` will throw an error because `featlist` and `annos` are of different length. Instead, you need to do the following:

```
fl <- featureList(featlist)

fl_annos <- dplyr::left_join(fl, annos, by = "id")
```

Now, you can annotate your list of spectra using `addAnnotations(featlist, fl_annos, annotationColumn = 4)`.

An analogous procedure works if you have your annotations stored in a peaktable that you have used for `mergeMSspectra()`. As the order of spectra in the list will not be same as the order of features in your peaktable, you need to do a matching with the output of `featureList()` as well.

Generate distance matrices

Once we have a list of `MS2spectrum` objects containing all the required information with or without annotation, we can generate distance matrices from (product ion) MS^2 spectra as well as from neutral loss patterns. These distance matrices serve as the basis for further analysis of the data. Both for MS^2 spectra and neutral loss patterns, cosine similarity is used as similarity metric:

$$cos(\theta) = \frac{\sum_i a_i \cdot b_i}{\sqrt{\sum_i {a_i}^2 \cdot \sum_i {b_i}^2}}$$

Distance matrix for product ion spectra

For most applications, analysing the similarity of product ion MS^2 spectra will be most useful. The generation of the distance matrix is done by just one simple command but it can take some time to calculate.

```
distmat <- distanceMatrix(annotatedSpeclist)
```

Distance matrix for neutral loss patterns

Common neutral losses and neutral loss patterns can convey information about structural similarity, as well, e.g. with nucleotides or glykosylated secondary metabolites. `CluMSID` offers the possibility to study neutral loss patterns independently from product ion spectra. The generation of a distance matrix is analogous, you just need to set the `type` argument to `"neutral_losses"`:

```
nlmat <- distanceMatrix(annotatedSpeclist, type = "neutral_losses")
```

Visualise distance/similarity data using multidimensional scaling (MDS)

One rather simple possibility to visually analyse the spectral similarity data is multidimensional scaling, a dimension reduction method that simplifies distances in *n*-dimensional space to those in two-dimensional space (*n* in this case being the number of consensus spectra or neutral loss patterns that were used to generate the distance matrix in the previous step). `CluMSID` offers a simple function to produce an MDS plot from the distance matrix with the option to highlight annotated metabolites and the possibility to generate an interactive plot using `plotly`.

Standard MDS plots are generated as follows:

For MS^2 spectra:

```
MDSplot(distmat, highlight_annotated = TRUE)
```

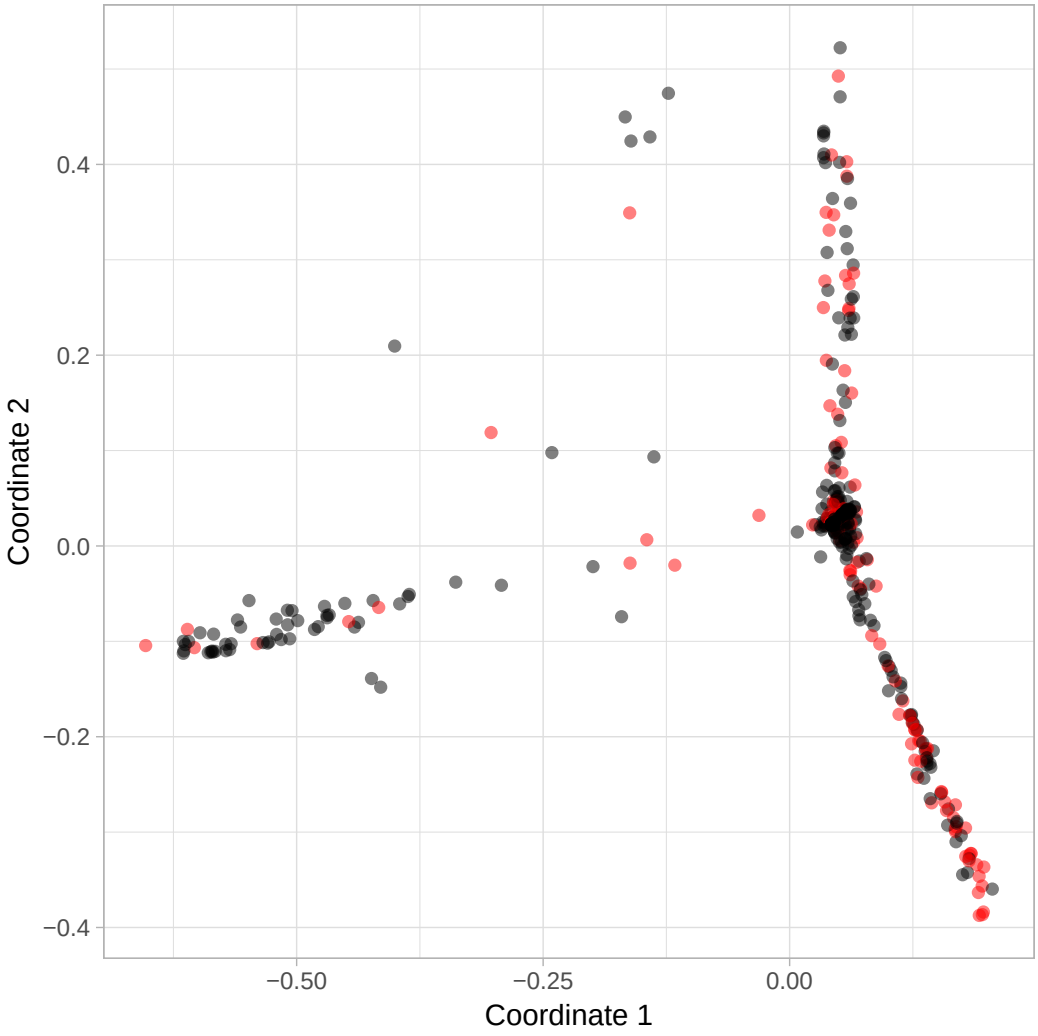

Figure A-1: Multidimensional scaling plot as a visualisation of MS^2 spectra similarities of the example data set. Red dots signify annotated spectra, black dots spectra from unknown metabolites.

For neutral loss patterns:

```
MDSplot(nlmat, highlight_annotated = TRUE)
```

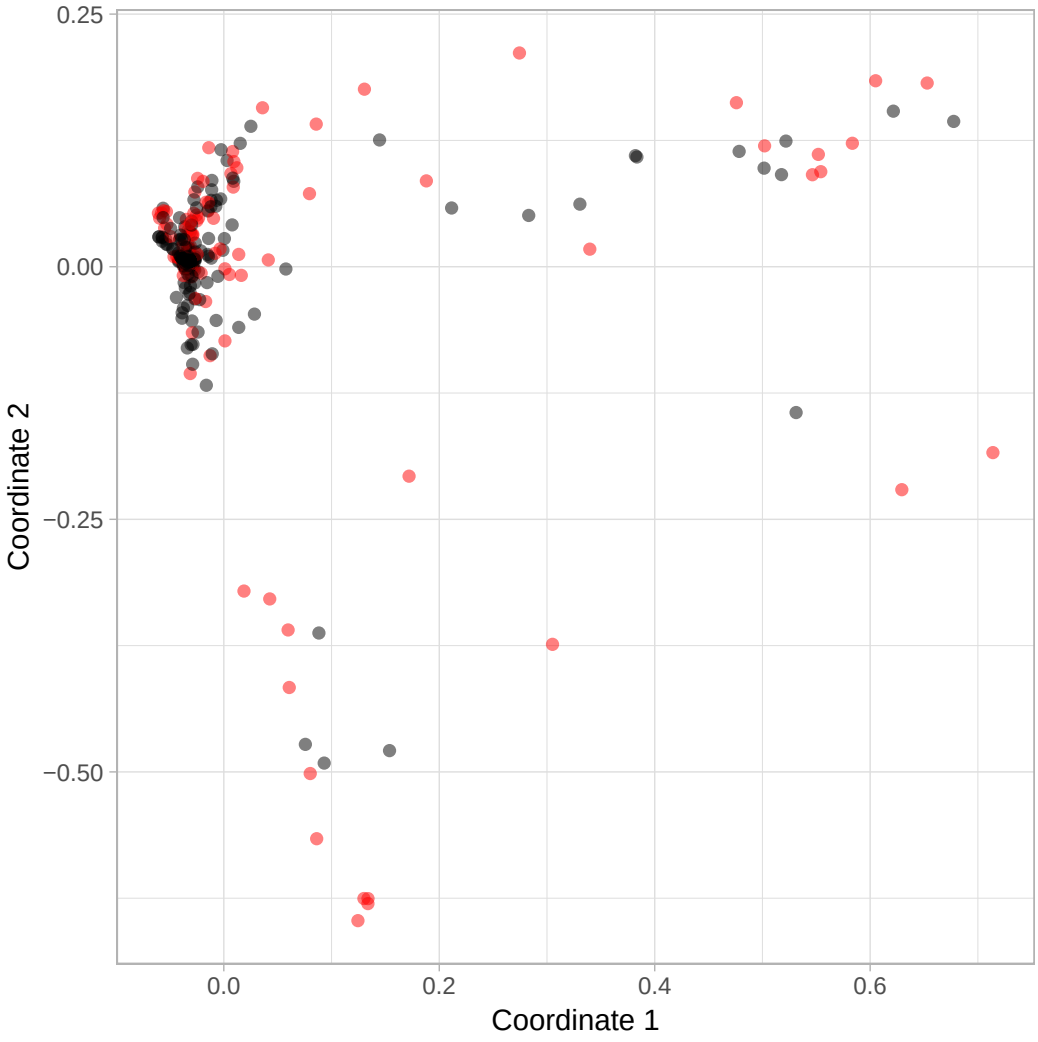

Figure A-2: Multidimensional scaling plot as a visualisation of neutral loss similarities of the example data set. Red dots signify annotated spectra, black dots spectra from unknown metabolites.

Interactive plots are zoomable and show feature names upon mouse-over. They are generated like normal MDS plots and can be viewed within RStudio or—after saving as html file using `htmlwidgets`—displayed in a normal web browser.

```
my_mds <- MDSplot(distmat, interactive = TRUE, highlight_annotated = TRUE)

htmlwidgets::saveWidget(my_mds, "mds.html")
```

This is how it looks like if you open the html file in Firefox and mouse over a feature:

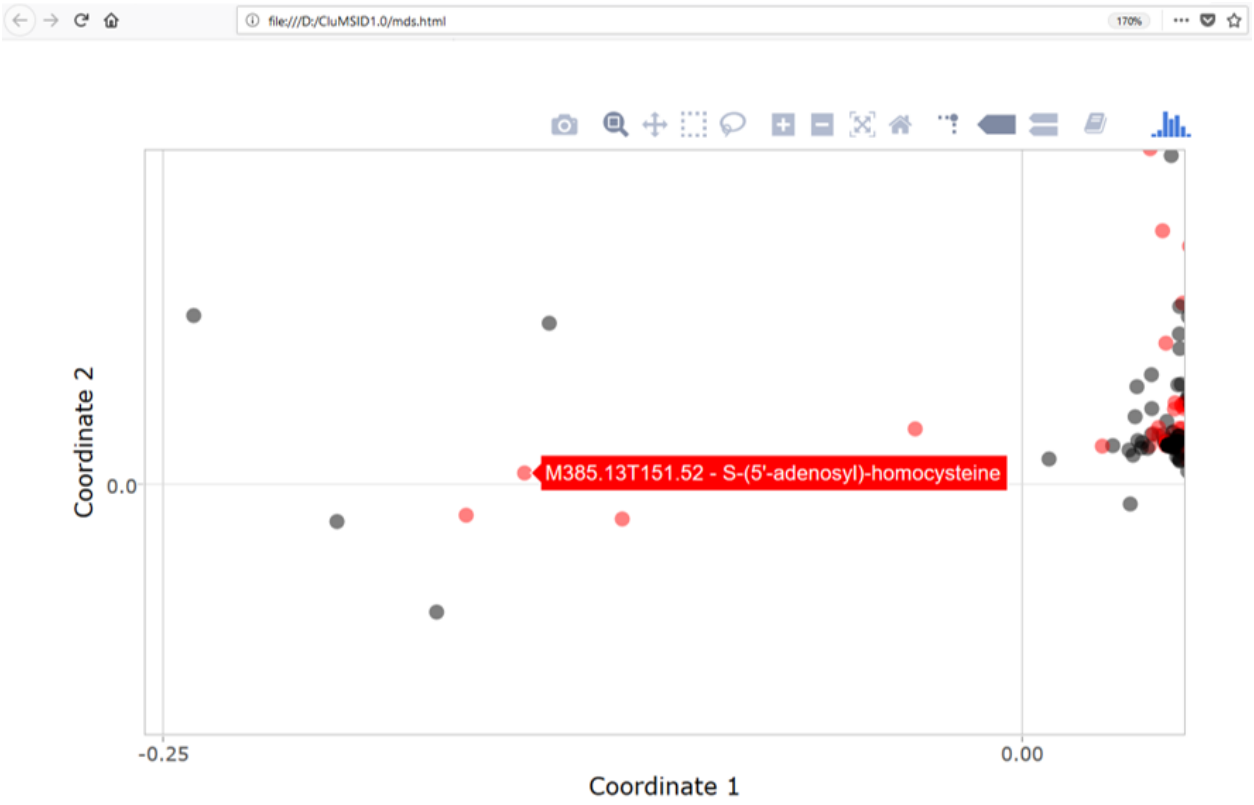

Figure A-3: Screenshot of the interactive version of the Multidimensional scaling plot visualising MS^2 spectra similarities of the example data set (cf Figure 1). Zoomed image section with tooltip displaying feature information upon mouse-over.

Perform density-based clustering using the OPTICS algorithm

For density-based clustering with `CluMSID`, the 'OPTICS' algorithm and its implementation in the `dbscan` package is used. Density-based clustering is a useful clustering method that often yields different results than hierarchical clustering and can thus provide additional insight into the data. `CluMSID` has two functions to perform density-based clustering, one for the reachability plot which is the most useful visualisation of OPTICS results and one that outputs a `data.frame` containing the cluster assignations for every feature.

Both functions require as arguments a distance matrix as well as three parameters for the underlying functions `dbscan::optics` and `dbscan::extractDBSCAN`: `eps`, `minPts` and `eps_cl`. Lowering the `eps` parameter (default is 10000) limits the size of the epsilon neighbourhood which from experience has very little effect on the results. `minPts` defaults to 3 in `CluMSID`. It defines how many points are considered for reachability distance calculation between clusters. The `dbscan::optics` default for `minPts` is 5. Users are encourage to experiment with this parameter. `eps_cl` is the reachability threshold to identify clusters and can be varied based on your data. Lowering `eps_cl` leads to a larger number of smaller clusters and vice versa for raising the value. In general, it is advisable to chose a higher `eps_cl` for MS^2 spectra than for neutral loss patterns, since the latter tend to show less similarity to each other. For details, please refer to the `dbscan` help for the `dbscan::optics` and `dbscan::extractDBSCAN` functions.

If the default parameters are used, the generation of an OPTICS reachability plots is very simple, shown here for MS^2 spectra and neutral loss patterns:

```
OPTICSplot(distmat)
```

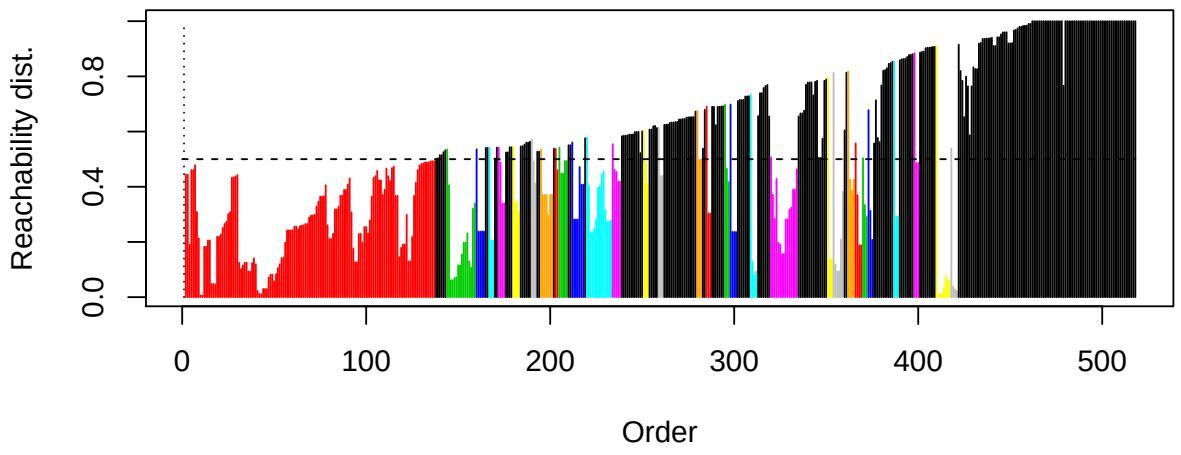

Figure A-4: Reachability distance plot resulting from OPTICS density based clustering of the MS^2 spectra similarities of the example data set. Bars represent features in OPTICS order with heights corresponding to the reachability distance to the next feature. The dashed horizontal line marks the reachability threshold that separates clusters. The resulting clusters are colour-coded with black representing noise, i.e. features not assigned to any cluster.

```
OPTICSplot(nlmat, eps_cl = 0.7)
```

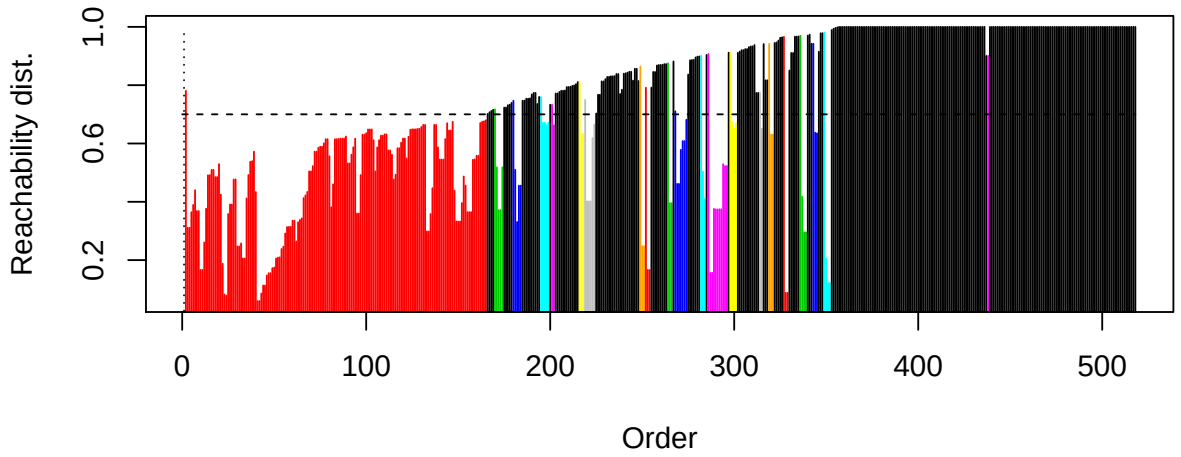

Figure A-5: Reachability distance plot resulting from OPTICS density based clustering of the neutral loss similarities of the example data set (cf Figure 4).

In the reachability plots, every line represents a feature and the height of the line is the reachability distance to the next feature in the OPTICS order. Thus, valleys represent groups of similar spectra or neutral loss patterns. The order and the cluster assignment can be studied using the `OPTICStbl` function that outputs a three-column `data.frame` with feature id, cluster assignment and OPTICS order. The order of features in the `data.frame` corresponds to the original order in the input distance matrix. Features that were not assigned to a cluster are black in the reachability plot and have the cluster ID 0. `OPTICStbl` takes the same arguments as `OPTICSplot`. The two functions have to be run with exactly the same parameters to assure compatibility of results.

```
OPTICStable <- OPTICStbl(distmat)

head(OPTICStable)
#>                                 feature cluster_ID OPTICS_order
#> 1            M146.17T59.35 - spermidine          1            1
#> 2 M129.14T58.57 - spermidine (fragment)          1            3
#> 3  M112.11T57.8 - spermidine (fragment)          1            4
#> 4                         M251.16T60.64          0          185
#> 5                         M212.85T65.02          0          518
#> 6                         M290.85T64.76          0          517
```

Perform hierarchical clustering

In Depke *et al.* 2017, hierarchical clustering proved the most useful method to unveil structural similarities between features. analogous to density-based clustering, `CluMSID` offers two functions, one for plots and one for a `data.frame` with cluster assignments, both taking a distance matrix as the only compulsory argument. The other two parameters are `h` (defaults to `0.95`), the height where the tree should be cut (see `stats::cutree` for details) and `type` that determines the type visualisation:

- `heatmap`: a heatmap displaying pairwise similarities/distances along with cluster dendrograms
- `dendrogram` (default): a circular dendrogram with colour code for cluster assignment

Create a heatmap

Heatmaps of our example data for MS^2 and neutral loss pattern similarity are created as follows (with reduced label font size by changing `cexRow` and `cexCol` as well as `margins` of the underlying `heatmap.2` function):

```
HCplot(distmat, type = "heatmap",
       cexRow = 0.06, cexCol = 0.06,
       margins = c(5,5))
```

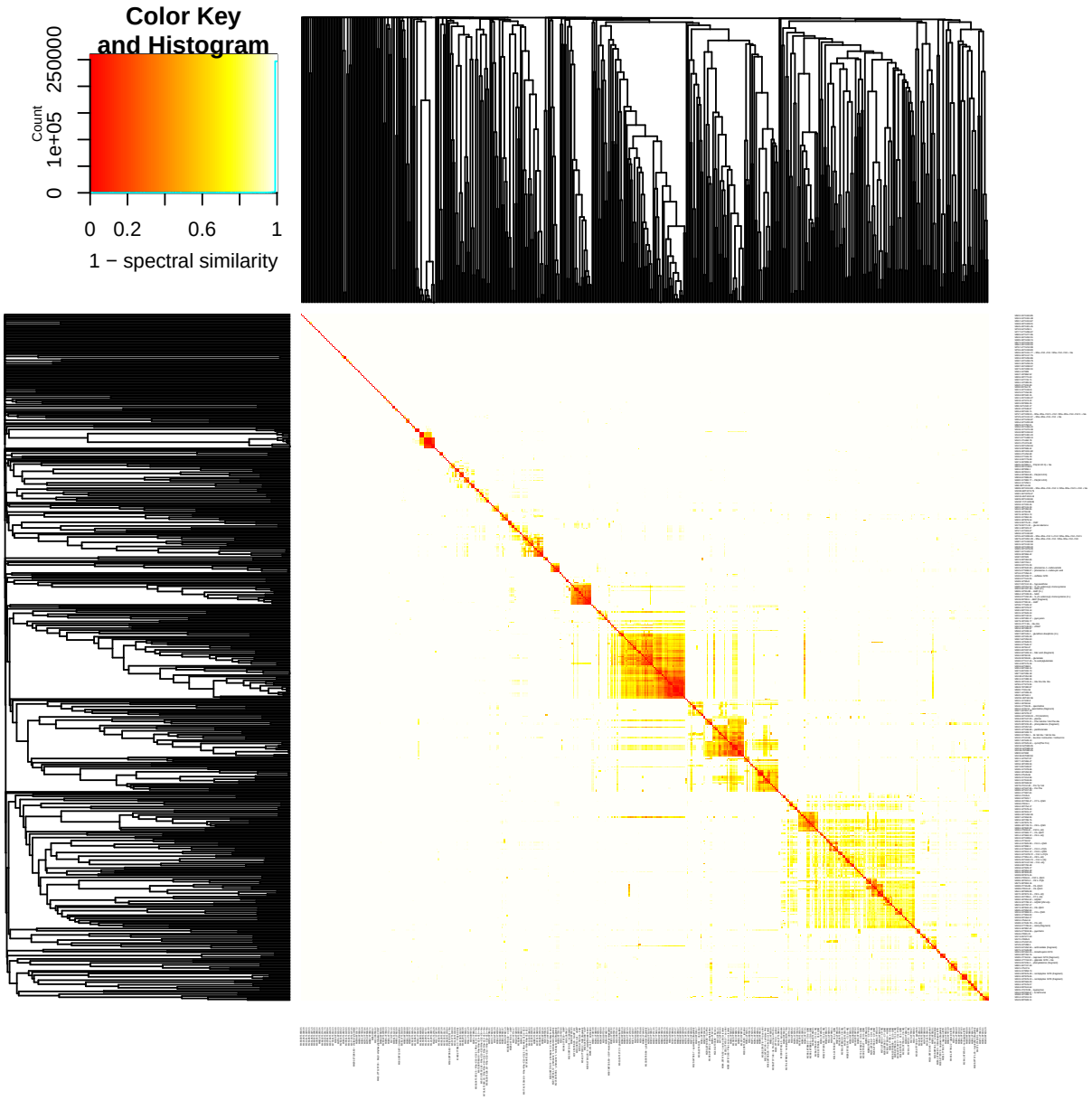

Figure A-6: Symmetric heat map of the distance matrix displaying MS^2 spectra similarities of the example data set along with dendrograms resulting from hierarchical clustering based on the distance matrix. The colour encoding is shown in the top-left insert.

```
HCplot(nlmat, type = "heatmap",
       cexRow = 0.06, cexCol = 0.06,
       margins = c(5,5))
```

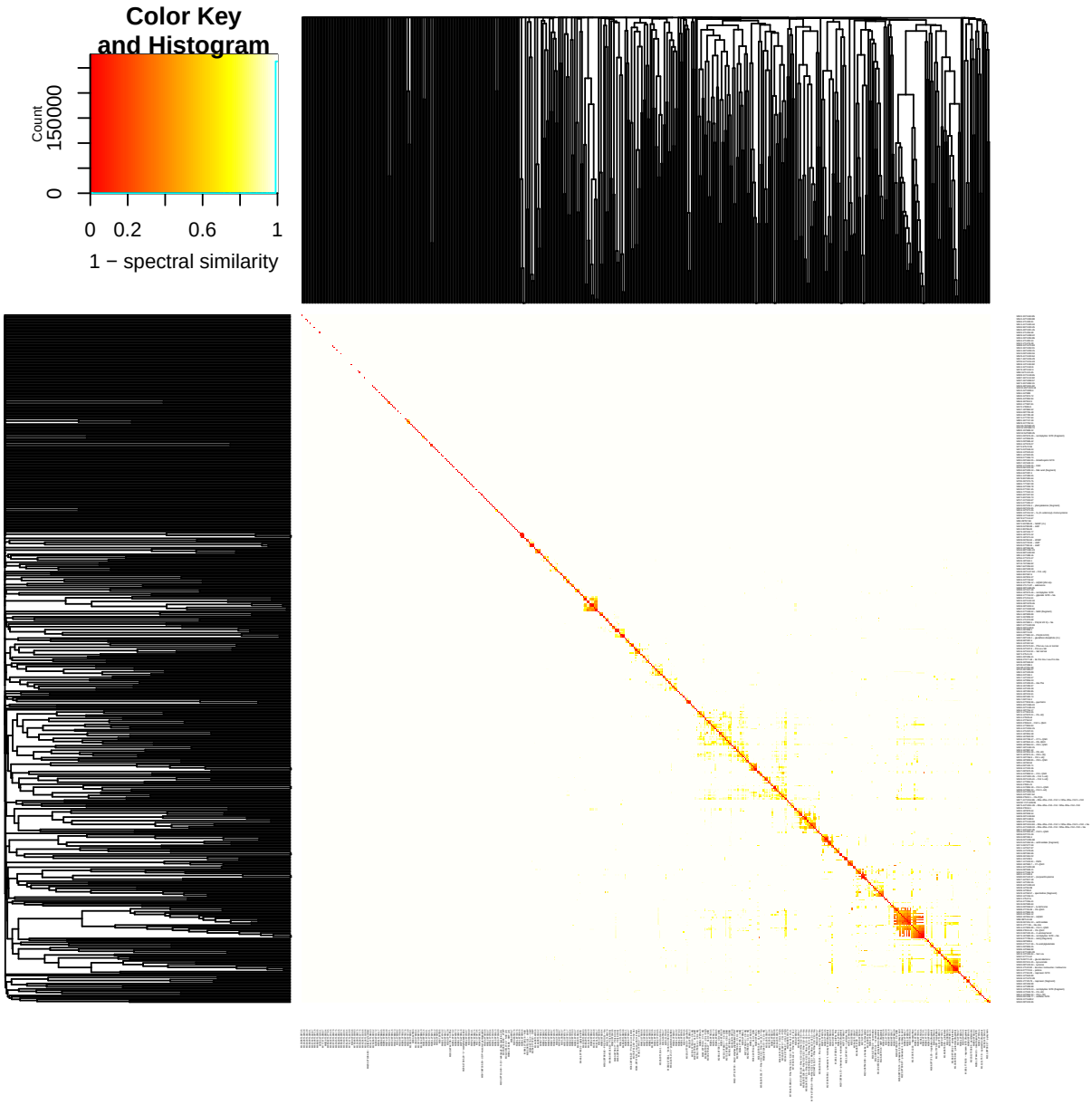

Figure A-7: Symmetric heat map of the distance matrix displaying neutral loss similarities of the example data set along with dendrograms resulting from hierarchical clustering based on the distance matrix. The colour encoding is shown in the top-left insert.

Obviously, it makes sense to export the plots to larger pdf or png files (e.g. 2000 × 2000 pixels) to examine them closely. If exported to pdf, the feature names remain searchable (`Ctrl+F` in Windows).

Create a dendrogram

With the dendrogram, too, it is advisable to export is to pdf in a large format, e.g. as follows:

```
pdf(file = "CluMSID_dendro.pdf", width = 20, height = 20)
HCplot(distmat)
```

```
dev.off()
```

The plot from our example data looks like this:

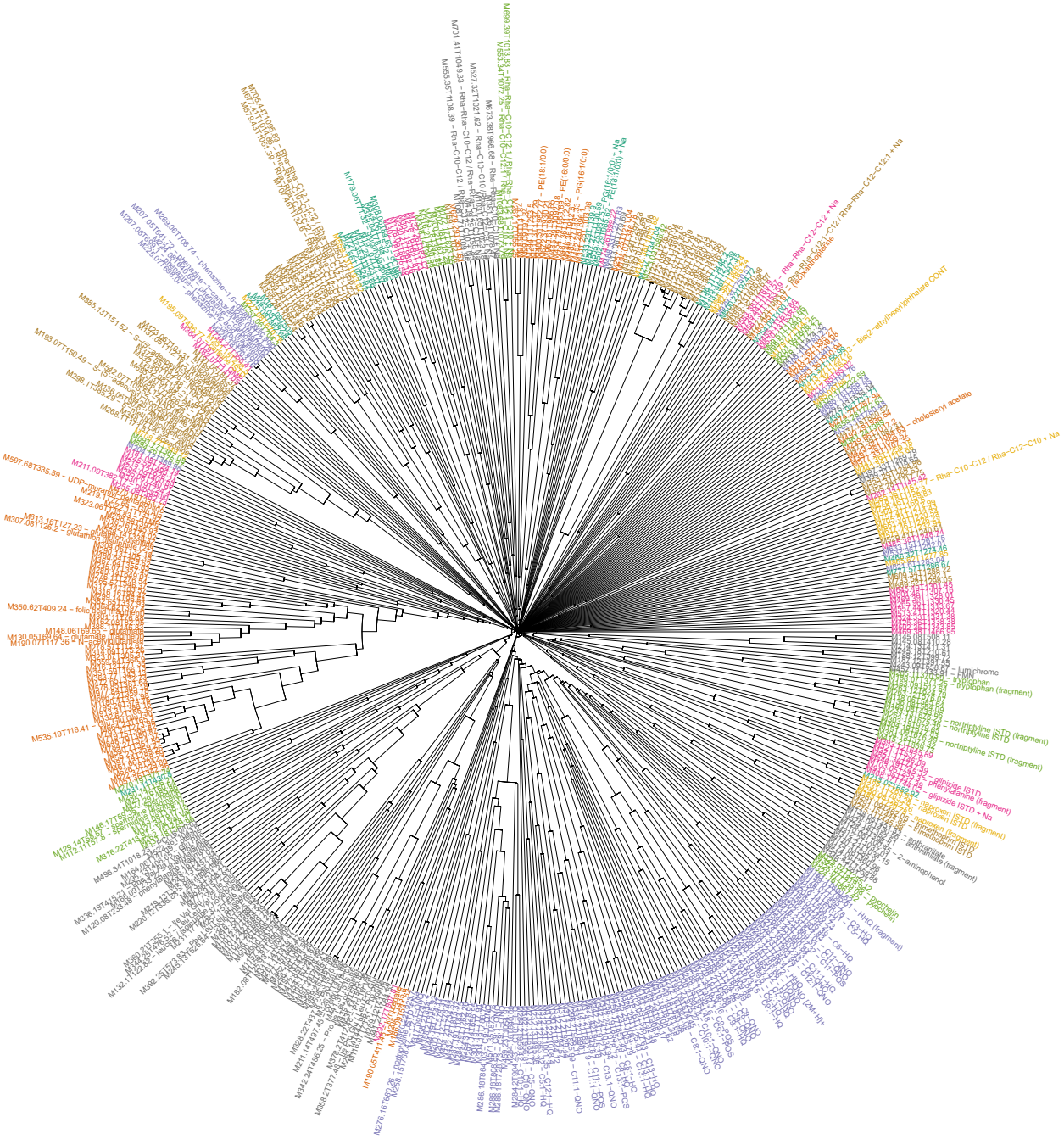

Figure A-8: Circularised dendrogram as a result of agglomerative hierarchical clustering with average linkage as agglomeration criterion based on MS^2 spectra similarities of the example data set. Each leaf represents one feature and colours encode cluster affiliation of the features. Leaf labels display feature IDs, along with feature annotations, if existent. Distance from the central point is indicative of the height of the dendrogram.

The clusters are colour-coded and if exported to pdf, the tip labels containing feature ID and annotation are searchable.The height of the dendrogram's branching points serves as another piece of information when interpreting the clustered data as it signifies similarity of features.

For a detailed example of how to interpret, please refer to Depke *et al.* 2017, where `CluMSID` helped to identify new members of several classes of secondary metabolites in *Pseudomonas aeruginosa*.

Like with density-based clustering, it is also possible to generate a list of features with respective cluster assignments using `HCtbl`. As mentioned above for `OPTISplot` and `OPTICStbl`, it is crucial to run `HCplot` and `HCtbl` using the same parameters.

```
HCtable <- HCtbl(distmat)

head(HCtable)
#>                                 feature cluster_ID
#> 1            M146.17T59.35 - spermidine          1
#> 2 M129.14T58.57 - spermidine (fragment)          1
#> 3  M112.11T57.8 - spermidine (fragment)          1
#> 4                         M251.16T60.64          1
#> 5                         M212.85T65.02          2
#> 6                         M290.85T64.76          3
```

Generate a correlation network

As a new functionality, `CluMSID` offers the possibility to analyse the similarity data using weighted correlation networks. These networks offer some advantages with respect to standard clustering methods, most notably that they do not strictly assign every feature to a distinct cluster but also represent similarities between features that would fall into different clusters in hierarchical or density-based clustering. Thus, correlation networks potentially contain more useful information for data interpretation. On the downside, the interpretation is also complicated by this lack of concrete cluster assignments. E.g., we cannot simply look up which features belong to the same cluster in order to examine their spectra closely but we have to go back to the correlation network visualisation and search for connected features manually.

`networkplot` requires some arguments:

- `distmat`: *matrix*; a distance matrix like for all other functions described above
- `interactive`: *logical*; Similar to `MDSplot`, correlation network can be generate as interactive plots that are zoomable and display feature IDs on mouse-over. If that is desired, set `interactive` to `TRUE` (default is `FALSE`).
- `show_labels`: *logical*; whether to display feature IDs in the (non-interactive) plot (default is `FALSE`, ignored if `interacive = TRUE`)
- `label_size`: *numeric*; font size of feature ID labels (default is `1.5`, which is way smaller than the default in `GGally::ggnet2`, `4.5`)
- `highlight_annotated`: *logical*; whether to plot dots for features with annotation in a different colour (same as in `MDSplot`, default is `FALSE`)
- `min_similarity`: *numeric*; the minimum similarity (1 – distance) threshold (similarities below this threshold will be ignored, default is `0.1`)
- `exclude_singletons`: *logical*; whether to exclude features from the plot that do not have connections to other features, particularly useful with data sets containing very dissimilar spectra, e.g. neutral loss patterns or MS^1 pseudospectra (default is `FALSE`)

A standard non-interactive correlation network for the MS^2 example data can be plotted like this:

```
networkplot(distmat, highlight_annotated = TRUE,
            show_labels = TRUE, interactive = FALSE)
```

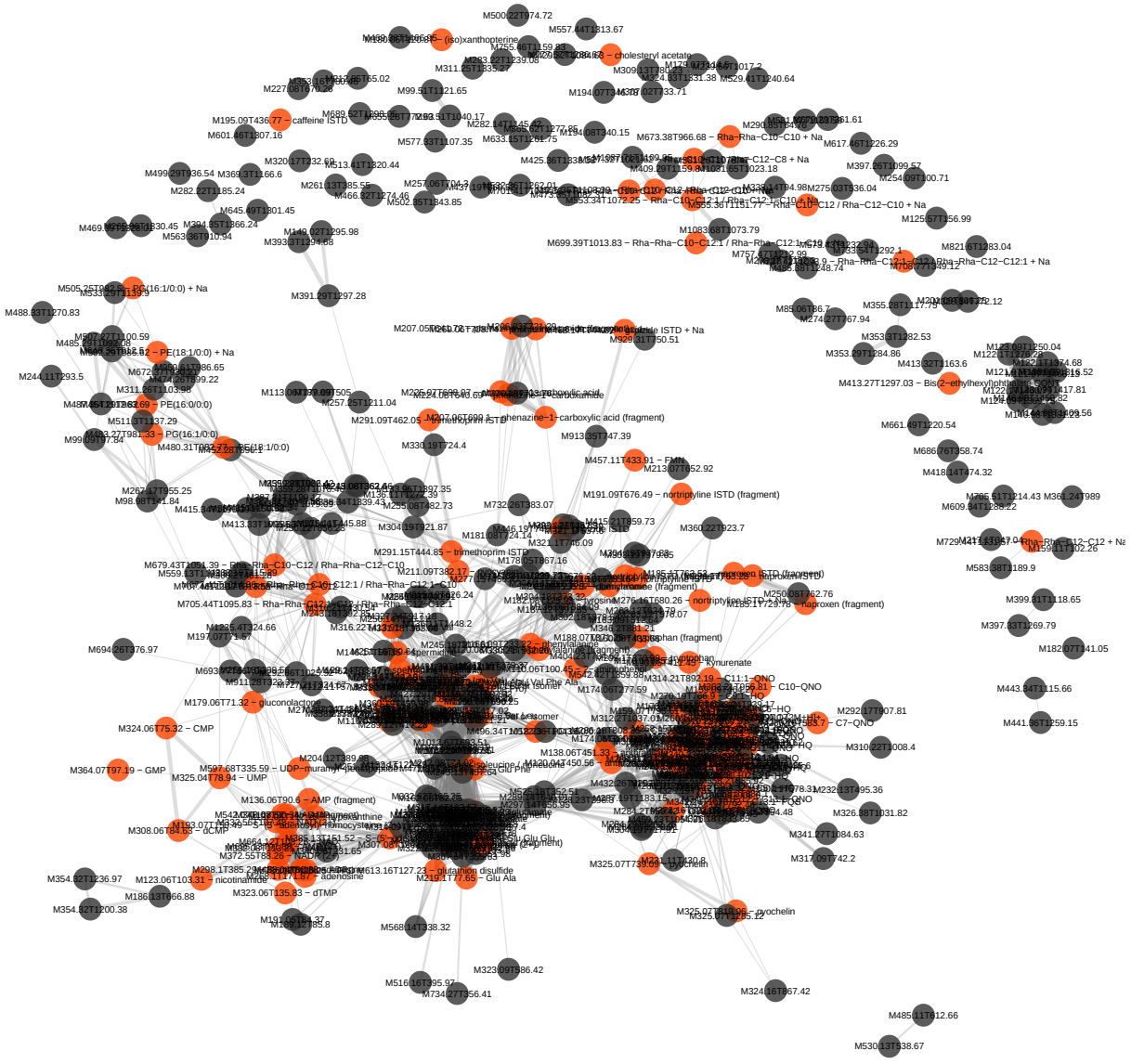

Figure A-9: Correlation network plot based on MS^2 spectra similarities of the example data set. Grey dots indicate non-identified features, orange dots identified ones. Labels display feature IDs, along with feature annotations, if existent. Edge widths are proportional to spectral similarity of the connected features.

As you can guess from this busy plot, it makes sense to use the interactive visualisation. Just like with `MDSplot`, you can view the interactive plot within RStudio or save it as html and view it in web browser.

```
my_net <- networkplot(distmat, interactive = TRUE,
                      highlight_annotated = TRUE)

htmlwidgets::saveWidget(my_net, "net.html")
```

This is how it looks like if you open the html file in Firefox, zoom in on a cluster and mouse over a feature:

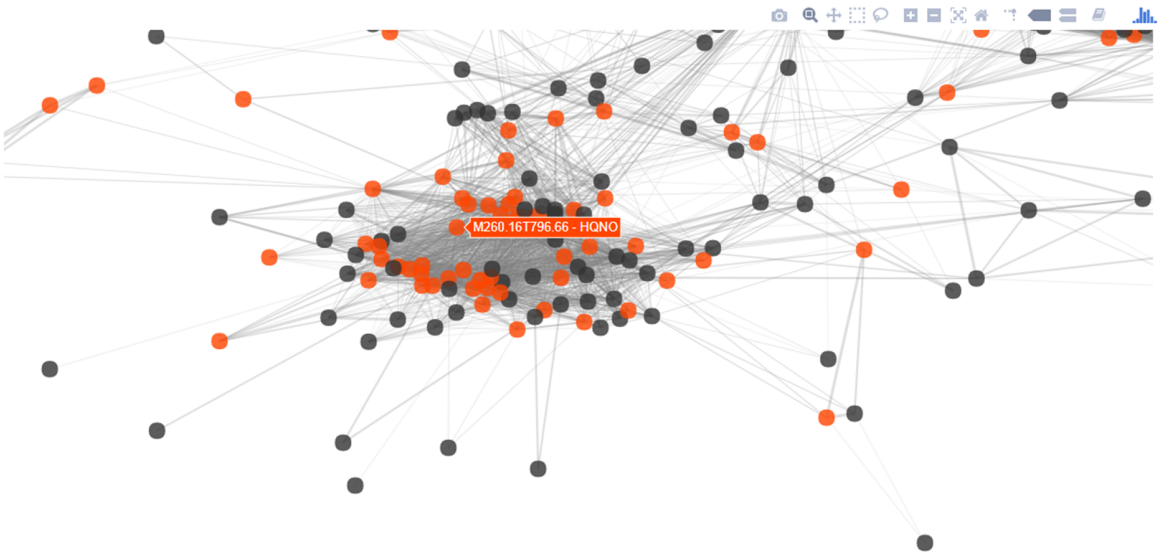

Figure A-10: Screenshot of the interactive version of the Correlation network plot based on MS^2 spectra similarities of the example data set (cf Figure 9). Zoomed image section with tooltip displaying feature information upon mouse-over.

Please be aware that the spatial arrangement of the data points in the plot has a random component, i.e. while the relative position of the points (the distance to each other) is always the same, the absolute position varies and will not be the same even if the same command is executed twice.

The pairwise similarity of spectra or neutral loss patterns of features expressed by the cosine score is signified by the width of the line connecting the two features. All pairwise similarities greater than `min_similarity` result in a connecting line in the plot. The spatial proximity in which the features are mapped onto the plot is determined by the multivariate method underlying the network generation.

As we have already noticed after inspection of the heatmaps on p.13–14, the neutral loss patterns show much less similarity to each other than the MS^2 spectra data. Thus, we expect quite a few neutral loss patterns that do not show any similarity to another neutral loss pattern. This expectation justifies the exclusion of these 'singletons' from the correlation network analysis. To do so, just set `exclude_singletons` to `TRUE`:

```
networkplot(nlmat, highlight_annotated = TRUE,
            show_labels = TRUE, exclude_singletons = TRUE)
```

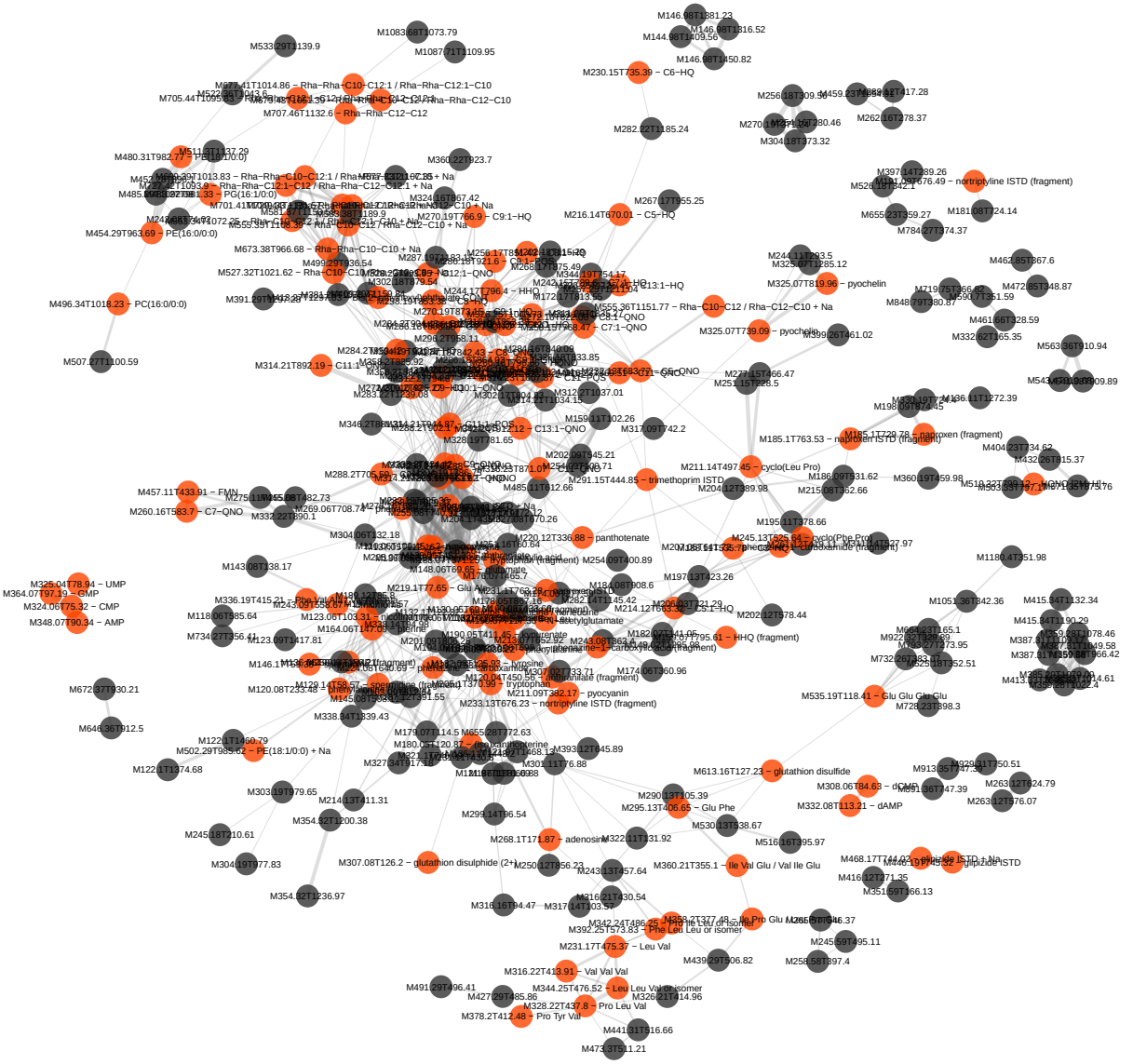

Figure A-11: Correlation network plot based on neutral loss similarities of the example data set. Grey dots indicate non-identified features, orange dots identified ones. Labels display feature IDs, along with feature annotations, if existent. Edge widths are proportional to spectral similarity of the connected features.

Additional functionalities

Multidimensional scaling, density-based clustering, hierarchical clustering and correlation network analysis are the main `CluMSID` tools to analyse MS^2 spectra or neutral loss pattern similarity data, however, the package contains some additional functionalities that may facilitate data analysis in some cases and can also be used in other contexts with or without the above-mentioned unsupervised methods.

Access individual spectra from a list of spectra by various slot entries

Accessing S4 objects within lists is not trivial. Therefore, `CluMSID` offers a function to access individual or several `MS2spectrum` objects by their slot entries. `getSpectrum()` requires the following arguments:

- `featlist`: a `list` that contains only objects of class `MS2spectrum`
- `slot`: the slot to be searched (invalid `slot` arguments will produce errors):
  - `id`
  - `annotation`
  - `precursor` (*m/z* of precursor ion)
  - `rt` (retention time of precursor)
- `what`: the search term or number, must be *character* for `id` and `annotation` and *numeric* for `precursor` and `rt`
- `mz.tol`: the tolerance used for precursor ion *m/z* searches, defaults to 1E-05 (10ppm)
- `rt.tol`: the tolerance used for precursor ion retention time searches, defaults to 30s; high values can be used to specify retention time ranges (see example)

Some examples will demonstrate the use of `getSpectrum()`:

1. Accessing a spectrum by its ID. For this, the exact feature ID must be known:

```
getSpectrum(annotatedSpeclist, "id", "M244.17T796.4")
#> An object of class "MS2spectrum"
#>  id: M244.17T796.4
#>  annotation: HHQ
#>  precursor: 244.1700
#>  retention time: 796.4
#>  polarity: positive
#>  MS2 spectrum with 98 fragment peaks
#>  neutral loss pattern with 81 neutral losses
```

2. Accessing a spectrum by its annotation. For this, the exact annotation has to be known as well, other annotations will produce a message:

```
getSpectrum(annotatedSpeclist, "annotation", "HHQ")
#> An object of class "MS2spectrum"
#>  id: M244.17T796.4
#>  annotation: HHQ
#>  precursor: 244.1700
#>  retention time: 796.4
#>  polarity: positive
#>  MS2 spectrum with 98 fragment peaks
#>  neutral loss pattern with 81 neutral losses
```

```
getSpectrum(annotatedSpeclist, "annotation", "C7-HQ")
#> No spectrum with that annotation.
```

3. Accessing spectra by their precursor ion *m/z*. If the list contains more than one spectrum with a precursor ion *m/z* within the tolerance, the output is again a list of `MS2spectrum` objects that meet the specified criterion:

```
getSpectrum(annotatedSpeclist, "precursor", 286.18, mz.tol = 1E-03)
#> [[1]]
#> An object of class "MS2spectrum"
#>  id: M286.18T728.73
#>  annotation: C9:1-QNO
#>  precursor: 286.1799
#>  retention time: 728.73
#>  polarity: positive
#>  MS2 spectrum with 4 fragment peaks
#>  neutral loss pattern with 2 neutral losses
#> [[2]]
#> An object of class "MS2spectrum"
#>  id: M286.18T808.85
#>  annotation: C9:1-QNO
#>  precursor: 286.1804
#>  retention time: 808.85
#>  polarity: positive
#>  MS2 spectrum with 7 fragment peaks
#>  neutral loss pattern with 5 neutral losses
#> [[3]]
#> An object of class "MS2spectrum"
#>  id: M286.18T864.03
#>  annotation: C9:1-QNO
#>  precursor: 286.1808
#>  retention time: 864.03
#>  polarity: positive
#>  MS2 spectrum with 183 fragment peaks
#>  neutral loss pattern with 167 neutral losses
#> [[4]]
#> An object of class "MS2spectrum"
#>  id: M286.18T921.6
#>  annotation: C9:1-PQS
#>  precursor: 286.1808
#>  retention time: 921.6
#>  polarity: positive
#>  MS2 spectrum with 3 fragment peaks
#>  neutral loss pattern with 1 neutral losses
```

4. Accessing spectra by their precursor retention time. Here, too, we can extract several `MS2spectrum` objects by setting a larger retention time tolerance. If we want to extract the spectra of all compounds that elute from 6min (360s) to 8min (480s), we proceed as follows:

```
six_eight <- getSpectrum(annotatedSpeclist, "rt", 420, rt.tol = 60)
length(six_eight)
#> [1] 75
```

Find spectra that contain a specific fragment or neutral loss

Another pair of accessory functions is `findFragment()` and `findNL()` which are used to find spectra that contain a specific fragment ion or neutral loss. Analogous to `getSpectrum()`, they need as arguments a list of `MS2spectrum` objects, the *m/z* of the fragment or neutral loss of interest and the respective *m/z* tolerance in ppm (default is 10ppm). The two functions can be useful in many situation, e.g. when working with lipid data where head groups and fatty acids often give characteristic fragments or neutral losses. In the world of *P. aeruginosa* secondary metabolites, alkylquinolones (AQs) play an important role and most of the AQ MS^2 spectra contain a signature fragment with an *m/z* of 159.068. Based on this fragment *m/z*, we can create a list of putative AQs:

```
putativeAQs <- findFragment(annotatedSpeclist, 159.068)
#> 70 spectra were found that contain a fragment of m/z 159.068 +/- 10 ppm.
```

An example for common neutral losses are nucleoside monophospates that all loose ribose-5'-monophosphate, resulting in a neutral loss of 212.009 in ESI-(+). Using `findNL()` we find CMP, UMP, AMP and GMP.

```
findNL(annotatedSpeclist, 212.009)
#> 4 neutral loss patterns were found that contain a neutral loss of m/z 212.009 +/- 10 ppm.
#> [[1]]
#> An object of class "MS2spectrum"
#>  id: M324.06T75.32
#>  annotation: CMP
#>  precursor: 324.0591
#>  retention time: 75.32
#>  polarity: positive
#>  MS2 spectrum with 8 fragment peaks
#>  neutral loss pattern with 8 neutral losses
#> [[2]]
#> An object of class "MS2spectrum"
#>  id: M325.04T78.94
#>  annotation: UMP
#>  precursor: 325.0429
#>  retention time: 78.94
#>  polarity: positive
#>  MS2 spectrum with 5 fragment peaks
#>  neutral loss pattern with 5 neutral losses
#> [[3]]
#> An object of class "MS2spectrum"
#>  id: M348.07T90.34
#>  annotation: AMP
#>  precursor: 348.0707
#>  retention time: 90.34
#>  polarity: positive
#>  MS2 spectrum with 21 fragment peaks
#>  neutral loss pattern with 19 neutral losses
#> [[4]]
#> An object of class "MS2spectrum"
#>  id: M364.07T97.19
#>  annotation: GMP
#>  precursor: 364.0659
#>  retention time: 97.19
#>  polarity: positive
#>  MS2 spectrum with 6 fragment peaks
#>  neutral loss pattern with 6 neutral losses
```

Match one spectrum against a set of spectra

If you are mainly interested in one or a few number of spectra or neutral loss patterns, it may be sufficient to match one feature at a time against a larger set of spectra. This set of spectra can be all spectra contained in one mzXML file like in all the examples in this tutorial or they could be a spectral library, as long as its format in R is a list of `MS2spectrum` objects.

The `getSimilarities()` function requires several arguments:

- `spec`: The spectrum to be compared to other spectra. Can be either an object of class `MS2spectrum` or a two-column numerical matrix that contains fragment mass-to-charge ratios in the first and intensities in the second column.
- `speclist`: The set of spectra to which `spec` is to be compared. Must be a list where every entry is an object of class `MS2spectrum`. Can be generated from an mzXML file as shown above or constructed using `new("MS2spectrum", ...)` for every list entry (see example).
- `type`: Specifies whether MS^2 spectra or neutral loss patterns are to be compared. Must be either 'spectrum' (default) or 'neutral_losses'.
- `hits_only`: Logical that indicates whether the result should contain only similarities greater than zero (see example).

In the first example, we want to find all MS^2 spectra in our example data set that are similar to the spectrum of pyocyanin, an important secondary metabolite from *Pseudomonas aeruginosa* and therefore match the pyocyanin spectrum against our `annotatedSpeclist`. Because we have already identified pyocyanin in the data set, we can use `getSpectrum` to extract the `MS2spectrum` object from `annotatedSpeclist`. We do not want to search all 518 elements of the result vector, so we set `hits_only` to `TRUE` to exclude spectra that have 0 similarity to the pyocyanin spectrum.

```
pyo <- getSpectrum(annotatedSpeclist, "annotation", "pyocyanin")

sim_pyo <- getSimilarities(pyo, annotatedSpeclist, hits_only = TRUE)
sim_pyo
#>  M110.06T100.45  M123.06T103.31  M332.56T107.48  M332.08T113.21
#>    0.0235166588    0.0071763662    0.0032575891    0.0035153018
#>  M182.08T125.93  M166.09T233.22  M120.08T233.48   M103.05T235.3
#>    0.0414005385    0.0394723541    0.0492390806    0.0826780036
#>  M174.06T277.59  M220.12T336.88  M525.18T352.51   M243.08T362.4
#>    0.0391004892    0.0205482303    0.0060019991    0.0145904545
#>  M188.07T371.25   M205.1T370.99  M211.09T382.17  M187.12T391.55
#>    0.0176900909    0.0179895663    1.0000000000    0.0210280136
#>  M188.12T399.72  M254.09T400.89  M160.08T433.66  M291.15T444.85
#>    0.0105392131    0.2071528536    0.0489638040    0.0106479317
#>  M120.04T450.56  M138.06T451.33   M176.07T465.7  M491.29T496.41
#>    0.0287432023    0.0202198052    0.0275059908    0.0610208210
#>  M255.08T482.73  M245.59T495.11  M145.08T508.11  M163.09T512.64
#>    0.6451546287    0.2583432230    0.0473127795    0.0034167239
#>  M188.11T535.78    M321.1T537.6  M243.09T558.67  M136.08T584.09
#>    0.0057005179    0.0293635312    0.0116275804    0.0132716679
#>  M118.06T585.64  M215.12T626.24  M224.08T640.69  M213.07T652.92
#>    0.0203921366    0.3252546561    0.0325490977    0.0083842257
#>  M216.14T670.01  M227.08T670.26  M264.18T675.46  M233.13T676.23
#>    0.0009299928    0.0034818309    0.0172023762    0.0143332573
#>  M225.07T698.07   M207.06T699.1   M257.06T704.3  M226.18T703.76
#>    0.0253940205    0.0230298767    0.0028192053    0.0255571995
#>  M325.07T739.09  M181.08T724.14   M330.19T724.4  M255.08T740.91
#>    0.0010572974    0.1283755709    0.5019236568    0.2030839014
#>  M446.19T745.32   M321.1T746.09  M891.36T747.39   M231.1T763.28
```

```
#>    0.0750793676    0.1017149711    0.0714108632    0.0070294838
#>   M185.1T763.53   M288.2T765.88  M258.15T768.47  M328.14T772.12
#>    0.0094225379    0.0018840838    0.0168976240    0.0009052790
#>   M242.15T789.6  M304.19T786.75  M260.16T796.66   M244.17T796.4
#>    0.0071606643    0.0066966285    0.0141834395    0.0106077162
#>  M159.07T795.61  M314.21T825.99  M326.18T833.85  M286.18T864.03
#>    0.0124577125    0.0006167122    0.0023434969    0.0205024121
#>  M270.19T873.15  M268.17T875.49  M178.05T867.16  M198.09T874.45
#>    0.0089744139    0.0011687551    0.0413446455    0.0064665757
#>    M170.1T885.9    M288.2T902.1   M184.08T908.6   M272.2T910.42
#>    0.0062571323    0.0100270721    0.0036517997    0.0086043375
#>   M312.2T929.44  M314.21T944.87   M296.2T958.11  M298.22T984.33
#>    0.0085388744    0.0128215188    0.0054678320    0.0065812089
#>  M500.22T974.72  M304.19T977.83  M303.19T979.65 M340.23T1007.62
#>    0.0396920510    0.0059045590    0.0049045093    0.0052002486
#> M324.23T1025.79 M314.21T1034.15 M326.25T1043.73 M679.43T1051.39
#>    0.0005826366    0.0030626495    0.0005424581    0.0008290325
```

We get 84 spectra that have a non-zero similarity to the pyocyanin spectrum, including pyocyanin itself with a similarity of `1`. Of course, we can further filter the data by subsetting the result vector in order to exclude spectra that have only minimal similarity, e.g. `M679.43T1051.39` with a cosine similarity of only `0.0008` (the last element in the vector).

In the second example, we generate a new `speclist`, e.g. from a spectral library. We look at the unknown feature that has most similarity to pyocyanin. As pyocyanin is contained in `annotatedSpeclist` itself, we have to look at the second highest similarity. Again, we use `getSpectrum()` to extract the object from `annotatedSpeclist`:

```
highest_sim <- sort(sim_pyo, decreasing = TRUE)[2]

sim_spec <- getSpectrum(annotatedSpeclist, "id", names(highest_sim))
sim_spec
#> An object of class "MS2spectrum"
#>  id: M255.08T482.73
#>  annotation:
#>  precursor: 255.0761
#>  retention time: 482.73
#>  polarity: positive
#>  MS2 spectrum with 5 fragment peaks
#>  neutral loss pattern with 3 neutral losses
```

We see that the feature is not annotated. We are interested whether this feature also shows similarity to other members of the phenazine family of *P. aeruginosa* secondary metabolites. Some phenazines are contained in `annotatedSpeclist` but some are not, so we make a new `speclist` called `phenazines` and add the missing spectra manually from an in-house library:

```
phenazines <- list()
phenazines[[1]] <- getSpectrum(annotatedSpeclist,
                               "annotation", "pyocyanin")
phenazines[[2]] <- getSpectrum(annotatedSpeclist,
                               "annotation", "phenazine-1-carboxamide")
phenazines[[3]] <- getSpectrum(annotatedSpeclist,
                               "annotation", "phenazine-1-carboxylic acid")
phenazines[[4]] <- getSpectrum(annotatedSpeclist,
                               "annotation", "phenazine-1,6-dicarboxylic acid")
phenazines[[5]] <- new("MS2spectrum", id = "lib_entry_1",
```

```
                        annotation = "1-hydroxyphenazine",
                        spectrum = matrix(c(168.0632, 14,
                                            169.0711, 288,
                                            170.0743, 33,
                                            179.0551, 62,
                                            197.0653, 999),
                                          byrow = TRUE,
                                          ncol = 2))
phenazines[[6]] <- new("MS2spectrum", id = "lib_entry_2",
                        annotation = "2-hydroxy-phenazine-1-carboxylic acid",
                        spectrum = matrix(c(167.0621, 43,
                                            179.0619, 93,
                                            180.0650, 12,
                                            195.0564, 40,
                                            223.0509, 999,
                                            224.0541, 142,
                                            241.0611, 60),
                                          byrow = TRUE,
                                          ncol = 2))
phenazines[[7]] <- new("MS2spectrum", id = "lib_entry_3",
                        annotation = "pyocyanin (library spectrum)",
                        spectrum = matrix(c(168.0690, 58,
                                            183.0927, 152,
                                            184.0958, 19,
                                            196.0640, 118,
                                            197.0674, 15,
                                            211.0873, 999,
                                            212.0905, 145),
                                          byrow = TRUE,
                                          ncol = 2))

getSimilarities(sim_spec, phenazines, hits_only = FALSE)
#> M211.09T382.17 M224.08T640.69 M225.07T698.07 M269.06T708.74     lib_entry_1
#>      0.6451546      0.0000000      0.0000000      0.0000000       0.0000000
#>    lib_entry_2    lib_entry_3
#>      0.0000000      0.6375061
```

As a result, we get the interesting information that the MS^2 spectra similarity of our unknown feature seems to be specific to pyocyanin (both the experimental and the library spectrum).

Convert `MSnbase` objects to class `MS2spectrum`

The `MSnbase` package—which is commonly used for proteomics applications and is also associated with XCMS3—has two classes for (MS^2) spectra, `Spectrum` and `Spectrum2` which contain spectra along with metainformation. These metainformation differ from those contained in `MS2spectrum` objects and are not very well suited for metabolomics applications. Still, it is possible to use `CluMSID` functions with objects of those two classes by converting them to `MS2spectrum` objects using `as.MS2spectrum()`:

```
CluMSID_object <- as.MS2spectrum(MSnbase_object)
# or alternatively
CluMSID_object <- as(MSnbase_object, "MS2spectrum")
```

Split polarities from polarity-switching runs

As polarity-switching and similar methords are gaining importance in LC-MS/MS metabolomics, `CluMSID` offers the possibility to process LC-MS/MS data containing spectra of different polarities. As spectra from positive and negative ionisation show different fragmentation mechanisms and patterns, it does not appear to be useful to compare spectra of different polarity to each other. Therefore, `CluMSID` provides a function to separate positive and negative spectra from each other. This has to be done in the very beginning of the analysis to not interfere with spectral merging. Positive and negative spectra can than be processed independently from each other as shown above.

A schematic workflow would look like this:

```
raw_list_mixedpolarities <- extractMS2spectra("raw_file_mixedpolarities.mzXML")

raw_list_positive <- splitPolarities(raw_list_mixedpolarities, "positive")
raw_list_negative <- splitPolarities(raw_list_mixedpolarities, "negative")

speclist_positive <- mergeMS2spectra(raw_list_positive)
speclist_negative <- mergeMS2spectra(raw_list_negative)
```

... and so on as described in this tutorial.

Use MS^1 pseudospectra instead of or in addition to MS^2 data

MS^1 pseudospectra are groups of peaks/ions that derive or are assumed to derive from the same compound. They consist of peaks for in-source fragment, adducts etc. Pseudospectra can contain structural information about analytes, e.g. about moieties that easily fragment even in MS^1 mode without CID. Thus, it might sometimes be useful to study similarities between pseudospectra analogously to those between MS^2 spectra. `CluMSID` makes use of the `CAMERA` package to assign peaks to pseudospectra. A custom S4 class named `pseudospectrum` is used which is very similar to the `MS2spectrum` class. For obvious reasons, it does not contain a precursor ion m/z slot and thus no neutral loss pattern, either. The `pcgroup` defined by `CAMERA` is used as ID, an annotation can be added if desired.

Extract pseudospectra

To extract pseudospectra, you first have to process your data using the `CAMERA` package, either in R or via XCMSonline, where this is done automatically. There are two possibilities to use the `extractPseudospectra()` function in `CluMSID`: either with an `xsAnnotate` object which you generate with `CAMERA` in R or with a `data.frame` that contains data on m/z, retention time, intensity and `pcgroup`, e.g. the results table from XCMSonline.

The latter is demonstrated with the XCMSonline results table already used to generate a peak table. If the column names are not changed, the `data.frame` can be supplied as-is and `intensity_columns` does not have to be specified. We want to exclude pseudospectra that have only one peak, so we set `min_peaks = 2`.

```
pstable <-
    read_delim(file = system.file("extdata",
                                  "TD035_XCMS.annotated.diffreport.tsv",
                                  package = "CluMSIDdata"),
               delim = "\t")

pseudospeclist <- extractPseudospectra(pstable, min_peaks = 2)
```

As a result, we get a list with 198 pseudospectra that we can now process further.

Create distance matrix for pseudospectra

The creation of a distance matrix is analogous to the procedure for MS^2 spectra:

```
pseudodistmat <- distanceMatrix(pseudospeclist)
```

Generate a correlation network for pseudospectra

The distance matrix can now be used for MDS, clustering and correlation networks just like described above. For demonstration, we generate a correlation network:

```
networkplot(pseudodistmat, show_labels = TRUE, exclude_singletons = TRUE)
```

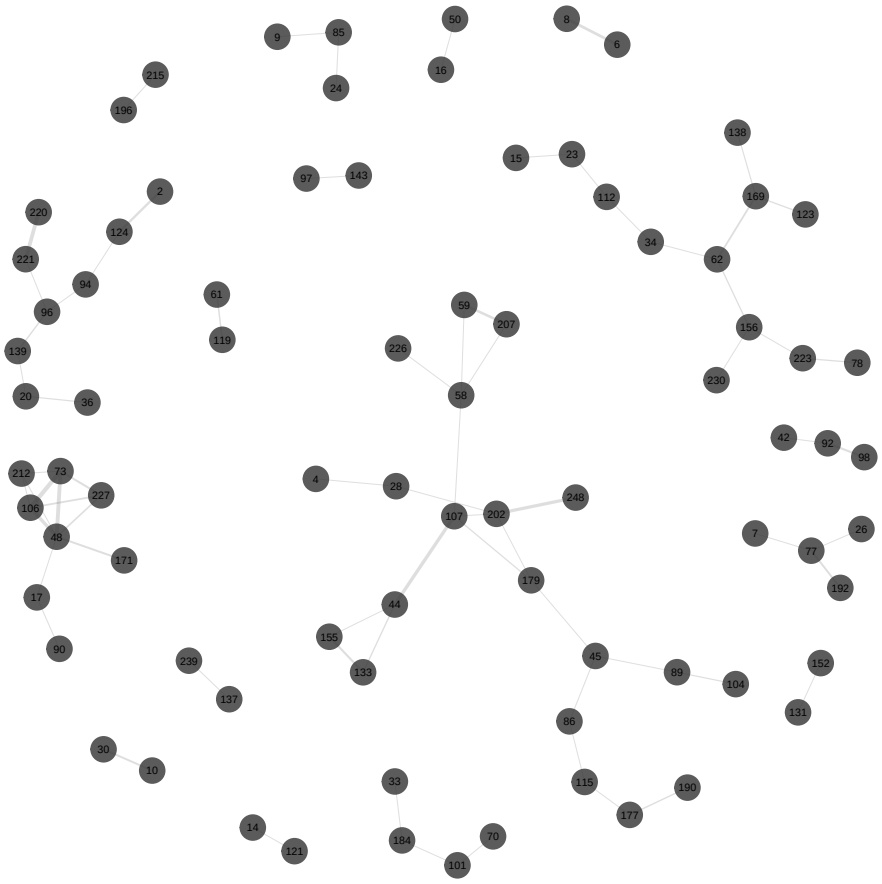

Figure A-12: Correlation network plot based on similarities of pseudospectra of the example data set. Grey dots indicate non-identified features, orange dots identified ones. Labels display `CAMERA`'s pseudospectra IDs. Edge widths are proportional to spectral similarity of the connected features.

With the exclusion of singletons, we get a much less busy plot than for MS^2 data but we still find quite a few connections that may prove informative.

Clustering Mass Spectra from Low Resolution GC-EI-MS Data Using CluMSID

Tobias Depke

December 31, 2018

Contents

Introduction

Although originally developed for high resolution LC-MS/MS data, CluMSID can also be used to find similarities in GC-EI-MS data, i.e. data from hard ionisation mass spectrometry.

As the peak picking and spectral merging differs considerably from data dependent ESI-MS/MS, we cannot use the standard `CluMSID` functions `extractMS2spectra()` and `mergeMS2spectra()`. In fact, the analysis of mass spectra from hard ionisation mass spectrometry resembles the one of MS^1 pseudospectra in ESI-MS. Thus, we can use the CluMSID function `extractPseudospectra()` in conjunction with pseudspectra generated by the `CAMERA` package.

Since `xcms` and `CAMERA` sometimes have difficulties in handling GC-EI-MS data, we use the `metaMS` package that enables workflows specialised to the analysis of such data. We also require the `metaMSdata` package from which we import the `FEMSsettings` object that contains `xcms` and `CAMERA` settings for GC-EI-MS data.

```
library(CluMSID)
library(CluMSIDdata)
library(metaMS)
library(metaMSdata)
data(FEMsettings)
```

Data import and preprocessing

As example data, we use GC-EI-MS metabolomics data from pooled cell extracts of *Pseudomonas aeruginosa* measured on a Thermo Scientific ITQ linear ion trap that has been converted to netCDF using Thermo Xcalibur. A netCDF file is available in the `CluMSIDdata` package:

```
pool <- system.file("extdata",
                    "1800802_TD_pool_total_1.cdf",
                    package = "CluMSIDdata")
```

To generate a list of (pseudo)spectra, we first need an xsAnnotate object as generated by `CAMERA`. In the case of GC-MS data, it is more convenient to use to use the `metaMS` function `runCAMERA()` than actual `CAMERA` functions. `metaMS::runCAMERA` requires an `xcmsSet` object which we generate by using `xcms::xcmsSet` on our netCDF file (we can do that in one go). We used standard GC-MS settings for `runCAMERA()` as they are proposed in the `metaMS` vignette.

```
xA <- runCAMERA(xcmsSet(pool),
                chrom = "GC",
                settings = metaSetting(TSQXLS.GC, "CAMERA"))
```

Extraction and annotation of spectra

From the `xsAnnotate` object, we can now extract the (pseudo)spectra using the `CluMSID` function `extractPseudospectra()` function as we would do for MS^1 pseudospectra from LC-ESI-MS data.

```
pslist <- extractPseudospectra(xA, min_peaks = 0)
```

Adding annotations is not as easy as with LC-(DDA-)MS/MS data, because only the retention time and the spectrum itself describe the feature and no precursor *m/z* is available. Thus, feature annotations/identifications made in a different programme, in this case MetaboliteDetector, have to be compared to the spectra in the `pslist` object.

Like with LC-(DDA-)MS/MS data, we can use `writeFeaturelist()` and `addAnnotations()` to add external annotations. The table output from `writeFeaturelist()` will give `NA` for all precursor *m/z*.

```
writeFeaturelist(pslist, "GC_pre.csv")
```

To facilitate manual annotation, it helps to plot the spectra along with the relevant information for every feature/pseudospectrum. That can be done by CluMSID's `specplot` function:

```
specplot(pslist[[27]])
```

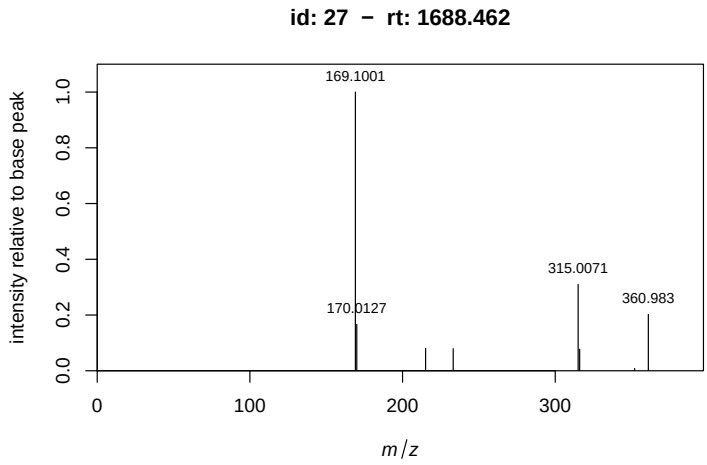

Figure B-1: Barplot for pseudospectrum 27, displaying fragment *m/z* on the x-axis and intensity normalised to the maximum intensity on the y-axis.

In this example, we load the list of feature annotations from `CluMSIDdata`:

```
apslist <- addAnnotations(featlist = pslist,
                          annolist = system.file("extdata",
                                                 "GC_post.csv",
                                                 package = "CluMSIDdata"))
```

Generation of distance matrix

This list of spectra in turn serves as an input for `distanceMatrix()`. As we are dealing with low resolution data, we have to adjust the *m/z* tolerance. The default value, 10ppm, is suitable for time-of-flight mass spectrometers while linear ion traps or single quadrupoles which are commonly used in GC-EI-MS only have unit mass resolution, equivalent to a relative mass error of 0.02 to 0.001 depending on the *m/z* of the analyte. We chose 0.02 to be tolerant enough for low molecular weight analytes:

```
pseudodistmat <- distanceMatrix(apslist, mz_tolerance = 0.02)
```

Data exploration

Starting from this distance matrix, we can use all the data exploration functions that `CluMSID` offers. In this example workflow, we look at a cluster dendrogram:

```
HCplot(pseudodistmat, type = "heatmap",
       cexRow = 0.3, cexCol = 0.3,
       margins = c(7,7))
```

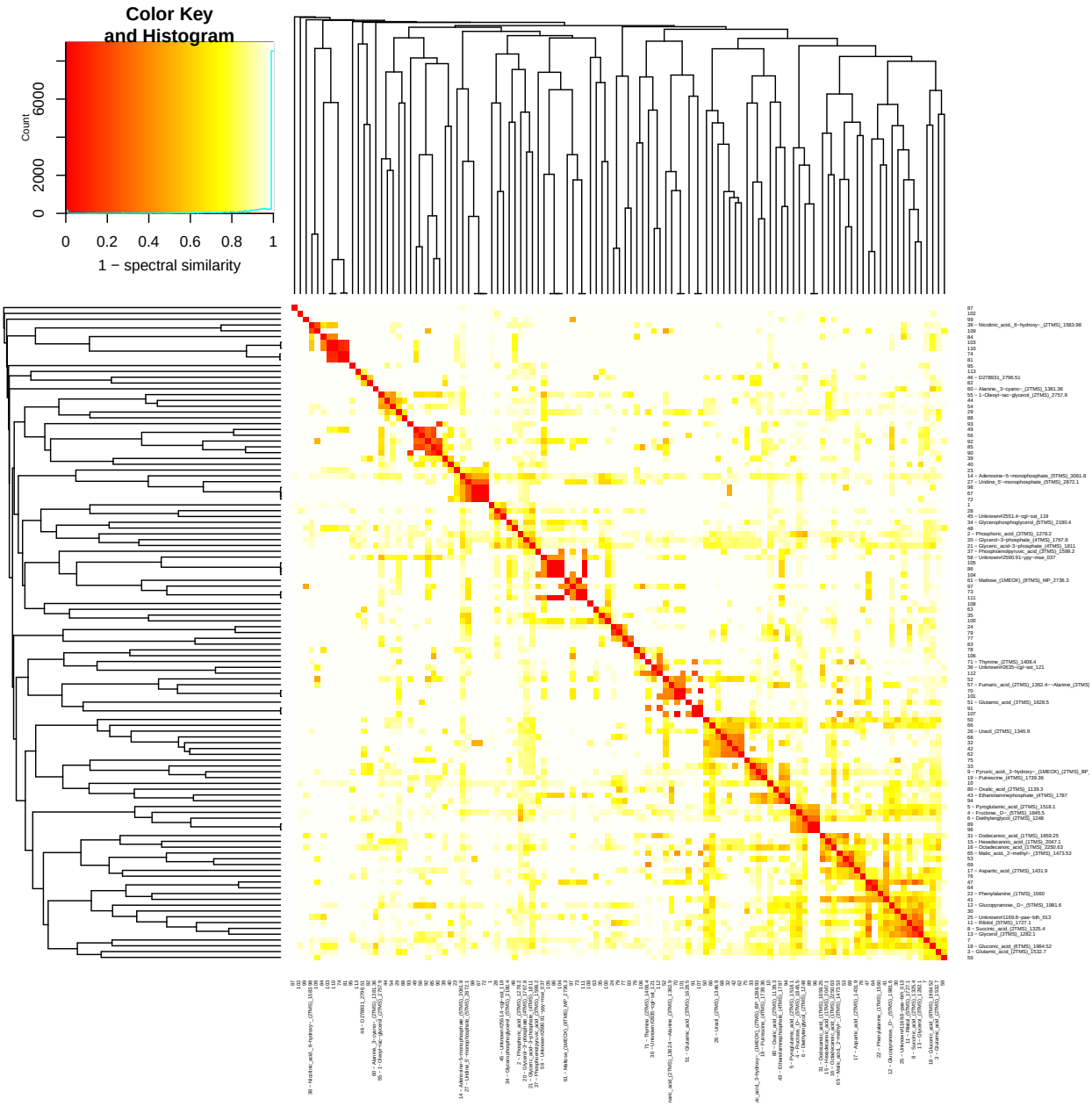

Figure B-2: Symmetric heat map of the distance matrix displaying pseudospectra similarities of the GC-EI-MS example data set along with dendrograms resulting from hierarchical clustering based on the distance matrix. The colour encoding is shown in the top-left insert.

It is directly visible that the resulting clusters are not as dense as with the LC-MS/MS example data. In turn, there are more between-cluster similarities.

This also shows in the correlation network, resulting in a chaotic plot when used with the default minimal similarity of `0.1`:

```
networkplot(pseudodistmat, highlight_annotated = TRUE,
            show_labels = TRUE, exclude_singletons = TRUE)
```

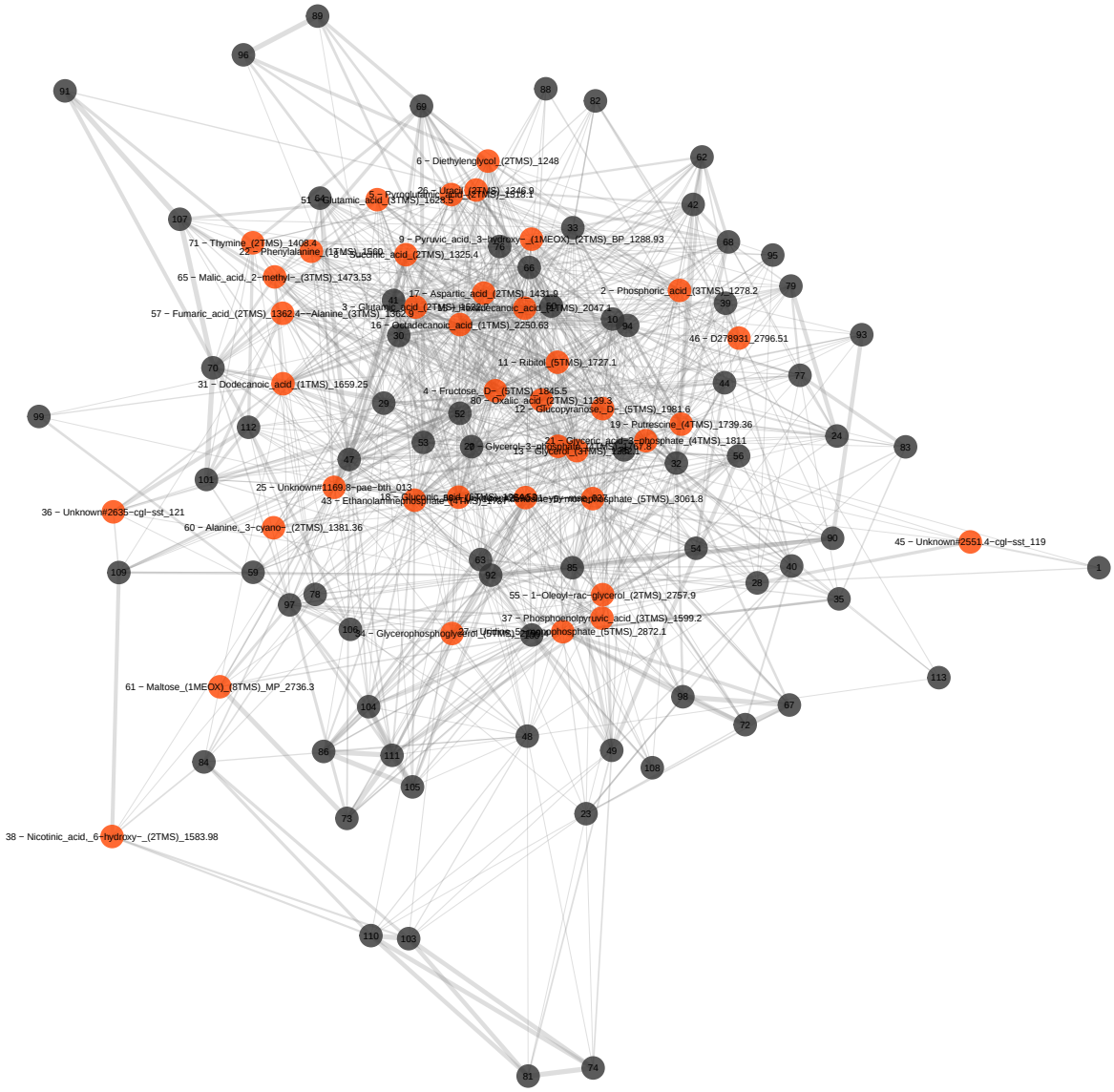

Figure B-3: Correlation network plot based on pseudospectra similarities of the GC-EI-MS example data set, generated with the default similarity threshold of `0.1`. Grey dots indicate non-identified features, orange dots identified ones. Labels display feature IDs, along with feature annotations, if existent. Edge widths are proportional to spectral similarity of the connected features.

By choosing a higher similarity threshold of e.g. `0.4`, it is far easier to identify clusters:

```
networkplot(pseudodistmat, highlight_annotated = TRUE,
            show_labels = TRUE, exclude_singletons = TRUE,
            min_similarity = 0.4)
```

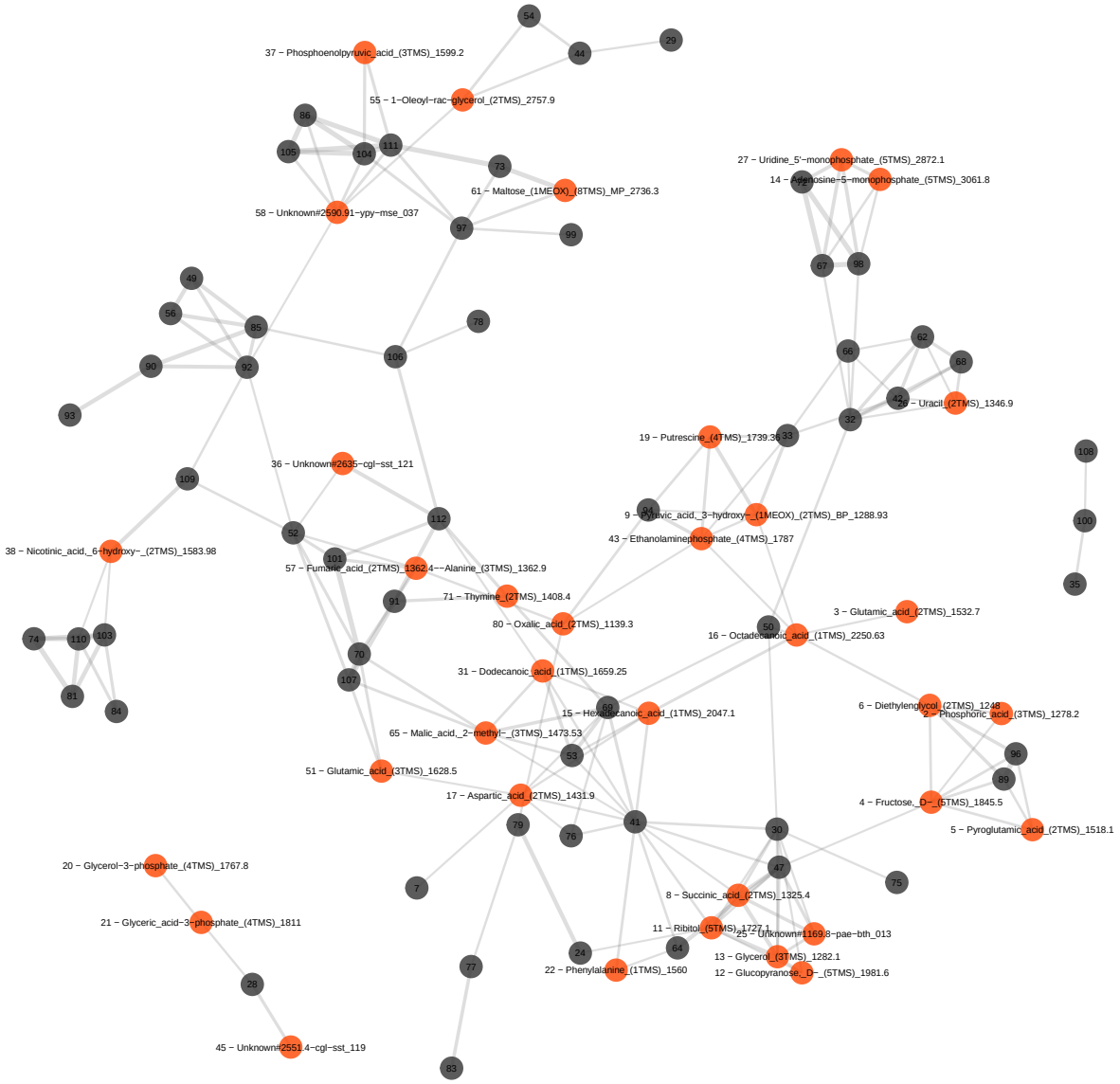

Figure B-4: Correlation network plot based on pseudospectra similarities of the GC-EI-MS example data set, generated with the custom similarity threshold of `0.4`. Grey dots indicate non-identified features, orange dots identified ones. Labels display feature IDs, along with feature annotations, if existent. Edge widths are proportional to spectral similarity of the connected features.

Presumably, the high between-cluster similarities are due to the low resolution data and the resulting fact, that fragment with different chemical composition but same unit resolution mass cannot be distinguished.

We can also use hierarchical clustering to identify clusters of similar (pseudo-)spectra. Here, too, we have to adjust `h` to account for higher between-cluster similarities:

```
HCplot(pseudodistmat, h = 0.7)
```

Figure B-5: Circularised dendrogram as a result of agglomerative hierarchical clustering with average linkage as agglomeration criterion based on pseudospectra similarities of the GC-EI-MS example data set. Each leaf represents one feature and colours encode cluster affiliation of the features. Leaf labels display feature IDs, along with feature annotations, if existent. Distance from the central point is indicative of the height of the dendrogram.

We see that e.g. octadecanoic acid, hexadecanoic acid and dodecanoic acid form a nice cluster as well as the phosphorate containing metabolites phosphoenolpyruvic acid, glyceric acid-3-phosphate, glycerol-3-phosphate and phosphoric acid itself.

It is also apparent that some features have a similarity of 1 and could therefore represent the same compound, like e.g. the features 98, 67 and 72. Those three features cluster together with AMP and UMP, suggesting that they could be nucleotides as well.

To illustrate the use of `CluMSID`'s accessory function with this type of data, we take another look at nucleotides: A signature fragment for nucleotides in GC-EI-MS is m/z 315 that derives from pentose-5-phosphates. We see this fragment in Figure 1, the spectrum of UMP (derivatised with 5 TMS groups). We can use `findFragment` to see if there are more spectra outside the cluster that freature this fragment. As we deal with unit masses, we would like to find m/z of 315 $\pm$ 0.5 which we can do by setting `tolerance = 0.5/315`:

```
fragmentlist <- findFragment(apslist, mz = 315, tolerance = 0.5/315)
#> 6 spectra were found that contain a fragment of m/z 315 +/- 1587.30158730159 ppm.

vapply(X = fragmentlist, FUN = accessID, FUN.VALUE = integer(1))
#> [1]  2 14 20 21 27 35
```

We find four more spectra that contain a 315 fragment that could be investigated closer.

Conclusion

In conclusion, every annotation method is extremely limited if only low resolution data is available and so is `CluMSID`. Still, we see that the tool works independently of chromatography and mass spectrometry method and even has the potential to give some good hints for feature annotation in GC-EI-MS metabolomics.

Clustering Mass Spectra from High Resolution DI-MS/MS Data Using CluMSID

Tobias Depke

December 31, 2018

Contents

Introduction

Although originally developed for liquid chromatography-tandem mass spectrometry (LC-MS/MS) data, `CluMSID` can also be used with direct infusion-tandem mass spectrometry (DI-MS/MS) data.

Generally, the missing retention time dimension makes feature annotation in metabolomics harder but if only direct infusion data is at hand, `CluMSID` can help to get an overview of the chemodiversity of a sample measured by DI-MS/MS.

In this example, we will use a similar sample (1µL *Pseudomonas aeruginosa* PA14 cell extract) as in the General Tutorial, measured on the same machine, a Bruker maxis^{HD} qTOF operated in ESI-(+) mode with auto-MS/MS but without chromatographic separation.

Data import

We load the file from the `CluMSIDdata` package:

```
library(CluMSID)
library(CluMSIDdata)

DIfile <- system.file("extdata",
                      "PA14_maxis_DI.mzXML",
                      package = "CluMSIDdata")
```

Data preprocessing

The extraction of spectra works the same way as with LC-MS/MS data:

```
ms2list <- extractMS2spectra(DIfile)
length(ms2list)
#> [1] 373
```

Merging of redundant spectra is less straightforward when retention time is not available. Depending on the MS/MS method it can be next to impossible to decide whether two spectra with the same precursor *m/z* and similar fragmentation patterns derive from the same analyte or from two different but structurally similar ones.

In this example, we would like to merge spectra with identical precursor ions only if they were recorded one right after another. We can do so by setting `rt_tolerance` to 1 second:

```
featlist <- mergeMS2spectra(ms2list, rt_tolerance = 1)
length(featlist)
#> [1] 349
```

We see that we have hardly reduced the number of spectra in the list. If we would decide to merge all spectra with identical precursor *m/z* from the entire run, we could do so by setting `rt_tolerance` to the duration of the run, in this case approx. 250 seconds:

```
testlist <- mergeMS2spectra(ms2list, rt_tolerance = 250)
length(testlist)
#> [1] 75
```

The resulting number of spectra is drastically lower but the danger of merging spectra that do not actually derive from the same analyte is also very big.

Generation of distance matrix

In this very explorative example, we skip the integration of previous knowledge on feature identities and generate a distance matrix right away:

```
distmat <- distanceMatrix(featlist)
```

Data exploration

Starting from this distance matrix, we can use all the data exploration functions that `CluMSID` offers. In this example workflow, we look at a cluster dendrogram:

```
HCplot(distmat)
```

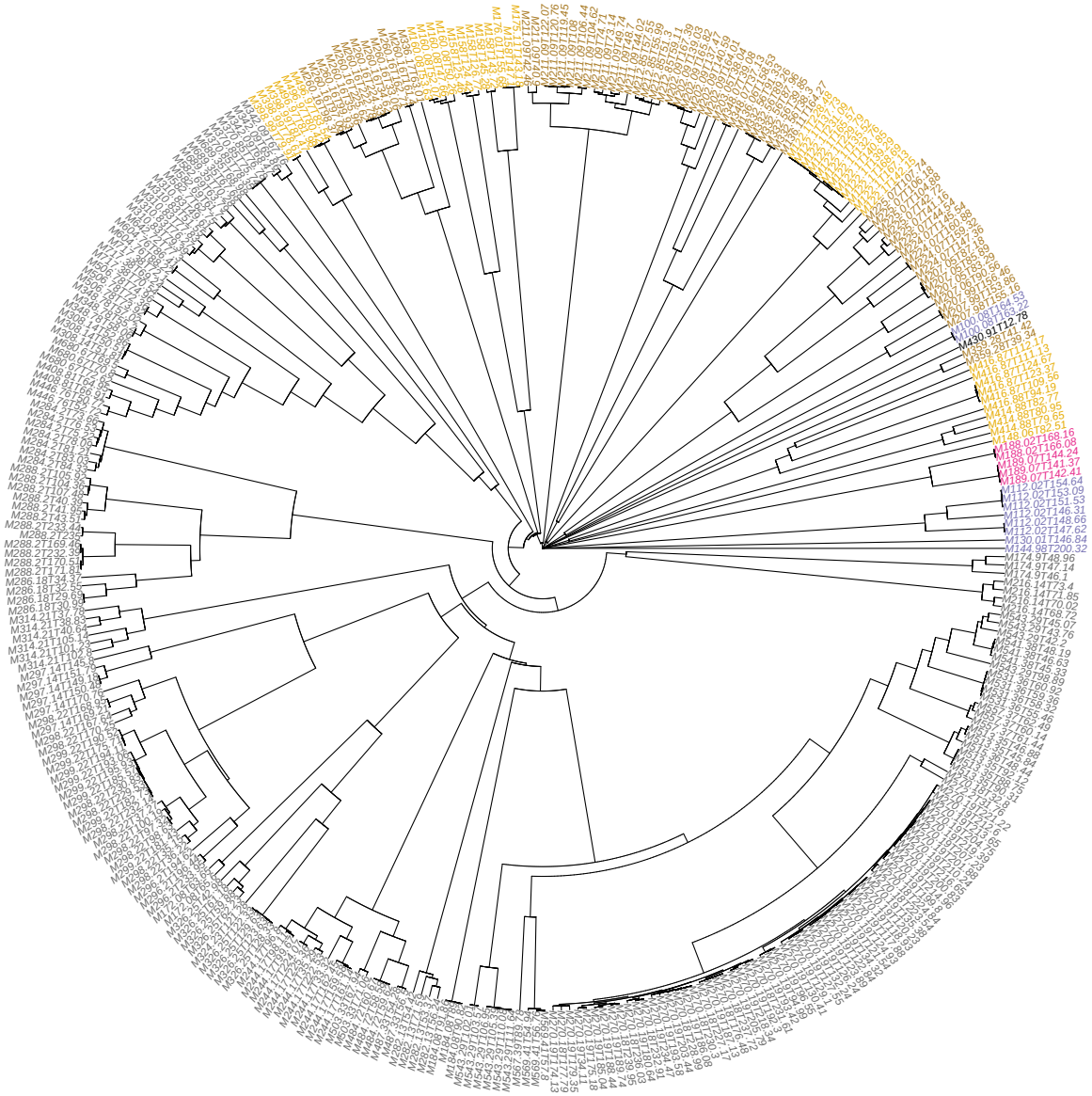

Figure C-1: Circularised dendrogram as a result of agglomerative hierarchical clustering with average linkage as agglomeration criterion based on MS^2 spectra similarities of the DI-MS/MS example data set. Each leaf represents one feature and colours encode cluster affiliation of the features. Leaf labels display feature IDs, along with feature annotations, if existent. Distance from the central point is indicative of the height of the dendrogram.

It is directly obvious that we have some spectra that are nearly identical and thus most likely derive from the same analyte, e.g. the many spectra with a precursor m/z of 270.19. But we still see nice clustering of similar spectra with different precursor m/z, e.g. the huge gray cluster that contains a lot of different alkylquinolone type metabolites (see General Tutorial).

Conclusion

In conclusion, `CluMSID` is very useful to provide an overview of spectral similarities within DI-MS/MS runs but wherever annotation is in the focus, one should not do without the additional layer of information created by chromatographic separation.

Clustering Mass Spectra from Low Resolution LC-MS/MS Data Using CluMSID

Tobias Depke

December 31, 2018

Contents

Introduction

As described in the GC-EI-MS tutorial, CluMSID can also be used to analyse low resolution data—although using low resolution data comes at a cost.

In this example, we will use a similar sample (1µL *Pseudomonas aeruginosa* PA14 cell extract) as in the General Tutorial, measured with similar chromatography but on a different mass spectrometer, a Bruker amaZon ion trap instrument operated in ESI-(+) mode with auto-MS/MS. In addition to introducing a workflow for low resolution LC-MS/MS data, this example also demonstrates that CluMSID can work with data from different types of mass spectrometers.

Data import

We load the file from the `CluMSIDdata` package:

```
library(CluMSID)
library(CluMSIDdata)

lowresfile <- system.file("extdata",
                          "PA14_amazon_lowres.mzXML",
                          package = "CluMSIDdata")
```

Data preprocessing

The extraction of spectra works the same way as with high resolution LC-MS/MS data:

```
ms2list <- extractMS2spectra(lowresfile)
length(ms2list)
#> [1] 1989
```

Like in the GC-EI-MS example, we have to adjust `mz_tolerance` to a much higher value compared to high resolution data, while the retention time tolerance can remain unchanged.

```
featlist <- mergeMS2spectra(ms2list, mz_tolerance = 0.02)
```

```
length(featlist)
#> [1] 525
```

We see that we have similar numbers of spectra as in the General Tutorial, because we tried to keep all parameters except for the mass spectrometer type comparable.

Generation of distance matrix

As we do not have low resolution spectral libraries at hand, we skip the integration of previous knowledge on feature identities in this example and generate a distance matrix right away:

```
distmat <- distanceMatrix(featlist)
```

Data exploration

Starting from this distance matrix, we can use all the data exploration functions that `CluMSID` offers.

When we now make an MDS plot, we learn that the similarity data is very different from the comparable high resolution data:

```
MDSplot(distmat)
```

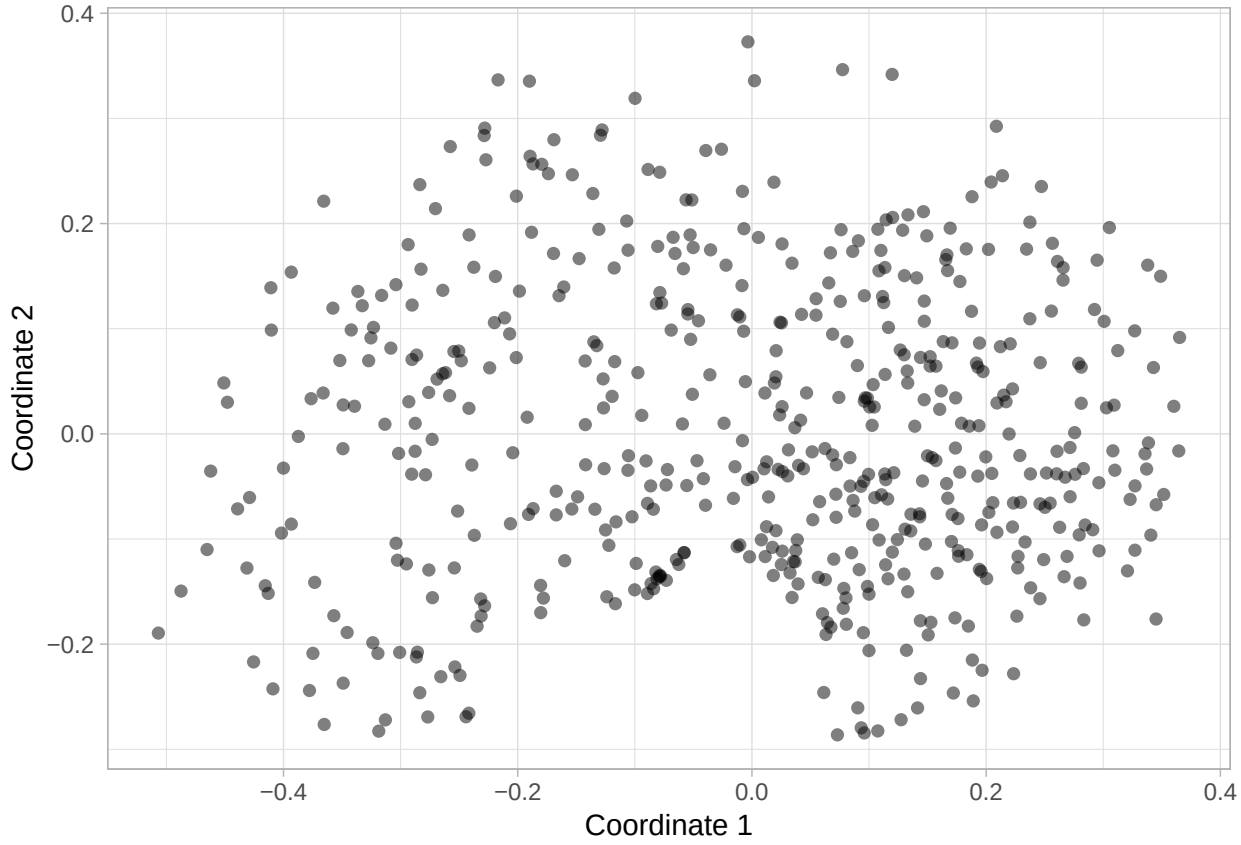

Figure D-1: Multidimensional scaling plot as a visualisation of MS^2 spectra similarities of the low resolution LC-MS/MS example data set. Black dots signify spectra from unknown metabolites.

To get a better overview of the data and the general similarity behaviour, we create a heat map of the distance matrix:

```
HCplot(distmat, type = "heatmap",
       cexRow = 0.1, cexCol = 0.1,
       margins = c(6,6))
```

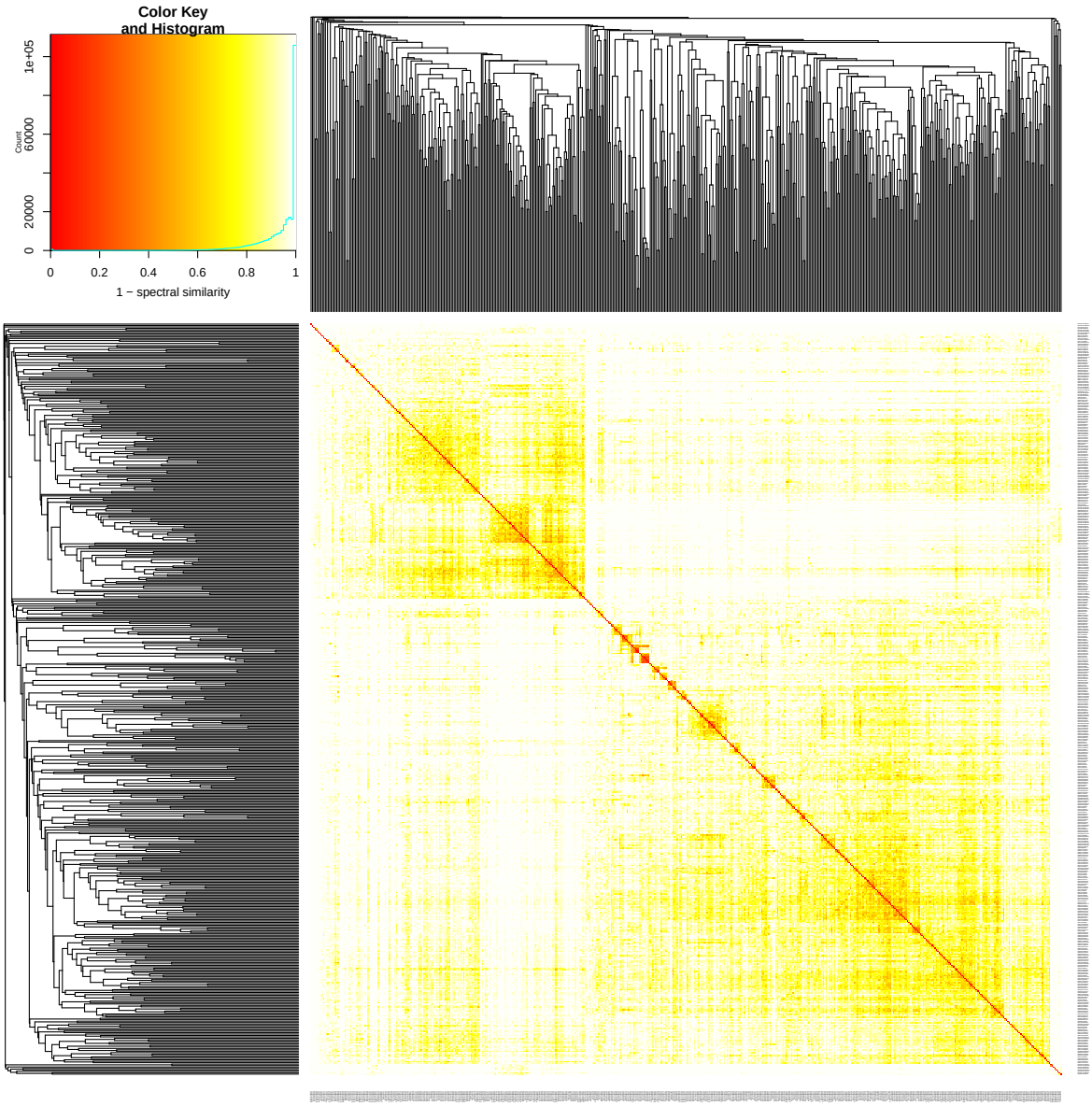

Figure D-2: Symmetric heat map of the distance matrix displaying MS^2 spectra similarities of the low resolution LC-MS/MS example data set. along with dendrograms resulting from hierarchical clustering based on the distance matrix. The colour encoding is shown in the top-left insert.

We clearly see that the heat map is generally a lot "warmer" than in the General Tutorial (an intuition that is supported by the histogram in the top-left corner), i.e. we have a higher general degree of similarity between spectra. That is not surprising as the m/z information has much fewer levels than in high resolution data and

fragments of different sum formula are more likely to have indistinguishable mass-to-charge ratios.

We also see that some more or less compact clusters can be identified. This is easier to inspect in the dendrogram visualisation:

```
HCplot(distmat, h = 0.8, cex = 0.5)
```

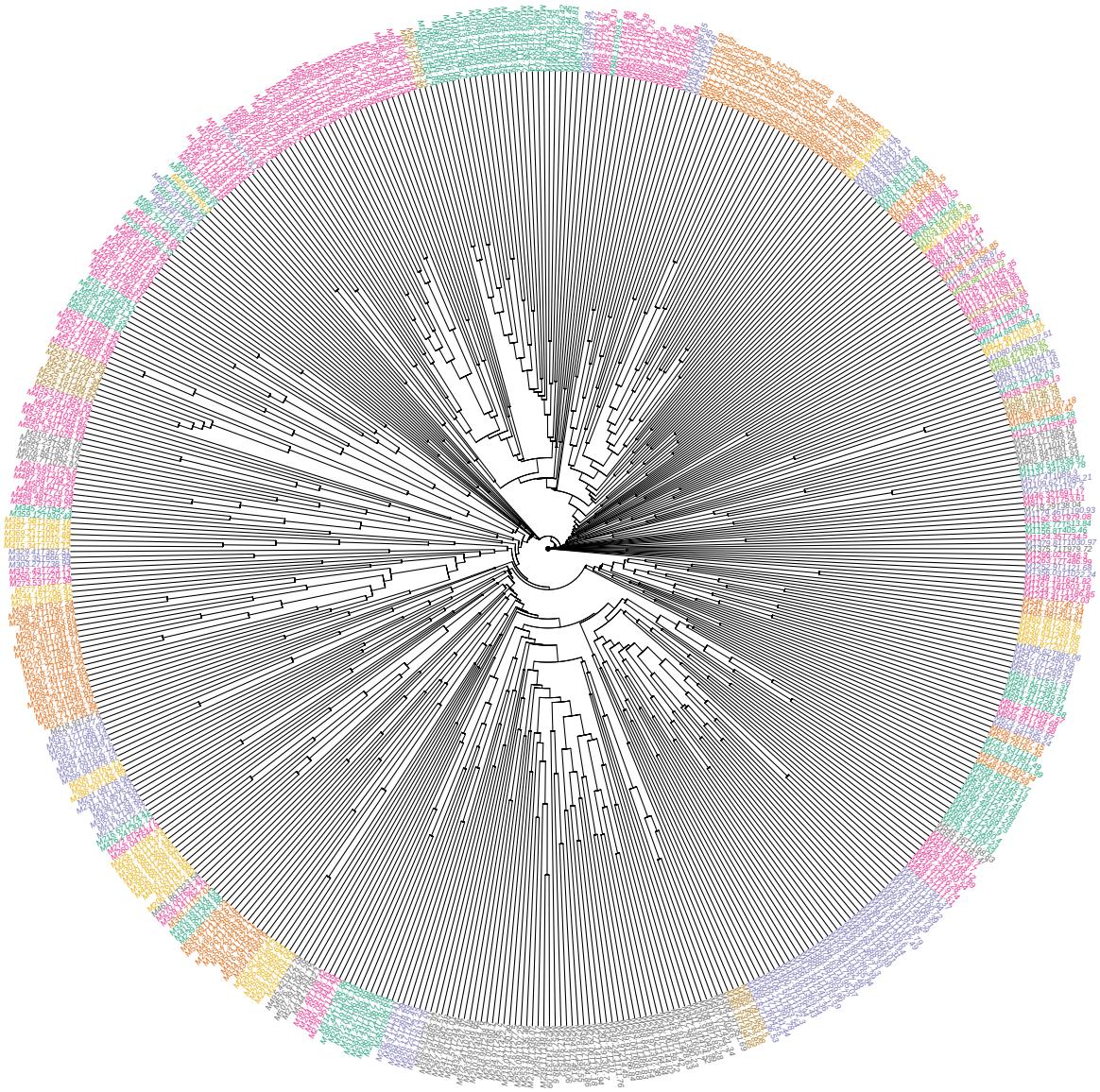

Figure D-3: Circularised dendrogram as a result of agglomerative hierarchical clustering with average linkage as agglomeration criterion based on MS^2 spectra similarities of the low resolution LC-MS/MS example data set. Each leaf represents one feature and colours encode cluster affiliation of the features. Leaf labels display feature IDs. Distance from the central point is indicative of the height of the dendrogram.

Conclusion

In conclusion, CluMSID is capable of processing low resolution LC-MS/MS data and if high resolution data is not available, it can be very useful to provide an overview of spectral similarities in low resolution data, thereby helping metabolite annotation if some individual metabolites can be identified by comparison to authentic standards. However, concerning feature annotation, high resolution methods should always be favoured for the many benefits they provide.

Using CluMSID with a Publicly Available MetaboLights Data Set

Tobias Depke

December 31, 2018

Contents

Introduction

In this tutorial, we would like to demonstate the use of CluMSID with a publicly available LC-MS/MS data set deposited on MetaboLights. We chose data set MTBLS433 that can be accessed on the MetaboLights web page (https://www.ebi.ac.uk/metabolights/MTBLS433) and which has been published in the following article:

Kalogiouri, N. P., Alygizakis, N. A., Aalizadeh, R., & Thomaidis, N. S. (2016). Olive oil authenticity studies by target and nontarget LC-QTOF-MS combined with advanced chemometric techniques. *Analytical and bioanalytical chemistry*, 408(28), 7955-7970.

The authors analysed olive oil of various providence using reversed-phase ultra high performance liquid chromatography-electrospray ionisation quadrupole time of flight tandem mass spectrometry in negative mode with auto-MS/MS fragmentation.

As a representative pooled sample is not provided, we will combine MS^2 data from several runs and use the peak picking done by the authors of the study for the merging of MS^2 spectra. Some metabolite annotations are also included in the MTBLS433 data set which we will integrate into our analysis.

```
library(CluMSID)
library(CluMSIDdata)
library(tidyverse)
```

Extract MS^2 spectra from multiple *.mzML files

For demonstration, not all files from the analysis will be included into the analysis. Four data files from the data set have been chosen that represent olive oil samples from different regions in Greece:

- `YH1_GA7_01_10463.mzML`: YH1, from Komi
- `AX1_GB5_01_10470.mzML`: AX1, from Megaloxori
- `LP1_GB3_01_10467.mzML`: LP1, from Moria
- `BR1_GB6_01_10471.mzML`: BR1, from Agia Paraskevi

Note that these are mzML files that can be processed the exact same way as mzXML files.

Furthermore, we would like to use the peak picking and annotation data from the original authors which we can read from the file `m_mtbls433_metabolite_profiling_mass_spectrometry_v2_maf.tsv`.

First, we extract MS^2 spectra from the respective files separately by using `extractMS2spectra()`. Then, we just combine the resulting lists into one list using base R functionality:

```
YH1 <- system.file("extdata", "YH1_GA7_01_10463.mzML",
                   package = "CluMSIDdata")
AX1 <- system.file("extdata", "AX1_GB5_01_10470.mzML",
                   package = "CluMSIDdata")
LP1 <- system.file("extdata", "LP1_GB3_01_10467.mzML",
                   package = "CluMSIDdata")
BR1 <- system.file("extdata", "BR1_GB6_01_10471.mzML",
                   package = "CluMSIDdata")

YH1list <- extractMS2spectra(YH1)
AX1list <- extractMS2spectra(AX1)
LP1list <- extractMS2spectra(LP1)
BR1list <- extractMS2spectra(BR1)

raw_oillist <- c(YH1list, AX1list, LP1list, BR1list)
```

Merge spectra with external peak list

First, we import the peak list by reading the respective table and filtering for the relevant information. We only need the columns `metabolite_identification`, `mass_to_charge` and `rentention_time` and we would like to replace `"unknown"` with an empty field in the `metabolite_identification` column. Plus, the features do not have a unique identifier in the table but we can easily generate that from *m/z* and RT. Note that the retention time in the raw data is given in seconds and in the data table it is in minutes, so we have to convert. For the sake of consistency, we also change the column names. We use `tidyverse` syntax but users can do as they prefer.

```
raw_mtbls_df <- system.file("extdata",
                "m_mtbls433_metabolite_profiling_mass_spectrometry_v2_maf.tsv",
                package = "CluMSIDdata")

mtbls_df <- readr::read_delim(raw_mtbls_df, "\t") %>%
    mutate(metabolite_identification =
               str_replace(metabolite_identification, "unknown", "")) %>%
    mutate(id = paste0("M", mass_to_charge, "T", retention_time)) %>%
    mutate(retention_time = retention_time * 60) %>%
    select(id,
           mass_to_charge,
           retention_time,
           metabolite_identification) %>%
    rename(mz = mass_to_charge,
           rt = retention_time,
           annotation = metabolite_identification)
```

This peak list, or its first three columns, can now be used to merge spectra. We exclude spectra that do not match to any of the peaks in the peak list. As we are not very familiar with instrumental setup, we set the limits for retention time and *m/z* deviation a little wider. To make an educated guess on mass accuracy, we take a look at an identified metabolite, its measured *m/z* and its theoretical *m/z*. We use arachidic acid $[M–H]^-$, whose theoretical *m/z* is 311.2956:

```
## Define theoretical m/z
th <- 311.2956

## Get measured m/z for arachidic acid data from mtbls_df
ac <- mtbls_df %>%
    filter(annotation == "Arachidic acid") %>%
    select(mz) %>%
    as.numeric()

## Calculate relative m/z difference in ppm
abs(th - ac)/th * 1e6
#> [1] 28.91143
```

So, we will work with an an *m/z* tolerance of ±30ppm (which seems rather high for a high resolution mass spectrometer).

```
oillist <- mergeMS2spectra(raw_oillist,
                           peaktable = mtbls_df[,1:3],
                           exclude_unmatched = TRUE,
                           rt_tolerance = 60,
                           mz_tolerance = 3e-5)
```

Add annotations

To add annotations, we use `mtbls_df` as well, as described in the General Tutorial:

```
fl <- featureList(oillist)
fl_annos <- dplyr::left_join(fl, mtbls_df, by = "id")

annolist <- addAnnotations(oillist, fl_annos, annotationColumn = 6)
```

Generate distance matrix

For the generation of the distance matrix, too, we use an *m/z* tolerance of ±30ppm:

```
distmat <- distanceMatrix(annolist, mz_tolerance = 3e-5)
```

Explore data

To explore the data, we have a look at a cluster dendrogram:

```
HCplot(distmat, h = 0.7, cex = 1)
```

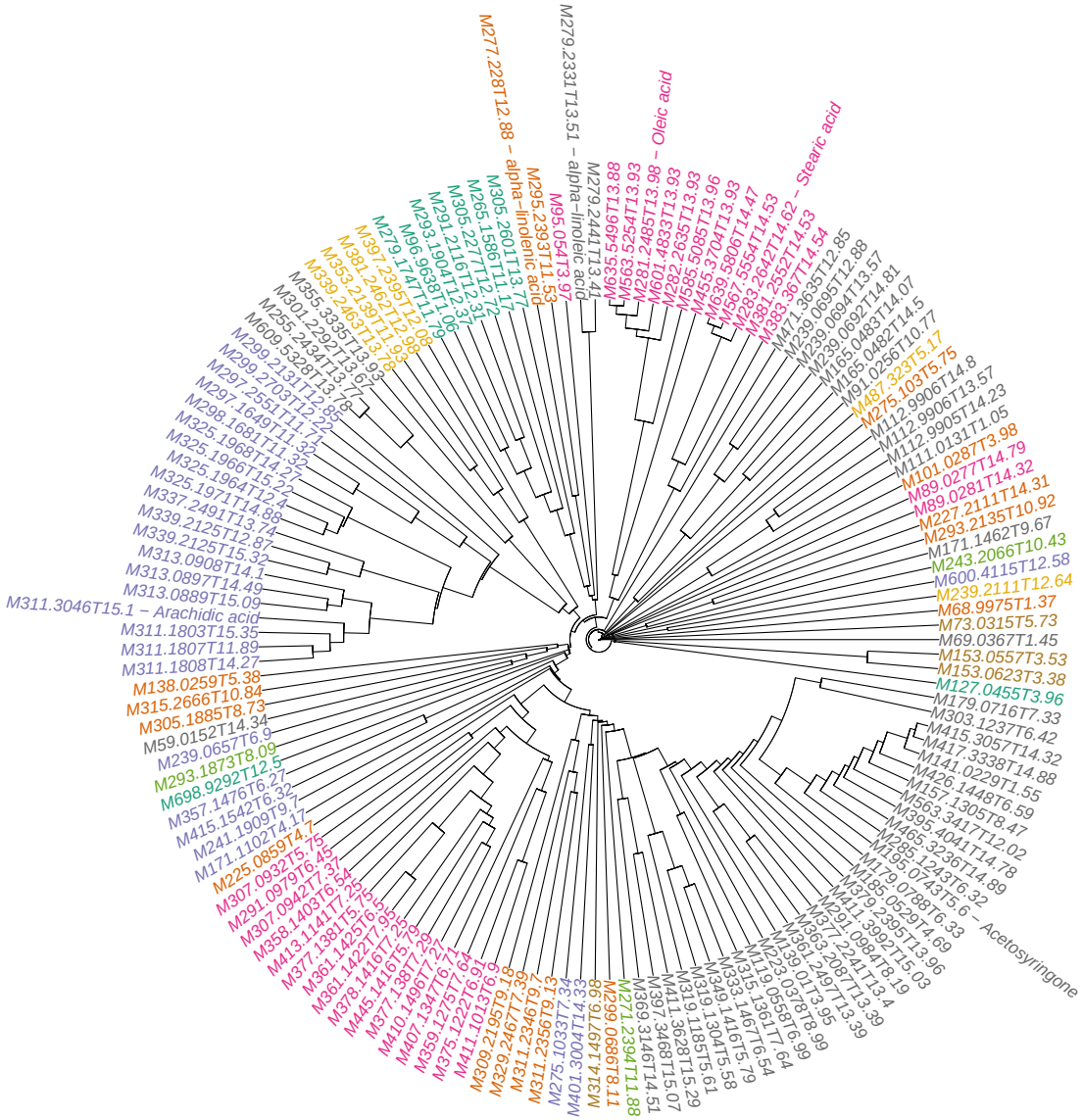

Figure E-1: Circularised dendrogram as a result of agglomerative hierarchical clustering with average linkage as agglomeration criterion based on MS^2 spectra similarities of the MTBLS433 LC-MS/MS example data set. Each leaf represents one feature and colours encode cluster affiliation of the features. Leaf labels display feature IDs. Distance from the central point is indicative of the height of the dendrogram.

Since it was not in the focus of their study, the authors identified only a few metabolites. If we look at the positions of these metabolites in the cluster dendrogram, we see that the poly-unsaturated fatty acids alpha-linolenic acid and alpha-linolenic acid are nicely separated from the saturated fatty acids stearic acid and arachidic acid. We would expect the latter to cluster together but a look at the spectra reveals that stearic acid barely produces any fragment ions and mainly contains the unfragmented $[M–H]^–$ parent ion:

```
specplot(getSpectrum(annolist, "annotation", "Stearic acid"))
```

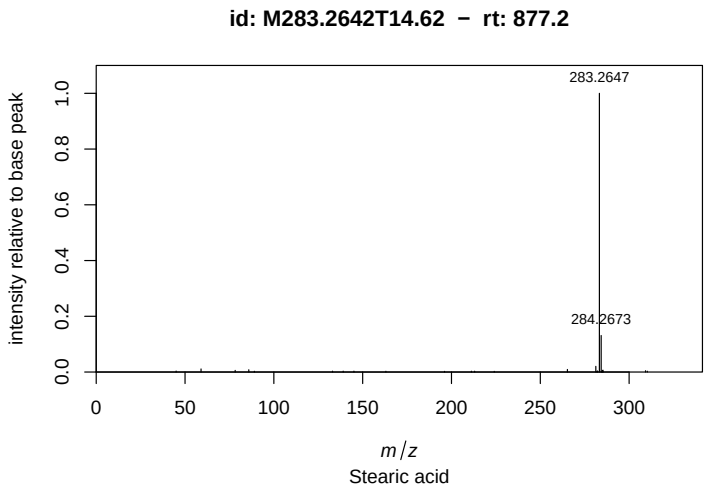

Figure E-2: Barplot for the feature M283.2642T14.62, identified as stearic acid, displaying fragment m/z on the x-axis and intensity normalised to the maximum intensity on the y-axis.

In contrast, arachidic acid produces a much richer spectrum:

```
specplot(getSpectrum(annolist, "annotation", "Arachidic acid"))
```

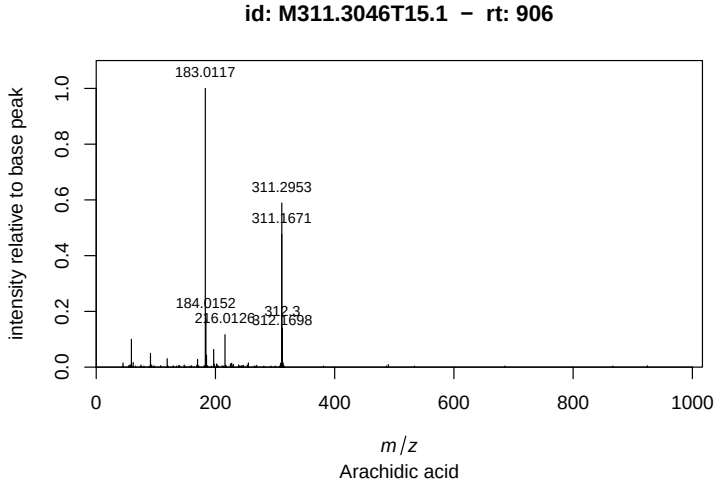

Figure E-3: Barplot for the feature M311.3046T15.1, identified as arachidic acid, displaying fragment m/z on the x-axis and intensity normalised to the maximum intensity on the y-axis.

Inspecting the features that cluster close to arachidic acid shows that many of them have an exact m/z that conforms with other fatty acids of different chain length or saturation (within the m/z tolerance), e.g. the neighbouring feature M339.2125T15.32 that could be arachidonic acid $[M+Cl]^-$.

Looking at oleic acid $[M–H]^-$, we see that it clusters very closely to M563.5254T13.93, whose m/z is consistent with oleic acid $[2M\text{-}H]^-$ and some other possible adducts.

As a last example, the only identified metabolite that does not belong to the class of fatty acids is acetosyringone, a phenolic secondary plant metabolite. It forms part of a rather dense cluster in the dendrogram, suggesting high spectral similarities to the other members of the cluster. It would be interesting to try to annotate more of these metabolite to find out if they are also phenolic compounds.

Conclusion

In conclusion, we demonstrated how to use `CluMSID` with a publicly available data set from the MetaboLights repository and how to include external information such as peak lists or feature annotations into a `CluMSID` workflow. In doing so, we had a look on a few example findings that could help to annotate more of the features in the data set and thereby showed the usefulness of `CluMSID` for samples very different from the ones in the other tutorials.