Analysis on serum dataset

Contents

1 Abstract

Untargeted mass spectrometry is a robust tool for biological research, but researchers universally time consumed by dataset parsing. We developed MCnebula, a novel visualization strategy proposed with multidimensional view, termed multi-chemical nebulae, involving in scope of abundant classes, classification, structures, sub-structural characteristics and fragmentation similarity. Many state-of-the-art technologies and popular methods were incorporated in MCnebula workflow to boost chemical discovery. Notably, MCnebula can be applied to explore classification and structural characteristics of unknown compounds that beyond the limitation of spectral library. MCnebula was integrated in R package and public available for custom R statistical pipeline analysis. Now, MCnebula2 (R object-oriented programming with S4 system) is further available for more friendly applications.

2 Introduction

We know that the analysis of untargeted LC-MS/MS dataset generally begin with feature detection. It detects 'peaks' as features in MS^1 data. Each feature may represents a compound, and assigned with MS^2

spectra. The MS^2 spectra was used to find out the compound identity. The difficulty lies in annotating these features to discover their compound identity, mining out meaningful information, so as to serve further biological research. Herein, a classified visualization method, called MCnebula, was used for addressing this difficulty. MCnebula utilizes the state-of-the-art computer prediction technology, SIRIUS workflow (SIRIUS, ZODIAC, CSI:fingerID, CANOPUS)[1–5], for compound formula prediction, structure retrieve and classification prediction. MCnebula integrates an abundance-based classes (ABC) selection algorithm into features annotation: depending on the user, MCnebula focuses chemical classes with more or less features in the dataset (the abundance of classes), visualizes them, and displays the features they involved; these classes can be dominant structural classes or sub-structural classes. With MCnebula, we can switch from untargeted to targeted analysis, focusing precisely on the compound or chemical class of interest to the researcher.

3 Set-up

Load the R package used for analysis. In the following analysis process, to illustrate the source of the function, we use the symbol `::` to mark the functions, e.g., `dplyr::filter`. The functions that were not marked may source from MCnebula2 or the packages that R (version 4.2) loaded by default.

```
library(MCnebula2)
library(exMCnebula2)
```

4 Integrate data and Create Nebulae

4.1 Initialize analysis

Set SIRIUS project path and its version to initialize mcnebula object.

```
mcn <- mcnebula()
mcn <- initialize_mcnebula(mcn, "sirius.v4", ".")
ion_mode(mcn) <- "pos"
```

Create a temporary folder to store the output data.

```
tmp <- paste0(tempdir(), "/temp_data")
dir.create(tmp, F)
export_path(mcn) <- tmp
```

In order to demonstrate the process of analyzing data with MCnebula2, we provide a 'mcnebula' object that was extracted in advance using the `collate_used` function, which means that all the data used in the subsequent analysis has already stored in this 'mcnebula' object, without the need to obtain it from the original Project directory. This avoids the hassle of downloading and storing a dozen GB of raw files. The following, we use the collated dataset containing 6501 features with chemical formula identification. This dataset was origin and processed from the research in article: https://doi.org/10.1016/j.cell.2020.07.040

```
exfiles <- system.file("extdata", package = "exMCnebula2")
```

Load the '.rdata' file.

```
load(paste0(exfiles, "/mcn_serum6501.rdata"))
mcn <- mcn_serum6501
rm(mcn_serum6501)
```

4.2 Filter candidates

Suppose we predicted a potential compound represented by LC-MS/MS spectrum, and obtained the candidates of chemical molecular formula, structure and chemical class. These candidates include both positive and negative results: for chemical molecular formula and chemical structure, the positive prediction

was unique; for chemical class, multiple positive predictions that belong to various classification were involved. We did not know the exact negative and positive. Normally, we ranked and filtered these according to the scores. There were numerious scores, for isotopes, for mass error, for structural similarity, for chemical classes… Which score selected to rank candidates depends on the purpose of research. Such as:

- To find out the chemical structure mostly be positive, ranking the candidates by structural score.
- To determine whether the potential compound may be of a certain chemical classes, ranking the candidates by the classified score.

Ether by `filter_formula()`, `filter_structure()` or `filter_ppcp()`, the candidate with top score can be obtained. However, for the three module (formula, structure, classes), sometimes thier top score candidates were not in line with each other. That is, thier top score towards different chemical molecular formulas. To find out the corresponding data in other modules, `create_reference()` should be performed to establish the 'specific_candidate' for subsequent filtering.

```
mcn <- filter_structure(mcn)
mcn <- create_reference(mcn)
mcn <- filter_formula(mcn, by_reference = T)
```

4.3 Filter chemical classes

The PPCP (Posterior Probability of Classification Prediction) data for each 'feature' contains the prediction of thousands of classes for the potential compound (even if the chemical structure was unknown). See http://www.nature.com/articles/s41587-020-0740-8 for details about the prediction. The data contains attributes of:

- `class.name`: name of classes.
- `pp.value`: value of posterior probability. `hierarchy`: hierarchy of classes in the taxonomy. See https://jcheminf.biomedcentral.com/articles/10.1186/s13321-016-0174-y for details about hierarchy and taxonomy of chemical classification.
- …

The method `create_stardust_classes()` use these inner attributes to filter classes candidates for each 'feature'.

Compared to the chemical class filtering within PPCP data by `create_stardust_classes()`, the filtering within 'stardust_classes' data by `cross_filter_stardust()` is fundamentally different.

- For `create_stardust_classes()`, the PPCP data belongs to each 'feature'. When performing the filtering, only simple threshold conditions or absolute conditions are set to filter the chemical classes; there is no crossover between the different attributes and no crossover between the 'features'. Therefore, we consider this as 'inner' filtering.
- For `cross_filter_stardust()`, the data of the chemical classes and their classified 'features', i.e. 'stardust_classes' data, were combined and then grouped upon the chemical classes. After grouping, each chemical class has a certain quantity of "features". When filtering, statistics may be performed on 'features' data within a group; statistics may be performed on these data in conjunction with 'features_annotation' data; and statistics may be performed to compare groups with each other. As its crossover, we consider this as 'cross' filtering.

Use `help(cross_filter_stardust)` to get more details about the algorithm.

```
mcn <- create_stardust_classes(mcn)
mcn <- create_features_annotation(mcn)
mcn <- cross_filter_stardust(mcn,
  max_ratio = 0.05, cutoff = 0.4,
  identical_factor = 0.6
)
```

```
classes <- unique(stardust_classes(mcn)$class.name)
table.filtered.classes <- backtrack_stardust(mcn)
```

Manually filter some repetitive classes or sub-structural classes. By means of Regex matching, we obtained a number of recurring name of chemical classes that would contain manay identical compounds as their sub-structure.

```
classes
```

```
##  [1] "Pyrans"                                    "Pyridines and derivatives"
##  [3] "Pyrroles"                                  "Ketones"
##  [5] "Benzopyrans"                               "Glycerolipids"
##  [7] "Indoles and derivatives"                   "Sugar acids and derivatives"
##  [9] "Glycerophospholipids"                      "Steroids and steroid derivatives"
## [11] "Prenol lipids"                             "Branched fatty acids"
## [13] "Unsaturated fatty acids"                   "Hydroxy fatty acids"
## [15] "Hydroxy acids and derivatives"             "Pyranones and derivatives"
## [17] "Lineolic acids and derivatives"            "Steroidal glycosides"
## [19] "Acyl carnitines"                           "Glycinated bile acids and derivatives"
## [21] "Diacylglycerols"                           "Carboxylic acid salts"
## [23] "Oxosteroids"                               "Hydroxysteroids"
## [25] "Phosphatidylcholines"                      "Lysophosphatidylcholines"
## [27] "Bile acids, alcohols and derivatives"      "Steroid glucuronide conjugates"
## [29] "Diterpenoids"                              "Bilirubins"
## [31] "Tertiary alcohols"                         "Beta hydroxy acids and derivatives"
## [33] "Terpene glycosides"                        "Hydroxy bile acids, alcohols and derivatives"
## [35] "Glycerophosphocholines"                    "Substituted pyrroles"
## [37] "Indoles"                                   "Long-chain fatty acids"
## [39] "Organic cations"                           "Organic salts"
## [41] "Vinylogous acids"
```

```
pattern <- c("stero", "fatty acid", "pyr", "hydroxy", "^orga")
dis <- unlist(lapply(pattern, grep, x = classes, ignore.case = T))
dis <- classes[dis]
dis
```

```
##  [1] "Steroids and steroid derivatives"          "Steroidal glycosides"
##  [3] "Oxosteroids"                               "Hydroxysteroids"
##  [5] "Steroid glucuronide conjugates"            "Branched fatty acids"
##  [7] "Unsaturated fatty acids"                   "Hydroxy fatty acids"
##  [9] "Long-chain fatty acids"                    "Pyrans"
## [11] "Pyridines and derivatives"                 "Pyrroles"
## [13] "Benzopyrans"                               "Pyranones and derivatives"
## [15] "Substituted pyrroles"                      "Hydroxy fatty acids"
## [17] "Hydroxy acids and derivatives"             "Hydroxysteroids"
## [19] "Beta hydroxy acids and derivatives"        "Hydroxy bile acids, alcohols and derivatives"
## [21] "Organic cations"                           "Organic salts"
```

```
dis <- dis[-1]
```

4.4 Create Nebulae

Create Nebula-Index data. This data created based on 'stardust_classes' data.

```
mcn <- backtrack_stardust(mcn, dis, remove = T)
mcn <- create_nebula_index(mcn)
```

Whether it is all filtered by the algorithm provided by MCnebula2's function or custom filtered for some chemical classes, we now have a data called 'nebula_index'. This data records a number of chemical classes and the 'features' attributed to them. The subsequent analysis process or visualization will be based on it. Each chemical class is considered as a 'nebula' and its classified 'features' are the components of these 'nebulae'. In the visualization, these 'nebulae' will be visualized as networks. Formally, we call these 'nebulae' formed on the basis of 'nebula_index' data as Child-Nebulae. In comparison, when we put all the 'features' together to form a large network, then this 'nebula' is called Parent-Nebulae.

```
mcn <- compute_spectral_similarity(mcn)
mcn <- create_parent_nebula(mcn)
mcn <- create_child_nebulae(mcn)
```

4.5 Visualize Nebulae

Create layouts for Parent-Nebula or Child-Nebulae visualizations.

```
mcn <- create_parent_layout(mcn)
mcn <- create_child_layouts(mcn)
mcn <- activate_nebulae(mcn)
```

The available chemical classes for visualization and its sequence in storage.

```
table.nebulae <- visualize(mcn)
```

```
## [INFO] MCnebula2: visualize
```

```
##  Specify item as following to visualize:
```

```
table.nebulae
```

```
## # A tibble: 22 x 3
##      seq hierarchy class.name
##    <int>     <dbl> <chr>
##  1     1         5 Acyl carnitines
##  2     2         4 Bile acids, alcohols and derivatives
##  3     3         4 Bilirubins
##  4     4         5 Carboxylic acid salts
##  5     5         5 Diacylglycerols
##  6     6         4 Diterpenoids
##  7     7         3 Glycerolipids
##  8     8         4 Glycerophosphocholines
##  9     9         3 Glycerophospholipids
## 10    10         5 Glycinated bile acids and derivatives
## # ... with 12 more rows
```

Draw and save as .png or .pdf image files.

```
p <- visualize(mcn, "parent")
ggsave(f4.61 <- paste0(tmp, "/parent_nebula.png"), p)
pdf(f4.62 <- paste0(tmp, "/child_nebula.pdf"), 12, 14)
visualize_all(mcn)
dev.off()
```

5 Nebulae for Downstream analysis

In general, Parent-Nebulae (Fig. 1) is too informative to show, so Child-Nebulae (Fig. 2) was used to dipict the abundant classes of features (metabolites) in a grid panel, intuitively. In a bird's eye view of Child-

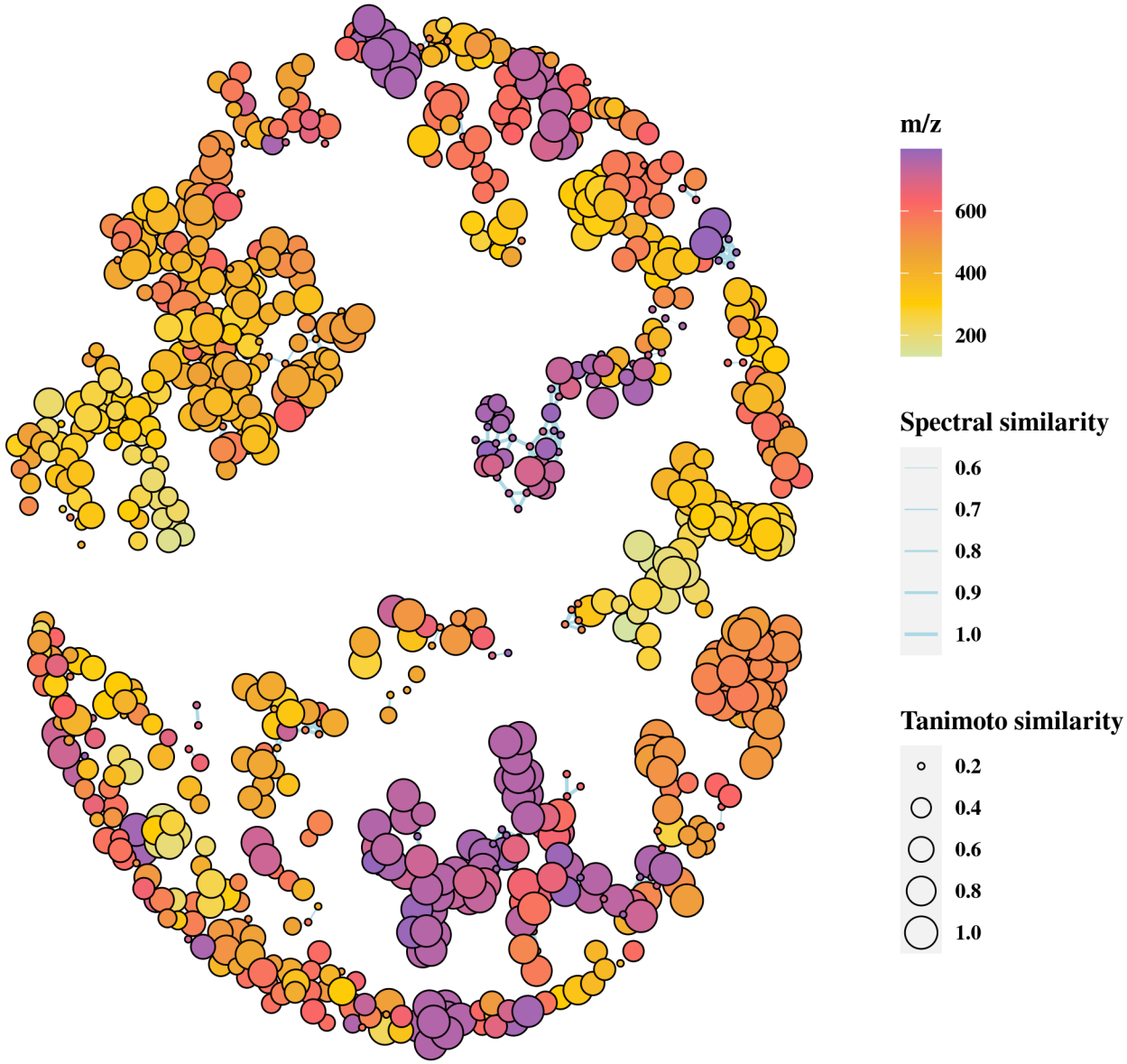

Figure 1: Parent-Nebula

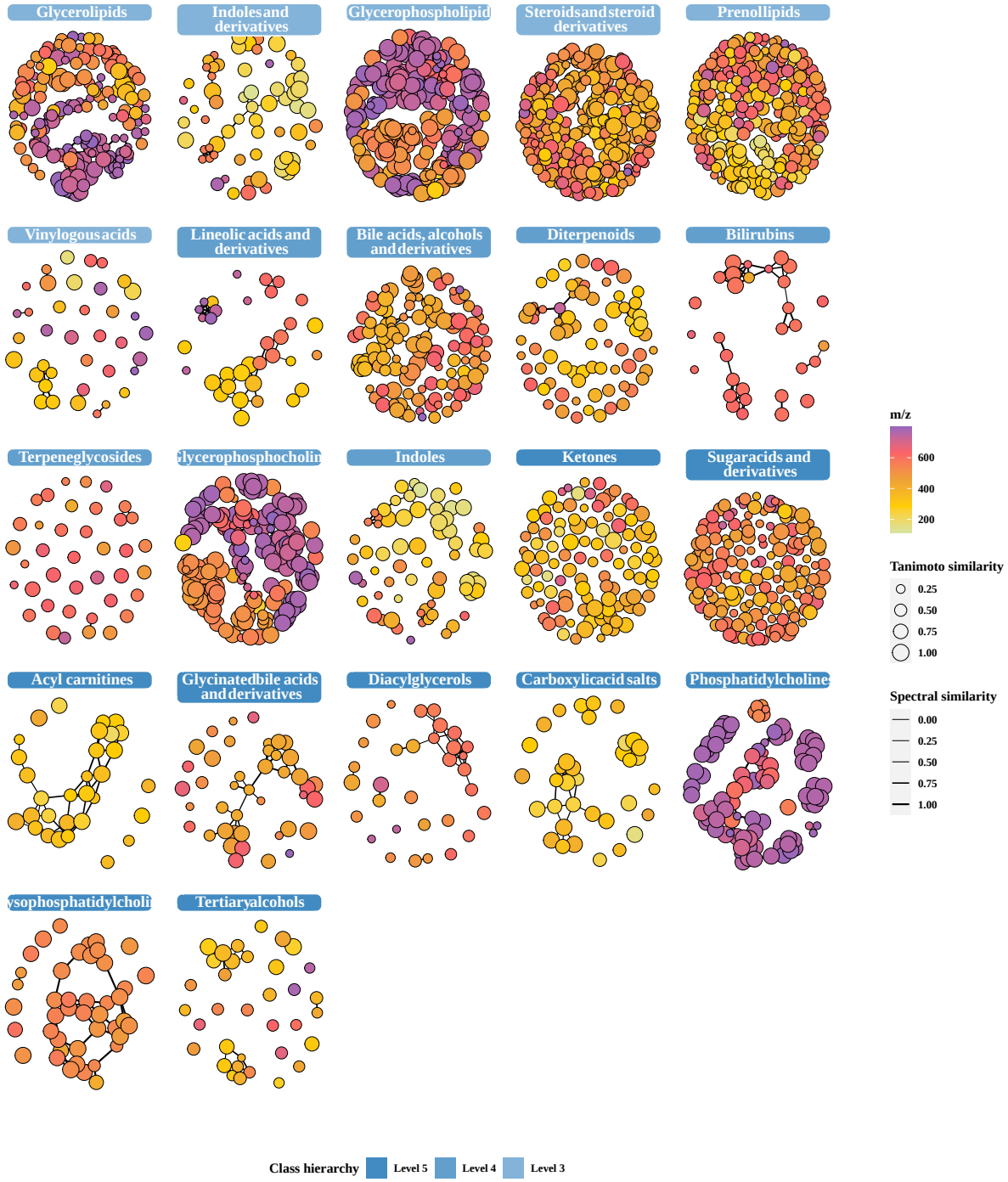

Figure 2: Child-Nebulae

Nebulae, we can obtain many characteristics of features, involving classes distribution, structure identified accuracy, as well as spectral similarity within classes.

5.1 Statistic analysis

Next we perform a statistical analysis with quantification data of the features. Note that the SIRIUS project does not contain quantification data of features, so our object `mcn` naturally does not contain that either. We need to get it from elsewhere.

```
utils::untar(paste0(exfiles, "/serum.tar.gz"), exdir = tmp)
origin <- data.table::fread(paste0(tmp, "/serum_origin_mmc3.tsv"),
  skip = 1
)
origin <- tibble::as_tibble(origin)
```

Its original data can obtained from: https://www.cell.com/cms/10.1016/j.cell.2020.07.040/attachment/f13178d1-d1ee-4179-9d33-227a02e604f1/mmc3.xlsx. Now, let's check the columns in the table.

```
origin
```

```
## # A tibble: 5,280 x 225
##    Unique_ID `m/z`    RT Subnetwork Molecular_Informati~ Spectral_Librar~ Superclass Class Subclass
##        <int> <dbl> <dbl>      <int> <chr>                <chr>            <chr>      <chr> <chr>
##  1       349  540.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
##  2       228  548.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
##  3      1963 1092.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
##  4       971  547.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
##  5        13  546.  6.57         45 <NA>                 <NA>             <NA>       <NA>  <NA>
##  6      4146  541.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
##  7       854  289.  4.12         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
##  8      4046 1036.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
##  9      1374  788.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
## 10      2304  789.  6.57         -1 <NA>                 <NA>             <NA>       <NA>  <NA>
## # ... with 5,270 more rows, and 216 more variables: Direct_Parent <chr>, Infection_pvalue <dbl>,
## #   Infection_FC <dbl>, Infection_BH_CV <dbl>, Infection_BH_Sig <chr>, Mortality_pvalue <chr>,
## #   Mortality_FC <chr>, Mortality_BH_CV <dbl>, Mortality_BH_Sig <chr>, EFS_Rank <int>,
## #   MW_Rank <int>, Avg_Rank <dbl>, ANOVA_pvalue <dbl>, `Cluster_#` <int>, ANOVA_CV <dbl>,
## #   ANOVA_BH_Sig <chr>, NN1 <dbl>, NN10 <dbl>, NN11 <dbl>, NN12 <dbl>, NN13 <dbl>, NN14 <dbl>,
## #   NN15 <dbl>, NN2 <dbl>, NN3 <dbl>, NN4 <dbl>, NN5 <dbl>, NN6 <dbl>, NN7 <dbl>, NN8 <dbl>,
## #   NN9 <dbl>, HN1 <dbl>, HN10 <dbl>, HN2 <dbl>, HN3 <dbl>, HN4 <dbl>, HN5 <dbl>, HN6 <dbl>, ...
```

Remove the rest of the columns and keep only the columns for ID, m/z, retention time, and quantification.

```
keep <- grep("^[A-Z]{2}[0-9]{1,3}$", colnames(origin))
quant <- dplyr::select(origin, oid = 1, mz = 2, rt = 3, dplyr::all_of(keep))
```

The IDs in the data `quant` are different from the IDs in the object `mcn`, so we need to align them first, according to mz and rt (they originate from the same batch of samples). In the following, we have allowed the two sets of data to be merged as accurately as possible in the form of evaluation of score:

- Score = (1 - rt.difference / rt.tolerance) * rt.weight + (1 - mz.defference / mz.tolerance) * mz.weight

```
meta_col <- dplyr::select(
  features_annotation(mcn), .features_id,
  mz, rt.secound
)
meta_col$rt.min <- meta_col$rt.secound / 60
merged <- align_merge(meta_col, quant, ".features_id",
```

```
  rt.main = "rt.min",
  rt.sub = "rt"
)
merged <- dplyr::select(
  merged, -mz.main, -mz.sub, -rt.min, -rt,
  -rt.secound
)
```

Due to the differences in feature detection algorithms, some of the features inevitably do not get matched.

```
merged
```

```
## # A tibble: 3,680 x 202
##    .features_id   oid         NN1       NN10      NN11     NN12     NN13     NN14     NN15      NN2
##    <chr>        <int>       <dbl>      <dbl>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1 1             2435  0.0000298   0.0000176   7.3 e-6  8.34e-6  4.62e-6 NA       NA        4.99e-6
##  2 100           1614  0.000101    0.0000681   6.34e-5  3.94e-5  9.5 e-5  9.34e-5  6.68e-5  1.06e-4
##  3 1000           372  0.000623    0.000787    3.50e-4  1.85e-4  5.79e-4  4.27e-4  7.19e-4  6.70e-4
##  4 1001          1001  0.00110     0.000138    1.66e-4  1.57e-4  2.27e-4  1.87e-4  2.40e-4  6.67e-4
##  5 1002          2364 NA          NA          NA       NA       NA       NA       NA       NA
##  6 1003           452 NA          NA          NA       NA       NA        8.49e-5 NA       NA
##  7 1004          2962  0.0000214  NA          NA       NA       NA       NA       NA        1.5 e-5
##  8 1005          1313  0.00000545  0.0000462   3.43e-5  1.46e-5  3.36e-5  1.36e-4  2.65e-5  6.82e-5
##  9 1006           883  0.000119    0.0000681   1.89e-4  5.92e-5  5.06e-5  8.37e-5  3.21e-5  1.47e-4
## 10 1007          2889  0.0000567   0.000283    2.35e-4  1.91e-4  4.64e-4  2.35e-4  3.43e-4  7.47e-5
## # ... with 3,670 more rows, and 192 more variables: NN3 <dbl>, NN4 <dbl>, NN5 <dbl>, NN6 <dbl>,
## #   NN7 <dbl>, NN8 <dbl>, NN9 <dbl>, HN1 <dbl>, HN10 <dbl>, HN2 <dbl>, HN3 <dbl>, HN4 <dbl>,
## #   HN5 <dbl>, HN6 <dbl>, HN7 <dbl>, HN8 <dbl>, HN9 <dbl>, HS1 <dbl>, HS10 <dbl>, HS11 <dbl>,
## #   HS12 <dbl>, HS13 <dbl>, HS14 <dbl>, HS15 <dbl>, HS16 <dbl>, HS17 <dbl>, HS18 <dbl>,
## #   HS19 <dbl>, HS2 <dbl>, HS20 <dbl>, HS21 <dbl>, HS22 <dbl>, HS23 <dbl>, HS24 <dbl>, HS25 <dbl>,
## #   HS26 <dbl>, HS27 <dbl>, HS28 <dbl>, HS29 <dbl>, HS3 <dbl>, HS30 <dbl>, HS31 <dbl>, HS32 <dbl>,
## #   HS33 <dbl>, HS34 <dbl>, HS35 <dbl>, HS36 <dbl>, HS37 <dbl>, HS38 <dbl>, HS39 <dbl>, ...
```

Create the metadata table and store it in the `mcn` object along with the quantification data.

```
gp <- c(NN = "^NN", HN = "^HN", HS = "^HS", HM = "^HM")
metadata <- MCnebula2:::group_strings(colnames(merged), gp, "sample")
metadata$group_name <- vapply(metadata$group, switch,
  FUN.VALUE = character(1),
  NN = "non-hospital & non-infected", HN = "hospital & non-infected",
  HS = "hospital & survival", HM = "hospital & mortality"
)
metadata$supergroup <- vapply(metadata$group, switch,
  FUN.VALUE = character(1),
  NN = "control groups", HN = "control groups", HS = "infection groups",
  HM = "infection groups"
)
features_quantification(mcn) <- dplyr::select(merged, -oid)
sample_metadata(mcn) <- metadata
```

Variance analysis was used as a way to detect whether there were differences between the experimental and control groups and whether the differences were significant. Linear models are an effective tool for variance analysis, and it permit very general analyses. The 'limma' package[6] integrates a number of functions for creating linear models and regression analysis. The statistical analysis provided in MCnebula2 is mainly built around the functions in the 'limma' package.

In the following we use the `binary_comparison` function for variance analysis. Note that the quantification data in `origin` has been normalized. To accommodate the downstream analysis of gene expression that the `limma` package was originally used for, we should log2-transform and centralize this data.

```
mcn <- binary_comparison(mcn, (HS + HM) - (NN + HN), HM - HS,
  fun_norm = function(x) scale(log2(x), scale = F)
)
top.list <- top_table(statistic_set(mcn))
```

To verify the validity of the above variance analysis, the data columns were merged to obtain the IDs from the original analysis.

```
top.list <- lapply(top.list, merge, y = dplyr::select(
  merged,
  .features_id, oid
), by = ".features_id", all.x = T, sort = F)
top.list <- lapply(top.list, tibble::as_tibble)
```

Verify with the `EFS_Rank` and `MW_Rank` column in the `origin` data. (The original authors used the two methods to rank the features.)

```
origin_top50 <- dplyr::filter(origin, EFS_Rank <= 50 | MW_Rank <=
  50)
inter. <- lapply(top.list, function(df) {
  match <- head(df, n = 50)$oid %in% origin_top50$Unique_ID
  oid <- head(df, n = 50)$oid[match]
  list(table.match = table(match), oid = oid)
})
lapply(inter., function(x) x$table.match)
```

```
## $`(HS + HM) - (NN + HN)`
## match
## FALSE  TRUE
##    48     2
##
## $`HM - HS`
## match
## FALSE  TRUE
##    13    37
```

Let's see which compounds were identified that intersected our ranking and the original ranking of features.

```
inter.compound <- dplyr::filter(origin, Unique_ID %in% inter.[[2]]$oid)
table(inter.compound$Spectral_Library_Match, useNA = "if")
```

```
##
## Decanoyl-L-carnitine          L-THYROXINE                <NA>
##                    1                    1                  27
```

Interestingly, these two compounds were critical compounds in the original study.

5.2 Set tracer in Child-Nebulae

Tracking top features obtained by Variance analysis in Nebulae provides insight not only into the chemical classes of these top features, but also into other features (may be analogous metabolites). Other features are not among the top ranked features, but they may contain key features that were missed due to algorithmic specificity. By tracking top features, it is possible to revisit all features at the overall data level.

```
n <- 50
tops <- unique(unlist(lapply(top.list, function(df) df$.features_id[1:n])))
palette_set(melody(mcn)) <- colorRampPalette(palette_set(mcn))(length(tops))
mcn2 <- set_tracer(mcn, tops)
mcn2 <- create_child_nebulae(mcn2)
mcn2 <- create_child_layouts(mcn2)
mcn2 <- activate_nebulae(mcn2)
mcn2 <- set_nodes_color(mcn2, use_tracer = T)
```

Draw and save the image.

```
pdf(f6.2 <- paste0(tmp, "/tracer_child_nebula.pdf"), 12, 14)
visualize_all(mcn2)
dev.off()
```

A part of the top features are marked with colored nodes in Child-Nebulae (Fig. 3) . These features are at least identified with chemical molecular formula. Those that are not identified, or the Nebula-Index data do not contain the chemical class to which these features belong, are absent from the Figure.

5.3 Quantification in Child-Nebulae

Show Fold Change (HM versus HS) in Child-Nebulae.

```
palette_gradient(melody(mcn2)) <- c("blue", "grey90", "red")
mcn2 <- set_nodes_color(mcn2, "logFC", top.list[[2]])
pdf(f7.1 <- paste0(tmp, "/logFC_child_nebula.pdf"), 12, 14)
visualize_all(mcn2, fun_modify = modify_stat_child)
dev.off()
```

Each Child-Nebula separately shows the overall content variation of the chemical class to which it belongs (Fig. 4) .

5.4 Annotate Nebulae

Now, the available Nebulae contains:

```
table.nebulae2 <- visualize(mcn2)
```

```
## [INFO] MCnebula2: visualize
```

```
##  Specify item as following to visualize:
```

```
table.nebulae2
```

```
## # A tibble: 16 x 3
##      seq hierarchy class.name
##    <int>     <dbl> <chr>
##  1     1         5 Acyl carnitines
##  2     2         4 Bile acids, alcohols and derivatives
##  3     3         5 Carboxylic acid salts
##  4     4         5 Diacylglycerols
##  5     5         3 Glycerolipids
##  6     6         4 Glycerophosphocholines
##  7     7         3 Glycerophospholipids
##  8     8         5 Ketones
##  9     9         4 Lineolic acids and derivatives
## 10    10         5 Lysophosphatidylcholines
## 11    11         3 Prenol lipids
```

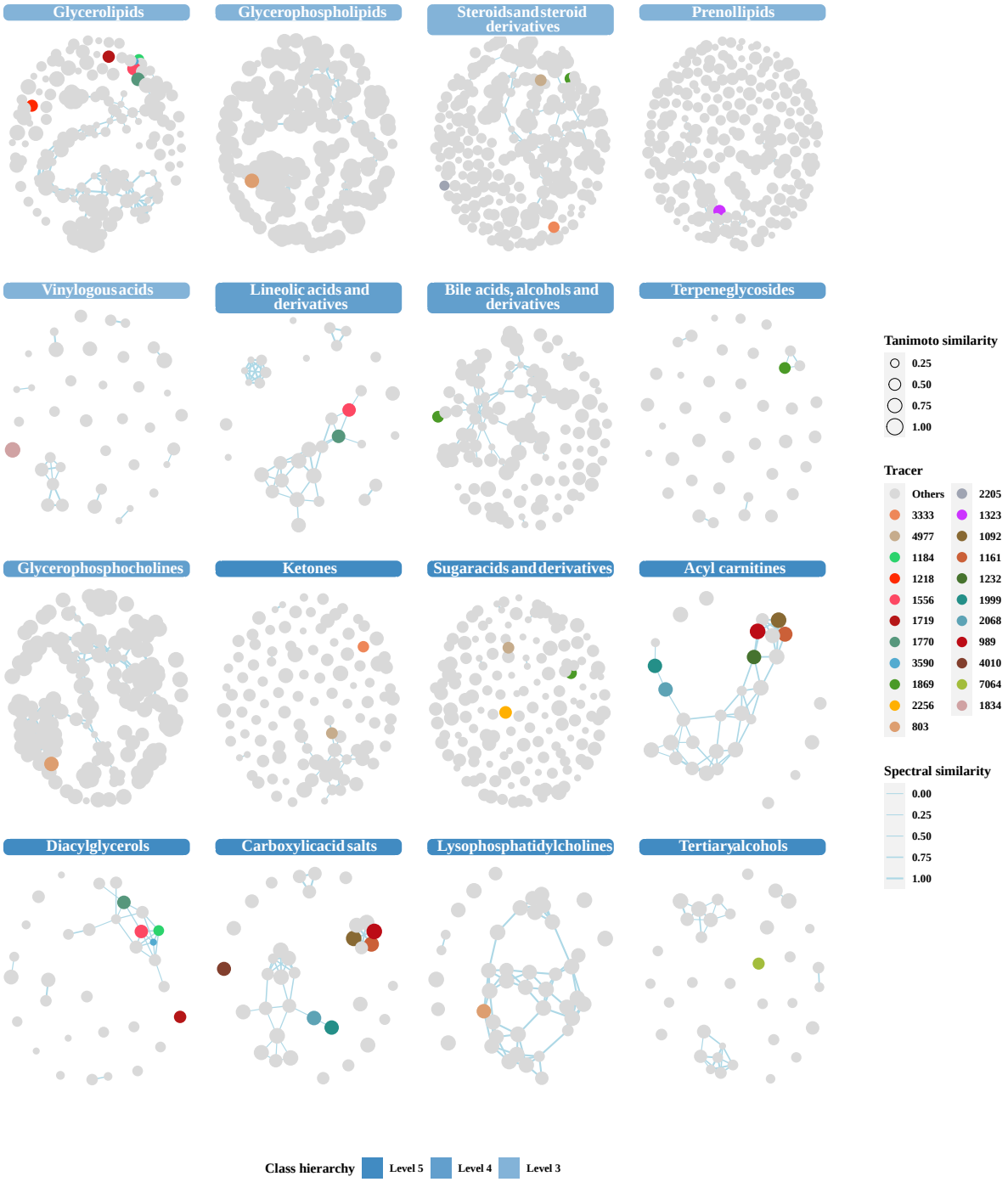

Figure 3: Tracing top features in Child-Nebulae

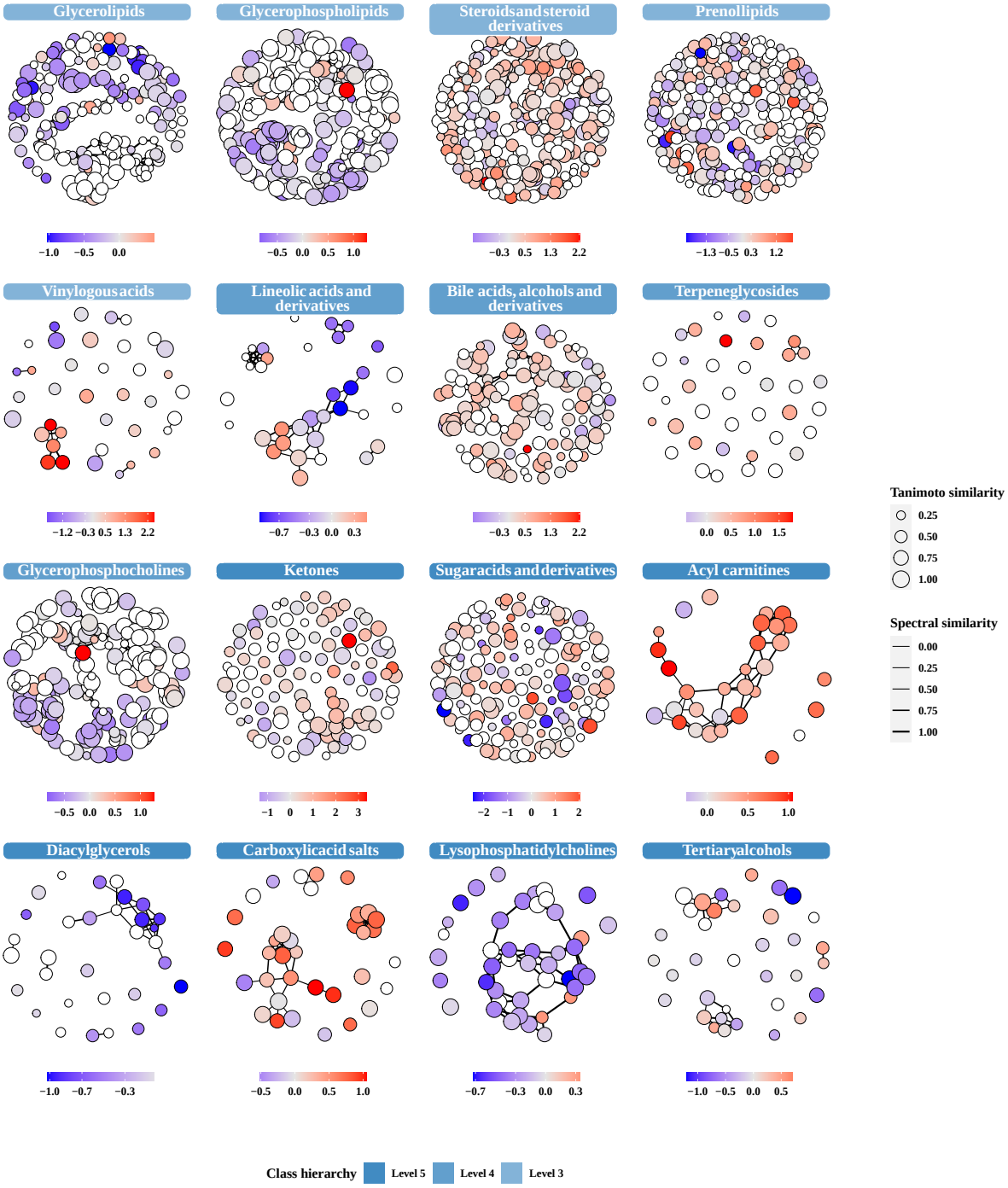

Figure 4: Show log2(FC) in Child-Nebulae

```
## 12    12         3 Steroids and steroid derivatives
## 13    13         5 Sugar acids and derivatives
## 14    14         4 Terpene glycosides
## 15    15         5 Tertiary alcohols
## 16    16         3 Vinylogous acids
```

Next, let us focus on Acyl carnitines, a class that was highlighted in the original research and also appears in Child-Nebulae, marked by plural top features (Likewise, we annotated two other chemical classes of Nebulae).

```
mcn2 <- set_nodes_color(mcn2, use_tracer = T)
palette_stat(melody(mcn2)) <- c(
  NN = "#B6DFB6", HN = "#ACDFEE",
  HS = "#EBA9A7", HM = "grey70"
)
ac <- "Acyl carnitines"
lpc <- "Lysophosphatidylcholines"
ba <- "Bile acids, alcohols and derivatives"
mcn2 <- annotate_nebula(mcn2, ac)
mcn2 <- annotate_nebula(mcn2, lpc)
mcn2 <- annotate_nebula(mcn2, ba)
```

Draw and save the image.

```
p <- visualize(mcn2, ac, annotate = T)
ggsave(f8.2 <- paste0(tmp, "/ac_child.pdf"), p, height = 5)
p <- visualize(mcn2, lpc, annotate = T)
ggsave(f8.2.2 <- paste0(tmp, "/lpc_child.pdf"), p, height = 5)
p <- visualize(mcn2, ba, annotate = T)
ggsave(f8.2.3 <- paste0(tmp, "/ba_child.pdf"), p, height = 5)
```

See results (Fig. 5) . The annotated Nebula presents a multi-dimensional annotation for each FEATURES, involving chemical classes, chemical structures, quantification etc.

Use the `show_node` function to get the annotation details for a feature. For example:

```
ef <- "2068"
pdf(f8.4 <- paste0(tmp, "/features_", ef, ".pdf"), 10, 4)
show_node(mcn2, ef)
dev.off()
```

See results (Fig. 6) .

5.5 Query compounds

The `features_annotation(mcn)` contains the main annotation information of all the features, i.e., the identity of the compound. Next, we would query the identified compounds based on the 'inchikey2d' column therein. Note that the stereoisomerism of the compounds is difficult to be determined due to the limitations of MS/MS spectroscopy. Therefore, we used InChIKey 2D (representing the molecular backbone of the compound) to query the compound instead of InChI.

First we need to format and organize the annotated data of features to get the non-duplicated 'inchikey2d'. We provide a function with a pre-defined filtering algorithm to quickly organize the table. By default, this function filters the data based on 'tani.score' (Tanimoto similarity), and then sorts and de-duplicates it.

```
feas <- features_annotation(mcn2)
feas <- format_table(feas, export_name = NULL)
key2d <- feas$inchikey2d
```

Create a folder to store the acquired data.

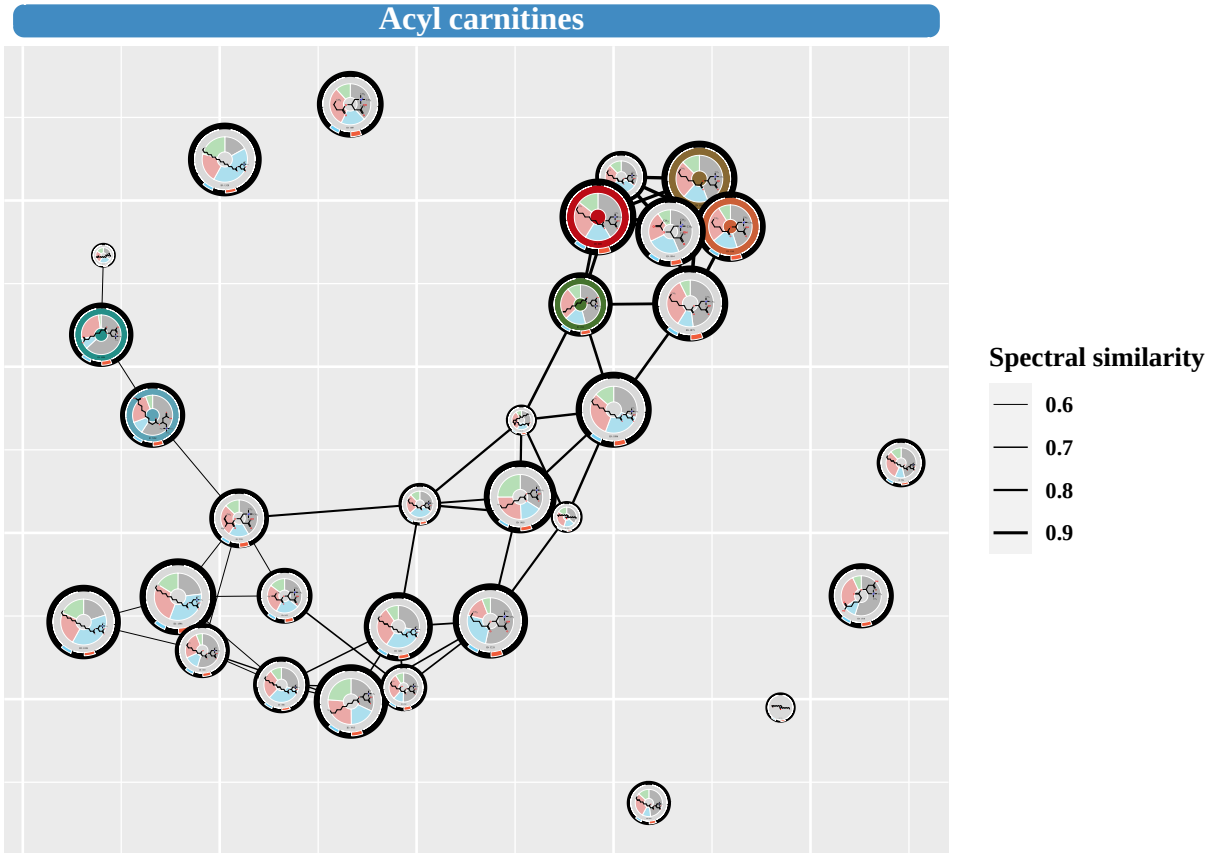

Figure 5: Annotated Nebulae: Acyl carnitines

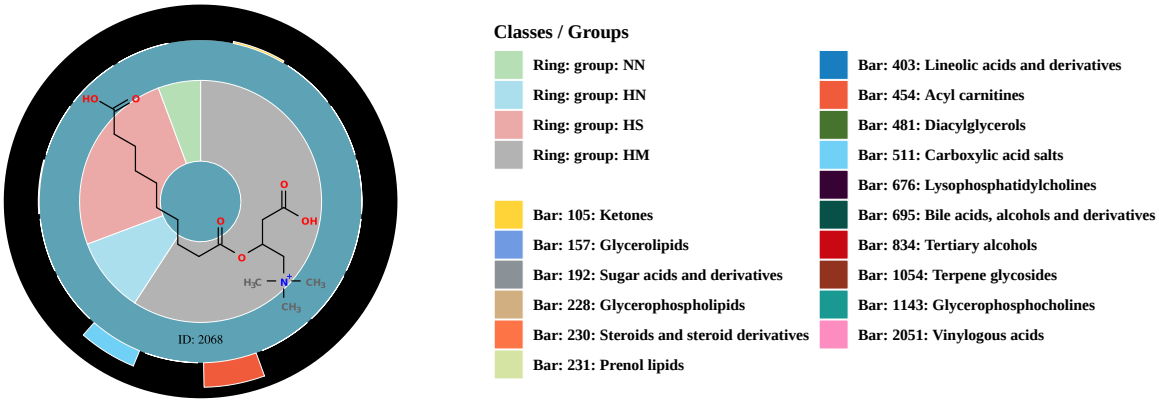

Figure 6: The annotated feature of ID: 2068

```
tmp2 <- paste0(tmp, "/query")
dir.create(tmp2, F)
```

Query the compound's InChIKey, chemical class, IUPUA name. If your system is not Linux, the multi-threading below may pose some problems, please remove the parameters `curl_cl = 4` and `classyfire_cl = 4`.

```
key.rdata <- query_inchikey(key2d, tmp2, curl_cl = 4)
class.rdata <- query_classification(key2d, tmp2, classyfire_cl = 4)
iupac.rdata <- query_iupac(key2d, tmp2, curl_cl = 4)
```

We will also query for synonyms of compounds, but this is done in 'CID' (PubChem's ID), so some transformation is required.

```
key.set <- extract_rdata_list(key.rdata)
cid <- lapply(key.set, function(data) data$CID)
cid <- unlist(cid, use.names = F)
syno.rdata <- query_synonyms(cid, tmp2, curl_cl = 4)
```

Screen for unique synonyms and chemical classes for all compounds.

```
syno <- pick_synonym(key2d, key.rdata, syno.rdata, iupac.rdata)
feas$synonym <- syno
class <- pick_class(key2d, class.rdata)
feas$class <- class
feas.table <- rename_table(feas)
write_tsv(feas.table, paste0(tmp, "/compounds_format.tsv"))
```

The formatted table as following:

```
feas.table
```

```
## # A tibble: 1,086 x 13
##    Synonym        ID    `Precursor m/z` `Mass error (p~` `RT (min)` Formula Adduct `Tanimoto simi~`
##    <chr>          <chr>           <dbl>            <dbl>      <dbl> <chr>   <chr>              <dbl>
##  1 Stearoyllysop~ 803              546.               8          6.6 C26H54~ [M + ~              0.84
##  2 3-beta,5-beta~ 4977             509.              -2.3        4.2 C27H40~ [M + ~              0.54
##  3 Etiocholanedi~ 3333             289.               8.5        4.1 C19H28~ [M + ~              0.51
##  4 Isoleucylprol~ 885              229.              -6.1        0.4 C11H20~ [M + ~              0.54
##  5 sebacoylcarni~ 2068             346.               5.7        3.4 C17H31~ [M + ~              0.84
##  6 Dimethylguano~ 2231             312.              -0.8        0.7 C12H17~ [M + ~              0.99
##  7 N6-Threonylca~ 2199             413.              -1.4        2.1 C15H20~ [M + ~              0.7
##  8 Dextrothyroxi~ 1960             778.               1          4.7 C15H11~ [M + ~              0.99
##  9 Hexanoylcarni~ 1161             260.              -6.6        3   C13H25~ [M + ~              0.91
## 10 octanoylcarni~ 1092             288.              -5.2        4.1 C15H29~ [M + ~              0.98
## # ... with 1,076 more rows, and 5 more variables: `InChIKey planar` <chr>, `log2(FC)` <dbl>,
## #   `P-value` <dbl>, `Q-value` <dbl>, Class <chr>
```

Filtering our results based on the ranking of 'features' (top 25 of EFS and MWU) and identification by Wozniak et al.

```
origin_top50 <- dplyr::select(origin_top50,
  oid = 1, EFS_Rank,
  MW_Rank, Spectral_Library_Match
)
feas.otop <- dplyr::select(merged, .features_id, oid)
feas.otop <- merge(origin_top50, feas.otop, all.x = T, by = "oid")
feas.otop <- merge(feas.otop, features_annotation(mcn2),
```

```
  all.x = T,
  by = ".features_id"
)
feas.otop.format <- format_table(feas.otop,
  filter = NULL, select = c(
    "oid",
    "EFS_Rank", "MW_Rank", "Spectral_Library_Match", .select_format
  ),
  export_name = c(
    oid = "# Original ID", EFS_Rank = "# EFS_Rank",
    MW_Rank = "# MW_Rank", Spectral_Library_Match = "# Spectral_Library_Match",
    .export_name
  )
)
write_tsv(feas.otop.format, paste0(tmp, "/oTop50_compounds_format.tsv"))
```

5.6 Pathway enrichment

A plural number of chemical classes of interest were selected and the IDs of their features were obtained. These features were filtered with the statistic analysis data (Q value < 0.05).

```
focus <- c("Acyl carnitines", "Lysophosphatidylcholines", "Bile acids, alcohols and derivatives")
focus <- select_features(mcn, focus,
  q.value = 0.05, logfc = 0.3,
  coef = 1:2
)
```

```
focus.key2d <- maps(feas, focus, ".features_id", "inchikey2d")
focus.cid <- lapply(focus.key2d, function(key2d) {
  set <- key.set[names(key.set) %in% key2d]
  unlist(lapply(set, function(df) df$CID), use.names = F)
})
keggids <- cid.to.kegg(unlist(focus.cid, use.names = F))
focus.kegg <- maps(
  keggids, lapply(focus.cid, as.character),
  "Query", "KEGG"
)
```

Let's see what we get.

```
focus.kegg
```

```
## $`Acyl carnitines`
## 11953814 11953821   439829
## "C02838" "C03299" "C02862"
##
## $`Bile acids, alcohols and derivatives`
## 53477753 42622727 53477907 22833540 23617285     6675
## "C03033" "C03033" "C05462" "C05466" "C01921" "C05122"
##
## $Lysophosphatidylcholines
##   460602    86555   497299 11757087 24779458 11005824 24779463 24779473   460604 52924053 52924051
## "C04230" "C04102" "C04230" "C04230" "C04230" "C04230" "C04230" "C04230" "C04230" "C04230" "C04230"
```

Using package 'FELLA' for pathway enrichment analysis. The following step create a 'database' for enrichment (It is quite time consuming and can take up to 30 minutes). Subsequently, load the data.

```
db.dir <- init_fella(tmp, "hsa")
db.data <- load_fella(db.dir)
```

Perform enrichment.

```
focus.enrich <- enrich_fella(focus.kegg, db.data)
focus.graph <- graph_fella(focus.enrich, db.data, "pagerank")
names(focus.graph) <- names(focus.kegg)
```

Some compounds are not present in the KEGG graph and are only background compounds. Let's check the enrichment results.

```
!vapply(focus.graph, is.null, logical(1))
```

```
##                      Acyl carnitines Bile acids, alcohols and derivatives
##                                FALSE                                 TRUE
##            Lysophosphatidylcholines
##                                TRUE
```

5.7 Heatmap analysis

Since the quantitative data obtained by merging contain many missing values, they need to be processed before plotting the heat map: For each subset of data, the missing values will be filled with the average value; if the set is all missing values, they will be filled with zero.

```
hp.data <- handling_na(features_quantification(mcn2), metadata = sample_metadata(mcn2))
```

Convert wide data to long data; log transform the values; if there is a value 0, replace it with 1/10 of the minimum value of the value column.

```
hp.data <- log_trans(hp.data)
```

Draw heat maps for three chemical classes.

```
hp.lst <- plot_heatmap(focus, hp.data, metadata, pal_group = palette_stat(mcn2))
ggsave(f12.31 <- paste0(tmp, "/ac_heatmap.pdf"), hp.lst[[1]],
  width = 13, height = 4
)
ggsave(f12.32 <- paste0(tmp, "/ba_heatmap.pdf"), hp.lst[[2]],
  width = 13, height = 7
)
ggsave(f12.33 <- paste0(tmp, "/lpc_heatmap.pdf"), hp.lst[[3]],
  width = 13, height = 4
)
```

As shown in figure, Fig. 9, 10, and 11 ACs and BAs implied a high correlation with disease, while LPCs showed a relatively weak correlation.

6 Verify Identification

In research of Wozniak et al, a subset of ACs compounds were identified. In addition, four top rank metabolites were identified. However, Most of the top 25 metabolites (EFS rank) were not identified.

```
ac_names <- c(
  "Palmitoyl-carnitine", "Octanoyl-carnitine", "Acetyl-carnitine",
  "Hexanoyl-carnitine", "Decanoyl-carnitine"
)
```

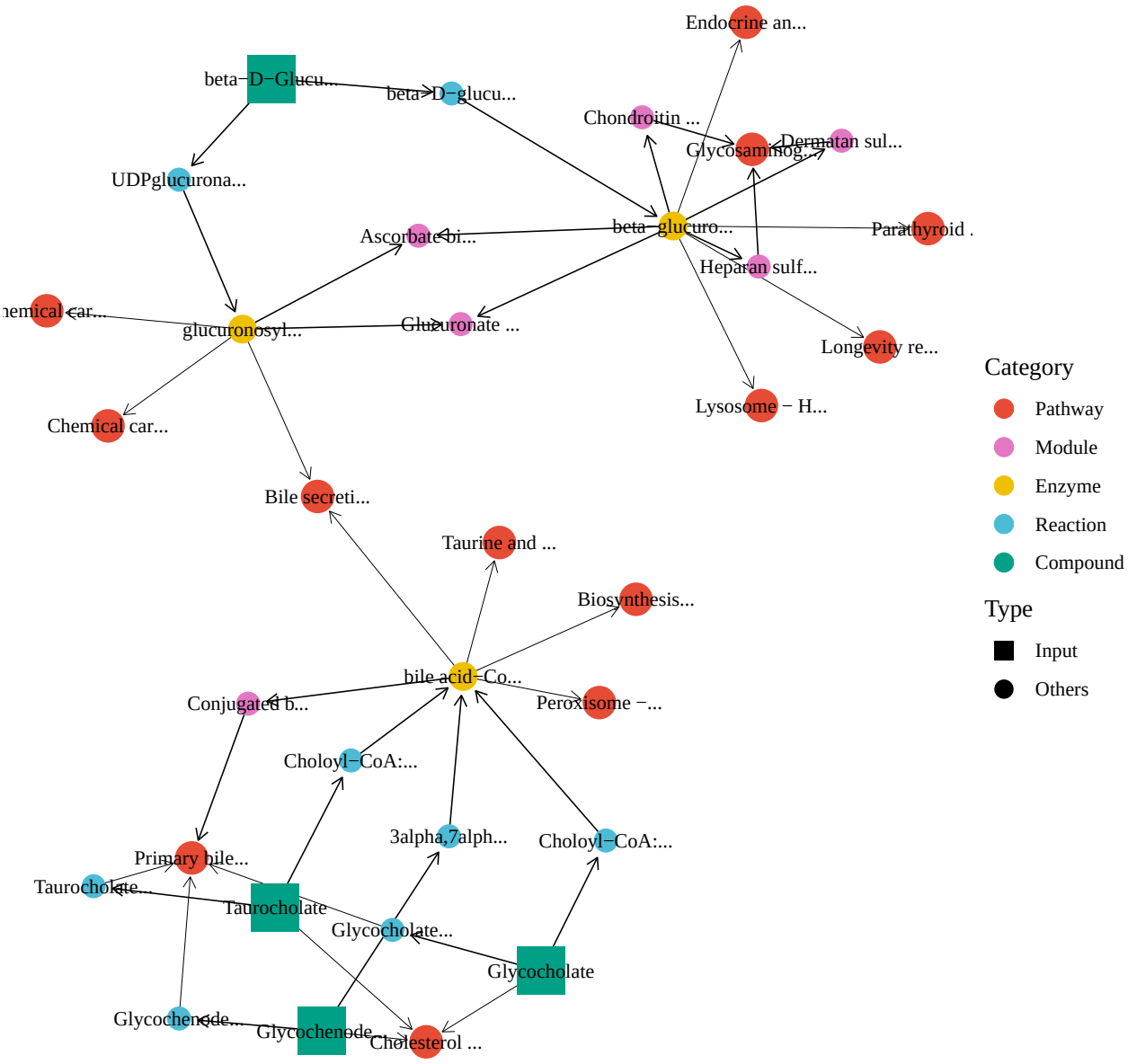

Figure 7: Enrichment of Pagerank of BA compounds

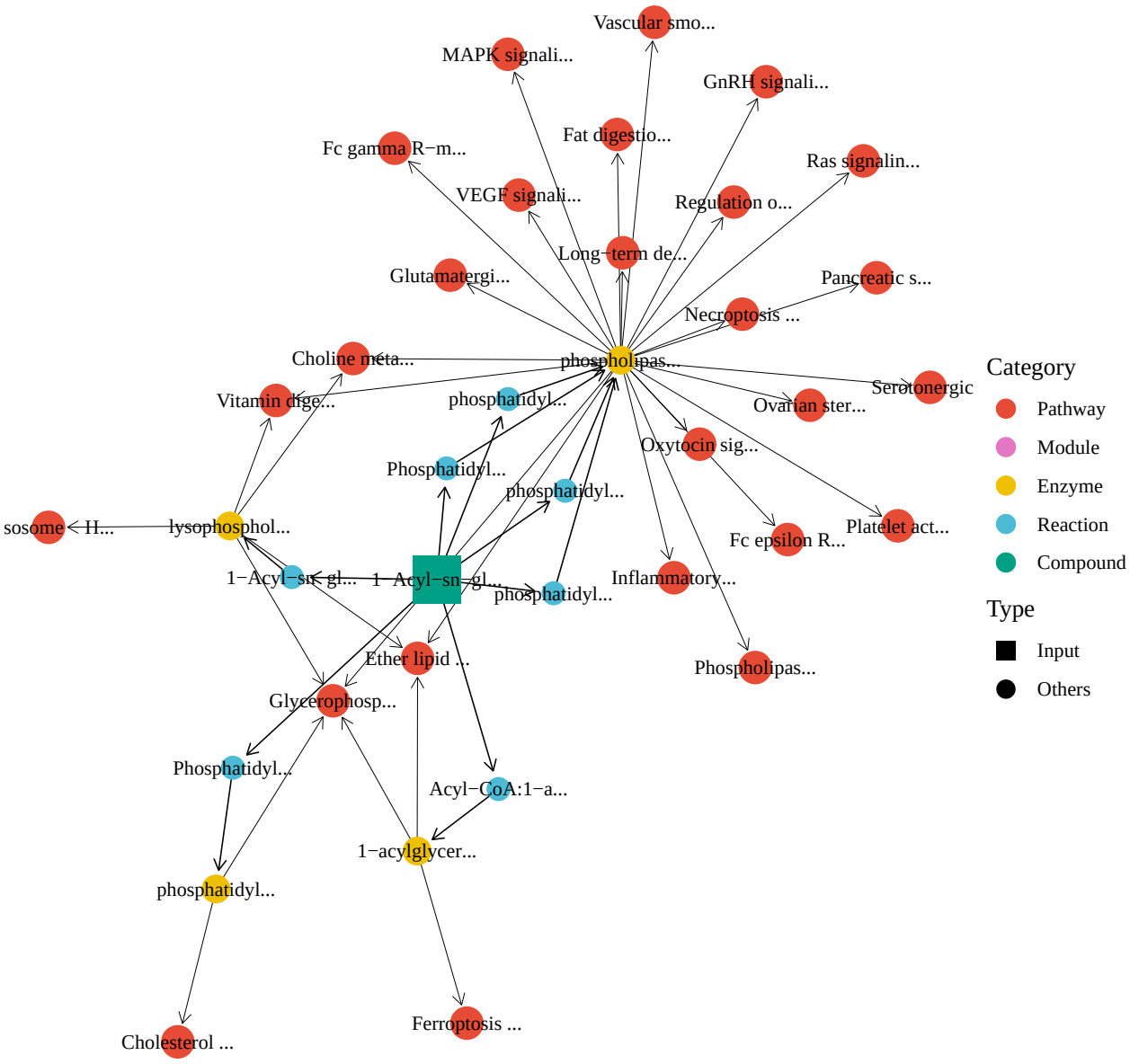

Figure 8: Enrichment of Pagerank of LPC compounds

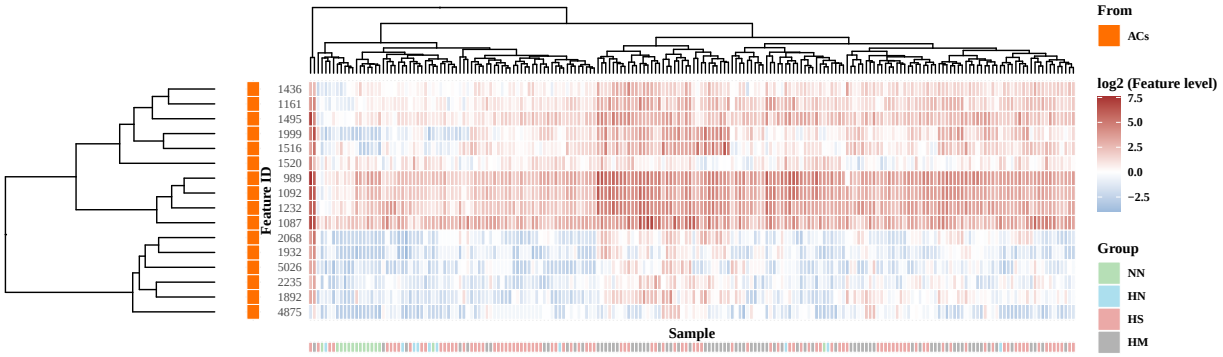

Figure 9: Heatmap of ACs

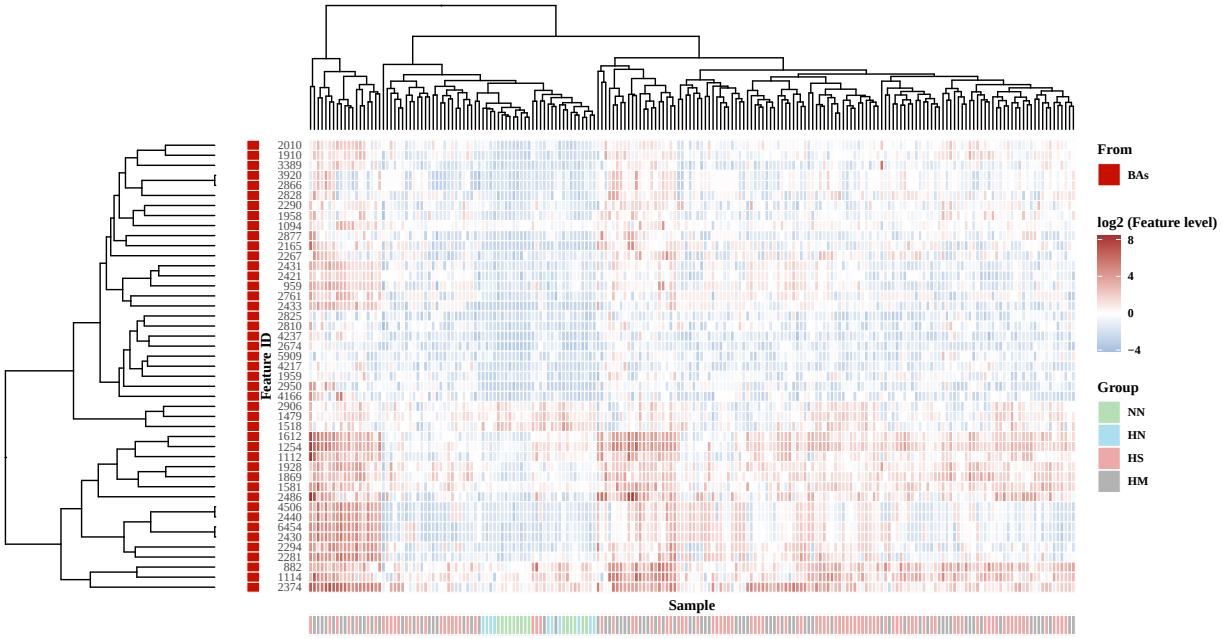

Figure 10: Heatmap of BAs

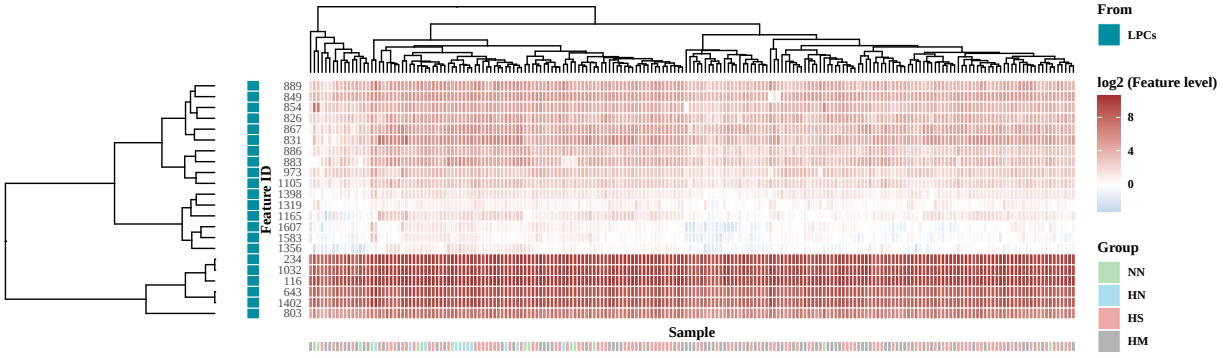

Figure 11: Heatmap of LPCs

```
ac_inchikey2d <- c(
  "XOMRRQXKHMYMOC", "CXTATJFJDMJMIY", "RDHQFKQIGNGIED",
  "VVPRQWTYSNDTEA", "LZOSYCMHQXPBFU"
)
names(ac_inchikey2d) <- ac_names
other_4m <- c(
  "Hepc", "D-erythro-Sphingosine-1-phosphate", "L-THYROXINE",
  "Decanoyl-L-carnitine"
)
other_4m_inchikey2d <- c(
  "BDPQVGIMLZYZQA", "DUYSYHSSBDVJSM",
  "XUIIKFGFIJCVMT", "LZOSYCMHQXPBFU"
)
other_4m.seq <- unlist(lapply(other_4m, grep,
  x = origin$Spectral_Library_Match,
  ignore.case = T
))
other_4m <- origin[other_4m.seq, ]
origin.top25 <- c(
  349, 746, 854, 228, 320, 971, 2532, 670, 92,
  1363, 13, 798, 1379, 1947, 4146, 736, 1656, 464, 731, 289,
  4431, 3865, 476, other_4m$Unique_ID
)
origin.top25.featureID <- dplyr::filter(merged, oid %in% origin.top25)$.features_id
```

Confirm which compounds were identified:

```
ac_inchikey2d %in% features_annotation(mcn2)$inchikey2d
```

```
## [1] TRUE TRUE TRUE TRUE TRUE
```

```
other_4m_inchikey2d %in% features_annotation(mcn2)$inchikey2d
```

```
## [1] FALSE  TRUE  TRUE  TRUE
```

```
reIdentify.origin.top25 <- dplyr::filter(
  features_annotation(mcn2),
  .features_id %in% origin.top25.featureID
)
reIdentify.origin.top25 <- dplyr::select(
  reIdentify.origin.top25,
  .features_id, tani.score, inchikey2d
)
print(reIdentify.origin.top25, n = Inf)
```

```
## # A tibble: 21 x 3
##    .features_id tani.score inchikey2d
##    <chr>             <dbl> <chr>
##  1 1011               0.93 AEHPOYAOLCAMIU
##  2 1071               0.44 PRHHYVQTPBEDFE
##  3 1104               0.85 SATGKQGFUDXGAX
##  4 1184               0.45 RWXWCAZLEFJOFU
##  5 1446              NA    <NA>
##  6 1671               0.4  XRDKWFXOXXUQJS
##  7 1869               0.54 ONLXJASEXIXGRM
##  8 1960               0.99 XUIIKFGFIJCVMT
```

```
##  9 2045              0.41 OVEVHVURWWTPFC
## 10 3333              0.51 RAJWOBJTTGJROA
## 11 3590             NA    <NA>
## 12 4353             NA    <NA>
## 13 5344             NA    <NA>
## 14 6237             NA    <NA>
## 15 630               0.51 KPGYIOMDVKQYDK
## 16 6646              0.19 RCDRLZYAKDYXMM
## 17 803               0.84 IHNKQIMGVNPMTC
## 18 886               1    RJZVWDTYEWCUAR
## 19 952               0.99 DUYSYHSSBDVJSM
## 20 981               0.39 OAUYENAPBFTAQT
## 21 989               0.99 LZOSYCMHQXPBFU
```

7 Session infomation

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Pop!_OS 22.04 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.10.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
##  [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] exMCnebula2_0.0.0.9000 MCnebula2_0.0.9000     testthat_3.1.4         ggplot2_3.3.6
## [5] nvimcom_0.9-142
##
## loaded via a namespace (and not attached):
##   [1] ModelMetrics_1.2.2.2   R.methodsS3_1.8.2      tidyr_1.2.0            bit64_4.0.5
##   [5] knitr_1.39             R.utils_2.11.0         styler_1.7.0           data.table_1.14.2
##   [9] rpart_4.1.16           doParallel_1.0.17      KEGGREST_1.36.2        hardhat_1.2.0
##  [13] RCurl_1.98-1.7         generics_0.1.2         preprocessCore_1.58.0  BiocGenerics_0.42.0
##  [17] callr_3.7.0            usethis_2.1.6          RSQLite_2.2.14         RApiSerialize_0.1.2
##  [21] future_1.26.1          bit_4.0.4              BiocStyle_2.24.0       xml2_1.3.3
##  [25] lubridate_1.8.0        ggsci_2.9              assertthat_0.2.1       viridis_0.6.2
##  [29] gower_1.0.1            xfun_0.31              evaluate_0.15          fansi_1.0.3
##  [33] scrime_1.3.5           caTools_1.18.2         igraph_1.3.2           DBI_1.1.2
##  [37] htmlwidgets_1.5.4      ChemmineOB_1.34.0      stats4_4.2.1           purrr_0.3.4
##  [41] ellipsis_0.3.2         dplyr_1.0.9            backports_1.4.1        bookdown_0.27
##  [45] grImport2_0.2-0        RcppParallel_5.1.5     vctrs_0.5.1            Biobase_2.56.0
##  [49] remotes_2.4.2          Cairo_1.6-0            caret_6.0-93           cachem_1.0.6
```

```
##  [53] withr_2.5.0            ggforce_0.3.3          checkmate_2.1.0        treeio_1.20.0
##  [57] prettyunits_1.1.1      svglite_2.1.0          cluster_2.1.3          FELLA_1.16.0
##  [61] ape_5.6-2              lazyeval_0.2.2         crayon_1.5.2           edgeR_3.38.1
##  [65] recipes_1.0.3          pkgconfig_2.0.3        tweenr_1.0.2           GenomeInfoDb_1.32.2
##  [69] ProtGenerics_1.28.0    nlme_3.1-159           pkgload_1.2.4          nnet_7.3-17
##  [73] devtools_2.4.3         rlang_1.0.6            globals_0.15.1         lifecycle_1.0.3
##  [77] affyio_1.66.0          rsvg_2.3.1             rprojroot_2.0.3        polyclip_1.10-0
##  [81] MetaboAnalystR_3.3.0   Matrix_1.5-1           aplot_0.1.6            base64enc_0.1-3
##  [85] processx_3.6.0         mzR_2.30.0             png_0.1-7              viridisLite_0.4.0
##  [89] stringfish_0.15.7      bitops_1.0-7           R.oo_1.25.0            pROC_1.18.0
##  [93] KernSmooth_2.23-20     Biostrings_2.64.0      blob_1.2.3             pdftools_3.2.1
##  [97] stringr_1.4.0          parallelly_1.32.0      qpdf_1.2.0             R.cache_0.15.0
## [101] jpeg_0.1-9             gridGraphics_0.5-1     S4Vectors_0.34.0       scales_1.2.0
## [105] memoise_2.0.1          magrittr_2.0.3         plyr_1.8.7             gplots_3.1.3
## [109] zlibbioc_1.42.0        compiler_4.2.1         tinytex_0.40           RColorBrewer_1.1-3
## [113] clue_0.3-61            pcaMethods_1.88.0      affy_1.74.0            cli_3.5.0
## [117] XVector_0.36.0         listenv_0.8.0          patchwork_1.1.1        pbapply_1.5-0
## [121] ps_1.7.0               htmlTable_2.4.0        Formula_1.2-4          MASS_7.3-58
## [125] tidyselect_1.2.0       vsn_3.64.0             stringi_1.7.6          highr_0.9
## [129] yaml_2.3.5             askpass_1.1            locfit_1.5-9.5         MALDIquant_1.21
## [133] latticeExtra_0.6-29    ggrepel_0.9.1          fastmatch_1.1-3        tools_4.2.1
## [137] future.apply_1.9.0     parallel_4.2.1         rstudioapi_0.13        MsCoreUtils_1.8.0
## [141] qs_0.25.4              foreach_1.5.2          foreign_0.8-82         crmn_0.0.21
## [145] classyfireR_0.3.8      gridExtra_2.3          prodlim_2019.11.13     mzID_1.34.0
## [149] farver_2.1.0           ggraph_2.0.5           utils.tool_0.0.0.9000  digest_0.6.29
## [153] BiocManager_1.30.18    ggtext_0.1.1           lava_1.7.0             Rcpp_1.0.8.3
## [157] gridtext_0.1.4         siggenes_1.70.0        ncdf4_1.19             MSnbase_2.22.0
## [161] httr_1.4.3             colorspace_2.0-3       brio_1.1.3             XML_3.99-0.10
## [165] fs_1.5.2               IRanges_2.30.0         splines_4.2.1          yulab.utils_0.0.4
## [169] tidytree_0.3.9         graphlayouts_0.8.0     multtest_2.52.0        ggplotify_0.1.0
## [173] plotly_4.10.0          sessioninfo_1.2.2      systemfonts_1.0.4      jsonlite_1.8.0
## [177] ggtree_3.4.0           tidygraph_1.2.1        timeDate_4021.107      glasso_1.11
## [181] ggfun_0.0.6            ipred_0.9-13           ggimage_0.3.1          R6_2.5.1
## [185] Hmisc_4.7-0            pillar_1.7.0           htmltools_0.5.2        glue_1.6.2
## [189] fastmap_1.1.0          BiocParallel_1.30.3    class_7.3-20           codetools_0.2-18
## [193] fgsea_1.22.0           pkgbuild_1.3.1         utf8_1.2.2             lattice_0.20-45
## [197] tibble_3.1.7           gtools_3.9.2.2         magick_2.7.3           survival_3.4-0
## [201] limma_3.52.1           rmarkdown_2.14         desc_1.4.1             munsell_0.5.0
## [205] GenomeInfoDbData_1.2.8 iterators_1.0.14       impute_1.70.0          reshape2_1.4.4
## [209] gtable_0.3.0
```

Reference

1. Dührkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, et al. SIRIUS 4: A rapid tool for turning tandem mass spectra into metabolite structure information. Nature Methods. 2019 Apr;16(4):299–302.
2. Böcker S, Letzel MC, Lipták Z, Pervukhin A. SIRIUS: Decomposing isotope patterns for metabolite identification†. Bioinformatics. 2009 Jan;25(2):218–24.
3. Dührkop K, Shen H, Meusel M, Rousu J, Böcker S. Searching molecular structure databases with tandem mass spectra using CSI:FingerID. Proceedings of the National Academy of Sciences. 2015 Oct;112(41):12580–5.

4. Ludwig M, Nothias L-F, Dührkop K, Koester I, Fleischauer M, Hoffmann MA, et al. Database-independent molecular formula annotation using Gibbs sampling through ZODIAC. Nature Machine Intelligence. 2020 Oct;2(10):629–41.

5. Dührkop K, Nothias L-F, Fleischauer M, Reher R, Ludwig M, Hoffmann MA, et al. Systematic classification of unknown metabolites using high-resolution fragmentation mass spectra. Nature Biotechnology. 2021 Apr;39(4):462–71.

6. Smyth GK. Limma: Linear Models for Microarray Data. In: Gentleman R, Carey VJ, Huber W, Irizarry RA, Dudoit S, editors. Bioinformatics and Computational Biology Solutions Using R and Bioconductor. New York: Springer-Verlag; 2005. p. 397–420.