Lab 5: Multi-omics Integration using R/Bioconductor

Omics Data Science

Himel Mallick

Contents

Setup

Install the following R packages to get started.

```
library(curatedMetagenomicData)
library(dplyr)
library(tidyverse)
library(IntegratedLearner)
# devtools::install_github('himelmallick/IntegratedLearner')
library(SuperLearner)
options(java.parameters = "-Xmx8000m")  # Needed for BART
library(caret)
library(tidyverse)
library(cowplot)
library(mcmcplots)
library(bartMachine)
library(ROCR)
library(multiview)
library(forcats)
library(vegan)
library(MMUPHin)
library(multiview)
library(glmnet)
library(pROC)
theme_set(theme_bw())
```

Introduction - Multi-omics Integration

In multiview data analysis, there are two main types of approaches: early fusion and late fusion. Early fusion combines all datasets into a single representation, which then serves as input for a supervised learning model. In contrast, late fusion builds individual models for each data view and combines their predictions using a second-level model as the final predictor. However, these traditional paradigms treat the data views

in isolation and do not allow for interactions or dependencies between them. A more advanced method, called cooperative learning, which is also known as intermediate fusion, combines the best of both worlds by encouraging predictions from different data views to align through an agreement parameter (ρ).

Multi-omics Prediction and Classification of Binary IBD Disease Status

In this lab, we showcase examples of various integration paradigms (early, late, and intermediate fusion) using the R packages `multiview` and `IntegratedLearner`.

Input data

We use the publicly available Inflammatory Bowel Diseases (IBD) data from the curatedMetagenomicData package, where we aim to predict IBD disease status based on both taxonomic (species abundances) and functional (pathway abundances) profiles.

```
################## Load iHMP data #

se_relative <- sampleMetadata |>
    filter(study_name == "HMP_2019_ibdmdb") |>
    returnSamples("relative_abundance", rownames = "short")

se_pathway <- sampleMetadata |>
    filter(study_name == "HMP_2019_ibdmdb") |>
    returnSamples("pathway_abundance", rownames = "short")
```

We will first prepare the input sample metadata and feature table for both relative abundance and pathway abundance data.

```
########################## Create sample metadata #

sample_metadata <- select(as.data.frame(colData(se_relative)),
    c("study_name", "disease", "subject_id"))

# Define response variable & sample id
sample_metadata$Y <- ifelse(sample_metadata$disease == "IBD",
    1, 0)
sample_metadata <- sample_metadata %>%
    select(subject_id, Y) %>%
    rename(subjectID = subject_id)

########################### Create Species Features #

feature_species <- as.data.frame(assay(se_relative))
rownames(feature_species) <- sub(".*s__", "", rownames(feature_species))

########################### Create Pathway Features #

feature_pwys <- as.data.frame(assay(se_pathway))
feature_pwys <- rownames_to_column(feature_pwys, "ID")
feature_pwys <- feature_pwys %>%
    filter(!grepl("\\|", ID)) %>%
    filter(!ID %in% c("UNMAPPED", "UNINTEGRATED")) %>%
    column_to_rownames("ID") %>%
    as.data.frame()
```

```
################################ Combine Species and
################################ Pathways #

feature_table <- bind_rows(feature_species, feature_pwys)

############################################################ Check
############################################################ row
############################################################ names
############################################################ of
############################################################ feature_table
############################################################ and
############################################################ sample_metadata
############################################################ #

all(colnames(feature_table) == rownames(sample_metadata))
#> [1] TRUE
```

We will then create a metadata table for the features. This table captures high-level information related to the features (e.g., which layer they belong to).

```
#################################### Create metadata
#################################### table for features #

rowID <- rep(c("Species", "Pathways"), c(nrow(feature_species),
    nrow(feature_pwys)))
feature_metadata <- cbind.data.frame(featureID = rownames(feature_table),
    featureType = rowID)
rownames(feature_metadata) <- feature_metadata$featureID

################ Sanity check #

all(rownames(feature_table) == rownames(feature_metadata))  # TRUE
#> [1] TRUE
all(colnames(feature_table) == rownames(sample_metadata))  # TRUE
#> [1] TRUE
```

Further data pre-processing is necessary to handle near-zero-variance features in this dataset. Ultimately, 483 features are retained, consisting of 360 pathways and 123 species. A combination of variance and prevalence filtering is applied to the feature table, while related metadata is cleaned to ensure the retention of matching samples.

```
################ feature_table #

feature_table_t <- as.data.frame(t(feature_table))
abd_threshold = 0
prev_threshold = 0.1
nzv <- nearZeroVar(feature_table_t)
features_var_filtered <- feature_table_t[, -nzv]
features_var_filtered <- as.data.frame(features_var_filtered)
features_filtered <- features_var_filtered[, colSums(features_var_filtered >
    abd_threshold) > nrow(features_var_filtered) * prev_threshold]
feature_table <- as.data.frame(t(features_filtered))
```

```
#################### feature_metadata #

feature_metadata <- feature_metadata[rownames(feature_table),
    ]
table(feature_metadata$featureType)
#>
#> Pathways  Species
#>      358      124

###################### Sanity check again #

all(rownames(feature_table) == rownames(feature_metadata))  # TRUE
#> [1] TRUE
all(colnames(feature_table) == rownames(sample_metadata))  # TRUE
#> [1] TRUE
```

Following the preprocessing, we conduct additional input data preparation, which includes configuring parameters for sample splitting and 5-fold cross-validation. It's important to note that this dataset contains repeated measures (i.e., multiple samples per subject). To address this, we will conduct 5-fold cross-validation at the subject level. The source code is derived from the `IntegratedLearner` R package.

```
# Set parameters and extract subject IDs for sample
# splitting
seed <- 1
set.seed(seed)
subjectID <- unique(sample_metadata$subjectID)

################################### Trigger 5-fold CV (Outer
################################### Loop) #

subjectCvFoldsIN <- createFolds(1:length(subjectID), k = 5, returnTrain = TRUE)

######################################### Curate
######################################### subject-level
######################################### samples per fold
######################################### #

obsIndexIn <- vector("list", 5)
for (k in 1:length(obsIndexIn)) {
    x <- which(!sample_metadata$subjectID %in% subjectID[subjectCvFoldsIN[[k]]])
    obsIndexIn[[k]] <- x
}
names(obsIndexIn) <- sapply(1:5, function(x) paste(c("fold",
    x), collapse = ""))

################################ Set up data for SL training
################################ #

cvControl = list(V = 5, shuffle = FALSE, validRows = obsIndexIn)

################################################## Stacked
################################################## generalization
################################################## input
################################################## data
```

```
################################################## preparation
################################################## #

feature_metadata$featureType <- as.factor(feature_metadata$featureType)
name_layers <- with(droplevels(feature_metadata), list(levels = levels(featureType)),
    nlevels = nlevels(featureType))$levels
SL_fit_predictions <- vector("list", length(name_layers))
SL_fit_layers <- vector("list", length(name_layers))
names(SL_fit_layers) <- name_layers
names(SL_fit_predictions) <- name_layers
X_train_layers <- vector("list", length(name_layers))
names(X_train_layers) <- name_layers
layer_wise_predictions_train <- vector("list", length(name_layers))
names(layer_wise_predictions_train) <- name_layers
```

Following the preprocessing, we run the late fusion algorithm described in the `IntegratedLearner` paper.To this end, we subset the data for each feature type and conduct the analysis on each layer for the first-stage learning to predict the outcome. In other words, we fit a machine learning algorithm per layer (called a 'base learner'), utilizing the `SuperLearner` R package.

```
############################################################ Subset
############################################################ data
############################################################ per
############################################################ omics
############################################################ and
############################################################ run
############################################################ each
############################################################ individual
############################################################ layers
############################################################ #

for (i in seq_along(name_layers)) {

    # Prepare single-omic input data
    include_list <- feature_metadata %>%
        filter(featureType == name_layers[i])
    t_dat_slice <- feature_table[rownames(feature_table) %in%
        include_list$featureID, ]
    dat_slice <- as.data.frame(t(t_dat_slice))
    Y = sample_metadata$Y
    X = dat_slice
    X_train_layers[[i]] <- X

    # Run user-specified base learner
    SL_fit_layers[[i]] <- SuperLearner(Y = Y, X = X, cvControl = cvControl,
        SL.library = c("SL.randomForest"), family = binomial())

    # Append the corresponding y and X to the results
    SL_fit_layers[[i]]$Y <- sample_metadata["Y"]
    SL_fit_layers[[i]]$X <- X

    # Remove redundant data frames and collect pre-stack
    # predictions
    rm(t_dat_slice)
```

```
    rm(dat_slice)
    rm(X)
    SL_fit_predictions[[i]] <- SL_fit_layers[[i]]$Z

    # Re-fit to entire dataset for final predictions
    layer_wise_predictions_train[[i]] <- SL_fit_layers[[i]]$SL.predict

}
```

The next step of the analysis is to combine the layer-wise cross-validated predictions using a meta-model (meta_learner) to generate final predictions based on all available data points for the stacked model. Here, we will use `glmnet` as the meta-learner. However, other choices are also possible.

```
############################### Prepare stacked input data #

combo <- as.data.frame(do.call(cbind, SL_fit_predictions))
names(combo) <- name_layers

############################### Set aside final predictions
############################### #

combo_final <- as.data.frame(do.call(cbind, layer_wise_predictions_train))
names(combo_final) <- name_layers

#################### Stack all models #

# Run user-specified meta learner
SL_fit_stacked <- SuperLearner(Y = Y, X = combo, cvControl = cvControl,
    verbose = TRUE, SL.library = "SL.glmnet", family = binomial())

# Extract the fit object from superlearner
model_stacked <- SL_fit_stacked$fitLibrary[[1]]$object
stacked_prediction_train <- predict.SuperLearner(SL_fit_stacked,
    newdata = combo_final)$pred

# Append the corresponding y and X to the results
SL_fit_stacked$Y <- sample_metadata["Y"]
SL_fit_stacked$X <- combo
```

In contrast to late fusion, in early fusion, we will simply supply a combined representation of the data and we will use the `random forest` method for building an integrated prediction model.

```
################################### EARLY FUSION
################################### (CONCATENATED MODEL)
################################### #

fulldat <- as.data.frame(t(feature_table))

# Early Fusion using Random Forest
SL_fit_concat <- SuperLearner(Y = Y, X = fulldat, cvControl = cvControl,
    SL.library = "SL.randomForest", family = binomial())

# Extract the fit object from SuperLearner
model_concat <- SL_fit_concat$fitLibrary[[1]]$object
```

```
# Append the corresponding y and X to the results
SL_fit_concat$Y <- sample_metadata["Y"]
SL_fit_concat$X <- fulldat
```

Finally, we consider the intermediate fusion approach, which combines ideas from both late fusion and early fusion by integrating the usual squared-error loss of predictions with an "agreement" (fusion) penalty (ρ) so that the predictions from different data views agree.

The intermediate fusion adjusts the degree of fusion in an adaptive manner, where the test set prediction error is estimated with a cross-validation method. By varying the weight of the fusion penalty - hyperparameter ρ, we obtain the early and late fusion approaches as special cases. If $\rho = 0$, we have a simple form of early fusion; if $\rho = 1$, we obtain a simple form of late fusion. For $0 < \rho < 1$, we obtain the intermediate fusion.

Here, as examples, we will prepare the input data, fit the multiview model, and run the cross-validation in this section.

```
############################## Prepare data for multiview #

# Separate omics layers
feature_metadata$featureType <- as.factor(feature_metadata$featureType)
name_layers <- with(droplevels(feature_metadata), list(levels = levels(featureType)),
    nlevels = nlevels(featureType))$levels

# Define a list
dataList <- vector("list", length = length(name_layers))
names(dataList) <- name_layers
table(feature_metadata$featureType)
#>
#> Pathways  Species
#>      358      124
dataList[[1]] <- t(feature_table[1:123, ])
dataList[[2]] <- t(feature_table[124:nrow(feature_table), ])

# Extract y and X's
dataList <- lapply(dataList, as.matrix)
dataList <- lapply(dataList, scale)

######################### Run cross-validation #

set.seed(1234)
cvfit <- cv.multiview(dataList, Y, family = binomial(), alpha = 0.5)
DD <- as.data.frame(as.matrix(coef_ordered(cvfit, s = "lambda.min",
    alpha = 0.5)))
DD$standardized_coef <- as.numeric(DD$standardized_coef)
DD$coef <- as.numeric(DD$coef)
```

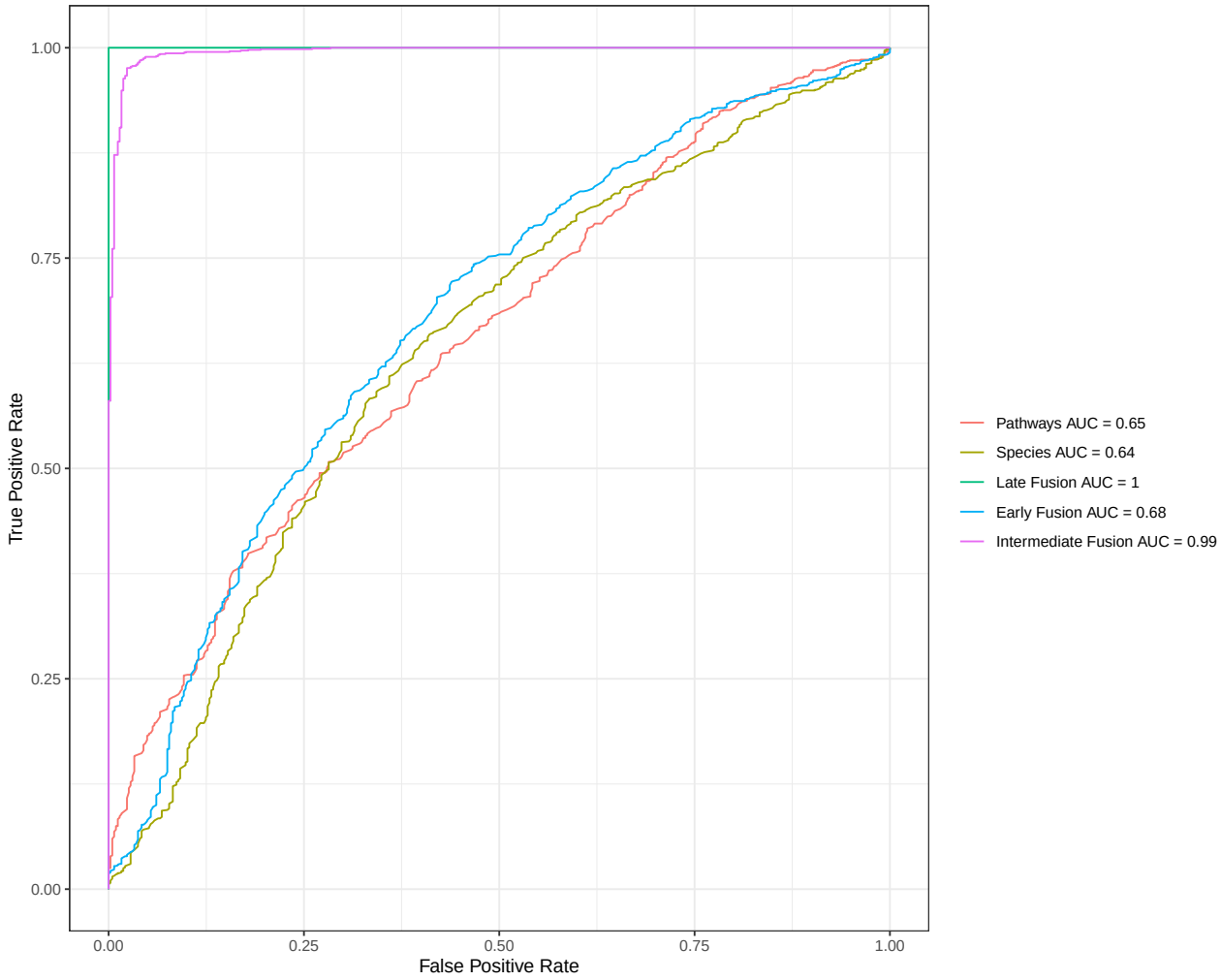
In this section, we visualize the prediction performance by gathering all predictions and extracting the data necessary for ROC plotting, corresponding to each integration paradigm.

```
############################################################ Gather
############################################################ all
############################################################ predictions
############################################################ and
############################################################ ROC
############################################################ curve
```

```
############################################################ data
############################################################ preparation
############################################################ #

coop_pred <- predict(cvfit, newx = dataList, s = "lambda.min",
    alpha = 0.5, type = "response")
yhat.train <- cbind(combo, stacked_prediction_train, SL_fit_concat$Z,
    coop_pred)
colnames(yhat.train) <- c(colnames(combo), "Late Fusion", "Early Fusion",
    "Intermediate Fusion")

# Extract ROC plot data
list.ROC <- vector("list", length = ncol(yhat.train))
names(list.ROC) <- colnames(yhat.train)

# Loop over layers
for (k in 1:length(list.ROC)) {
    preds <- yhat.train[, k]
    pred <- prediction(preds, Y)
    AUC <- round(performance(pred, "auc")@y.values[[1]], 2)
    perf <- performance(pred, "sens", "spec")
    list.ROC[[k]] <- data.frame(sensitivity = slot(perf, "y.values")[[1]],
        specificity = 1 - slot(perf, "x.values")[[1]], AUC = AUC,
        layer = names(list.ROC)[k])
}

# Combine
ROC_table <- do.call("rbind", list.ROC)
```

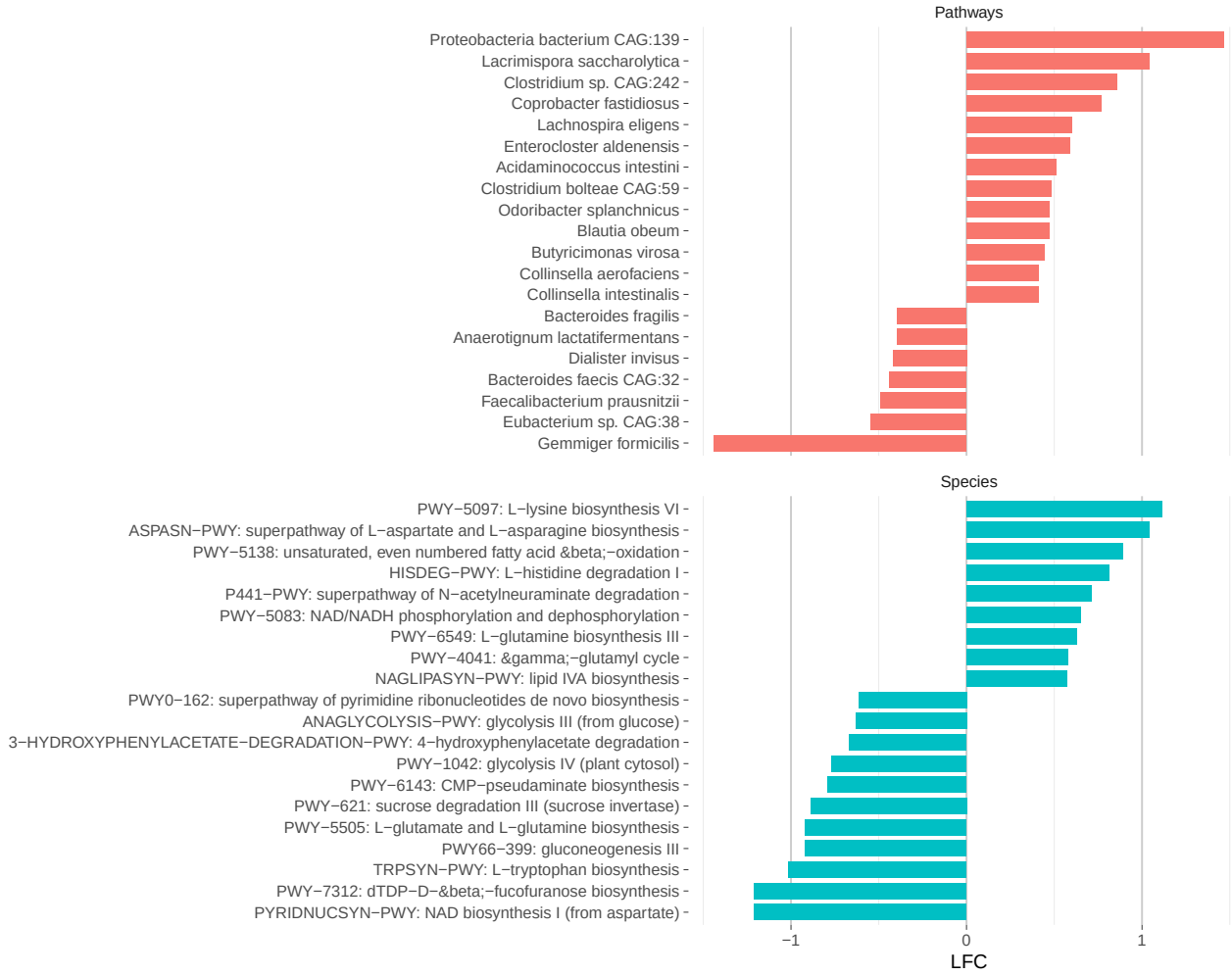
Based on the ROC plot described below, we observe that the AUC is 0.65 when considering only the pathway abundance data in the model, and 0.64 for the model including only the species abundance data. The AUC increases to 0.67 when using the early fusion model and reaches 1 with the late fusion model. In contrast, the intermediate fusion model achieves an AUC of 0.99. Overall, most integrated classifiers outperform individual layers in distinguishing between IBD and non-IBD controls.

```
# Prepare data for plotting
plot_data <- ROC_table
plot_data$displayItem <- paste(plot_data$layer, " AUC = ", plot_data$AUC,
    sep = "")
plot_data$displayItem <- factor(plot_data$displayItem, levels = unique(plot_data$displayItem))

# ROC curves
p <- ggplot(plot_data, aes(x = specificity, y = sensitivity,
    group = displayItem)) + geom_line(aes(x = specificity, y = sensitivity,
    color = displayItem)) + theme(legend.position = "bottom",
    legend.background = element_blank(), legend.box.background = element_rect(colour = "black")) +
    theme_bw() + xlab("False Positive Rate") + ylab("True Positive Rate") +
    theme(legend.position = "right", legend.direction = "vertical") +
    labs(color = "")

# Print
print(p)
```



Finally, we visualize the results for the top 20 features for each layer based on the intermediate fusion.

```
# Only plot UP TO TOP 20 PER LAYER
DD <- DD %>%
    group_by(view) %>%
    top_n(n = 20, wt = abs(standardized_coef))

# Visualization
p <- DD %>%
    mutate(view_col = fct_reorder(view_col, standardized_coef)) %>%
    ggplot(aes(x = view_col, y = standardized_coef, fill = view,
        width = 0.75)) + geom_bar(stat = "identity", show.legend = FALSE,
    width = 1) + coord_flip() + facet_wrap(~view, scales = "free_y",
    nrow = 2) + ylab("LFC") + xlab("") + ggtitle("") + theme_bw() +
    theme(strip.background = element_blank(), panel.grid.major = element_line(colour = "grey80"),
        panel.border = element_blank(), axis.ticks = element_line(size = 0),
        panel.grid.minor.y = element_blank(), panel.grid.major.y = element_blank())
p
```



Introduction to Meta-analysis (Horizontal Integration)

Meta-analysis is a statistical analysis that combines the results of multiple scientific studies by appropriately weighting the individual study-specific effect sizes. A common meta-analytic combination strategy is to take a weighted average of the study-specific summary measures. In fixed-effects meta-analysis, the weights are based on the assumption that there is a single true parameter underlying all the studies, while in random-effects meta-analysis, the weights are based on a model where the true parameter varies across studies according to a probability distribution.

Meta-analysis is particularly useful when individual studies are small, and a joint analysis of these studies can enhance the precision of effect size estimation and improve statistical power. Especially in microbiome studies, individual effect sizes from each study tend to be small. Only through meta-analysis can these small effects reach statistical significance. Therefore, by jointly analyzing these studies, we amplify the collective voice of the data, offering a more comprehensive understanding of the effects under investigation.

In this portion, our focus is on using the R/Bioconductor package `MMUPHin` (Meta-Analysis Methods with a Uniform Pipeline for Heterogeneity in microbiome studies) to perform the meta-analysis of microbiome data. We utilize publicly available cancer microbiome datasets from the R/Bioconductor package `curatedMetagenomicData`.

Meta-analysis of cancer microbiome studies using relative abundance data

Recent work has indicated a possible involvement of the gut microbiome in influencing the efficacy of immune checkpoint inhibitor (ICI) treatment strategies for PD-1/PD-L1 (Programmed Cell Death Protein 1/Programmed Cell Death Ligand 1) targeting checkpoint inhibitors. Several investigations into the microbiome have provided clinical data suggesting that the gut microbiome modulates the response to inhibitors of the PD-1/PD-L1 axis. However, published cancer microbiome studies face challenges, such as limited sample sizes, inconsistent data analysis methods, and a lack of functionally relevant consensus signatures.

Elucidation of the mechanisms by which the gut microbiome alters the function of the immune system to enable or promote cancer development may reveal novel pathways to explore in cancer therapy. Consequently, a high-quality re-analysis of public cancer microbiome data through a systematic meta-analysis approach could provide valuable insights into the microbiome's role in cancer development and progression in a rapid and cost-effective manner.

Using advanced melanoma as a model, extensively studied in the context of PD-1/PD-L1 immune checkpoint inhibitors, our aim is to identify functional biomarkers positively and negatively associated with ICI response by analyzing both taxonomic and functional profiles with ICI response.

In the following section, we provide detailed examples of how to perform batch effect correction and meta-analytic differential abundance testing on publicly available cancer microbiome data from multiple studies. We use the R package `MMUPHin` for these tasks. `MMUPHin` is a recently developed R/Bioconductor package designed for meta-analysis of microbiome taxonomic and functional profiles. It is agnostic to the data type and leverages another R/Bioconductor package, `MaAsLin2`, as a backend to conduct the meta-analysis. `MaAsLin2` is particularly designed to be applicable to various microbial community data types (taxonomy or functional profiles) and environments (human or otherwise). It is modular, including implementations of alternative normalization/transformation schemes and statistical models (e.g., for amplicon vs. shotgun metagenomic profiles). Leveraging this flexible framework under the hood, `MMUPHin` performs normalization, batch effect correction, and meta-analysis using the default random effects meta-regression.

Here, we will first perform the batch effect correction analysis. We use both the relative abundance data and the pathway abundance data from 5 public datasets. All these datasets are available from the `curatedMetagenomicData` package, with a total of 285 subjects included.

```
####################### Load melanoma data #

se_relative <- sampleMetadata |>
    filter(study_name %in% c("FrankelAE_2017", "GopalakrishnanV_2018",
        "LeeKA_2022", "MatsonV_2018", "PetersBA_2019", "WindTT_2020")) |>
    returnSamples("relative_abundance", rownames = "short")
```

First, let's look into the relative abundance data.

Performing batch (study) effect adjustment with adjust_batch for relative abundance

In this analysis, we are using two input files. The first input file is the relative abundance data file, which consists of a feature-by-sample matrix. The second input is the metadata associated with the relative abundance file, which includes the study names and combines overall response rate (ORR), progression-free survival at 12 months (PFS12), as well as the response to anti-PD-1 (anti_PD_1) as the response variable. By the end of this step, we obtain a batch-adjusted relative abundance matrix.

```
######################### Create sample metadata #

data_meta <- select(as.data.frame(colData(se_relative)), c("study_name",
    "ORR", "PFS12", "anti_PD_1"))

# Define response variable
```

```
data_meta$resvar <- NA
data_meta$anti_PD_1[data_meta$anti_PD_1 == "responder"] <- "yes"
data_meta$anti_PD_1[data_meta$anti_PD_1 == "non_responder"] <- "no"
data_meta$resvar[!is.na(data_meta$anti_PD_1)] <- data_meta$anti_PD_1[!is.na(data_meta$anti_PD_1)]
data_meta$resvar[!is.na(data_meta$PFS12)] <- data_meta$PFS12[!is.na(data_meta$PFS12)]
data_meta$resvar[!is.na(data_meta$ORR)] <- data_meta$ORR[!is.na(data_meta$ORR)]

# Filter individuals without response variable
data_meta <- data_meta[!is.na(data_meta$resvar), c("study_name",
    "resvar")]

# Convert the 'study_name' to factor variable
data_meta$study_name <- as.factor(data_meta$study_name)

############################ Create Species Features #
############################ Transpose the abundance matrix
############################ and change the value of
############################ abundance data to proportion
############################ unit
data_abd <- assay(se_relative)
data_abd <- data_abd/100

# Match the individuals in the data_abd
data_abd <- data_abd[, colnames(data_abd) %in% rownames(data_meta)]

# Change the rownames names for variables of interest
rownames(data_abd) <- sub(".*s__", "", rownames(data_abd))

# Use adjust_batch to correct for differences in the five
# studies, while controlling for the effect of cases versus
# control (variable resvar in data_meta).
fit_adjust_batch <- adjust_batch(feature_abd = data_abd, batch = "study_name",
    covariates = "resvar", data = data_meta, control = list(verbose = FALSE))

data_abd_adj <- fit_adjust_batch$feature_abd_adj
```

Evaluation for batch effect adjustment

After obtaining the adjusted abundance matrix, we can evaluate the improvement of the batch effect adjustment. Here, the total variation from study difference will be assessed through the PERMANOVA[1]4 test.

```
D_before <- vegdist(t(data_abd))
D_after <- vegdist(t(data_abd_adj))

set.seed(1)
fit_adonis_before <- adonis(D_before ~ study_name, data = data_meta)
fit_adonis_after <- adonis(D_after ~ study_name, data = data_meta)

print(fit_adonis_before$aov.tab)
#> Permutation: free
#> Number of permutations: 999
#>
```

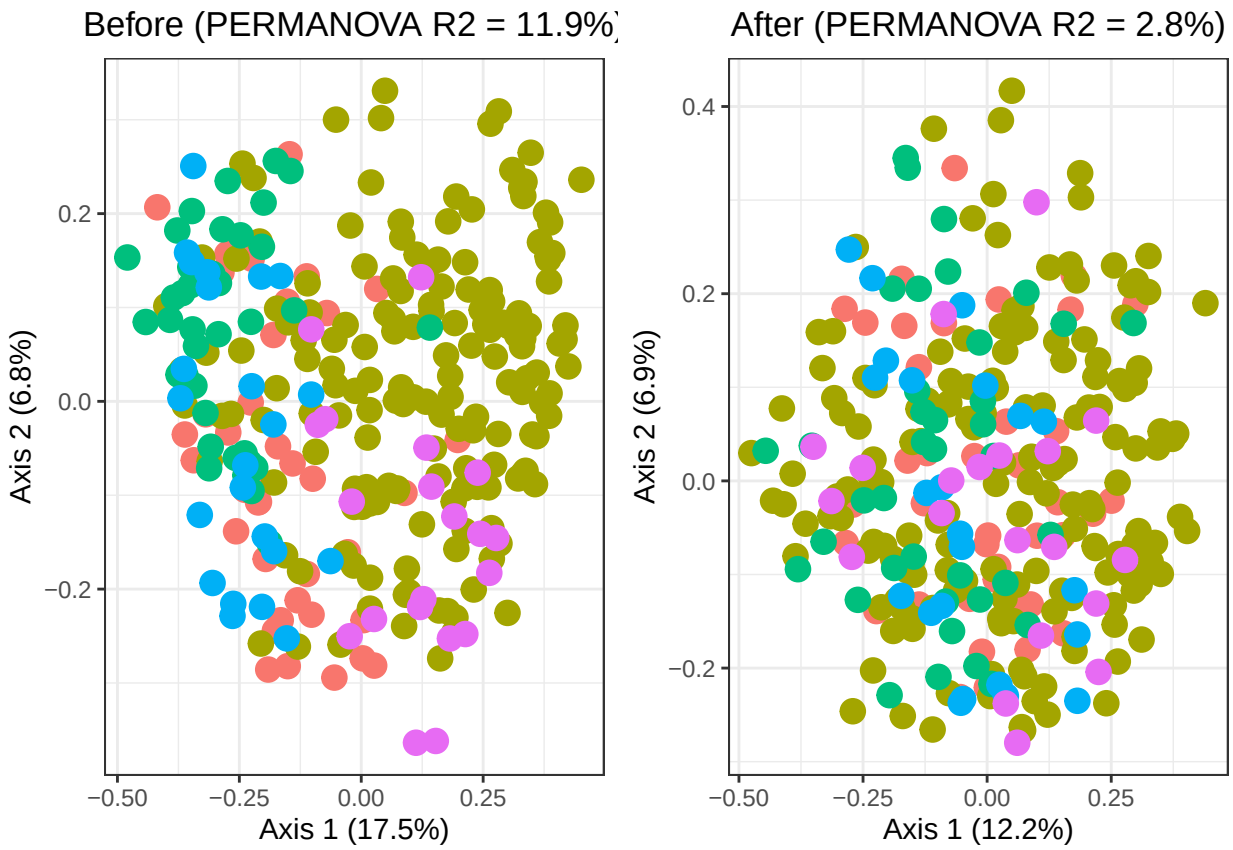
```
#> Terms added sequentially (first to last)
#>
#>             Df SumsOfSqs MeanSqs F.Model      R2 Pr(>F)
#> study_name   4    11.069 2.76725  9.4978 0.11947  0.001 ***
#> Residuals  280    81.580 0.29136         0.88053
#> Total      284    92.649                 1.00000
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
print(fit_adonis_after$aov.tab)
#> Permutation: free
#> Number of permutations: 999
#>
#> Terms added sequentially (first to last)
#>
#>             Df SumsOfSqs MeanSqs F.Model      R2 Pr(>F)
#> study_name   4     2.518 0.62956  2.0171 0.02801  0.001 ***
#> Residuals  280    87.389 0.31210         0.97199
#> Total      284    89.907                 1.00000
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Based on the results, we can see that the study differences can explain a total of 11.922% (R^2) of the variability in microbial abundance profiles before study effect adjustment, whereas after adjustment this was reduced to 2.806% (R^2).

Let's visualize the results of the PERMANOVA test.

```
############### Ordination # Before
R2_before <- round(fit_adonis_before$aov.tab[1, 5] * 100, 1)
pcoa_before <- cmdscale(D_before, eig = TRUE)
ord_before <- as.data.frame(pcoa_before$points)
percent_var_before <- round(pcoa_before$eig/sum(pcoa_before$eig) *
    100, 1)[1:2]
before_labels <- c(paste("Axis 1 (", percent_var_before[1], "%)",
    sep = ""), paste("Axis 2 (", percent_var_before[2], "%)",
    sep = ""))

before_phrase <- paste("Before (PERMANOVA R2 = ", R2_before,
    "%)", sep = "")
colnames(ord_before) <- c("PC1", "PC2")
ord_before$Study <- data_meta$study_name

# After
R2_after <- round(fit_adonis_after$aov.tab[1, 5] * 100, 1)
pcoa_after <- cmdscale(D_after, eig = TRUE)
ord_after <- as.data.frame(pcoa_after$points)
percent_var_after <- round(pcoa_after$eig/sum(pcoa_after$eig) *
    100, 1)[1:2]
after_labels <- c(paste("Axis 1 (", percent_var_after[1], "%)",
    sep = ""), paste("Axis 2 (", percent_var_after[2], "%)",
    sep = ""))

after_phrase <- paste("After (PERMANOVA R2 = ", R2_after, "%)",
    sep = "")
```

```
colnames(ord_after) <- c("PC1", "PC2")
ord_after$Study <- data_meta$study_name

# Ordination Plot
p_before <- ggplot(ord_before, aes(x = PC1, y = PC2, color = Study)) +
    geom_point(size = 4) + theme_bw() + xlab(before_labels[1]) +
    ylab(before_labels[2]) + ggtitle(before_phrase) + theme(plot.title = element_text(hjust = 0.5)) +
    theme(legend.position = "none")

p_after <- ggplot(ord_after, aes(x = PC1, y = PC2, color = Study)) +
    geom_point(size = 4) + theme_bw() + xlab(after_labels[1]) +
    ylab(after_labels[2]) + ggtitle(after_phrase) + theme(plot.title = element_text(hjust = 0.5)) +
    theme(legend.position = "none")

p <- plot_grid(p_before, p_after, ncol = 2)
p
```

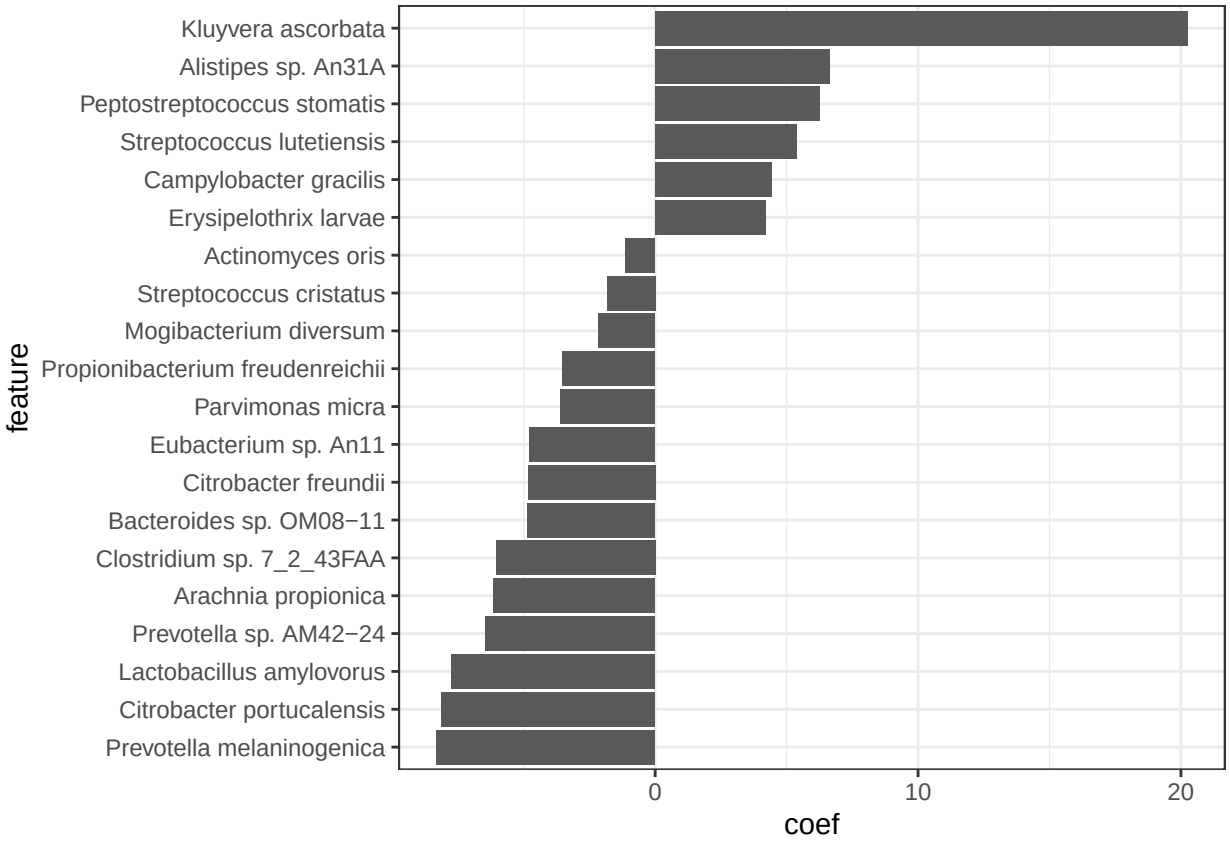


Meta-analytical differential abundance testing with lm_meta

Here, we illustrate the details of the meta-analytical differential abundance testing. We apply the Tweedie model (using the "CPLM" analysis method in `MaAsLin2`, also implemented in the R package `Tweedieverse`.

```
fit_lm_meta <- lm_meta(feature_abd = data_abd_adj, batch = "study_name",
    exposure = "resvar", data = data_meta, control = list(analysis_method = "CPLM",
        transform = "NONE"))

meta_fits <- fit_lm_meta$meta_fits
```

```
meta_fits %>%
    filter(qval.fdr < 0.05) %>%
    arrange(coef) %>%
    mutate(feature = factor(feature, levels = feature)) %>%
    ggplot(aes(y = coef, x = feature)) + geom_bar(stat = "identity") +
    coord_flip()
```



Based on the results, we see that there are 22 significant features in total, among which the *kluyvera ascorbata* has the strongest positive effect, while the *pseudomonas putida* species has the strongest negative effect.

Homework 5

- Note that, `IntegratedLearner` implements the a variety of algorithms based on both early and late fusion. Follow the `IntegratedLearner` tutorial to implement and compare the following algorithms (both early and late) in terms of AUC and plot the results. To reduce computational time, you can run these algorithms on a subset of features per layer. Make sure you add `options(java.parameters = "-Xmx8000m")` to your code.

Base learner	Meta learner
Random Forest	NNLS
glmnet	NNLS
BART	NNLS
KSVM	NNLS
XGBOOST	NNLS
Random Forest	Random Forest
glmnet	glmnet

Base learner	Meta learner
BART	BART
KSVM	KSVM
XGBOOST	XGBOOST

- Using the functions in this repository, re-run the intermediate fusion algorithm described above so that it selects the *rho* and *alpha* parameter from a grid of values. Compare the results with the early and late fusion approaches from Question 1. Choose `alpha = 0, rho = c(0, 0.5, 1)`. To reduce computational time, you can run these algorithms on a subset of features per layer.
- Repeat the meta-analysis above (same studies) on the HMP2 pathway abundances.

Session information

```
sessionInfo()
#> R version 4.4.2 (2024-10-31)
#> Platform: aarch64-apple-darwin20
#> Running under: macOS Sequoia 15.2
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> time zone: Asia/Kolkata
#> tzcode source: internal
#>
#> attached base packages:
#> [1] splines   stats4    stats     graphics  grDevices utils     datasets
#> [8] methods   base
#>
#> other attached packages:
#>  [1] pROC_1.18.5                    glmnet_4.1-8
#>  [3] Matrix_1.7-1                   MMUPHin_1.18.1
#>  [5] vegan_2.6-8                    permute_0.9-7
#>  [7] multiview_0.8                  ROCR_1.0-11
#>  [9] bartMachine_1.3.4.1            missForest_1.5
#> [11] randomForest_4.7-1.2           bartMachineJARs_1.2.1
#> [13] rJava_1.0-11                   mcmcplots_0.4.3
#> [15] coda_0.19-4.1                  cowplot_1.1.3
#> [17] caret_7.0-1                    lattice_0.22-6
#> [19] SuperLearner_2.0-29            gam_1.22-5
#> [21] foreach_1.5.2                  nnls_1.6
#> [23] IntegratedLearner_0.0.1        lubridate_1.9.4
#> [25] forcats_1.0.0                  stringr_1.5.1
#> [27] purrr_1.0.2                    readr_2.1.5
#> [29] tidyr_1.3.1                    tibble_3.2.1
#> [31] ggplot2_3.5.1                  tidyverse_2.0.0
#> [33] dplyr_1.1.4                    curatedMetagenomicData_3.12.0
#> [35] TreeSummarizedExperiment_2.12.0 Biostrings_2.72.1
#> [37] XVector_0.44.0                 SingleCellExperiment_1.26.0
```

```
#> [39] SummarizedExperiment_1.34.0     Biobase_2.64.0
#> [41] GenomicRanges_1.56.2            GenomeInfoDb_1.40.1
#> [43] IRanges_2.38.1                  S4Vectors_0.42.1
#> [45] BiocGenerics_0.50.0             MatrixGenerics_1.16.0
#> [47] matrixStats_1.4.1               knitr_1.49
#>
#> loaded via a namespace (and not attached):
#>   [1] fs_1.6.5                     DirichletMultinomial_1.46.0
#>   [3] httr_1.4.7                   RColorBrewer_1.1-3
#>   [5] numDeriv_2016.8-1.1          tools_4.4.2
#>   [7] doRNG_1.8.6                  metafor_4.6-0
#>   [9] R6_2.5.1                     lazyeval_0.2.2
#>  [11] mgcv_1.9-1                   withr_3.0.2
#>  [13] gridExtra_2.3                cli_3.6.3
#>  [15] textshaping_0.4.1            formatR_1.14
#>  [17] logging_0.10-108             denstrip_1.5.4
#>  [19] biglm_0.9-3                  labeling_0.4.3
#>  [21] mvtnorm_1.3-2                robustbase_0.99-4-1
#>  [23] pbapply_1.7-2                systemfonts_1.1.0
#>  [25] yulab.utils_0.1.8            scater_1.32.1
#>  [27] decontam_1.24.0              parallelly_1.41.0
#>  [29] itertools_0.1-3              rstudioapi_0.17.1
#>  [31] RSQLite_2.3.9                generics_0.1.3
#>  [33] shape_1.4.6.1                metadat_1.2-0
#>  [35] ggbeeswarm_0.7.2             DECIPHER_3.0.0
#>  [37] abind_1.4-8                  lifecycle_1.0.4
#>  [39] yaml_2.3.10                  mathjaxr_1.6-0
#>  [41] recipes_1.1.0                SparseArray_1.4.8
#>  [43] BiocFileCache_2.12.0         grid_4.4.2
#>  [45] blob_1.2.4                   ExperimentHub_2.12.0
#>  [47] crayon_1.5.3                 beachmat_2.20.0
#>  [49] KEGGREST_1.44.1              pillar_1.10.0
#>  [51] optparse_1.7.5               future.apply_1.11.3
#>  [53] codetools_0.2-20             glue_1.8.0
#>  [55] data.table_1.16.4            MultiAssayExperiment_1.30.3
#>  [57] vctrs_0.6.5                  png_0.1-8
#>  [59] treeio_1.28.0                gtable_0.3.6
#>  [61] cachem_1.1.0                 gower_1.0.2
#>  [63] xfun_0.49                    S4Arrays_1.4.1
#>  [65] mime_0.12                    prodlim_2024.06.25
#>  [67] pcaPP_2.0-5                  survival_3.8-3
#>  [69] timeDate_4041.110            iterators_1.0.14
#>  [71] tinytex_0.54                 hardhat_1.4.0
#>  [73] lava_1.8.0                   statmod_1.5.0
#>  [75] bluster_1.14.0               cplm_0.7-12.1
#>  [77] ipred_0.9-15                 nlme_3.1-166
#>  [79] bit64_4.5.2                  filelock_1.0.3
#>  [81] irlba_2.3.5.1                vipor_0.4.7
#>  [83] rpart_4.1.23                 colorspace_2.1-1
#>  [85] DBI_1.2.3                    nnet_7.3-19
#>  [87] tidyselect_1.2.1             bit_4.5.0.1
#>  [89] compiler_4.4.2               curl_6.0.1
#>  [91] BiocNeighbors_1.22.0         DelayedArray_0.30.1
```

```
#>  [93] scales_1.3.0                sfsmisc_1.1-20
#>  [95] DEoptimR_1.1-3-1            rappdirs_0.3.3
#>  [97] digest_0.6.37               minqa_1.2.8
#>  [99] rmarkdown_2.29              htmltools_0.5.8.1
#> [101] pkgconfig_2.0.3             sparseMatrixStats_1.16.0
#> [103] dbplyr_2.5.0                fastmap_1.2.0
#> [105] rlang_1.1.4                 UCSC.utils_1.0.0
#> [107] DelayedMatrixStats_1.26.0   farver_2.1.2
#> [109] jsonlite_1.8.9              BiocParallel_1.38.0
#> [111] ModelMetrics_1.2.2.2        BiocSingular_1.20.0
#> [113] magrittr_2.0.3              scuttle_1.14.0
#> [115] GenomeInfoDbData_1.2.12     munsell_0.5.1
#> [117] Rcpp_1.0.13-1               ape_5.8-1
#> [119] viridis_0.6.5               stringi_1.8.4
#> [121] Maaslin2_1.18.0             zlibbioc_1.50.0
#> [123] MASS_7.3-61                 AnnotationHub_3.12.0
#> [125] plyr_1.8.9                  parallel_4.4.2
#> [127] listenv_0.9.1               ggrepel_0.9.6
#> [129] hash_2.2.6.3                hms_1.1.3
#> [131] igraph_2.1.2                rngtools_1.5.2
#> [133] reshape2_1.4.4              ScaledMatrix_1.12.0
#> [135] BiocVersion_3.19.1          evaluate_1.0.1
#> [137] BiocManager_1.30.25         tzdb_0.4.0
#> [139] getopt_1.20.4               future_1.34.0
#> [141] rsvd_1.0.5                  tweedie_2.3.5
#> [143] tidytree_0.4.6              viridisLite_0.4.2
#> [145] class_7.3-22                ragg_1.3.3
#> [147] memoise_2.0.1               beeswarm_0.4.0
#> [149] AnnotationDbi_1.66.0        cluster_2.1.8
#> [151] timechange_0.3.0            globals_0.16.3
#> [153] mia_1.12.0
```