nature protocols

https://doi.org/10.1038/s41596-024-01020-z

Protocol

Using clusterProfiler to characterize multiomics data

In the format provided by the authors and unedited

Supplementary Note 1:

Parameters for differential analysis in case study I

DA function:

```
DA <- function(expr,
               meta,
               abundance = 'Abundance',
               group_colname = 'Diagnosis',
               force = TRUE,
               relative = FALSE,
               subset_group,
               diff_group,
               filter.p = 'pvalue', ...) {
  sign_group_colname <- paste0('Sign_', group_colname)
  mpse <- MPSE(expr)
  mpse <- mpse |> left_join(meta, by = "Sample")
  mpse |>
    dplyr::filter(!!as.symbol(group_colname) %in% subset_group) |>
    mp_diff_analysis(
      .abundance = !!as.symbol(abundance),
      .group = !!as.symbol(group_colname),
      force = force,
      relative = relative,
      filter.p = filter.p,
```

```
    ...
  ) |>
  mp_extract_feature() |>
    dplyr::filter(!!as.symbol(sign_group_colname) == diff_group) |>
    dplyr::pull(OTU)
}
```

We defined the function DA, which is essentially a second-level encapsulated function, to enhance code conciseness. The main approach involves utilizing the feature abundance table and sample information to construct an MPSE object. Following this, the mp_diff_analysis function is applied to conduct a differential analysis, from which the differential features are subsequently extracted. The parameters of this function are as follows:

expr: Stores the abundance matrix of microbial genes or metabolites, where rows represent features and columns represent samples.

meta: Sample grouping information, comprising at least two columns. The first column is the sample identifier, corresponding to the column names in the expr parameter. The second column represents sample grouping.

group_colname: The column name in the meta parameter where sample grouping information is located.

force: Whether to use the original abundance matrix to perform differential analysis without rarefaction.

relative: Whether to calculate the relative abundance to perform differential analysis.

subset_group: A vector of length 2, representing the two group names for inter-group differential analysis – the treatment group and the control group.

diff_group: The name of the treatment group within subset_group.

filter.p: A string specifying the method for filtering p-values, with the default being fdr. This implies that features with fdr <= .first.test.alpha will be retained. If set to pvalue, features with p-value <= .first.test.alpha will be kept.

...: Additional parameters passed to the MicrobiotaProcess::mp_diff_analysis function.

Given that we use a second-level encapsulated function, the primary goal is differential analysis and filtering of significantly different features between different groups. For a comprehensive understanding of the function used in this instance, please refer to this link: https://www.bioconductor.org/packages/release/bioc/vignettes/MicrobiotaProcess/inst/doc/MicrobiotaProcess.html#biomarker-discovery

Supplementary Note 2

This is the source code of the procedures with syntax highlighted.

Procedure 1: Functional Enrichment Analyses in Metabolomic and Metagenomic Data

Download data**:**

```
url <- "https://yulab-smu.top/clusterProfiler_protocol/examples"

IBD_files <- c("mg.meta.csv", "mg.expr.csv",
 "metabolism_meta.csv", "metabolism_expr.csv")


for (f in IBD_files) {
```

```
 download.file(file.path(url, "IBD_2_subtypes", f), destfile =
f)

}
```

1| Load the R packages into the R environment:

```
library(MicrobiotaProcess)

library(clusterProfiler)

library(ggplot2)

library(enrichplot)
```

2| Import metagenomic data:

```
meta_mg <- read.csv("mg.meta.csv")

metagenome <- read.csv("mg.expr.csv",

 row.names = 1,

 check.name = FALSE)
```

3| Define a function to perform differential analysis:

```
DA <- function(expr,

 meta,

 abundance = 'Abundance',

 group_colname = 'Diagnosis',

 force = TRUE,

 relative = FALSE,

 subset_group,

 diff_group,

 filter.p = 'pvalue', ...) {
```

```
sign_group_colname <- paste0('Sign_', group_colname)
mpse <- MPSE(expr)
mpse <- mpse |> left_join(meta, by = "Sample")
mpse |>
dplyr::filter(!!as.symbol(group_colname) %in% subset_group) |>
mp_diff_analysis(
.abundance = !!as.symbol(abundance),
.group = !!as.symbol(group_colname),
force = force,
relative = relative,
filter.p = filter.p,
...
) |>
mp_extract_feature() |>
dplyr::filter(!!as.symbol(sign_group_colname) == diff_group)
|>
dplyr::pull(OTU) |> suppressMessages()
}
```

4| Identify differentially abundant microbial genes:

```
groups <- c(CD = 'CD', UC = 'UC')
de_gene <- lapply(groups, function(x) {
DA(expr = metagenome,
meta = meta_mg,
subset_group = c(x, 'Control'),
```

```
 diff_group = x)
})
```

5| Perform enrichment analysis of differential genes**:**

```
gene_enrich_result <- compareCluster(geneClusters = de_gene,
 fun = "enrichKEGG",
 organism = "ko")
```

6| Visualize the functional enrichment result of differential genes:

```
dotplot(gene_enrich_result, facet = 'intersect', showCategory =
 10,
 split = "intersect", label_format = 60) +
 ggtitle("Functional enrichment of intestinal genes") +
 theme(plot.title = element_text(hjust = 1))
```

7| Import metabolomic data

```
meta_mb <- read.csv("metabolism_meta.csv")
metabolism <- read.csv("metabolism_expr.csv",
 row.names = 1,
 check.name = FALSE)
```

8| Identify differentially abundance metabolites

```
groups <- c(CD = 'CD', UC = 'UC')
de_cpd <- lapply(groups, function(x) {
 DA(expr = metabolism,
 meta = meta_mb,
 subset_group = c(x, 'Control'),
```

```
 diff_group = x)
})
```

9| Perform enrichment analysis of differential metabolites

```
cpd_enrich_result <- compareCluster(geneClusters = de_cpd,
 fun = "enrichKEGG",
 organism = "cpd")
```

10| Visualize the functional enrichment results of differential metabolites (Figure 2B):

```
dotplot(cpd_enrich_result, facet = 'intersect', showCategory =
10,
 split = "intersect", label_format = 60) +
 ggtitle("Functional enrichment of chemical compounds") +
 theme(plot.title = element_text(hjust = 1))
```

Procedure 2: Transcription Factor Analysis Pertaining to Cold Tolerance in *Phyllostachys edulis* (*PE*)

Download data:

```
count_files <- c("counts.txt", "group_info.txt")
url <- "https://yulab-smu.top/clusterProfiler_protocol/examples
/Phyllostachys_heterocycla"

for (f in count_files) {
 download.file(file.path(url, f), destfile = f)
}
```

```
annot_files <- c("regulation_from_motif_CE_Phe.txt",
 "Phe_TF_list.txt", "Phe_GO_annotation.txt"
)
for (f in annot_files) {
 download.file(file.path(url, "annot_data", f), destfile = f)
}
```

1| Load the R packages into the R environment:

```
library(DESeq2)
library(clusterProfiler)
library(ggplot2)
library(enrichplot)
library(aplot)
library(ggfun)
```

2| Read the expression matrix and sample grouping information:

```
counts <- read.delim("counts.txt")

group_info <- read.delim("group_info.txt")

group_info$group <- factor(group_info$group,
  levels = c("0h", "2h", "24h", "168h")
)
```

3| Create a DESeqDataSet object:

```
count_dds <- DESeqDataSetFromMatrix(countData = counts,
 colData = group_info,
 design = ~ group)
```

4| Use DESeq to perform default differential expression analysis, including logarithmic fold changes that incorporate data-driven prior distributions:

```
count_dds <- DESeq(count_dds)
```

5| Define groups (different time points) to be compared with the baseline group (i.e., 0h):

```
time_points <- setNames(object = c("168h", "24h", "2h"),
 nm = c("168h_vs_0h", "24h_vs_0h", "2h_vs_0h"))
```

6| Extract log2FC values:

```
all_result <- lapply(time_points, function(time_point) {
 result <- results(count_dds, tidy = TRUE,
 contrast = c("group", time_point, "0h"))
 setNames(object = result$log2FoldChange, nm = result$row) |>
 sort(decreasing = TRUE)
})
```

7| Prepare transcription factor annotation data:

```
tf_db <- read.delim(
 "regulation_from_motif_CE_Phe.txt", header = FALSE,
 colClasses = c("character", "NULL", "character", rep("NULL",
4))
) |> setNames(c("TF", "targetGene"))
```

8| Use compareCluster to perform batch GSEA for identifying perturbed transcription factors from each time point:

```
perturbed_TF_result <- compareCluster(all_result, fun = "GSEA",
```

```
 pvalueCutoff = .05,
 TERM2GENE = tf_db,
 seed = 1234)
```

9| Visualize the enrichment result using dotplot:

```
perturbed_TF_plot <- dotplot(perturbed_TF_result,
 showCategory = 25) +
 aes(shape = I(22)) +
 coord_flip() +
 theme_minimal() +
 theme_noxaxis() +
 xlab(NULL) +
 set_enrichplot_color(
 c("#6C8FAD", "#84ADA7", "#C7B398"),
 .fun = ggplot2::scale_fill_gradientn
)
```

10 | Construct a named list of TF target genes and extract a subset of the most significant TFs that are related to cold responses:

```
tf_id <- unique(get_plot_data(perturbed_TF_plot, "ID")[, 1])
tf_genes <- split(tf_db$targetGene, tf_db$TF)[tf_id]
```

11| Import the GO annotations of PE genes:

```
go_db <- read.delim(file = "Phe_GO_annotation.txt")
```

12 | Characterize TF functions based on GO enrichment analysis of TF target genes:

```
TF_GO_result <- compareCluster(tf_genes, fun = "enricher",
 TERM2GENE = go_db[, c(2, 1)],
 TERM2NAME = go_db[, c(2, 3)])
```

13 | Visualize TF function enrichment result:

```
TF_GO_plot <- dotplot(TF_GO_result, by = "count",
 showCategory = 3,
 label_format = 40) +
 theme_minimal() +
 theme(axis.text.x = element_text(vjust = 1, hjust = 1,
 angle = 30, size = 10)) +
 xlab(NULL) + ggtitle(NULL)
```

14| Import TF family annotation information:

```
tf_family <- read.delim("Phe_TF_list.txt", row.names = 1)
```

15 | Prepare TF family data for visualization:

```
family_data <- subset(tf_family, Gene_ID %in% tf_id)
family_data <- family_data[order(family_data$Family), ]
family_data$Gene_ID <- factor(family_data$Gene_ID,
 levels = family_data$Gene_ID)
```

16 | Visualize the TF family information:

```
tf_family_plot <- ggplot(data = family_data,
 aes(x = Gene_ID, y = 1, fill = Family)) +
```

```
 geom_tile() +
 scale_fill_discrete(type=c('#B3D3AA', '#4B6B5C', '#E88A71',
 '#DEAB76', '#CD574D', '#85C1BF', '#BF38AE',
 '#176D84', '#7D83B7', '#4040C6', '#994B41')) +
 ggfun::theme_nothing()
```

17| Create a composite plot that integrates all the pieces of information (Figure 3):

```
insert_top(tf_family_plot, perturbed_TF_plot, height = 5) |>
 insert_bottom(TF_GO_plot, height = 50)
```

Procedure 3: Single-cell transcriptomic cell type annotation.

Download data:

```
url <- "https://yulab-smu.top/clusterProfiler_protocol/examples
/single_cell/"
dir.create("hg19")
pbmc_files <- c("barcodes.tsv", "genes.tsv", "matrix.mtx")
for (f in pbmc_files) {
 download.file(
 file.path(url, "filtered_gene_bc_matrices/hg19/", f),
 method = "auto",
 destfile = file.path("hg19", f)
 )
}
download.file(
```

```
 file.path(url, "cell_marker_db/c8.all.v2023.1.Hs.symbols.gmt
"),
 method = "auto",
 destfile = "c8.all.v2023.1.Hs.symbols.gmt"
)
```

1| Load the R packages into the R environment:

```
library(Seurat)
library(CelliD)
library(clusterProfiler)
library(ggplot2)
library(ggrepel)
library(ggsc)
```

2| Import example PBMC data:

```
pbmc_counts <- Read10X(data.dir = "hg19")
pbmc <- CreateSeuratObject(counts = pbmc_counts, project = "pbm
c3k",
 min.cells = 3)
```

3| Quality control by removing unwanted cells:

```
pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^
MT-")
pbmc <- subset(pbmc,
 subset = nFeature_RNA > 200 & nFeature_RNA < 2500 & percent.mt
 < 5
)
```

4| Normalize the data:

```
pbmc <- NormalizeData(pbmc, normalization.method = "LogNormaliz
e")

pbmc <- ScaleData(pbmc)
```

5| Identify highly variable genes:

```
pbmc <- FindVariableFeatures(pbmc, selection.method = "vst",

 nfeatures = 2000)
```

6| Perform dimensional reduction by typing:

```
pbmc <- RunPCA(pbmc, features = VariableFeatures(object = pbm
c))

pbmc <- RunUMAP(pbmc, dims = 1:10)
```

7| Construct a K-nearest neighbor (KNN) graph:

```
pbmc <- FindNeighbors(pbmc, dims = 1:10)
```

8| Cluster cells into different groups:

```
pbmc <- FindClusters(pbmc, resolution = 0.5)
```

9| Find cluster biomarkers

```
pbmc <- RunMCA(pbmc)

cluster_markers <- GetGroupGeneSet(X = pbmc, n.features = 20)
```

10| Import cell marker gene set:

```
cell_marker_db <- read.gmt("c8.all.v2023.1.Hs.symbols.gmt")
```

11| Perform cell type enrichment analysis by typing:

```
cell_type_enrich_result <- compareCluster(cluster_markers,
```

```
 fun = "enricher", TERM2GENE = cell_marker_db
)
```

12| Predict cell type:

```
predict_cell_type <- function(enrich_result) {
 enrich_result <- as.data.frame(enrich_result)
 result <- split(
 enrich_result, enrich_result$Cluster
 ) |>
 vapply(function(x) {
 x$ID[which.min(x$p.adjust)]
 }, FUN.VALUE = character(1))
 cell_type <- gsub("_", " ", result) |>
 yulab.utils::str_wrap(18)
 names(cell_type) <- names(result)
 return(cell_type)
}
cell_type_predict <- predict_cell_type(cell_type_enrich_result)
```

13| Assign cell type identity to clusters

```
pbmc <- RenameIdents(pbmc, cell_type_predict)
```

14| Visualize predicted cell type identity (Figure 4B):

```
cols <- c('#B3D3AA', '#E88A71', '#DEAB76', '#CD574D',
 '#BF38AE', '#176D84', '#7D83B7', '#4040C6', '#994B41')
sc_dim(pbmc) +
```

```
sc_dim_geom_label(geom = ggrepel::geom_text_repel,
color = "black", bg.color = "white") +
scale_color_discrete(type=cols) +
theme(legend.position = "none")
```