Evaluation of MCnebula2

Contents

1 Introduction

This document provides the code to evaluate MCnebula2. Most of the data used in this is already included in the package 'exMCnebula2'. By downloading and installing the 'exMCnebula2' package, the following code can be run without trouble. The code in this document can be divided into two parts:

- The first part is the code used to generate the predicate dataset (for evaluation), which is time consuming to run, but we have already run it in advance and included the outcome data in the package 'exMCnebula2', so users do not need to rerun them, unless they are tested for feasibility.
- The second part of the code is the code for evaluating the results, including the data processing and data visualization modules, which is lightweight and can be run quickly to get the results.

2 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)
```

3 Initialisation

Create a temporary folder to store the output data.

```
tmp <- paste0(tempdir(), "/temp_data")
dir.create(tmp, F)
exfiles <- system.file("extdata", package = "exMCnebula2")
utils::untar(paste0(exfiles, "/evaluation.tar.gz"), exdir = tmp)
source(paste0(tmp, "/evaluation.R"))
```

4 Use simulated dataset

4.1 Convert .msp as .mgf

We downloaded a collated spectral data (.msp file) originating from GNPS from a third-party website. (http://prime.psc.riken.jp/compms/msdial/main.html#MSP) The .msp was converted as .mgf file and, at the same time, the simulated isotope peaks were added to MS1. Then the .mgf file could be used for SIRIUS computation. The following would not be evaluated, as the converted .mgf file has been included in .tar file.

```
if (!requireNamespace("rcdk", quietly = T)) install.packages("rcdk")
msp_to_mgf(
  name = "origin_gnps_pos.msp", id_prefix = "gnps",
  path = tmp, mass_level = "all", fun = "deal_with_msp_record"
)
```

4.2 Query classification for compounds

In order for this data to be used for evaluation, we need to first obtain the classes of these compounds. (The results has been included in .tar files. So the following need not to be run again, as its time-consuming.)

```
mgf_metadata <- data.table::fread(paste0(tmp, "/", "origin_gnps_pos.meta.tsv"))
dir.create(tmp1 <- paste0(tmp, "/query"))
key2d <- unique(stringr::str_extract(mgf_metadata$INCHIKEY, "^[A-Z]*"))
key.rdata <- query_inchikey(key2d, tmp1, curl_cl = 5)
class.rdata <- query_classification(key2d, tmp1, classyfire_cl = 5)
```

4.3 Add noise peak

The 'noise' include mass shift, peak intensity shift, and external noise peaks for original spectra. (refer to https://doi.org/10.1038/s41587-021-01045-9). First, we collected a subset of the noise peaks used to insert the spectra:

```
load(files[1])
non_noise <- dplyr::select(
  latest(mcn, "project_dataset", ".f3_spectra"),
  .features_id, mz, rel.int.
```

```r
)
non_noise <- split(non_noise, ~.features_id)
non_noise <- lapply(non_noise, dplyr::select, -.features_id)
rm(mcn)
origin_lst <- filter_mgf(NULL, paste0(tmp, "/gnps_pos.mgf"))
noise_pool <- collate_as_noise_pool(origin_lst, non_noise)
```

Two noise models were simulated:

```r
medium_noise_lst <- spectrum_add_noise(origin_lst,
  int.sigma = 1,
  global.sigma = 10 / 3 * 1e-06, indivi.sigma = 10 / 3 * 1e-06,
  sub.factor = 0.03, alpha = 0.2, .noise_pool = noise_pool
)
high_noise_lst <- spectrum_add_noise(origin_lst,
  int.sigma = 2^(1 / 2),
  global.sigma = 15 / 3 * 1e-06, indivi.sigma = 15 / 3 * 1e-06,
  sub.factor = 0.03, alpha = 0.4, .noise_pool = noise_pool
)
```

4.4 Output .mgf for SIRIUS

Export the above three list data to the .mgf format required by SIRIUS.

```r
lst <- list(
  origin = origin_lst, medium_noise = medium_noise_lst,
  high_noise = high_noise_lst
)
dir.create(tmp2 <- paste0(tmp, "/simutate"))
lapply(names(lst), function(name) {
  data <- data.table::rbindlist(lst[[name]])
  write.table(data, paste0(tmp2, "/", name, "_gnps_pos.sirius.mgf"),
    quote = F, col.names = F, row.names = F
  )
})
```

4.5 Output .mgf for GNPS

Export the above three list data to the .mgf format required by GNPS (FBMN). (https://doi.org/10.1038/s41592-020-0933-6). FBMN worklow required two types of data for upload, .mgf file and .csv (quantification table)

```r
quant_table <- simulate_gnps_quant(mgf_metadata, tmp2, return_df = T)
lapply(names(lst), function(name) {
  lst <- discard_level1(lst[[name]])
  lst <- pbapply::pblapply(lst, mgf_add_anno.gnps)
  data <- data.table::rbindlist(lst)
  data <- dplyr::mutate(data, V1 = gsub(
    "RTINSECONDS=", "RTINSECONDS=1000",
    V1
  ), V1 = gsub("CHARGE=+1", "CHARGE=1+", V1), V1 = gsub(
    "FEATURE_ID=gnps",
    "FEATURE_ID=", V1
  ))
  write.table(data, paste0(tmp2, "/", name, "_gnps_pos.gnps.mgf"),
    quote = F, col.names = F, row.names = F
  )
  quant <- dplyr::filter(quant_table, `row m/z` <= 800, `row ID` %in%
    names(lst))
  quant$`row ID` <- stringr::str_extract(quant$`row ID`, "[0-9]{1,}$")
  write.table(quant, paste0(tmp2, "/", name, "_gnps_pos.gnps.meta.csv"),
    sep = ",", row.names = F, col.names = T, quote = F
  )
})
```

5 Evaluate MCnebula2

Extract the required data from the pre-computed SIRIUS projects, using the following code:

```
dirs <- c("origin", "medium_noise", "high_noise")
lst <- lapply(dirs, function(dir) {
  mcn <- collated_used()
  mcn <- mcnebula()
  mcn <- initialize_mcnebula(mcn, "sirius.v4", dir)
  mcn <- collate_used(mcn)
})
```

5.1 Integrate data

Integrate data to classifiy the 'features' via MCnebula2. (The following .rdata files were not provided in packages of 'exMCnebula2'. But you can downloaded them via URL such as: https://raw.githubusercontent.com/Cao-lab-zcmu/utils_tool/master/inst/extdata/evaluationLarge/origin_gnps_pos.rdata)

```
files <- paste0(
  system.file("extdata/evaluationLarge", package = "utils.tool"),
  "/", c(
    "origin_gnps_pos.rdata", "medium_noise_gnps_pos.rdata",
    "high_noise_gnps_pos.rdata"
  )
)
lst <- lapply(files, function(file) {
  load(file)
  mcn <- filter_structure(mcn)
  mcn <- create_reference(mcn)
  mcn <- filter_formula(mcn, by_reference = T)
  mcn <- create_stardust_classes(mcn)
  mcn <- create_features_annotation(mcn)
  mcn <- cross_filter_stardust(mcn,
    min_number = 50, max_ratio = 0.1,
    cutoff = 0.4, identical_factor = 0.6
  )
  mcn <- create_nebula_index(mcn, force = T)
  if (file == files[1]) {
    mcn <- create_hierarchy(mcn)
    mcnHier. <- mcnebula()
    reference(mcn_dataset(mcnHier.))$hierarchy <- hierarchy(mcn)
    save(mcnHier., file = paste0(system.file("extdata/evaluation",
      package = "utils.tool"
    ), "/mcnHier.rdata"))
  }
  list(features_annotation = features_annotation(mcn), nebula_index = nebula_index(mcn))
})
names(lst) <- dirs
save(lst, file = paste0(
  system.file("extdata/evaluation", package = "utils.tool"),
  "/integrated.rdata"
))
```

5.2 Use pre-integrate data

Load the integrated data in the 'exMCnebula2' package. This data contains the results of the three levels, as well as the corresponding classified results and identification results.

```
load(paste0(tmp, "/integrated.rdata"))
```

5.3 Download results of finished jobs from GNPS service

In order to compare MCnebula2 with MolNetEnhancer in GNPS, we have pre-uploaded and completed jobs of FBMN and MolNetEnhancer. The URL of finished jobs in GNPS service were as following. (MolNetEnhancer: https://doi.org/10.1101/654459)

```
fbmn_lst <- list(
  origin = "https://gnps.ucsd.edu/proteosafe/status.jsp?task=05f492249df5413ba72a1def76ca973d",
  medium_noise = "https://gnps.ucsd.edu/proteosafe/status.jsp?task=c65abe76cd9846c99f1ae47ddbd34927",
  high_noise = "https://gnps.ucsd.edu/proteosafe/status.jsp?task=62b25cf2dcf041d3a8b5593fdbf5ac5e"
)
molnet_lst <- list(
  origin = "https://gnps.ucsd.edu/ProteoSAFe/status.jsp?task=9d9c7f83fa2046c2bf615a3dbe35ca62",
  medium_noise = "https://gnps.ucsd.edu/ProteoSAFe/status.jsp?task=7cc8b5a2476f4d4e90256ec0a0f94ca7",
  high_noise = "https://gnps.ucsd.edu/ProteoSAFe/status.jsp?task=f6d08a335e814c5eac7c97598b26fb80"
)
```

The data that would be used has been extracted.

```
molnet_files <- paste0(tmp, "/", "gnps_results")
molnet_lst <- sapply(c("origin", "medium_noise", "high_noise"),
  simplify = F, function(dir) {
    data.table::fread(paste0(molnet_files, "/", dir, "/ClassyFireResults_Network.txt"))
  }
)
```

5.4 Evaluate accuracy of classify

Evaluate the 'features' of each chemical class within the Nebula-Index: whether the compounds relative to the chemical class.

5.4.1 Load assessment data and reference data.

The evaluation and reference data have already been saved in the previous steps and only need to be loaded:

```
mgf_metadata <- data.table::fread(paste0(tmp, "/", "origin_gnps_pos.meta.tsv"))
id2key <- stringr::str_extract(mgf_metadata$INCHIKEY, "^[A-Z]*")
names(id2key) <- mgf_metadata$.id
id2key <- as.list(id2key)
```

```
load(paste0(tmp, "/mcnHier.rdata"))
class.db <- extract_rdata_list(paste0(tmp, "/classification.rdata"))
```

5.4.2 Filter assessment data.

Filtered according to: the common ‘features’ (at least identified to the chemical formula) in the three levels; whether included in reference data (`class.db`).

```
common <- lapply(lst, function(l) l$features_annotation$.features_id)
common <- common[[1]][(common[[1]] %in% common[[2]]) & (common[[1]] %in%
  common[[3]])]
keep <- id2key %in% names(class.db)
common <- common[common %in% names(id2key)[keep]]
lst <- lapply(lst, function(l) {
  l$nebula_index <- dplyr::filter(l$nebula_index, .features_id %in%
    common)
  return(l)
})
```

5.4.3 Count the results.

Assess the accuracy and derive ratios for each type of result.

```
res <- lapply(lst, function(l) {
  l <- lapply(split(l$nebula_index, ~class.name), function(df) df[[".features_id"]])
  stat_list <- sapply(names(l), simplify = F, function(class.name) {
    stat_classify(
      l[[class.name]], class.name, id2key, mcnHier.,
      class.db
    )
  })
  stat_table <- data.table::rbindlist(lapply(stat_list, table_app,
    prop = T
  ), fill = T, idcol = T)
  stat_table <- dplyr::rename(stat_table, class.name = .id)
  stat_table <- dplyr::summarise_all(stat_table, function(x) {
    ifelse(is.na(x),
      0, x
    )
  })
})
```

5.4.4 Post-filtering.

Filter out chemical classes with insufficient number of features; keep only those contained in ‘origin’.

```
res[[1]] <- dplyr::filter(res[[1]], sum >= 50)
keep <- res[[1]]$class.name
res[2:3] <- lapply(res[2:3], dplyr::filter, class.name %in% keep)
```

5.4.5 Distinguish the chemical class of the dominant structure.

Our reference data (`class.db`) contains only chemical classes for the dominant structure of a compound, but not for the sub-structure of a compound. Our evaluation data (`lst`) contain both chemical classes of substructures and classes of dominant structures. This makes the assessment biased. But they are easily distinguishable:

```
dominant_res <- lapply(res, dplyr::filter, false < 0.4)
sub_res <- lapply(res, dplyr::filter, false >= 0.4)
summarise <- function(df) {
  false <- round(mean(df$false) * 100, 1)
  sum <- round(mean(df$sum), 1)
  list(false = false, sum = sum)
}
summary <- lapply(dominant_res, summarise)
```

These chemical classes represent mostly local structures in compounds (they represent very small structures that are present in many other classes of compounds):

```
sub_res[[1]]$class.name
```

```
##   [1] "Aldehydes"
##   [2] "Alkaloids and derivatives"
##   [3] "Alkanolamines"
##   [4] "Alkyl glycosides"
##   [5] "Alpha-acyloxy carbonyl compounds"
##   [6] "Alpha,beta-unsaturated carbonyl compounds"
##   [7] "Amino fatty acids"
##   [8] "Aminosaccharides"
##   [9] "Amphetamines and derivatives"
##  [10] "Aniline and substituted anilines"
##  [11] "Aralkylamines"
##  [12] "Aromatic alcohols"
##  [13] "Aromatic monoterpenoids"
##  [14] "Aryl chlorides"
##  [15] "Aryl halides"
##  [16] "Benzamides"
##  [17] "Benzenediols"
##  [18] "Benzenetriols and derivatives"
##  [19] "Benzodiazines"
##  [20] "Benzodioxoles"
##  [21] "Benzofurans"
##  [22] "Benzoic acid esters"
##  [23] "Benzoic acids"
##  [24] "Benzoic acids and derivatives"
##  [25] "Benzoyl derivatives"
##  [26] "Beta hydroxy acids and derivatives"
##  [27] "Bicyclic monoterpenoids"
##  [28] "Branched fatty acids"
##  [29] "Butenolides"
##  [30] "Catechols"
##  [31] "Chlorobenzenes"
##  [32] "Chromones"
##  [33] "Cinnamic acid esters"
##  [34] "Cinnamic acids and derivatives"
##  [35] "Coumarans"
##  [36] "Coumaric acids and derivatives"
##  [37] "Cresols"
##  [38] "Delta valerolactones"
##  [39] "Dialkyl ethers"
##  [40] "Diarylethers"
##  [41] "Diazanaphthalenes"
##  [42] "Diazinanes"
##  [43] "Dihydrofurans"
##  [44] "Dimethoxybenzenes"
##  [45] "Dioxopiperazines"
##  [46] "Disaccharides"
##  [47] "Epoxides"
##  [48] "Fatty acid esters"
##  [49] "Fatty acids and conjugates"
##  [50] "Fatty acyl glycosides"
##  [51] "Fatty acyl glycosides of mono- and disaccharides"
##  [52] "Fatty alcohols"
##  [53] "Flavanones"
##  [54] "Furanones"
##  [55] "Furans"
##  [56] "Gamma butyrolactones"
##  [57] "Gluco/mineralocorticoids, progestogins and derivatives"
##  [58] "Hydroxy acids and derivatives"
##  [59] "Hydroxy fatty acids"
##  [60] "Hydroxybenzoic acid derivatives"
##  [61] "Hydroxycinnamic acid esters"
##  [62] "Hydroxycinnamic acids and derivatives"
##  [63] "Imidazoles"
##  [64] "Indoles"
##  [65] "Indoles and derivatives"
##  [66] "Iridoids and derivatives"
##  [67] "Isoflavones"
##  [68] "Isoindoles"
##  [69] "Isoindoles and derivatives"
##  [70] "Isoindolines"
##  [71] "Isoindolones"
##  [72] "Lactams"
##  [73] "Long-chain fatty acids"
##  [74] "Macrolactams"
##  [75] "Macrolides and analogues"
##  [76] "Medium-chain fatty acids"
##  [77] "Meta cresols"
##  [78] "Methoxybenzenes"
##  [79] "Methoxybenzoic acids and derivatives"
##  [80] "Methoxyphenols"
##  [81] "Monocyclic monoterpenoids"
##  [82] "N-alkylpiperazines"
##  [83] "Naphthalenes"
##  [84] "Naphthopyrans"
##  [85] "Organic carbonic acids and derivatives"
##  [86] "Organochlorides"
##  [87] "Organohalogen compounds"
##  [88] "Oxepanes"
##  [89] "Phenanthrenes and derivatives"
##  [90] "Phenethylamines"
##  [91] "Piperazines"
##  [92] "Piperidines"
##  [93] "Pyranochromenes"
##  [94] "Pyridines and derivatives"
##  [95] "Pyrimidones"
##  [96] "Pyrroles"
##  [97] "Pyrrolidines"
##  [98] "Pyrrolidones"
##  [99] "Quinazolines"
## [100] "Resorcinols"
## [101] "Secondary amines"
## [102] "Styrenes"
## [103] "Substituted pyrroles"
## [104] "Tertiary alcohols"
## [105] "Tertiary amines"
## [106] "Tetracarboxylic acids and derivatives"
## [107] "Tetrahydrofurans"
## [108] "Tetralins"
## [109] "Toluenes"
## [110] "Triazoles"
## [111] "Tricarboxylic acids and derivatives"
## [112] "Unsaturated fatty acids"
## [113] "Ureas"
## [114] "Vinylogous amides"
## [115] "Vinylogous esters"
```

5.4.6 Visualizaion

Visualize the results of evaluation:

```
vis_lst <- lapply(dominant_res, visualize_stat, mcn = mcnHier.)
pdfs <- list()
for (i in names(vis_lst)) {
  pdfs[[i]] <- paste0(tmp, "/", i, "_classified_accuracy.pdf")
  pdf(pdfs[[i]], 13, 11)
  draw(vis_lst[[i]])
  dev.off()
}
```

Totally 7524 compounds used for evaluation. For the origin dataset, the average false rate of MCnebula2 classifying is 17.8% (Fig. 1); the average classified number of 'features' is 161.7. For the medium noise dataset, the average false rate of MCnebula2 classifying is 19.4% (Fig. 2); the average classified number of 'features' is 159.8. For the high noise dataset, the average false rate of MCnebula2 classifying is 20.5% (Fig. 3). the average classified number of 'features' is 148.

See following:

```
summary
```

```
## $origin
## $origin$false
## [1] 17.8
##
## $origin$sum
## [1] 161.7
##
##
## $medium_noise
## $medium_noise$false
## [1] 19.4
##
```

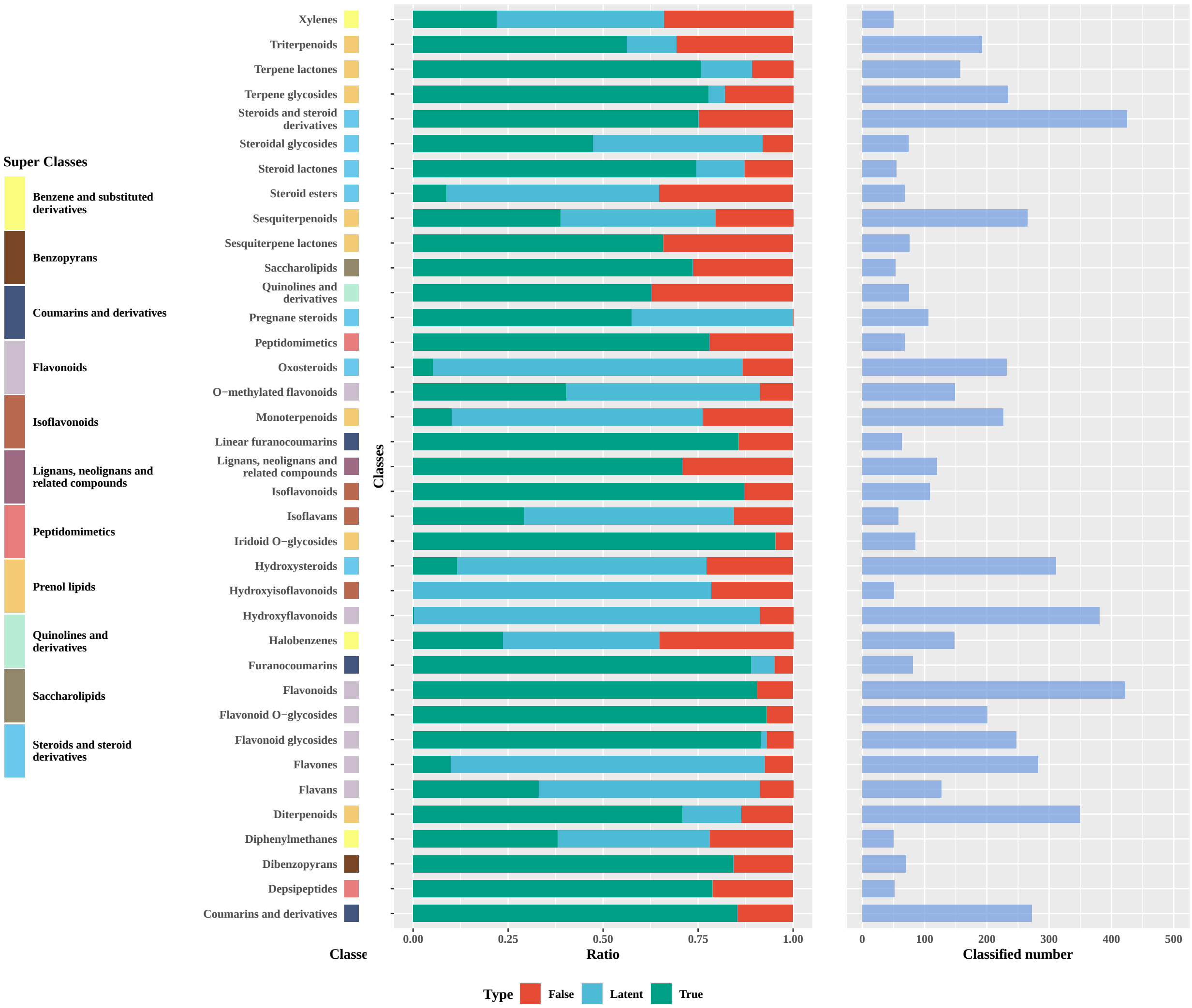

Figure 1: Classified accuracy (MCnebula2) of origin dataset

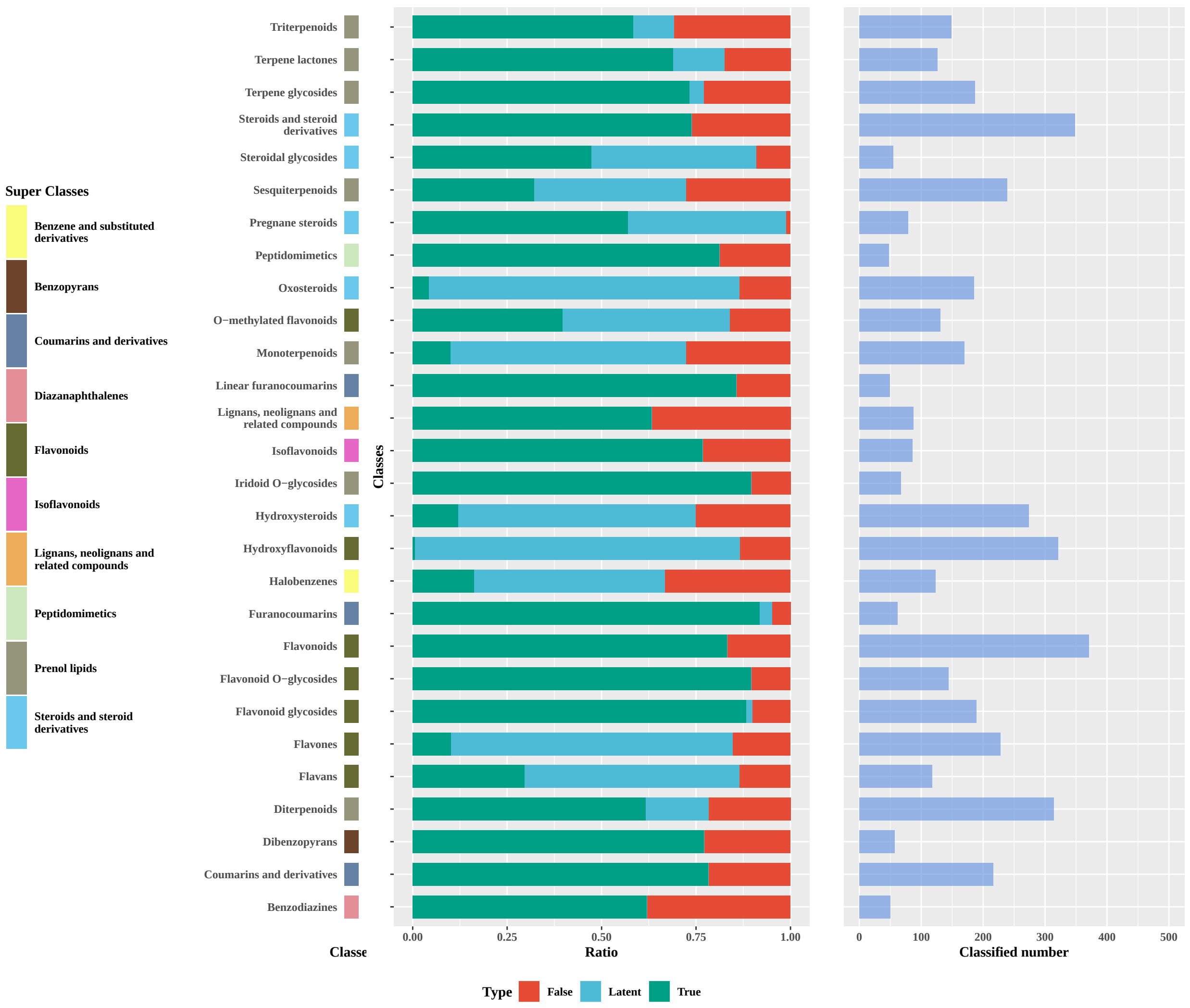

Figure 2: Classified accuracy (MCnebula2) of medium noise dataset

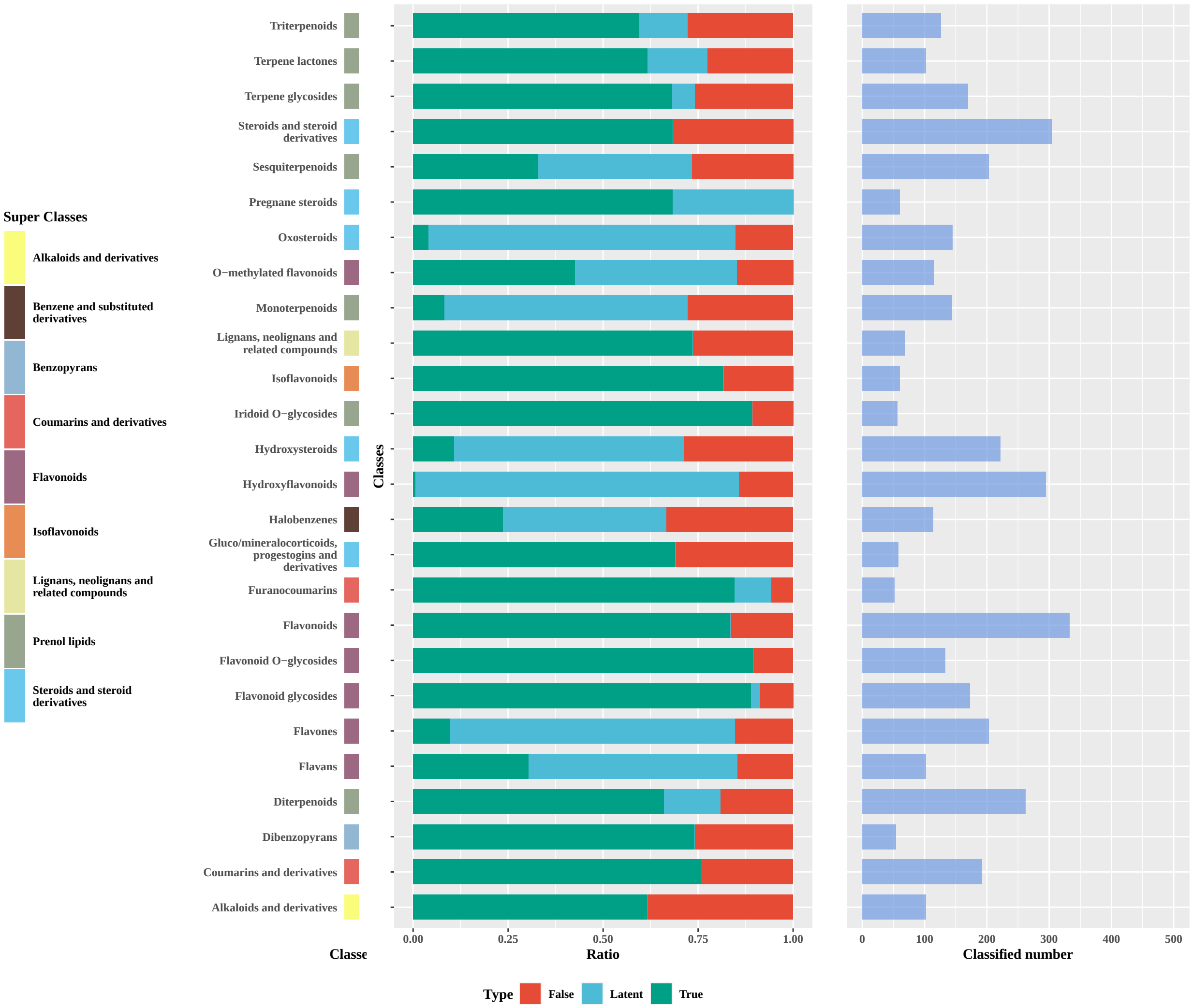

Figure 3: Classified accuracy (MCnebula2) of high noise dataset

```
## $medium_noise$sum
## [1] 159.8
##
##
## $high_noise
## $high_noise$false
## [1] 20.5
##
## $high_noise$sum
## [1] 148
```

Gather three levels (origin, medium_noise, high_noise) of results:

```
vis <- visualize_statComplex(dominant_res, mcnHier., weight = c(
  pl = 0.7,
  pm = 1.1, pr = 0.8
))
pdf(
  f2.577 <- paste0(tmp, "/gather_classified_accuracy.pdf"),
  15.5, 13
)
draw(vis)
dev.off()
```

See results (Fig. 4).

5.4.7 Compare with MolNetEnhancer

Pre-process the data.

```
molnet_lst <- lapply(molnet_lst, function(df) {
  df <- dplyr::select(df, .id = `cluster index`, ends_with("class"))
  df <- dplyr::mutate(df, .id = paste0("gnps", .id))
  df <- dplyr::filter(df, .id %in% !!common)
  lst <- lapply(2:4, function(n) {
    lst <- split(df, df[[n]])
    lst <- lst[!names(lst) %in% c("", "no matches")]
    lst <- lapply(lst, function(df) {
      if (length(df[[".id"]]) >= 50) {
        df[[".id"]]
      } else {
        NULL
      }
    })
  })
  lst <- unlist(lst, recursive = F)
  lst[!vapply(lst, is.null, logical(1))]
})
```

Count the results.

```
res_molnet <- lapply(molnet_lst, function(l) {
  stat_list <- sapply(names(l), simplify = F, function(class.name) {
    stat_classify(
      l[[class.name]], class.name, id2key, mcnHier.,
      class.db
    )
```

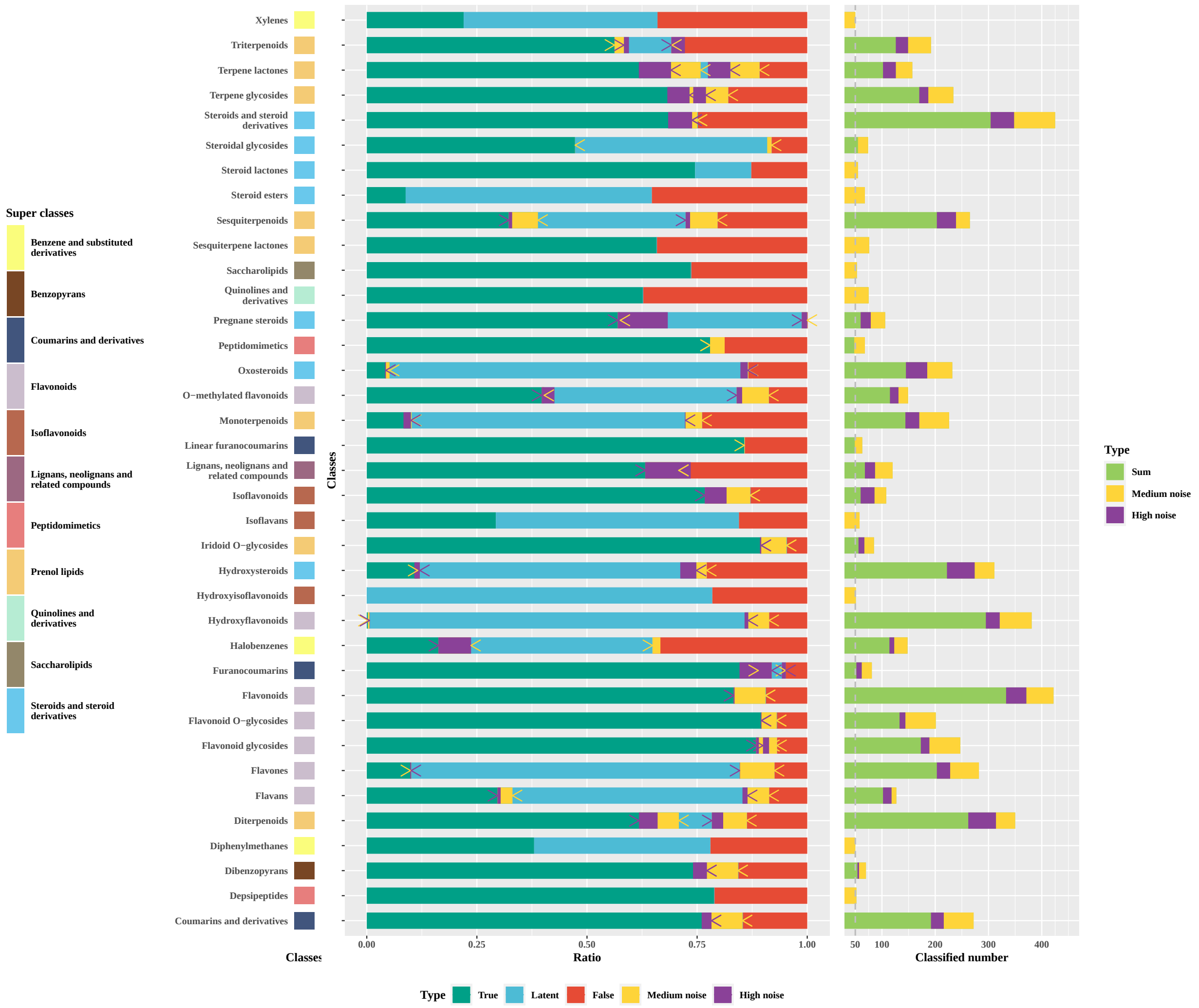

Figure 4: Classified accuracy (MCnebula2) of three levels dataset

```
  })
  stat_table <- data.table::rbindlist(lapply(stat_list, table_app,
    prop = T
  ), fill = T, idcol = T)
  stat_table <- dplyr::rename(stat_table, class.name = .id)
  stat_table <- dplyr::summarise_all(stat_table, function(x) {
    ifelse(is.na(x),
      0, x
    )
  })
})
res_molnet[[1]] <- dplyr::filter(res_molnet[[1]], sum >= 50)
keep <- res_molnet[[1]]$class.name
res_molnet[2:3] <- lapply(res_molnet[2:3], dplyr::filter, class.name %in%
  keep)
summary_molnet <- lapply(res_molnet, summarise)
```

Visualize the results.

```
vis_lst <- lapply(res_molnet, visualize_stat,
  mcn = mcnHier.,
  weight = c(pl = 1, pm = 0.8, pr = 0.7)
)
pdfs2 <- list()
for (i in names(vis_lst)) {
  pdfs2[[i]] <- paste0(tmp, "/", i, "_classified_accuracy_Molnet.pdf")
  w <- if (i == "origin") {
    15
  } else {
    11
  }
  pdf(pdfs2[[i]], w, 11)
  draw(vis_lst[[i]])
  dev.off()
}
vis <- visualize_statComplex(res_molnet, mcnHier., y_cut_left = c(
  50,
  700
), y_cut_right = c(800, 2000), y_cut_left_breaks = c(
  50,
  seq(100, 700, by = 200)
), y_cut_right_breaks = c(1200, 1600), )
pdf(
  f2.583 <- paste0(tmp, "/gather_classified_accuracy_Molnet.pdf"),
  18, 13
)
draw(vis)
dev.off()
```

For the origin dataset, the average false rate of MolNetEnhancer classifying is 13.3% (Fig. 5); the average classified number of 'features' is 285.4. For the medium noise dataset, the average false rate of MolNetEnhancer classifying is 6.8% (Fig. 6); the average classified number of 'features' is 196.2. For the high noise dataset, the average false rate of MolNetEnhancer classifying is 4.4% (Fig. 7); the average classified number of 'features' is 147.3.

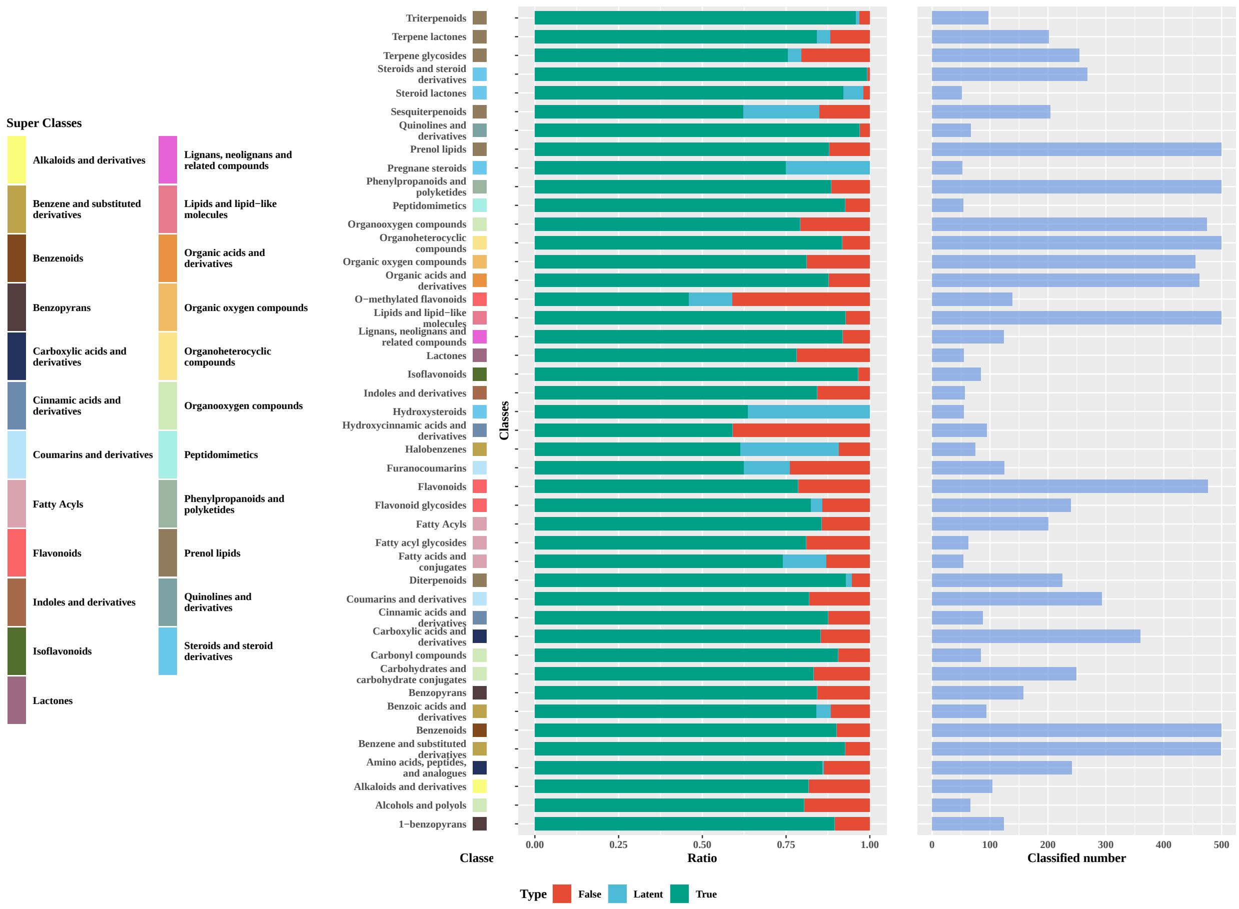

Figure 5: Classified accuracy (MolnetEnhancer) of origin dataset

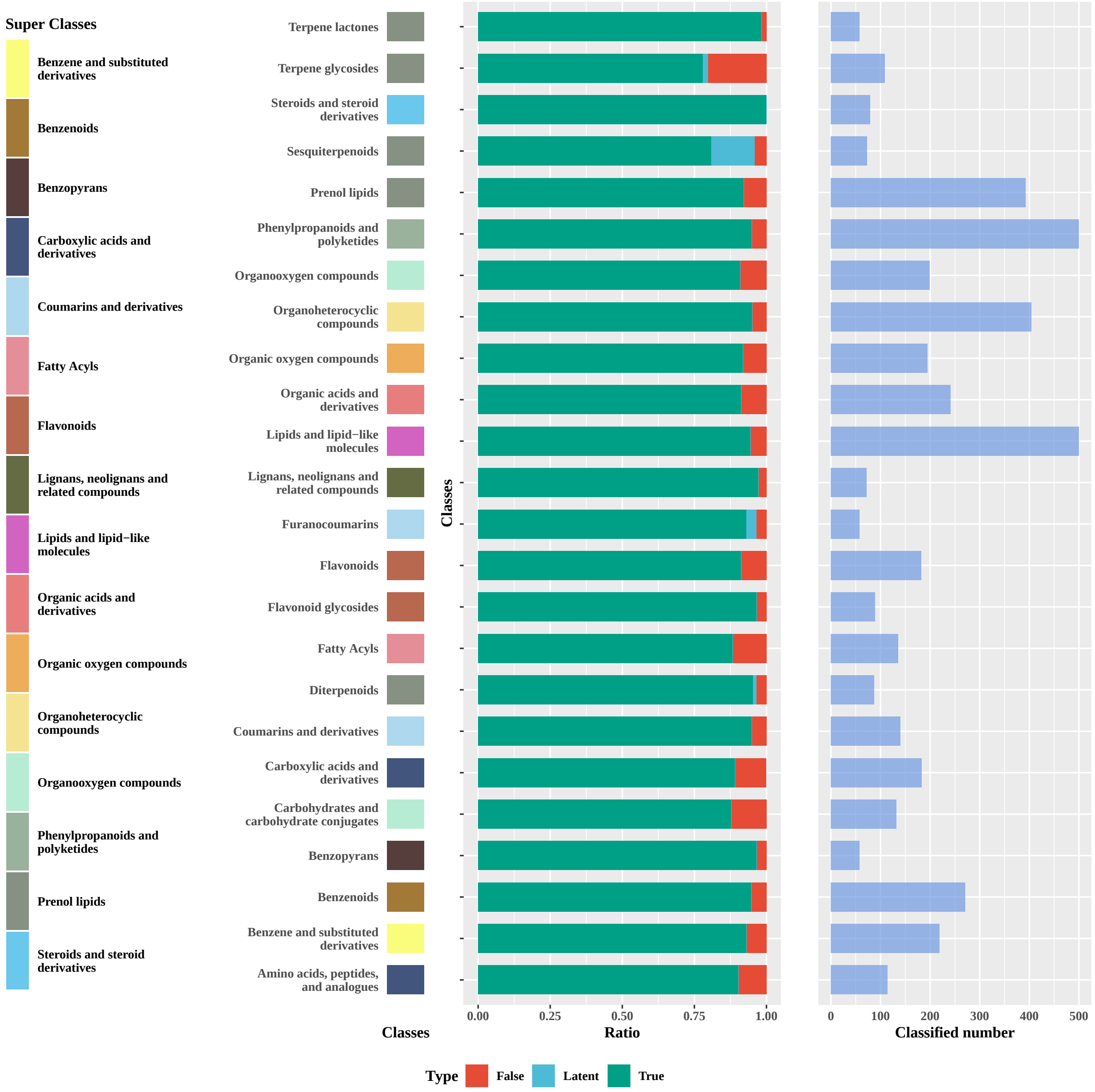

Figure 6: Classified accuracy (MolNetEnhancer) of medium noise dataset

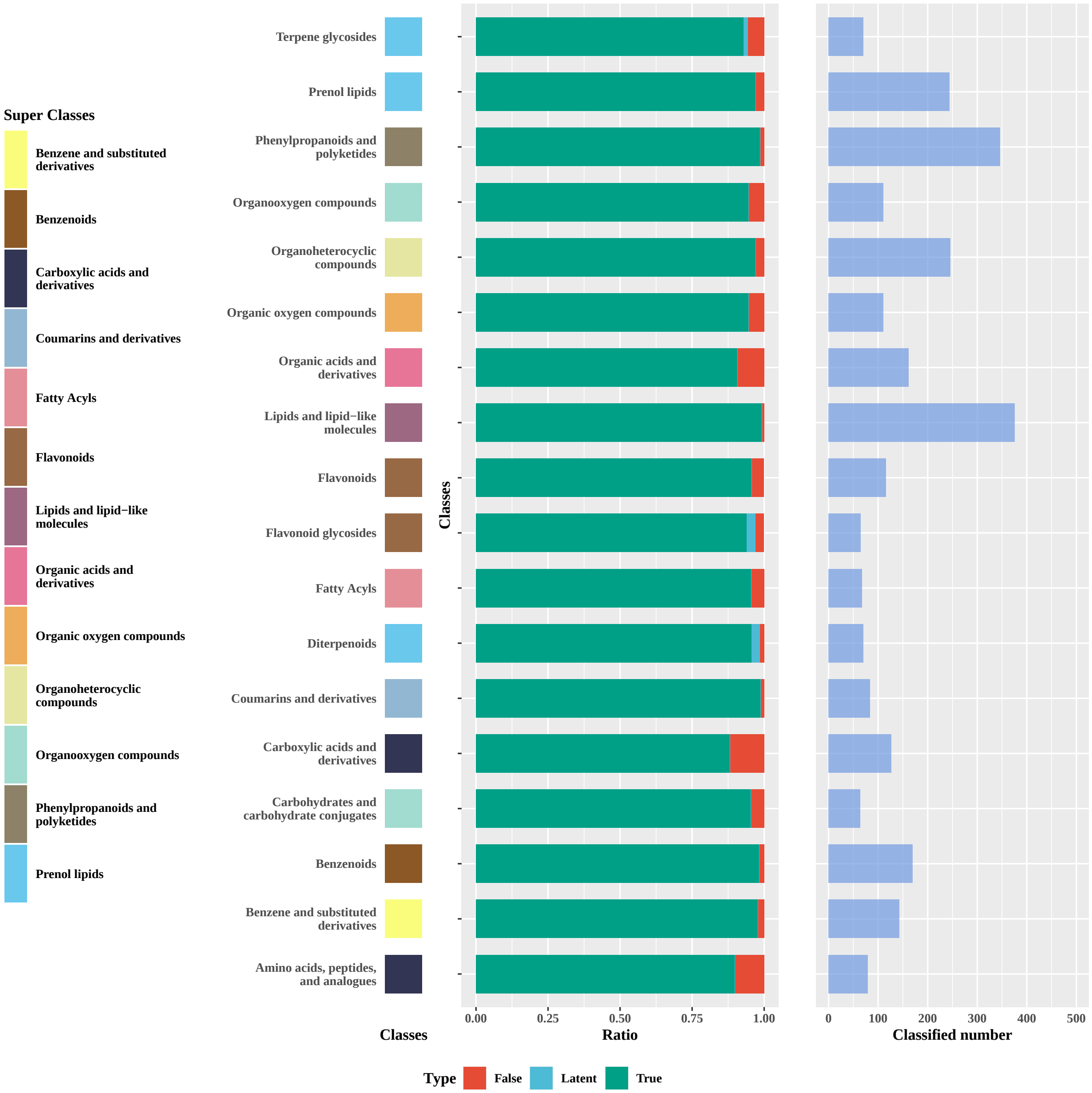

Figure 7: Classified accuracy (MolNetEnhancer) of high noise dataset

See following:

```
summary_molnet
```

```
## $origin
## $origin$false
## [1] 13.3
##
## $origin$sum
## [1] 285.4
##
##
## $medium_noise
## $medium_noise$false
## [1] 6.8
##
## $medium_noise$sum
## [1] 196.2
##
##
## $high_noise
## $high_noise$false
## [1] 4.4
##
## $high_noise$sum
## [1] 147.3
```

See results (Fig. 8).

5.4.8 Summary

The following data is available.

```
res_parallel <- attr(vis, "data")
comman_class <- unique(res_parallel[[1]]$class.name)
res_mcnebula <- dominant_res
res_molnet
summary_mcnebula <- summary
summary_molnet
```

Also required:

```
res_mcnebula_common <- lapply(res_mcnebula, dplyr::filter, class.name %in%
  comman_class)
res_molnet_common <- lapply(res_molnet, dplyr::filter, class.name %in%
  comman_class)
summary_mcnebula_common <- lapply(res_mcnebula_common, summarise)
summary_molnet_common <- lapply(res_molnet_common, summarise)
```

Draw the figure:

```
lst_molnet <- visualize_summary(summary_molnet_common)
lst_mcnebula <- visualize_summary(summary_mcnebula_common)
vis <- lapply(namel(lst_mcnebula, lst_molnet), function(lst) {
  bar <- as_grob(lst$p.num)
  ring <- list(
    p.precision = as_grob(lst$p.precision$p.m),
```

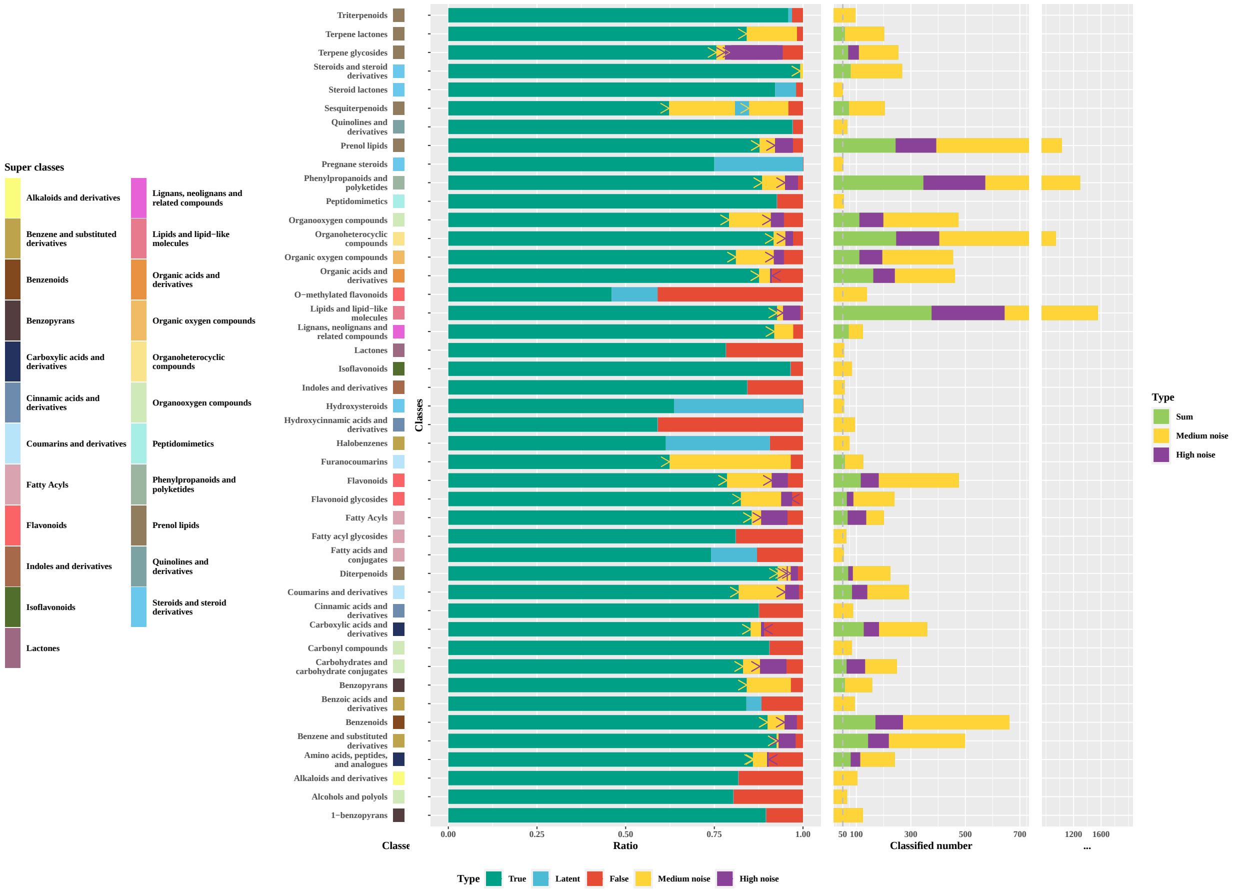

Figure 8: Classified accuracy (MolNetEnhancer) of three levels dataset

```
    p.ratioSt = as_grob(lst$p.ratioSt$p.m), p.recall = as_grob(lst$p.recall$p.m)
  )
  frame <- frame_col(
    c(p.ratioSt = 2, p.precision = 3, p.recall = 3),
    ring
  )
  frame_col(c(bar = 2, frame = 8), c(namel(bar), namel(frame)))
})
vis <- frame_row(c(lst_mcnebula = 1, lst_molnet = 1), vis)
label_mcnebula <- gtext90("MCnebula", "#4DBBD5FF")
label_molnet <- gtext90("GNPS", "#E64B35FF")
group_labels <- frame_row(
  c(label_mcnebula = 1, null = 0.1, label_molnet = 1),
  namel(label_mcnebula, label_molnet, null = nullGrob())
)
vis2 <- frame_col(c(group_labels = 0.1, vis = 5), namel(
  group_labels,
  vis
))
legend <- frame_col(
  c(null = 2, null = 2, null = 3, null = 3),
  list(
    null = nullGrob(), p.precision = lst_molnet$p.precision$p.l,
    p.ratioSt = lst_molnet$p.ratioSt$p.l
  )
)
vis3 <- frame_row(c(vis2 = 5, legend = 0.5), namel(vis2, legend))
label_sum <- gtext90("Sum number", "#709AE1FF", 0)
label_false <- gtext90("Precision", "#FED439FF", 0)
label_stab <- gtext90("Stability", "#91D1C2", 0)
label_recall <- gtext90("Recall", "#D5E4A2FF", 0)
title_labels <- frame_col(c(
  null = 0.2, label_sum = 2, null = 0.1,
  label_stab = 2, null = 0.1, label_false = 3, null = 0.1,
  label_recall = 3
), namel(label_sum, label_false, label_stab,
  label_recall,
  null = nullGrob()
))
vis4 <- frame_row(
  c(title_labels = 1, null = 0.5, vis3 = 15),
  namel(title_labels, vis3, null = nullGrob())
)
vis4 <- ggather(vis4, vp = viewport(, , 0.95, 0.95))
pdf(f2.59 <- paste0(tmp, "/evaluation_summary.pdf"), 17, 5)
draw(vis4)
dev.off()
```

Under the same conditions (common chemical classes), MCnebula outperforms the benchmark method (Fig. 9). The formula for Relative false rate and Recall (TP / (TP + FN)): - RelativeFalseRate = 1 - (1 - FalseRate) * (1 - AverageLostRate) - Recall = (1 - RelativeFalseRate) / (1 - RelativeFalseRate + AverageLostRate)

Combine figure…

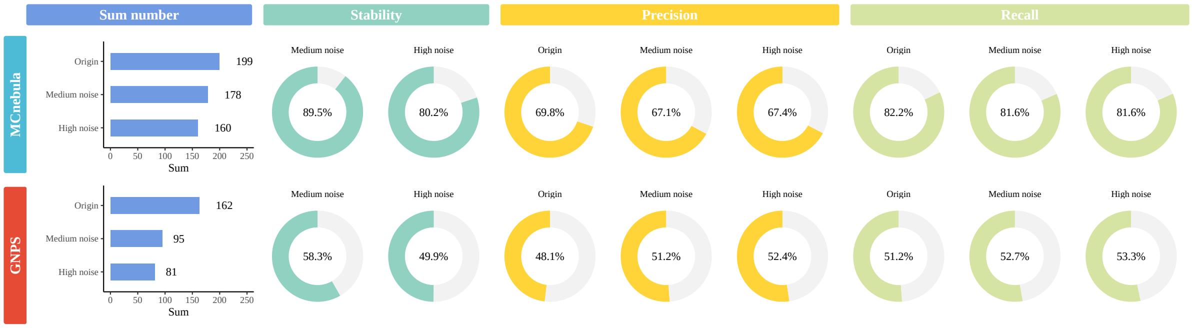

Figure 9: Evaluation sumary for MCnebula and benchmark method

```
vis_summary <- into(grecta("a"), vis4)
vis_classify <- visualize_comparison(dominant_res, res_molnet,
  from = c("MCnebula", "GNPS")
)
vis_classify <- into(grecta("b"), vis_classify)
vis_id <- visualize_idRes(list(`No cut-off` = idRes, `0.5 cut-off` = idRes.5))
vis_id <- into(grecta("c"), vis_id)
frame1 <- frame_col(c(vis_classify = 2, vis_id = 1), namel(
  vis_classify,
  vis_id
))
frame2 <- frame_row(c(vis_summary = 1, frame1 = 2), namel(
  vis_summary,
  frame1
))
frame2 <- ggather(frame2, vp = viewport(, , 0.95, 0.95))
pdf(fs <- paste0(tmp, "/compare_accuracy.pdf"), 18, 14)
draw(frame2)
dev.off()
```

5.5 Evaluate accuracy of identification

Evaluate the accuracy of identification with origin dataset:

```
identified <- dplyr::select(
  lst[[1]]$features_annotation, .features_id,
  inchikey2d, tani.score
)
identified <- dplyr::filter(identified, !is.na(tani.score))
index <- dplyr::filter(lst[[1]]$nebula_index, .features_id %in%
  identified$.features_id)
index <- split(index, ~class.name)
index <- index[names(index) %in% dominant_res[[1]]$class.name]
ref_inchikey2d <- dplyr::mutate(mgf_metadata, inchikey2d = stringr::str_extract(
  INCHIKEY,
  "^[A-Z]*"
))
ref_inchikey2d <- dplyr::select(ref_inchikey2d,
  .features_id = .id,
```

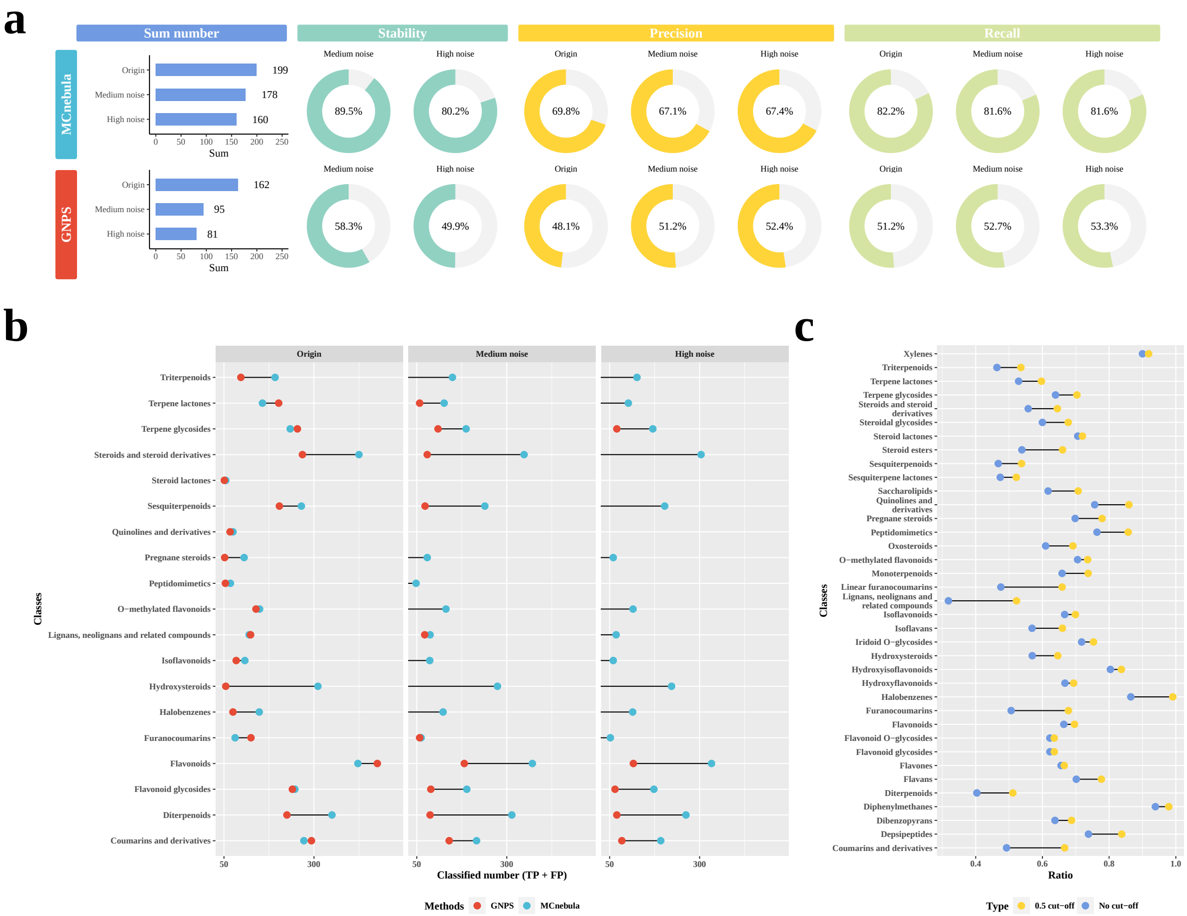

Figure 10: Overall summary

```
  inchikey2d
)
idRes <- stat_identification(index, identified, ref_inchikey2d)
idRes.summary <- summarise(idRes)
```

Set a cut-off for ‘tani.score’ (Tanimoto similarity).

```
identified.5 <- dplyr::filter(identified, tani.score >= 0.5)
index.5 <- lapply(index, dplyr::filter, .features_id %in% identified.5$.features_id)
idRes.5 <- stat_identification(index.5, identified.5, ref_inchikey2d)
idRes.5.summary <- summarise(idRes.5)
```

For all identified compounds, the average false rate was 37%. Set 0.5 for ‘tani.score’ as threshold, the average false rate was 29.4%.

Visualize the results:

```
vis <- visualize_idRes(list(`No cut-off` = idRes, `0.5 cut-off` = idRes.5))
pdf(f2.83 <- paste0(tmp, "/identified_accuracy.pdf"), 6, 9)
draw(vis)
dev.off()
```

See results (Fig. 11).

6 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] backports_1.4.1        Hmisc_4.7-0            systemfonts_1.0.4      plyr_1.8.7
##   [5] igraph_1.3.2           lazyeval_0.2.2         splines_4.2.1          BiocParallel_1.32.6
##   [9] ChemmineOB_1.34.0      usethis_2.1.6          GenomeInfoDb_1.34.9    digest_0.6.29
##  [13] yulab.utils_0.0.4      htmltools_0.5.2        viridis_0.6.2          magick_2.7.3
##  [17] fansi_1.0.3            checkmate_2.1.0        magrittr_2.0.3         memoise_2.0.1
```

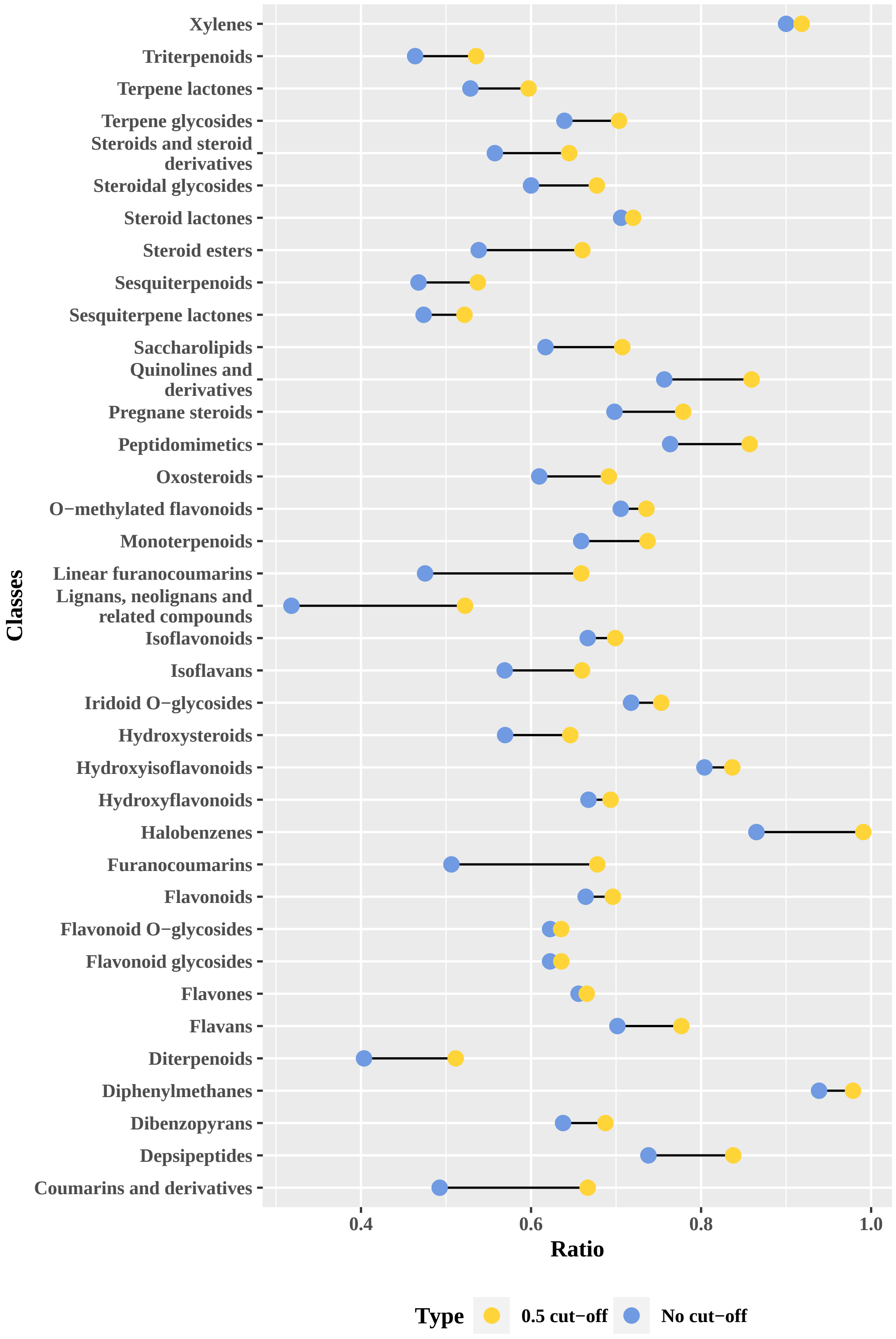

Figure 11: Identified accuracy of compounds in each classified chemical class

```
##  [21] grImport2_0.2-0        cluster_2.1.3          remotes_2.4.2          Biostrings_2.66.0
##  [25] graphlayouts_0.8.0     R.utils_2.11.0         svglite_2.1.0          askpass_1.1
##  [29] FELLA_1.16.0           prettyunits_1.1.1      jpeg_0.1-9             colorspace_2.0-3
##  [33] blob_1.2.3             ggrepel_0.9.1          xfun_0.31              dplyr_1.0.9
##  [37] callr_3.7.0            crayon_1.5.2           RCurl_1.98-1.7         jsonlite_1.8.0
##  [41] survival_3.4-0         ape_5.6-2              glue_1.6.2             polyclip_1.10-0
##  [45] gtable_0.3.0           zlibbioc_1.42.0        XVector_0.38.0         R.cache_0.15.0
##  [49] pkgbuild_1.3.1         BiocGenerics_0.44.0    scales_1.2.0           qpdf_1.2.0
##  [53] DBI_1.1.2              Rcpp_1.0.8.3           htmlTable_2.4.0        viridisLite_0.4.0
##  [57] gridtext_0.1.4         tidytree_0.3.9         gridGraphics_0.5-1     foreign_0.8-82
##  [61] bit_4.0.4              Formula_1.2-4          stats4_4.2.1           rsvg_2.3.1
##  [65] pdftools_3.2.1         htmlwidgets_1.5.4      httr_1.4.3             RColorBrewer_1.1-3
##  [69] ellipsis_0.3.2         pkgconfig_2.0.3        XML_3.99-0.10          R.methodsS3_1.8.2
##  [73] farver_2.1.0           nnet_7.3-17            utf8_1.2.2             labeling_0.4.2
##  [77] ggplotify_0.1.0        tidyselect_1.2.0       rlang_1.0.6            munsell_0.5.0
##  [81] tools_4.2.1            cachem_1.0.6           cli_3.5.0              generics_0.1.2
##  [85] RSQLite_2.2.14         devtools_2.4.3         evaluate_0.15          stringr_1.4.0
##  [89] fastmap_1.1.0          yaml_2.3.5             ggtree_3.4.0           processx_3.6.0
##  [93] knitr_1.39             bit64_4.0.5            fs_1.5.2               tidygraph_1.2.1
##  [97] purrr_0.3.4            KEGGREST_1.36.2        ggraph_2.0.5           nlme_3.1-159
## [101] pbapply_1.5-0          R.oo_1.25.0            aplot_0.1.6            xml2_1.3.3
## [105] BiocStyle_2.24.0       brio_1.1.3             compiler_4.2.1         rstudioapi_0.13
## [109] png_0.1-7              classyfireR_0.3.8      treeio_1.22.0          tibble_3.1.7
## [113] tweenr_1.0.2           stringi_1.7.6          ggimage_0.3.1          highr_0.9
## [117] ps_1.7.0               desc_1.4.1             lattice_0.20-45        Matrix_1.5-1
## [121] styler_1.7.0           ggsci_2.9              vctrs_0.5.1            pillar_1.7.0
## [125] lifecycle_1.0.3        BiocManager_1.30.18    data.table_1.14.2      bitops_1.0-7
## [129] patchwork_1.1.1        latticeExtra_0.6-29    R6_2.5.1               bookdown_0.27
## [133] gridExtra_2.3          IRanges_2.32.0         sessioninfo_1.2.2      codetools_0.2-18
## [137] MASS_7.3-58            assertthat_0.2.1       pkgload_1.2.4          rprojroot_2.0.3
## [141] withr_2.5.0            S4Vectors_0.36.2       GenomeInfoDbData_1.2.9 parallel_4.2.1
## [145] ggtext_0.1.1           rpart_4.1.16           ggfun_0.0.6            tidyr_1.2.0
## [149] rmarkdown_2.14         ggforce_0.3.3          base64enc_0.1-3
```