Metabolite Identification with MetFrag in R

C. Ruttkies, S. Neumann, A. Helmchen

28. Oktober 2014

Table of Contents

1 Introduction

In this section the MetFragR package is introduced and described. For a better understanding the functionality of a mass spectrometer (MS/MS) was outlined. Initially, the package shall be loaded.

```
library(rcdk)
library(metfRag)
```

For the understanding of the metabolism of several organisms different analysis methods are necessary. One of these methods is referred as mass spectrometry to detect and identify compounds in samples [1]. These compounds are partially unknown and have to be determined and annotated to understand other metabolic circles and pathways. On the other hand mass spectrometry can help to identify known proteins or posttranslational modifications (PTMs) like alteration caused by phosphate.

Actually, to obtain applicable results of an unknown molecule tandem mass spectrometry MS/MS or MS^2 is used, because additional structural hints and the exact mass of fragments of a compound are delivered [2]. Hence, two mass spectrometers are combined. In the first MS based on a given mass a single precursor ion is selected. The molecule is colliding with a neutral gas (collision-induced dissociation (CID)). This compound is then cleaved into several fragments which can be distinguished by a given mass [3].

This results in a MS/MS spectrum which contains only product ions from selected precursor ion. The spectra indicate the mass-to-charge ratio (m/z ratio) which used by the MS/MS to deflect specific ions to the analysator. The ion detector data produce a fragmentation spectrum with the m/z ratio against the abundance of each compound.

Hence, to use the statistical power of R we write the package *metfRag* which expand the features of MetFrag by the capabilities of R. Metfrag performs an in silico refragmentation of compounds. Therefor, compound libraries are compared with spectra of unknown metabolites [2]. Metfrag is written in Java so that it was integrated in metfRag by using *rJava* [4] The package 'metfRag' uses the precursor mass, the compound id or the formula to obtain a candidate list of compounds from KEGG or PubChem. Additionally, it provides functions to process the obtained candidate list and calculates ranking parameters like the optimistic, the pessimistic or the relative rank.

2 Methods of the MetFragR package

In this section the functions of this package are described. Firstly, the loading of compounds and the processing are explained. Then a ranking procedure is introduced to generate own ranking methods by using own properties or functions. Additionally, an example illustrates the functional principle on mentioned methods.

2.1 Structure file preparation

To load compounds from a chemical table file like SDF you can directly use the function from *rcdk*. In the example below a metfRag system file is used. The included SDF is the result of MetFusion processing of the MS/MS spectrum of the CASMI 2014 challenge 5 [5].

```
file <- "sdf/metfusion-category2-Challenge5-pubchem.sdf"
sdfile <- system.file(file, package = "metfRag")
mols <- load.molecules(sdfile)
```

2.2 Getting compounds from a database

To get a list of molecules without a file we send a query to available databases KEGG or PubChem by using the exact mass and the mass deviation (in ppm) to specify a compound. In addition to that, metfRag uses RCurl to retrieve the results which delivers a database [6]. Additionally, metfRag can be used to search by a database identifier/ID. KEGG uses three parameters among them also the mol weight. Let us view an example for PubChem to get some molecules but the same could be applied for KEGG:

```
params <- list(mass=174.05, range=0.001);
pubchem.mol <- db.pubchem.getId(params)[1:5,];
pubchem.mol[1:5]

## [[1]]
## [1] "84766583"
##
## [[2]]
## [1] "84654828"
##
## [[3]]
## [1] "84654827"
##
## [[4]]
## [1] "84654826"
##
## [[5]]
## [1] "84654825"
```

Each entry of the result is converted to an IAtomContainer which is described in the manual from *rcdk* [5]. Every container contains properties which can be altered by an user. If one send a request, PubChem add specific properties like IUPAC name and deliver the associated molecules. This information is necessary to process the molecules further with set operations.

KEGG use the Representational State Transfer (REST) paradigma to grant web access to own data. Therefor, metfRag use the package KEGGREST from Bioconductor [7]. Hence, each query and the obtained additional properties build two requests.

Similarly, metfRag use for the PubChem requests the paradigma PUG REST (power user gateway) which apply FTP to give a access to the data [8]. To obtain the results in a structured form metfRag uses XML to process the resulted XML files to IAtomContainer [9]. The number of candidates is limited by PubChem and metfRag to default value 100.

```
  pubchem.container <- db.pubchem.getMoleculeContainer(pubchem.mol)
  get.properties(pubchem.container[[1]])[1:5]

## $`cdk:Title`
## [1] "84766583"
##
## $`cdk:Remark`
## [1] NA
##
## $COMPOUND_CID
## [1] "84766583"
##
## $COMPOUND_CANONICALIZED
## [1] "1"
##
## $CACTVS_COMPLEXITY
## [1] "143"
```

2.3 Union, intersect and build difference of compounds

To perform set operations on IAtomContainers two compound lists are necessary. The operation occurs by using molecule properties which are equal in both container lists. Thereto, the database prefix from each molecule property was removed. To adept the equal properties we could omit the third link parameter or we call the function `showLinkOptions`.

```
  opt <- container.union(pubchem.container,pubchem.container)

## NULL

## Warning: Please choose a link from above. You could use the
##          function: comm.lib.showLinkOptions(set.a,set.b)
```

This show us the possible links of both container. To union the both sets we could call `container.union` and `container.intersect` to intersect two container sets. Then, the molecule property from first list was compared with each molecule property from second list. The doubled occurence of an element are filtered which means that the first element is in the list and the second element is discard.

```
  container.union(pubchem.container,
                  pubchem.container,
                  "MOLECULAR_FORMULA")

  container.intersect(pubchem.container,
                      pubchem.container,
                      "MOLECULAR_FORMULA")
```

If we would like to build the symmetric difference or asymmetric difference we could call `container.symmetric.difference` and `container.asymmetric.difference`.

```
  container.symmetric.difference(pubchem.container,
                                 pubchem.container,
                                 "MOLECULAR_FORMULA")

  container.asymmetric.difference(pubchem.container,
                                 pubchem.container,
                                 "MOLECULAR_FORMULA")
```

We could search a molecule by a link in a list of compounds. At that point we are interested in all positions where the molecule occurs. Thereto, we call the function `common.lib.lookup`. The link describes a property of the molecule. The value of a given molecule is validated against the properties in the compound list. To split a property like 'IUPAC_INCHIKEY' into parts the parameter 'split' is introduced. This parameter is a list with a seperator and the position of the substring which is used for the comparison.

```
  split <- list(sep="-", pos=3)
  common.lib.lookup(pubchem.container[[1]],
                    pubchem.container,
                    "IUPAC_INCHIKEY",
                    split)

## [1] 1 2 3 4 5
```

Another variant of this principle is the function `common.lib.lookupFirst`. Only the first occurence of a molecule is considered.

```
  split <- list(sep="-", pos=3)
  common.lib.lookupFirst(kegg.container[[1]],
                         pubchem.container,
                         "IUPAC_INCHIKEY",
                         split)
```

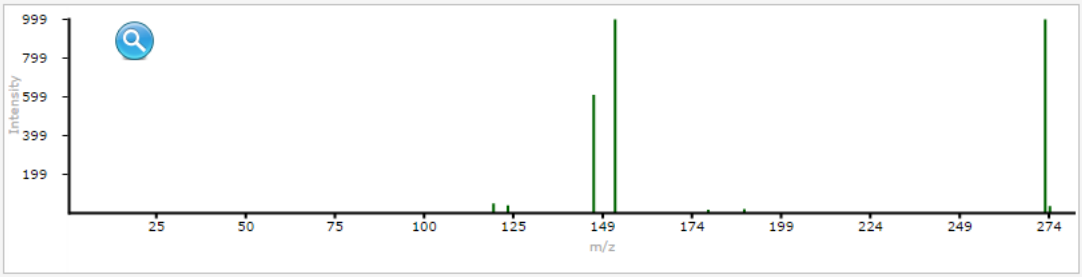
2.4 Scoring structures with MetFrag

To score received compounds from a database MetFragR uses two options. On the one hand scoring of a chemical table format SDF or mol file. For this option metfRag needs a path. Obligatory this package needs the mass-to-charge ratio values from the peak list, the intensity values and the exact mass of the precursor ion.

```
file <- "sdf/metfusion-category2-Challenge5.mf"
queryfile <- system.file(file, package = "metfRag")
challenge5 <- read.table(queryfile,
                         sep="\t",
                         col.names=c("mz", "inten"))

scoredMols <- score.molecules.from.sdf(sdfile,
                                       mzs=challenge5[,"mz"],
                                       ints=challenge5[,"mz"],
                                       exact.mass=290.0646,
                                       mz.abs=0.001,
                                       mz.ppm=5,
                                       search.ppm=5,
                                       pos.charge=TRUE,
                                       mode=1,
                                       tree.depth=2)
scoredMols <- mols
```

These example values are extracted from the MetFrag Website [10]. A MS/MS spectrum was generated by using the entered parameters (Figure 1).



Peak#	m/z	abs. int.	rel. int.
1	119.051	467.616	47.0
2	123.044	370.662	37.0
3	147.044	6078.145	607.0
4	153.019	10000.0	999.0
5	179.036	141.192	14.0
6	189.058	176.358	18.0
7	273.076	10000.0	999.0
8	274.083	318.003	32.0

Figure 1: MS/MS spectrum of the MetFrag Website with the neutral mass 272.06847 and the example values for each peak. They are characterised by the mass-to-charge values (m/z), the absolute intensities (abs.int) and the relative intensities (rel.int).

On the other hand the molecules can be passed as IAtomContainer list which is loaded by *rcdk*. For this reason the optional search.ppm parameter is not necessary. Previously, the compound list could have been generated with help of the database functions `db.kegg.getMoleculeContainer` or `db.pubchem.getMoleculeContainer` mentioned before in section 2.2.

```
mzs <- c(119.051,123.044,147.044,153.019,179.036,
         189.058,273.076,274.083)
ints <-c(467.616,370.662,6078.145,10000.0,141.192,
         176.358,10000.000,318.003)

pubchem.container <- score.molecules.from.container(
  pubchem.container,
  mzs,
  ints,
  272.06847)
```

After the scoring seven new scoring properties are added to the database specific properties.

```
  tail(get.properties(pubchem.container[[1]]), n=7)
```

```
## $MetFragID
## [1] "0"
##
## $PeakScore
## [1] 922.6
##
## $BondEnergyScore
## [1] 960
##
## $Score
## [1] 0.75
##
## $NoPeaksExplained
## [1] 1
##
## $Rank
## [1] 1
##
## $PeaksExplained
## [1] "119.051 47.0 "
```

2.5 Rank molecules among a given function or IAtomContainer properties

If we want like to rank given molcules we need to specify the ranking parameters. Accordingly, the values BC (candidates with a better score), WC (candiates with a worse score), EC (candidates with equal score) and TC (total number of candidates) were introduced.

With given parameters we can rank our molecules and determine the optimistic rank (OR), the pessimistic rank (PR) and the relative rank (RRP) of a molecule based on the condition [2].

$$PR = BC + EC + 1$$
$$OR = BC + 1$$
$$RRP = 0.5 \cdot (1 - \frac{BC - WC}{TC - 1})$$

For example we have a list of molecules. These molecules should be ranked by numeric molecule properties. To display all numeric properties the function `comm.lib.showNumberOptions` could be used. If you call the ranking function `scoring.getRanks` without a sorting property the output will be the same.

```
  comm.lib.showNumberOptions(pubchem.container)
```

With this parameter we get the possible sorting parameters by which the compounds are classified. In addition to this we need the common condition which we get by using the function `comm.lib.showLinkOptions`. As a result the condition list contain a name which determined by molecule property name and a value which determined by molecule property value.

```
  sorting <- list("PeakScore","Score")
  condition <- list(IUPAC_INCHIKEY="PAFJIHSCEHOAMQ-UHFFFAOYSA-N")
  scoring.getRanks(pubchem.container, sorting, condition)

## list()
```

We get a list with the ranking results based on the molecule with the condition properties sorted by the sorting parameters.

Additionally, we can rank by an own function. Hence, we declare a list as mentioned before. In contrast to the previous example a new list, including the defined scoring function, is added into the existing list. The first parameters are the sorting properties which used in the anonymous function. In our example the list was ordered by the PeakScore, the Score and the own function. We can force the splitting of compound properties with a split parameter which explained in section 2.3.

```
  scoring.getRanks(pubchem.container,
                   list("PeakScore",
                        "Score",
                        list("PeakScore", "Score", function(x, y) x*y)),
                   list(IUPAC_INCHIKEY="PAFJIHSCEHOAMQ"),
                   list(sep="-", pos=1))

## list()
```

2.6 Get fragments of a compound

To get fragments of a compound a SMILES string and the path to MetFrag binaries are required. Hence, the compound is fragmented by using the MetFrag fragmenter which use the most common substructure. The SMILES is available from the compound properties [2]. The fragments can be visualised by calling `view.molecule.2d`.

```
  path <- "D:/Documents/MetFrag/lib/"
  smiles <- "CN(C)CC(C1=C=C(C=C1)OC)C2(CCCCC2)O"
  frag.generateFragments(path, smiles)
```

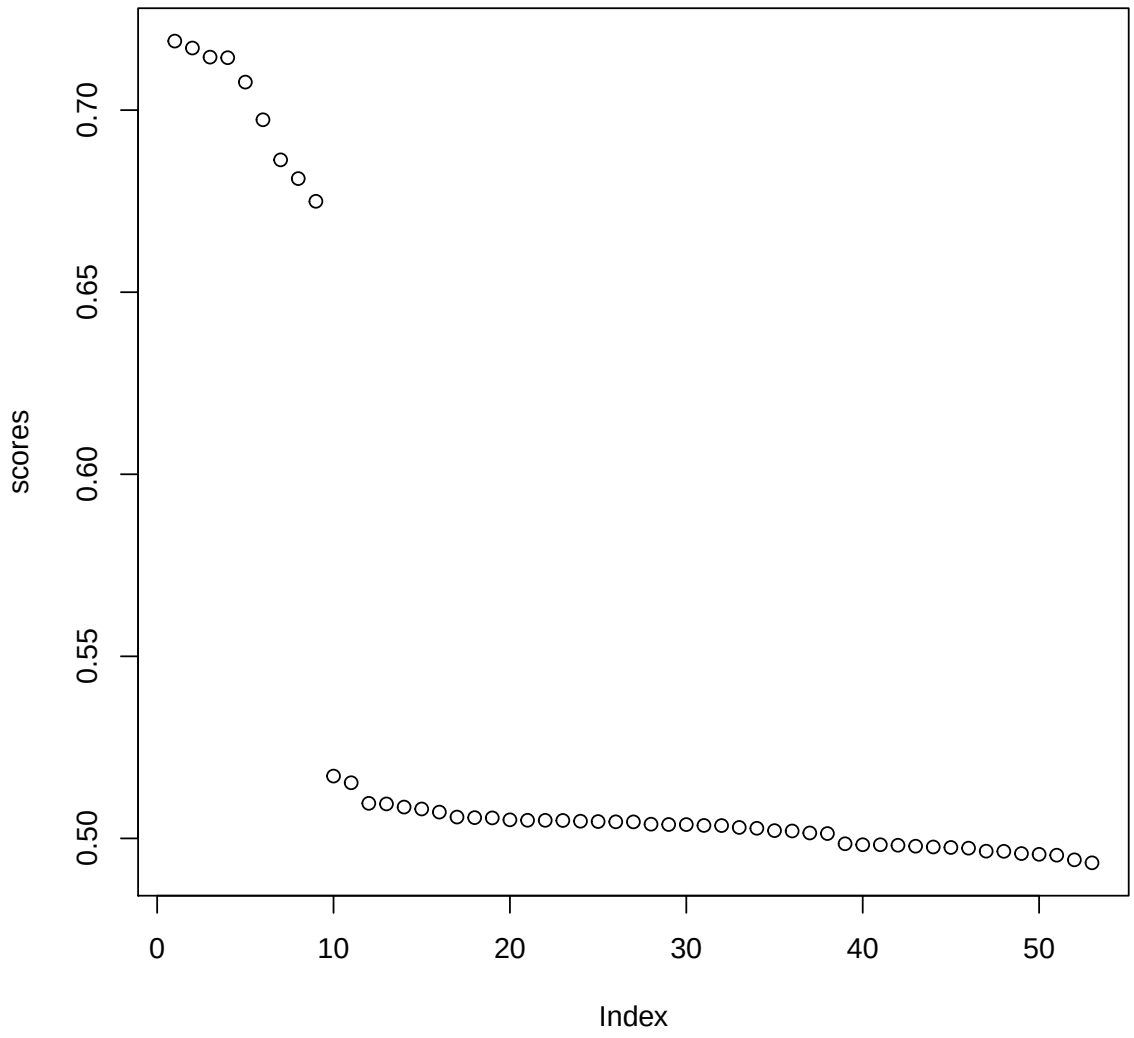
3 Visualisation of Results

3.1 Score distribution

First, we check the score distribution. Hence, we extract the numeric values of a list of molecules and save them in a list. These are displayed as a dot plot (Figure **??**) whereby the axis of ordinates shows the score values and the axis of abscissas shows the position of the scored values.

```
scores <- getScores(scoredMols, scoreprop="Score")
scores <- getScores(scoredMols, scoreprop="newscore")

plot(scores)
```



3.2 Plotting Maximum Common Substructure

To calculate the MCSS for given clusters the function `getClusterMCSS` is used. This describes a a 1:1 atom correspondence between two compounds. The bond connection and atom types are equal in both molecules in addition to the largest number of bonds or atoms. Such structural similarities are often used to predict bioactive compounds [11].

```
clusterreps <- getClusterMCSS(cluster, mols=scoredMols, k=7, which=1:7)
```

Each MCSS of a cluster can be plotted separately - with an overlay of the cluster number with the function `plotMol`. This function requires i. a. molecules as IAtomContainer objects and an optional watermark (Figure 2).

```
plotMol(clusterreps[[1]], watermark=1)
plotMol(clusterreps[[2]], watermark=2)
...
```

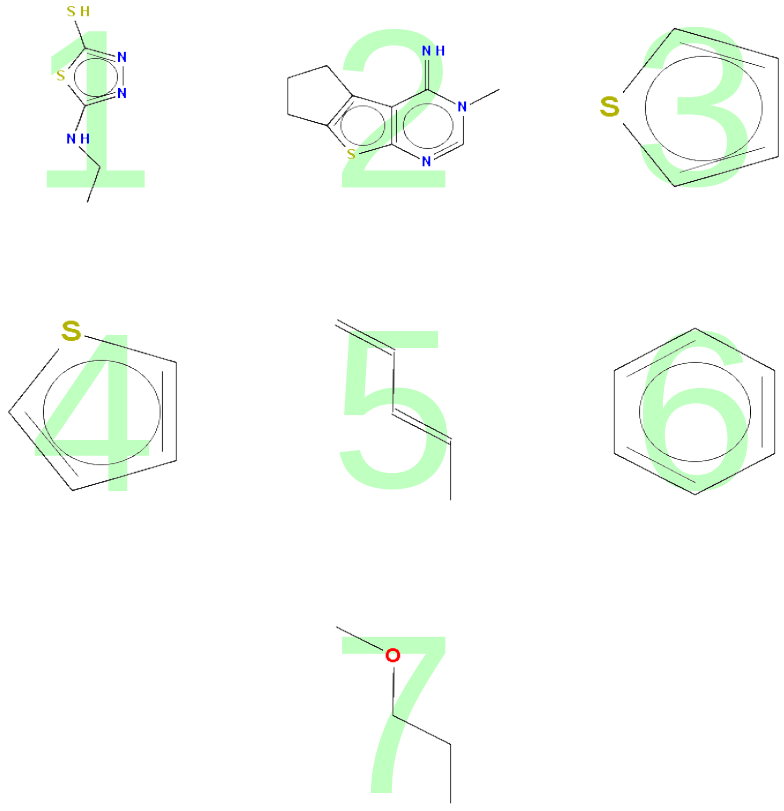


Figure 2: Plot of the most common substructure of each cluster with the number overlaid.

3.3 Candidate Clustering

3.3.1 Clustering with displayed cluster numbers

If we want to plot similar compounds in a dendrogram, firstly we have to create clusters. On that point we restrict the length of the list to the first 50 molecules and group them based on structural similarity. We could use a SD file or a list of compounds with the score property and the database property. In this example we get 10 groups.

```
scoredMols <- scoredMols[1:min(50, length(scoredMols))]
cluster <- hclust.mols(mols=scoredMols,
                       scoreprop="Score",
                       idprop="DatabaseID")
```

To plot the molecule cluster as a dendrogram with cluster numbers overlaid, firstly we create a plot either by applying `plot` or by using `hclust`. Then we create the numbers of each cluster. For this purpose the maximum common substructure (MCSS) is calculated with $k = 7$ clusters whereby all 7 clusters are framed with a rectangle with a black border color.

Thereto, the clusters are drawn into the consisting plot (Figure 3). Hence, the height are referred as Tanimoto distance which formal formulated as [12]

$$T_d(X, Y) = -\log_2 \left(\frac{\sum_i (X_i \wedge Y_i)}{\sum_i (X_I \vee Y_i)} \right)$$

In our case, this describe the structural similarity of two different compounds to a third compound.

```
plot(cluster, hang=-1)
myimages.clustNumbers(cluster, k=7, which=1:7, border=2)
```

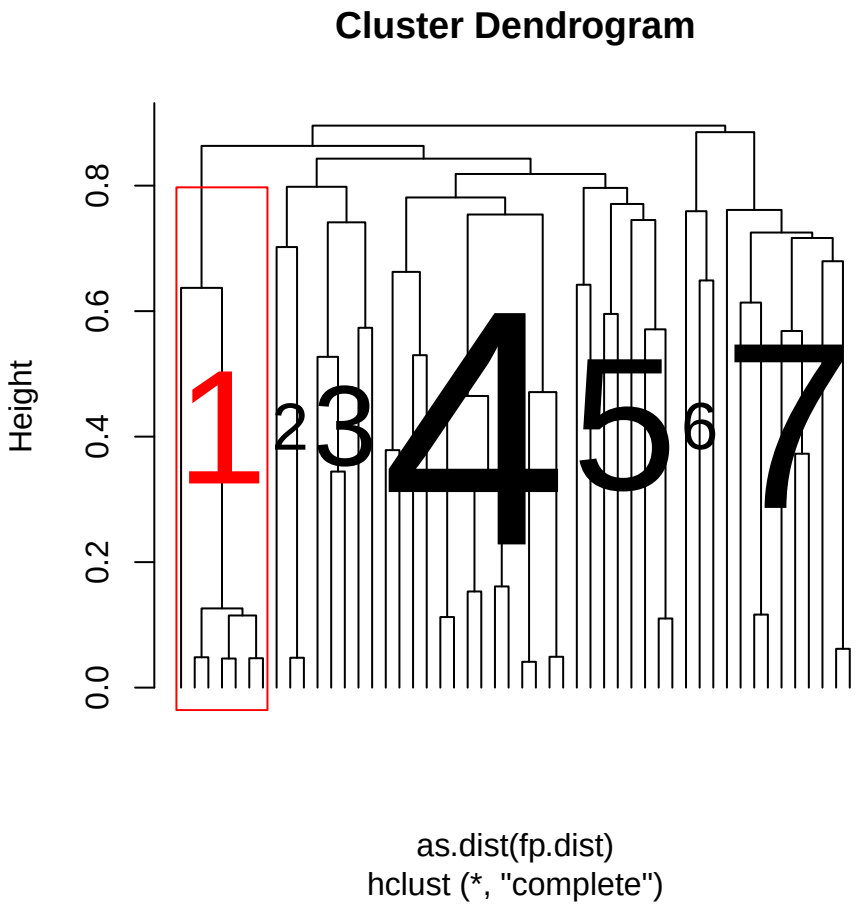


Figure 3: Example plot of clustered compounds. If the MCSS is most similar two of 50 molecules joined into a single cluster illustrate by a vertical line. In contrast to that, if a cluster split into new cluster the horizontal bar illustrate the dissimilarity. The cluster can numbered and marked with a coloured border.

3.3.2 Clustering with displayed MCSS

If we want to plot the clusters with the most common substructure overlaid instead of a cluster number, we create a cluster dendrogram as mentioned before. In the following, we use the function `myimages.hclust` to create MCSS below the cluster dendrogram (Figure 4).

```
plot(cluster, hang=-1)
myimages.hclust(cluster, mols=scoredMols, k=7, which=1:7, border=2)
```

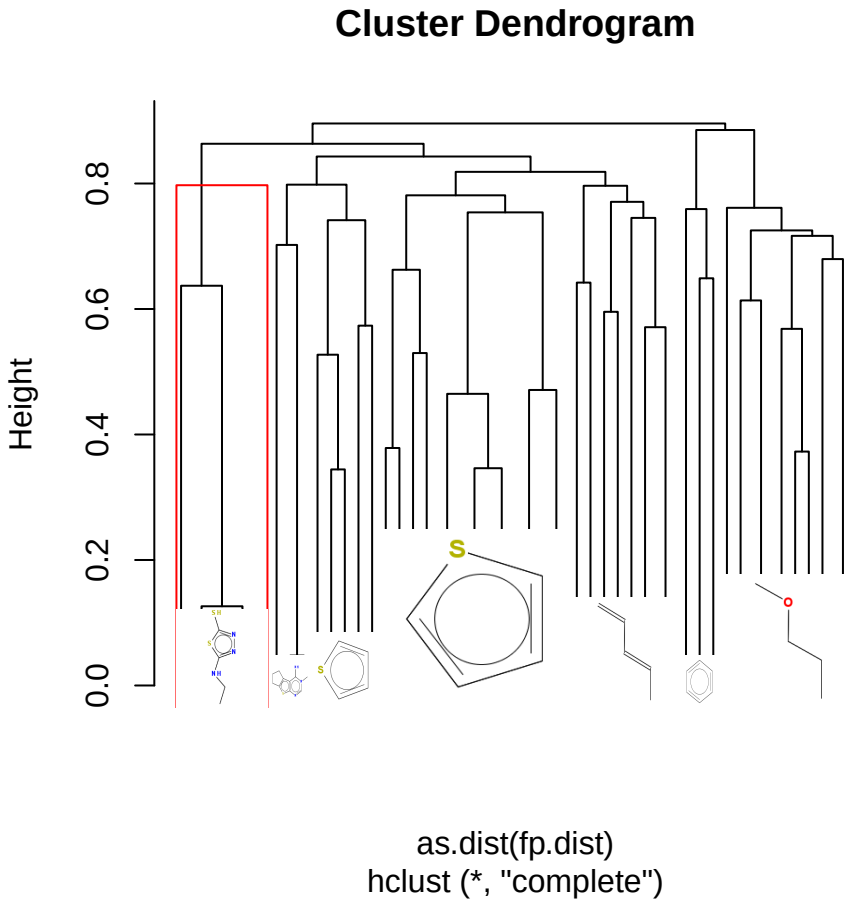


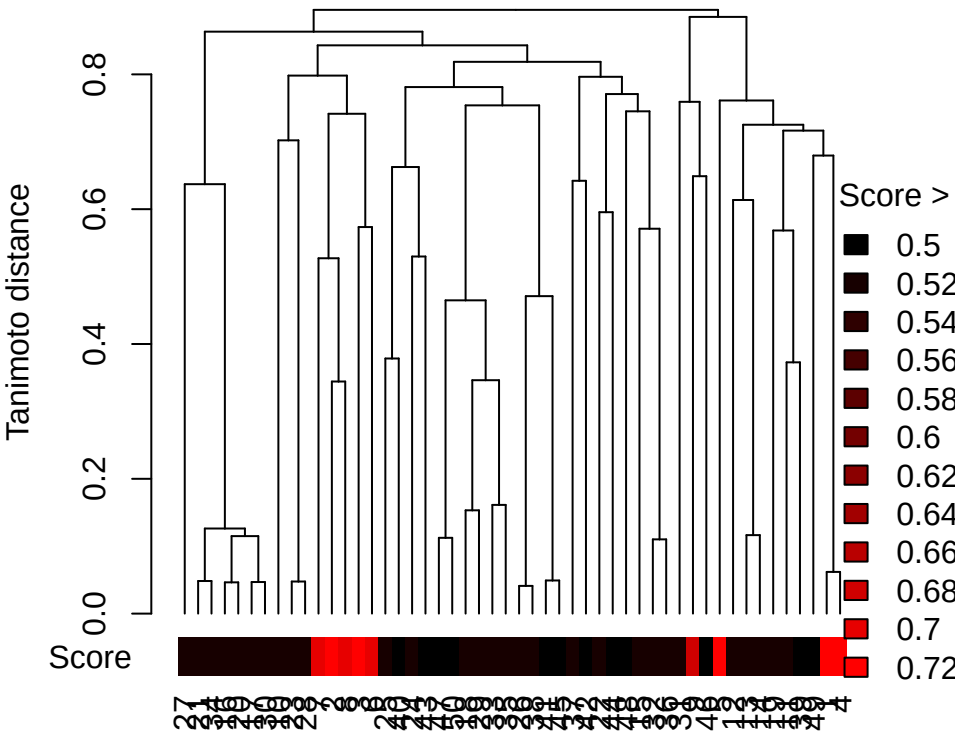
Figure 4: Cluster dendrogram of 50 sample compounds with given coloured border to mark a specific cluster.

3.3.3 Clustering with displayed Scores

Instead of the MCSS or the cluster numbers we can plot the score and the color gradient, respectively. Therefor, the function `plotCluster` is used. The function expects i. a. the list of the molecules and the score which are listed in the properties of molecules.

```
plotCluster(scoredMols, score="Score", h=0.2, scoreprop="newscore")
```

The color indicates the level of a score of the compounds where a light red color indicates a high and a black color a low score. Each compound contains a score which is plotted at the bottom of the figure as a red-black gradient. The height of the dendrogram is given as Tanimoto distance which is explained in section 3.3.1.



4 Summary

In this manual the package 'metfRag' was described. Based on the precursor mass a candidate list of 'KEGG' and 'PubChem' is returned. In future versions new databases will be added like 'ChemSpider' or 'ChEMBL' to get more putative candidates. The databases especially 'PubChem' possess more several search methods whereby three methods are implemented. It is necessary to expand the capabilities to retrieve more accurate candidates, for example the combination of search methods per database.

Several lists from different or the same databases can be processed by using simple set operations. Thereto, the properties of the molecules are used which are returned. Nevertheless only sets with equal properties are regared. To consider these compounds a homogenisation and preprocessing of the properties are necessary respectively. Hence, a thesaurus could applied to get the suspected semantic synonyms of several property names while comparing the property names of each compound list.

The functionality of ranking is implemented. A scored compound list can be classified while numeric properties are used to rank the molecules to a specific order. Additionally the user can write own functions which use molecule properties to calculate new values with which the ranking is determined. The optimistic and pessimistic rank as well as the relative rank can be calculated.

The scoring of a list of compounds performed by MetFrag which is written in Java and integrated through *rJava*.

Literatur

[1] Warwick B. Dunn. „Current trends and future requirements for the mass spectrometric investigation of microbial, mammalian and plant metabolomes.“ eng. In: *Phys Biol* 5.1 (2008), S. 011001. DOI: 10.1088/1478-3975/5/1/011001. URL: http://dx.doi.org/10.1088/1478-3975/5/1/011001.

[2] Sebastian Wolf u. a. „In silico fragmentation for computer assisted identification of metabolite mass spectra.“ eng. In: *BMC Bioinformatics* 11 (2010), S. 148. DOI: 10.1186/1471-2105-11-148. URL: http://dx.doi.org/10.1186/1471-2105-11-148.

[3] JEOL. *Tandem Mass Spectrometry (MS/MS) Mass Spectrometry - Essays and Tutorials*. JEOL USA , Inc. 2006. URL: http://www.jeolusa.com/DesktopModules/Bring2mind/DMX/Download.aspx?EntryId=78.

[4] Simon Urbanek. *rJava: Low-level R to Java interface*. R package version 0.9-6. 2013. URL: http://CRAN.R-project.org/package=rJava.

[5] Rajarshi Guha. „Chemical Informatics Functionality in R“. In: *Journal of Statistical Software* 18.6 (2007).

[6] Duncan Temple Lang. *RCurl: General network (HTTP/FTP/...) client interface for R*. R package version 1.95-4.1. 2013. URL: http://CRAN.R-project.org/package=RCurl.

[7] Dan Tenenbaum. *KEGGREST: Client-side REST access to KEGG*. R package version 1.4.1.

[8] Mark R. Southern und Patrick R. Griffin. „A Java API for working with PubChem datasets.“ eng. In: *Bioinformatics* 27.5 (2011), S. 741–742. DOI: 10.1093/bioinformatics/btq715. URL: http://dx.doi.org/10.1093/bioinformatics/btq715.

[9] Duncan Temple Lang. *XML: Tools for parsing and generating XML within R and S-Plus*. R package version 3.98-1.1. 2013. URL: http://CRAN.R-project.org/package=XML.

[10] URL: http://msbi.ipb-halle.de/MetFrag/.

[11] Yiqun Cao, Tao Jiang und Thomas Girke. „A maximum common substructure-based algorithm for searching and predicting drug-like compounds.“ eng. In: *Bioinformatics* 24.13 (2008), S. i366–i374. DOI: 10.1093/bioinformatics/btn186. URL: http://dx.doi.org/10.1093/bioinformatics/btn186.

[12] D. J. Rogers und T. T. Tanimoto. „A Computer Program for Classifying Plants.“ eng. In: *Science* 132.3434 (1960), S. 1115–1118. DOI: 10.1126/science.132.3434.1115. URL: http://dx.doi.org/10.1126/science.132.3434.1115.