Introduction to multi-omics data analysis

University of Turku

2022-01-14

Contents

Chapter 1

Overview

Welcome to the multi-omics data analysis workshop

Figure source: Moreno-Indias *et al.* (2021) Statistical and Machine Learning Techniques in Human Microbiome Studies: Contemporary Challenges and Solutions. Frontiers in Microbiology 12:11.

1.1 Introduction

This course is based on *miaverse* (mia = **MI**crobiome **A**nalysis) is an R/Bioconductor framework for microbiome data science. It extends another popular framework, phyloseq.

The miaverse consists of an efficient data structure, an associated package ecosystem, demonstration data sets, and open documentation. These are explained in more detail in the online book Orchestrating Microbiome Analysis.

This workshop material walks you through example workflows for multi-omics data analysis covering data access, exploration, analysis, visualization and reproducible reporting. **You can run the workflow by simply copy-pasting the examples.** For advanced material, you can test and modify further examples from the OMA book, or try to apply the techniques to your own data.

1.2 Learning goals

This workshop provides an overview of analytical tools for multi-omics studies in R. A particular focus is on multi-omics tools and techniques required to process microbial community data in combination with other omics.

After the workshop the participants should be able to preprocess and manipulate data, perform simple visualizations and statistical analyses, apply unsupervised and supervised machine learning, and produce robust and reproducible results.

Target audience Advanced students and applied researchers who wish to develop their skills in multi-omics analysis.

Venue The course is organized fully remotely in Zoom. The meeting requires a passcode (sent by email to the participants).

1.3 Acknowledgments

Citation "Introduction to microbiome data science (2021). URL: https://microbiome.github.io".

Borman et al. (2022)

We thank Felix Ernst, Sudarshan Shetty, and other miaverse developers who have contributed open resources that supported the development of the training material.

Contact Leo Lahti, University of Turku, Finland

Source code

The source code of this repository is fully reproducible and contains the Rmd files with executable code. All files can be rendered at one go by running the file main.R. You can check the file for details on how to clone the repository and convert it into a gitbook, although this is not necessary for the training.

- Landing page (html): workshop teaching material
- Source code (github): workshop teaching material

Chapter 2

Program

The workshop takes place on the 13th and 14th of January from 9am – 5pm (CET). Short breaks will be scheduled between sessions.

The practical sessions consists of a set of example multi-omics analysis workflows. It is assumed that you have already installed the required software. Do not hesitate to ask support from the course assistants.

2.1 Day 1

Lectures

- 9:15-10:00 - Welcome and introduction - Leo Lahti, Associate professor (UTU)
- 10:00-10:15 - Break
- 10:15-11:00 - Metagenomics - Katariina Pärnänen, Postdoctoral researcher (UTU)
- 11:00-11:15 - Break
- 11:15-12:00 - Metabolomics - Pande Putu Erawijantari, Postdoctoral researcher (UTU)
- 12:00-12:15 - Break
- 12:15-13:00 - Multiomics - Leo Lahti, Associate professor (UTU)
- 13:00-14:15 - Lunch break

Practical

- 14:15-17:00 - Tuomas Borman and Chouaib Benchraka, Research assistants (UTU)
- Topics:
  - Data import and data structures
  - Microbiome data exploration
  - Visualization

- R Code:
  - Installation
  - Day 1

2.2 Day 2

Lectures

- 9:15-11:00 (including a short break) - Unsupervised and supervised machine learning - Matti Ruuskanen, Postdoctoral researcher (UTU)
- 11:00-11:15 - Break
- 11:15-12:00 - Individual-based modeling - Gergely Boza, Research fellow (CER)
- 12:00-12:15 - Break
- 12:15-13:00 - Data integration - Leo Lahti, Associate professor (UTU)
- 13:00-14:15 - Lunch break

Practical

- 14:15-17:00 - Tuomas Borman, Matti Ruuskanen and Chouaib Benchraka (UTU)
- Topics:
  - Unsupervised learning: Beta-diversity and biclustering
  - Supervised learning: Regression and classification with random forests
  - Validation and interpretation of black box models
- R Code:
  - Installation
  - Day 2

Chapter 3

Getting started

3.1 Checklist (before the workshop)

Install the following software in advance in order to avoid unnecessary delays and leaving more time for the workshop contents.

- R (version >4.1.0)
- RStudio; choose "Rstudio Desktop" to download the latest version. Optional but preferred. For further details, check the Rstudio home page.
- For Windows users: Rtools; Follow the instructions to install the toolkit. This might be required to compile some of the packages required on this course.
- Install and load the required R packages

3.2 Support and resources

For online support on installation and other matters, you can join us at:

- Users: miaverse Gitter channel
- Developers: Bioconductor Slack #microbiomeexperiment channel (ask for an invitation)

3.3 Installing and loading the required R packages

This section shows how to install and load all required packages into the R session. Only uninstalled packages are installed.

```
if (!requireNamespace("BiocManager")) {
  install.packages("BiocManager")
}
```

```
# List of packages that we need from cran and bioc
packages <- c("ggplot2", "pheatmap", "stringr", "igraph", "ANCOMBC",
              "microbiome", "httpuv", "microbiomeDataSets", "mia", "caret", "ranger",
             "dplyr", "miaViz", "knitr", "kableExtra", "vegan", "ecodist", "biclust",
             "patchwork", "pdp", "MLmetrics", "precrec")
```

The following script tries to load all required packages, and if they are not available, installs them.

```
new.pkgs <- packages[!(packages %in% installed.packages()[, "Package"])]

if (length(new.pkgs)) {
    BiocManager::install(new.pkgs, ask=T)
}

sapply(packages, require, character.only = TRUE)
```

```
##            ggplot2           pheatmap            stringr             igraph
##               TRUE               TRUE               TRUE               TRUE
##            ANCOMBC         microbiome              httpuv microbiomeDataSets
##               TRUE               TRUE               TRUE               TRUE
##                mia              caret             ranger              dplyr
##               TRUE               TRUE               TRUE               TRUE
##             miaViz              knitr         kableExtra              vegan
##               TRUE               TRUE               TRUE               TRUE
##            ecodist            biclust          patchwork                pdp
##               TRUE               TRUE               TRUE               TRUE
##          MLmetrics            precrec
##               TRUE               TRUE
```

Chapter 4

Data

This section demonstrates how to import data in R.

4.1 Data structure

Such analysis using the miaverse framework, are based upon core data structures including SingleCellExperiment (SCE), SummarizedExperiment (SE), TreeSummarizedExperiment (TreeSE) and MultiAssayExperiment (MAE) (resources).

Multi-assay data can be stored in altExp slot of TreeSE or MAE data container.

Different data sets are first imported into SE or TreeSE data container similarly to the case when only one data set is present. After that different data sets are combined into the same data container. Result is one TreeSE object with alternative experiment in altExp slot, or MAE object with multiple experiment in its experiment slot.

4.2 Example data

As an example data, we use data from following publication: Hintikka L et al. (2021) Xylo-oligosaccharides in prevention of hepatic steatosis and adipose tissue inflammation: associating taxonomic and metabolomic patterns in fecal microbiotas with biclustering.

This example data can be loaded from microbiomeDataSets. The data is already in MAE format. It includes three different experiments: microbial abundance data, metabolite concentrations, and data about different biomarkers.

4.3 Importing data in R

```
library(stringr)

# Load the data
```

```
mae <- microbiomeDataSets::HintikkaXOData()

# Drop off those bacteria that do not include information in Phylum or lower levels
mae[[1]] <- mae[[1]][!is.na(rowData(mae[[1]])$Phylum), ]

# Clean taxonomy data, so that names do not include additional characters
rowData(mae[[1]]) <- DataFrame(apply(rowData(mae[[1]]), 2,
                                     str_remove, pattern = "._[0-9]__"))

mae
```

```
## A MultiAssayExperiment object of 3 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 3:
##  [1] microbiota: SummarizedExperiment with 12613 rows and 40 columns
##  [2] metabolites: SummarizedExperiment with 38 rows and 40 columns
##  [3] biomarkers: SummarizedExperiment with 39 rows and 40 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

Chapter 5

Microbiome data exploration

Now we have loaded the data set into R. Next, let us walk through some basic operations for data exploration to confirm that the data has all the necessary components.

5.1 Data structure

Let us now investigate how taxonomic profiling data is organized in R.

Dimensionality tells us how many taxa and samples the data contains. As we can see, there are 12613 taxa and 40 samples.

```
# mae[[1]]: indexing/retrieving the taxonomic data experiment
dim(mae[[1]])
```

```
## [1] 12613    40
```

The `rowData` slot contains a taxonomic table. This includes taxonomic information for each of the 12613 entries. With the `head()` command, we can print just the beginning of the table.

The `rowData` seems to contain information from 7 different taxonomy classes.

```
knitr::kable(head(rowData(mae[[1]]))) %>%
  kableExtra::kable_styling("striped",
                            latex_options="scale_down") %>%
  kableExtra::scroll_box(width = "100%")
```

The colData slot contains sample metadata. It contains information for all 40 samples. However, here only the 6 first samples are shown as we use the `head()` command. There are 6 columns, that contain information, e.g., about patients' status, and cohort.

	Phylum	Class	Order	Family	Genus	Species	OTU
GAYR01026362.62.2014	Proteobacteria	Alphaproteobacteria	Rickettsiales	Mitochondria	Solanum melongena (eggplant)	Solanum melongena (eggplant)	GAYR01026362.62.2014
CVJT01000011.50.2173	Firmicutes	Bacilli	Bacillales	Staphylococcaceae	Staphylococcus	Staphylococcus aureus	CVJT01000011.50.2173
KF625183.1.1786	Proteobacteria	Gammaproteobacteria	Enterobacteriales	Enterobacteriaceae	Klebsiella	Klebsiella oxytoca	KF625183.1.1786
AYSG01000002.292.2076	Firmicutes	Bacilli	Lactobacillales	Streptococcaceae	Streptococcus	Streptococcus thermophilus TH1435	AYSG01000002.292.2076
CCPS01000022.154.1916	Proteobacteria	Gammaproteobacteria	Enterobacteriales	Enterobacteriaceae	Escherichia-Shigella	Escherichia coli	CCPS01000022.154.1916
KJ923794.1.1762	Firmicutes	Bacilli	Bacillales	Staphylococcaceae	Staphylococcus	Staphylococcus aureus	KJ923794.1.1762

	Sample	Rat	Site	Diet	Fat	XOS
C1	C1	1	Cecum	High-fat	High	0
C2	C2	2	Cecum	High-fat	High	0
C3	C3	3	Cecum	High-fat	High	0
C4	C4	4	Cecum	High-fat	High	0
C5	C5	5	Cecum	High-fat	High	0
C6	C6	6	Cecum	High-fat	High	0

```
# For simplicity, classify all high-fat diets as high-fat, and all the low-fat
# diets as low-fat diets
colData(mae)$Diet <- ifelse(colData(mae)$Diet == "High-fat" |
                              colData(mae)$Diet == "High-fat + XOS",
                            "High-fat", "Low-fat")

knitr::kable(head(colData(mae))) %>%
  kableExtra::kable_styling("striped",
                            latex_options="scale_down") %>%
  kableExtra::scroll_box(width = "100%")
```

From here, we can draw summaries of the sample (column) data, for instance to see what is the diet distribution.

The command `colData(mae)$Diet` fetches the data from the column, and `table()` creates a table that shows how many times each class is present, and `sort()` sorts the table to ascending order.

There are 20 samples from mice having High-fat, and 20 Low-fat.

```
sort(table(colData(mae)$Diet))
```

```
##
## High-fat  Low-fat
##       20       20
```

5.1.1 Transformations

Microbial abundances are typically 'compositional' (relative) in the current microbiome profiling data sets. This is due to technical aspects of the data generation process (see e.g. Gloor et al., 2017).

The next example calculates relative abundances as these are usually easier to interpret than plain counts. For some statistical models we need to transform the data into other formats as explained in above link (and as we will see later).

```
# Calculates relative abundances, and stores the table to assays
mae[[1]] <- transformCounts(mae[[1]], method = "relabundance")
```

A variety of standard transformations for microbiome data are available through mia R package.

	Phylum	Class	Order	Family	Genus	Species	OTU
Proteobacteria	Proteobacteria	NA	NA	NA	NA	NA	GAYR01026362.62.2014
Firmicutes	Firmicutes	NA	NA	NA	NA	NA	CVJT01000011.50.2173
Cyanobacteria	Cyanobacteria	NA	NA	NA	NA	NA	GEMN01027092.33.1623
Tenericutes	Tenericutes	NA	NA	NA	NA	NA	AM277369.1.1548
Deferribacteres	Deferribacteres	NA	NA	NA	NA	NA	AYGZ01000001.327.1863
Actinobacteria	Actinobacteria	NA	NA	NA	NA	NA	JGZF01000005.1.1534

5.1.2 Aggregation

Microbial species can be called at multiple taxonomic resolutions. We can easily agglomerate the data based on taxonomic ranks. Here, we agglomerate the data at Phylum level.

```
se_phylum <- agglomerateByRank(mae[[1]], rank = "Phylum")

# Show dimensionality
dim(se_phylum)
```

```
## [1] 13 40
```

Now there are 13 taxa and 40 samples, meaning that there are 13 different Phylum level taxonomic groups. Looking at the `rowData` after agglomeration shows all Firmicutes are combined together, and all lower rank information is lost.

From the assay we can see that all abundances of taxa that belong to Firmicutes are summed up.

```
knitr::kable(head(rowData(se_phylum))) %>%
  kableExtra::kable_styling("striped",
                            latex_options="scale_down") %>%
  kableExtra::scroll_box(width = "100%")
```

If you are sharp, you have by now noticed that all the aggregated values in the above example are NA's (missing data). This is because the agglomeration is missing abundances for certain taxa, and in that case the sum is not defined by default (`na.rm = FALSE`). We can ignore the missing values in summing up the data by setting `na.rm = TRUE`; then the taxa that do not have information in specified level will be removed. Those taxa that do not have information in specified level are agglomerated at lowest possible level that is left after agglomeration.

```
temp <- rowData(agglomerateByRank(mae[[1]], rank = "Genus"))

# Prints those taxa that do not have information at the Genus level (NA)
knitr::kable(head(temp[which(is.na(temp$Genus)),])) %>%
  kableExtra::kable_styling("striped",
                            latex_options="scale_down") %>%
  kableExtra::scroll_box(width = "100%")
```

Here agglomeration is done similarly, but na.rm = TRUE

	Phylum	Class	Order	Family	Genus	Species	OTU
Family:uncultured	Proteobacteria	Alphaproteobacteria	Rhodospirillales	uncultured	NA	NA	JRJTB:01000:00983
Family:Ruminococcaceae	Firmicutes	Clostridia	Clostridiales	Ruminococcaceae	NA	NA	JRJTB:00751:00256
Order:Clostridiales	Firmicutes	Clostridia	Clostridiales	NA	NA	NA	JRJTB:03059:01977
Family:Lachnospiraceae	Firmicutes	Clostridia	Clostridiales	Lachnospiraceae	NA	NA	JRJTB:00738:02832
Family:Peptostreptococcaceae	Firmicutes	Clostridia	Clostridiales	Peptostreptococcaceae	NA	NA	JRJTB:01731:00274
Family:Pasteurellaceae	Proteobacteria	Gammaproteobacteria	Pasteurellales	Pasteurellaceae	NA	NA	JRJTB:01960:01703

```
temp2 <- rowData(agglomerateByRank(mae[[1]], rank = "Genus", na.rm = TRUE))

print(paste0("Agglomeration with na.rm = FALSE: ", dim(temp)[1], " taxa."))
```

```
## [1] "Agglomeration with na.rm = FALSE: 277 taxa."
```

```
print(paste0("Agglomeration with na.rm = TRUE: ", dim(temp2)[1], " taxa."))
```

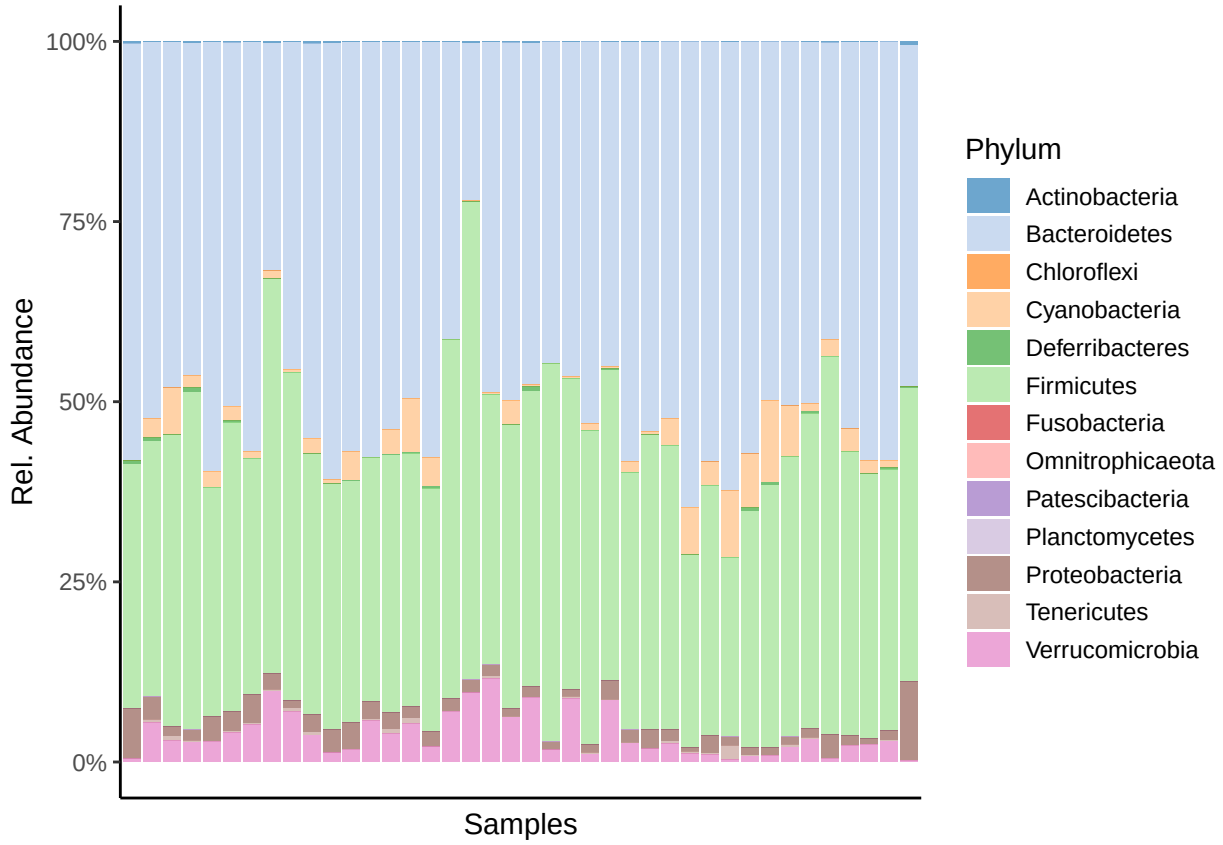
```
## [1] "Agglomeration with na.rm = TRUE: 262 taxa."
```

The mia package contains further examples on various data agglomeration and splitting options.

5.2 Visualization

The miaViz package facilitates data visualization. Let us plot the Phylum level abundances.

```
# Here we specify "relabundance" to be abundance table that we use for plotting.
# Note that we can use agglomerated or non-agglomerated mae[[1]] as an input, because
# the function agglomeration is built-in option.

# Legend does not fit into picture, so its height is reduced.
plot_abundance <- plotAbundance(mae[[1]], abund_values="relabundance", rank = "Phylum") +
  theme(legend.key.height = unit(0.5, "cm")) +
  scale_y_continuous(label = scales::percent)

plot_abundance
```

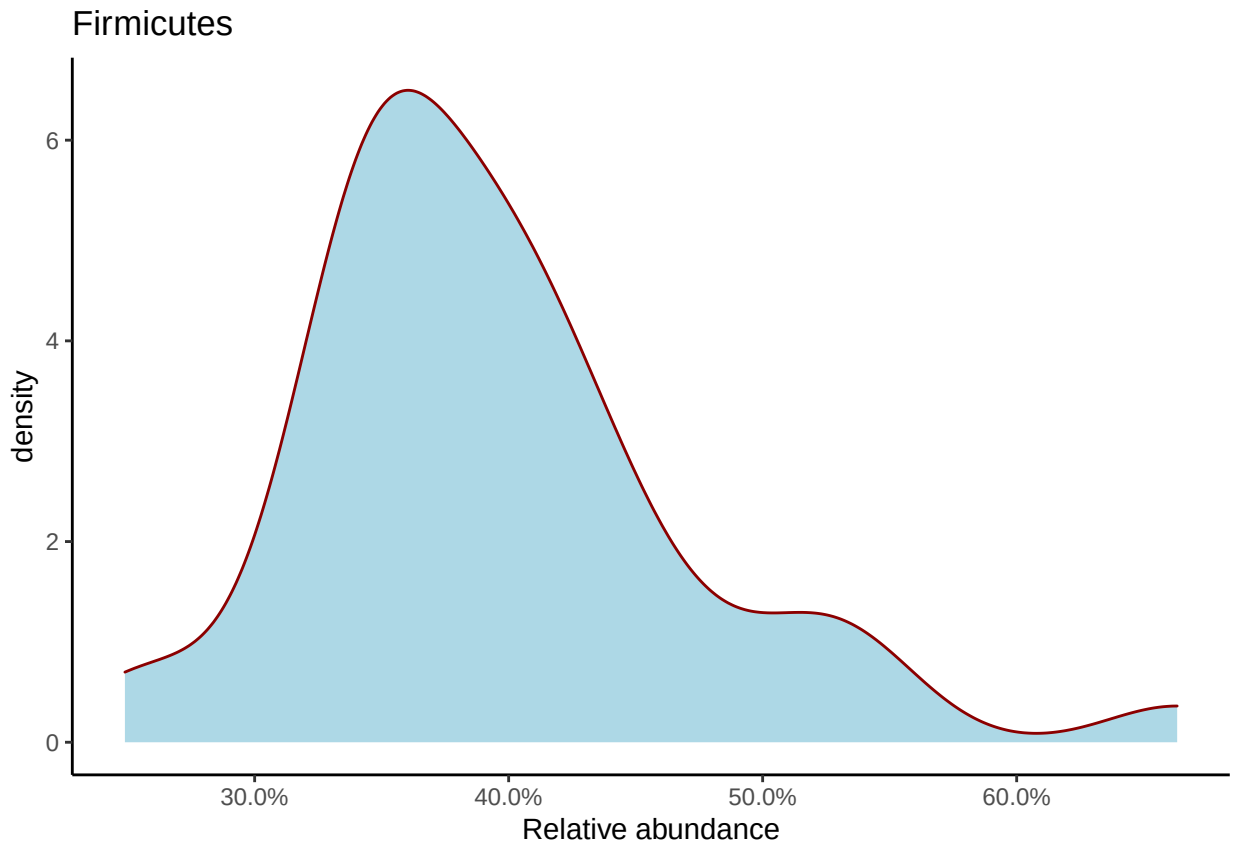


Density plot shows the overall abundance distribution for a given taxonomic group. Let us check the relative abundance of Firmicutes across the sample collection. The density plot is a smoothened version of a standard histogram.

The plot shows peak abundances around 30 %.

```
# Subset data by taking only Firmicutes
se_firmicutes <- se_phylum["Firmicutes"]

# Gets the abundance table
abundance_firmicutes <- assay(se_firmicutes, "relabundance")

# Creates a data frame object, where first column includes abundances
firmicutes_abund_df <- as.data.frame(t(abundance_firmicutes))
# Rename the first and only column
colnames(firmicutes_abund_df) <- "abund"

# Creates a plot. Parameters inside feom_density are optional. With
# geom_density(bw=1000), it is possible to adjust bandwidth.
firmicutes_abund_plot <- ggplot(firmicutes_abund_df, aes(x = abund)) +
  geom_density(color="darkred", fill="lightblue") +
  labs(x = "Relative abundance", title = "Firmicutes") +
  theme_classic() + # Changes the background
  scale_x_continuous(label = scales::percent)
```

```
firmicutes_abund_plot
```



For more visualization options and examples, see the miaViz vignette.

Chapter 6

Beta diversity

Beta diversity is another name for sample dissimilarity. It quantifies differences in the overall taxonomic composition between two samples.

Common indices include Bray-Curtis, Unifrac, Jaccard index, and the Aitchison distance. For more background information and examples, you can check the dedicated section in online book.

6.1 Examples of PCoA with different settings

Beta diversity estimation generates a (dis)similarity matrix that contains for each sample (rows) the dissimilarity to any other sample (columns).

This complex set of pairwise relations can be visualized in informative ways, and even coupled with other explanatory variables. As a first step, we compress the information to a lower dimensionality, or fewer principal components, and then visualize sample similarity based on that using ordination techniques, such as Principal Coordinate Analysis (PCoA). PCoA is a non-linear dimension reduction technique, and with Euclidean distances it is is identical to the linear PCA (except for potential scaling).

We typically retain just the two (or three) most informative top components, and ignore the other information. Each sample has a score on each of these components, and each component measures the variation across a set of correlated taxa. The top components are then easily visualized on a two (or three) dimensional display.

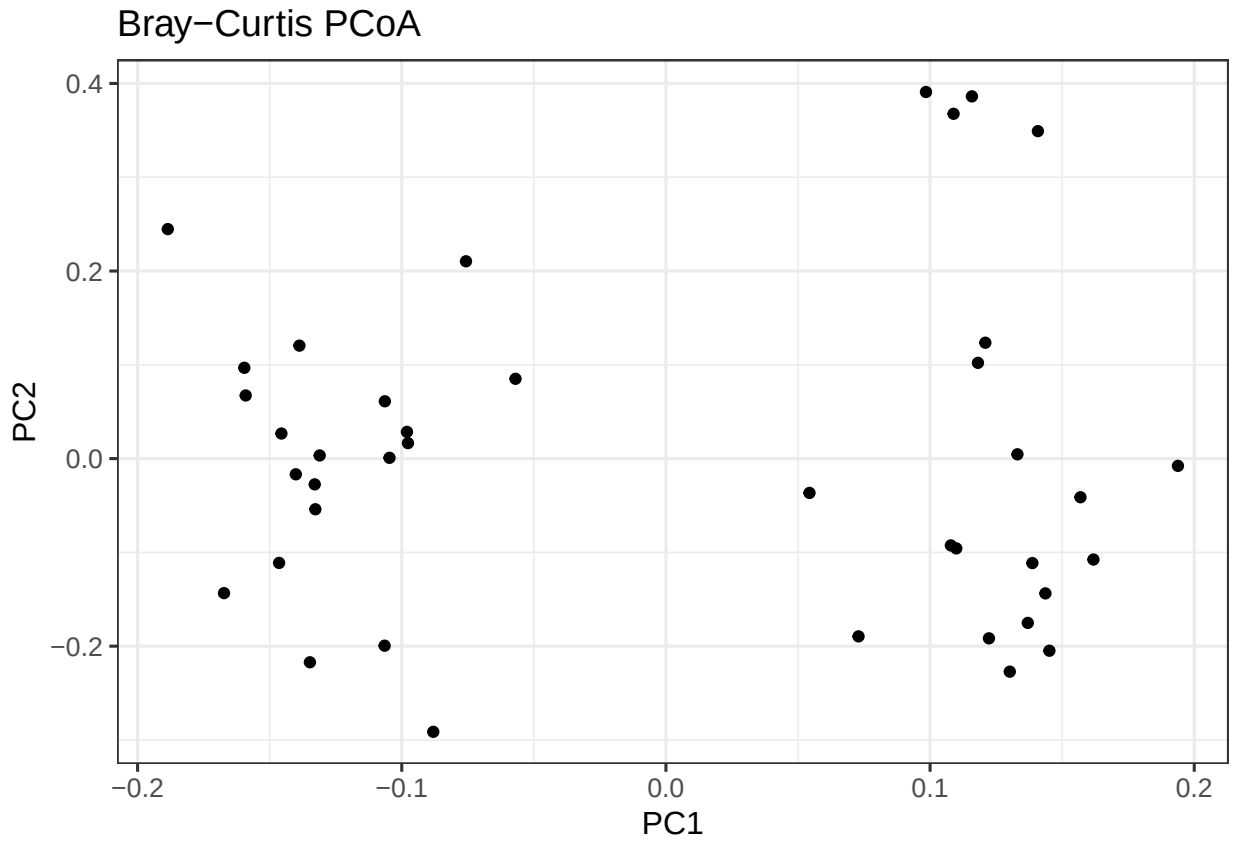
Let us next look at some concrete examples.

6.1.1 PCoA for ASV-level data with Bray-Curtis

Let us start with PCoA based on a Bray-Curtis dissimilarity matrix.

```
# Pick the relative abundance table
rel_abund_assay <- assays(mae[[1]])$relabundance

# Calculates Bray-Curtis distances between samples. Because taxa is in
# columns, it is used to compare different samples. We transpose the
# assay to get taxa to columns
bray_curtis_dist <- vegan::vegdist(t(rel_abund_assay), method = "bray")
```

```
# PCoA
bray_curtis_pcoa <- ecodist::pco(bray_curtis_dist)

# All components could be found here:
# bray_curtis_pcoa$vectors
# But we only need the first two to demonstrate what we can do:
bray_curtis_pcoa_df <- data.frame(pcoa1 = bray_curtis_pcoa$vectors[,1],
                                  pcoa2 = bray_curtis_pcoa$vectors[,2])

# Create a plot
bray_curtis_plot <- ggplot(data = bray_curtis_pcoa_df, aes(x=pcoa1, y=pcoa2)) +
  geom_point() +
  labs(x = "PC1",
       y = "PC2",
       title = "Bray-Curtis PCoA") +
  theme_bw(12) # makes titles smaller

bray_curtis_plot
```

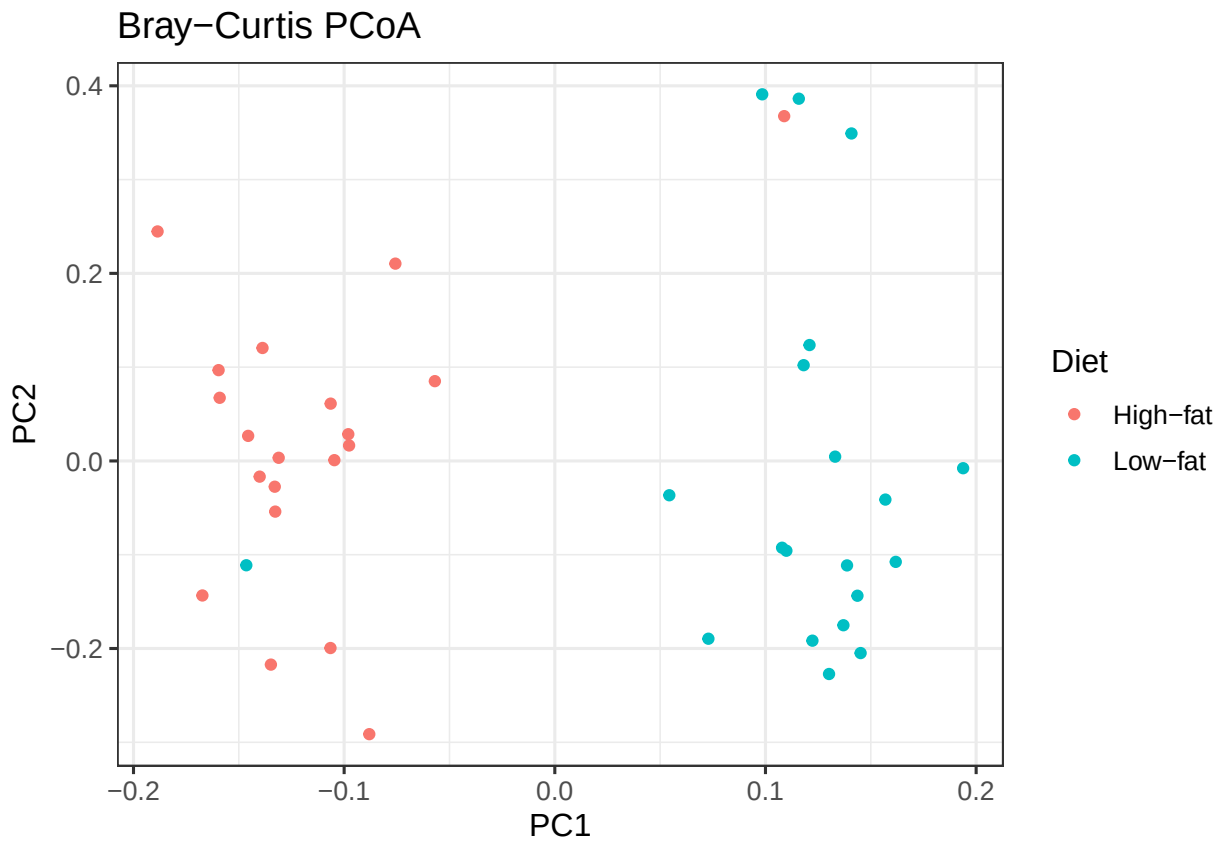


6.2 Highlighting external variables

We can map other variables on the same plot for example by coloring the points accordingly.

The following is an example with a discrete grouping variable (Diet) shown with colors:

```
# Add diet information to data.frame
bray_curtis_pcoa_df$Diet <- colData(mae)$Diet

# Creates a plot
plot <- ggplot(data = bray_curtis_pcoa_df, aes_string(x = "pcoa1", y = "pcoa2", color = "Diet")) +
  geom_point() +
  labs(x = "PC1",
       y = "PC2",
       title = "Bray-Curtis PCoA") +
  theme_bw(12)

plot
```



6.3 Estimating associations with an external variable

Next to visualizing whether any variable is associated with differences between samples, we can also quantify the strength of the association between community composition (beta diversity) and external factors.

The standard way to do this is to perform a so-called permutational multivariate analysis of variance (PERMANOVA). This method takes as input the abundance table, which measure of distance you want to base the test on and a formula that tells the model how you think the variables are associated with each other.

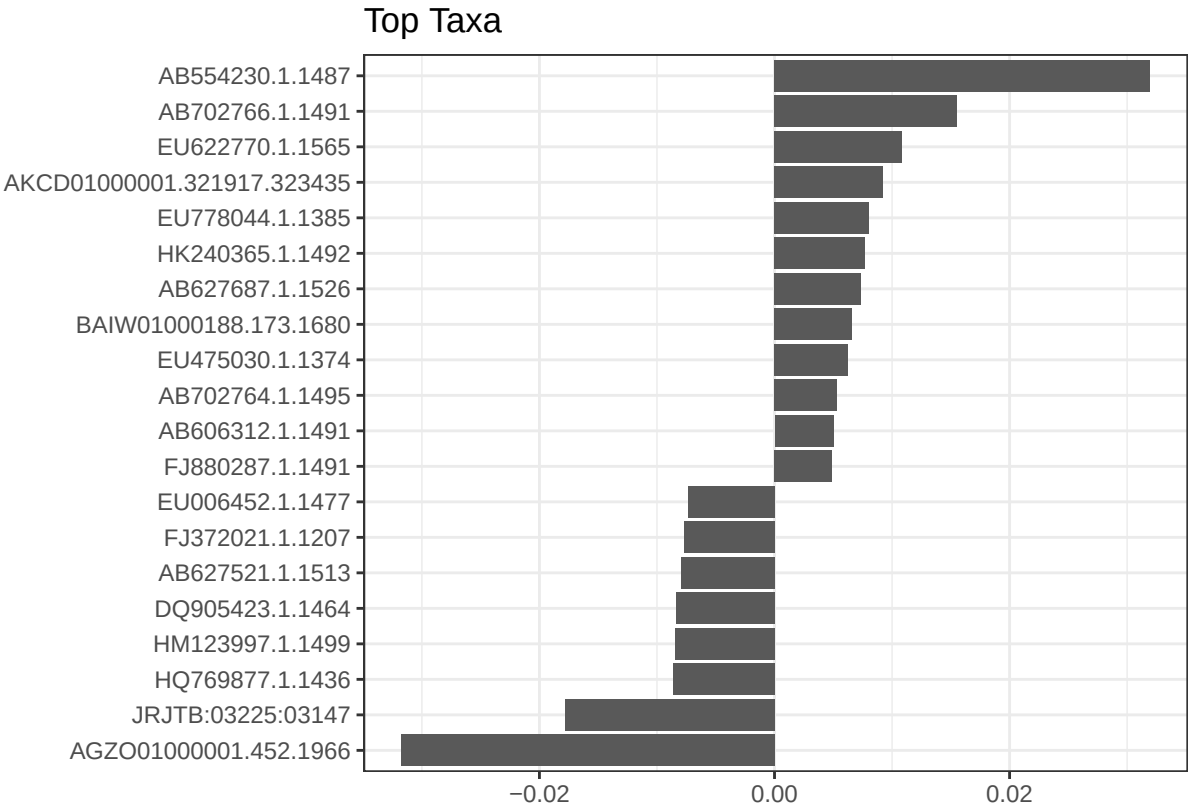
```
# First we get the relative abundance table
rel_abund_assay <- assays(mae[[1]])$relabundance

# again transpose it to get taxa to columns
rel_abund_assay <- t(rel_abund_assay)

# then we can perform the method
permanova_diet <- vegan::adonis(rel_abund_assay ~ Diet,
                                data = colData(mae),
                                permutations = 99)

# we can obtain a the p value for our predictor:
print(paste0("The test result p-value: ",
             as.data.frame(permanova_diet$aov.tab)["Diet", "Pr(>F)"]))
```

```
## [1] "The test result p-value: 0.01"
```

The diet variable is significantly associated with microbiota composition (p-value is less than 0.05).

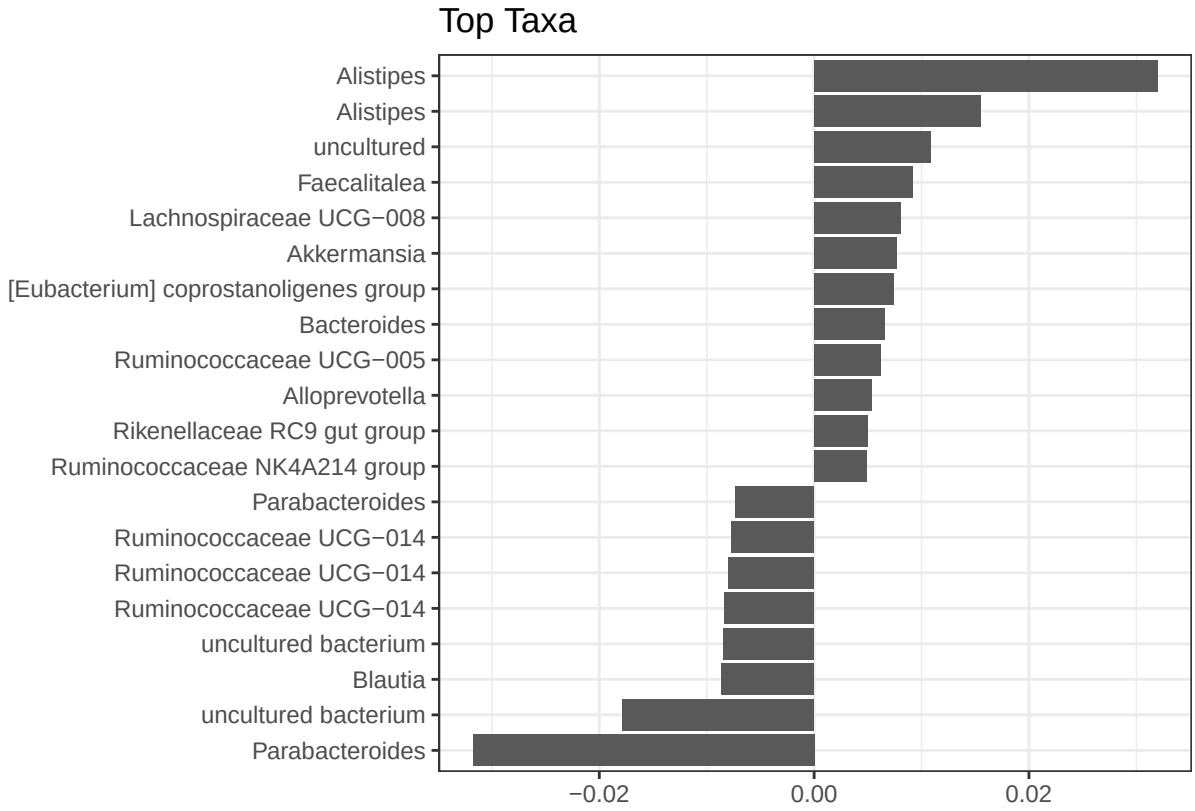
We can visualize those taxa whose abundances drive the differences between diets. We first need to extract the model coefficients of taxa:

```
# Gets the coefficients
coef <- coefficients(permanova_diet)["Diet1",]

# Gets the highest coefficients
top.coef <- sort(head(coef[rev(order(abs(coef)))],20))

# Plots the coefficients
top_taxa_coeffient_plot <- ggplot(data.frame(x = top.coef,
                                             y = factor(names(top.coef),
                          unique(names(top.coef)))),
                                  aes(x = x, y = y)) +
  geom_bar(stat="identity") +
  labs(x="", y="", title="Top Taxa") +
  theme_bw()

top_taxa_coeffient_plot
```



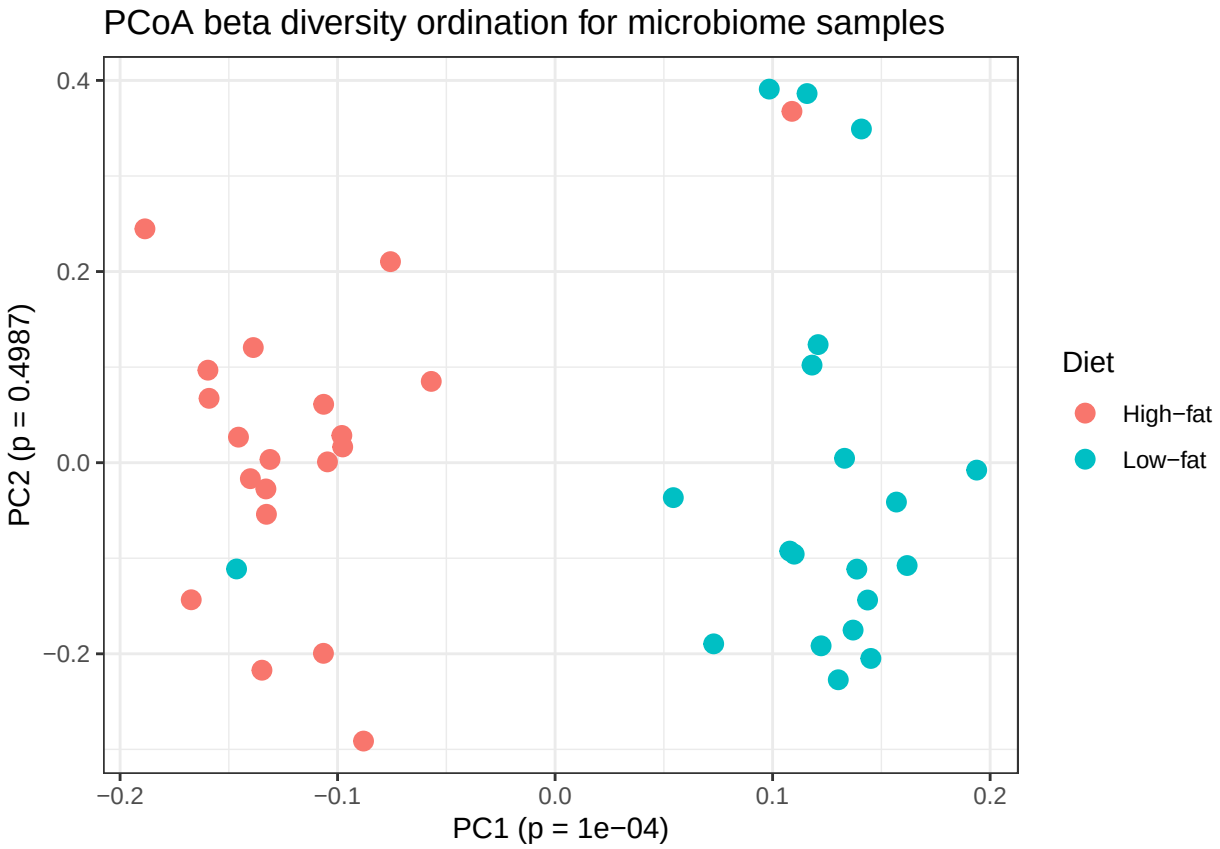
The above plot shows taxa as code names, and it is hard to tell which bacterial groups they represent. However, it is easy to add human readable names. We can fetch those from our rowData. Here we use Genus level names:

```
# Gets corresponding Genus level names and stores them to top.coef
names <- rowData(mae[[1]])[names(top.coef), ][,"Genus"]

# Adds new labels to the plot
top_taxa_coeffient_plot <- top_taxa_coeffient_plot +
  scale_y_discrete(labels = names) # Adds new labels
top_taxa_coeffient_plot
```



The same test can be conducted using the ordination from PCoA as follows:

```
bray_curtis_pcoa_df$Diet <- colData(mae)$Diet
p_values <- list()
for(pc in c("pcoa1", "pcoa2")){
  # Creates a formula from objects
  formula <- as.formula(paste0(pc, " ~ ", "Diet"))
  # Does the permanova analysis
  p_values[[pc]] <- vegan::adonis(formula, data = bray_curtis_pcoa_df,
                                  permutations = 9999, method = "euclidean"
  )$aov.tab["Diet", "Pr(>F)"]
}

# Creates a plot
plot <- ggplot(data = bray_curtis_pcoa_df, aes_string(x = "pcoa1", y = "pcoa2", color = "Diet")) +
  geom_point(size = 3) +
  labs(title = paste0("PCoA beta diversity ordination for microbiome samples"), x = paste0("PC1 (p = ", p_val
  theme_bw()

plot
```



There are many alternative and complementary methods for analysing community composition. For more examples, see a dedicated section on beta diversity in the online book.

6.4 Community typing

A dedicated section presenting examples on community typing is in the online book.

Chapter 7

Cross-correlation & Unsupervised learning

7.1 Cross-correlation

With cross-correlation analysis, we can analyze how strongly and how differently variables are associated between each other. For instance, we can analyze if higher presence of a specific taxon equals to higher levels of a biomolecule.

TASK

1. Run installation script to load packages into the session
2. Import HintikkaXO data
3. Subset *microbiota* data (rank = "Phylum", prevalence = 0.2, detection = 0.001) (subsetByPrevalentTaxa)
4. Apply clr-transform to *microbiota* data and log10-tranform to *metabolites* data (transformSamples)
5. Remove uncultured and ambiguous taxa (as it's hard to interpret their results) (USE THIS: *mae[[1]] <- mae[[1]][-grep("uncultured|Ambiguous_taxa", names(mae[[1]])),]*)
6. Calculate cross-correlation between *microbiota* (clr) and *metabolites* (log10) (Use *show_warnings = FALSE* as an argument) (testExperimentCrossCorrelation)
7. Create a heatmap from cross-correlation matrix pheatmap

7.2 Unsupervised learning

Unsupervised learning is a part of machine learning where we try to find information from unknown data. It is also called data mining. Usually this means finding of clusters, for instance. Cluster refers to group of samples/features that are similar between each other. For example, based on clinical data we can try to find patient groups that have similar response to used drug.

7.2.1 Biclustering

Biclustering is a clustering method, which simultaneously clusters rows and columns. In this example, the aim is to find clusters where subset of taxa share similar pattern over subset of metabolites. In our case, we try to find clusters where taxa and metabolites correlate similarly.

Check more from OMA which has dedicated chapter on biclustering.

```
# Load package
library(biclust)
```

```
## Loading required package: MASS
```

```
## Loading required package: grid
```

```
##
## Attaching package: 'grid'
```

```
## The following object is masked from 'package:Biostrings':
##
##     pattern
```

```
## Loading required package: colorspace
```

```
## Loading required package: lattice
```

```
# Find biclusters
bc <- biclust(corr$cor, method=BCPlaid(), fit.model = y ~ m,
              background = TRUE, shuffle = 100, back.fit = 0, max.layers = 10,
              iter.startup = 10, iter.layer = 100, verbose = FALSE)

bc
```

```
##
## An object of class Biclust
##
## call:
##  biclust(x = corr$cor, method = BCPlaid(), fit.model = y ~ m,
##      background = TRUE, shuffle = 100, back.fit = 0, max.layers = 10,
##      iter.startup = 10, iter.layer = 100, verbose = FALSE)
##
## There was one cluster found with
##   4 Rows and  6 columns
```

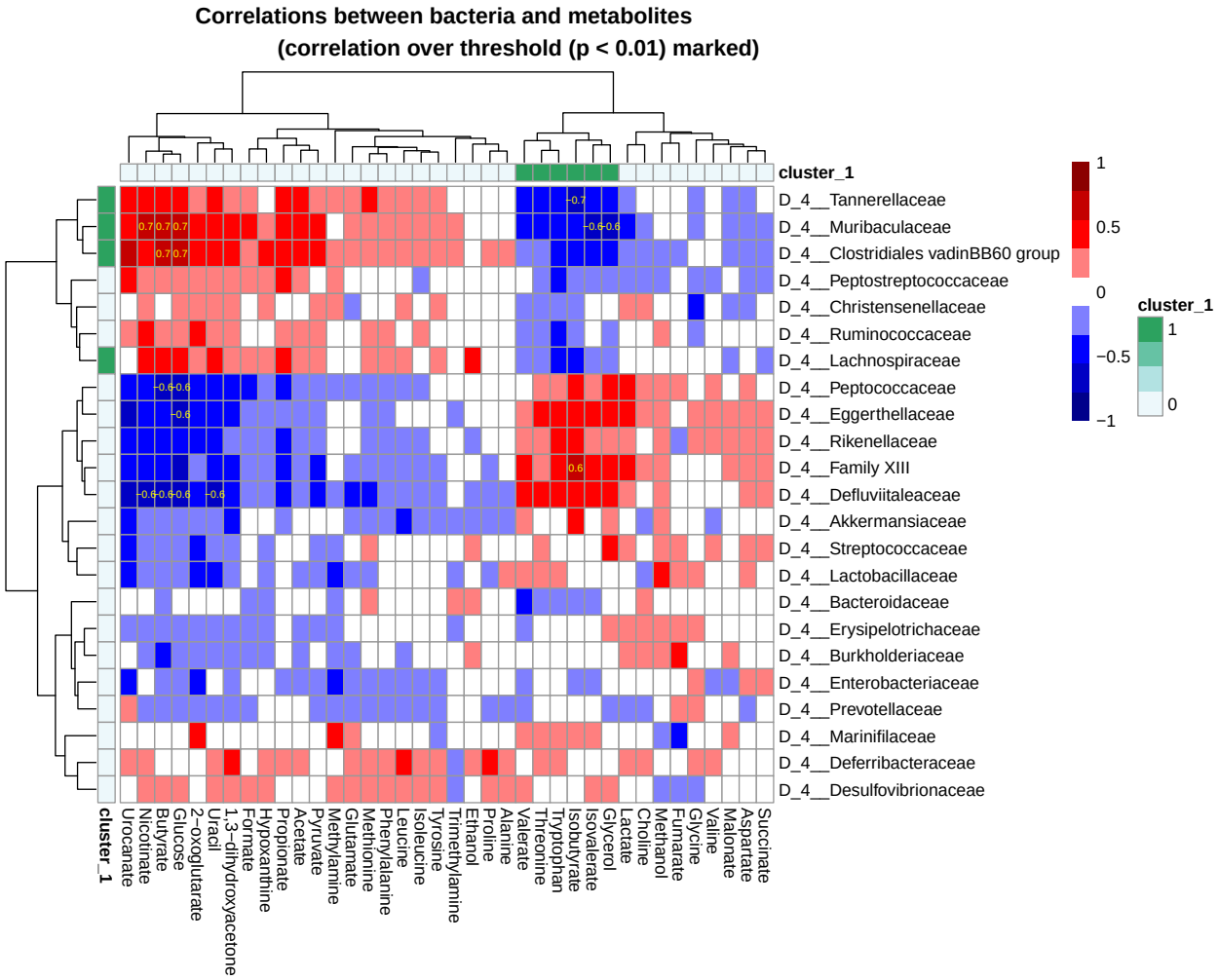
```
# Get biclusters
bicluster_rows <- bc@RowxNumber
bicluster_columns <- bc@NumberxCol

# Convert them into data.frames
```

```
bicluster_rows <- data.frame(bicluster_rows)
bicluster_columns <- data.frame(t(bc@NumberxCol))

# Adjust names of clusters
colnames(bicluster_rows) <- paste0("cluster_", 1:ncol(bicluster_rows))
colnames(bicluster_columns) <- paste0("cluster_", 1:ncol(bicluster_columns))

# Print biclusters for rows
head(bicluster_rows)
```

```
##   cluster_1
## 1     FALSE
## 2     FALSE
## 3     FALSE
## 4     FALSE
## 5     FALSE
## 6     FALSE
```

Now, we can add bicluster information into the heatmap that we already made.

```
# Convert boolean values into numeric
bicluster_columns[ , 1] <- as.numeric(bicluster_columns[, 1] )
bicluster_rows[ , 1] <- as.numeric(bicluster_rows[ , 1])

# Adjust their rownames
rownames(bicluster_columns) <- colnames(corr$cor)
rownames(bicluster_rows) <- rownames(corr$cor)

# Get correlation values that are over thresholds
p_threshold <- 0.01
corr_values <- ifelse(corr$p_adj<p_threshold, round(corr$cor,1), "")

# Create a heatmap
pheatmap(corr$cor,
         annotation_col = bicluster_columns,
         annotation_row = bicluster_rows,

         display_numbers = corr_values,

         main = paste0("Correlations between bacteria and metabolites
                       (correlation over threshold (p < ", p_threshold,") marked)"),

         breaks = breaks,
         color = colors,

         fontsize_number = 6,
         number_color = "yellow")
```


Correlations between bacteria and metabolites
(correlation over threshold (p < 0.01) marked)
cluster_1
D_4__Tannerellaceae
D_4__Muribaculaceae
D_4__Clostridiales vadinBB60 group
D_4__Peptostreptococcaceae
D_4__Christensenellaceae
D_4__Ruminococcaceae
D_4__Lachnospiraceae
D_4__Peptococcaceae
D_4__Eggerthellaceae
D_4__Rikenellaceae
D_4__Family XIII
D_4__Defluviitaleaceae
D_4__Akkermansiaceae
D_4__Streptococcaceae
D_4__Lactobacillaceae
D_4__Bacteroidaceae
D_4__Erysipelotrichaceae
D_4__Burkholderiaceae
D_4__Enterobacteriaceae
D_4__Prevotellaceae
D_4__Marinifilaceae
D_4__Deferribacteraceae
D_4__Desulfovibrionaceae
cluster_1
Urocanate
Nicotinate
Butyrate
Glucose
2−oxoglutarate
Uracil
1,3−dihydroxyacetone
Formate
Hypoxanthine
Propionate
Acetate
Pyruvate
Methylamine
Glutamate
Methionine
Phenylalanine
Leucine
Isoleucine
Tyrosine
Trimethylamine
Ethanol
Proline
Alanine
Valerate
Threonine
Tryptophan
Isobutyrate
Isovalerate
Glycerol
Lactate
Choline
Methanol
Fumarate
Glycine
Valine
Malonate
Aspartate
Succinate
1
0.5
0
−0.5
−1
cluster_1
1
0

Chapter 8

Supervised learning

Machine learning models are highly flexible and can be used to model differences in samples, similarly to (frequentist) statistics. However, the analysis workflow with these methods is very different from the frequentist analyses. These models learn a function to predict values of the dependent variable, given data. Between different models, the algorithms vary greatly, but generally all regression and classification models can be used similarly in a machine learning workflow.

Machine learning models do not usually output p-values, but they are designed to predict the outcome (value or class) of the dependent variable based on data. Thus, if we want to know how good our model is, we need to divide our data to training and test (or validation) sets. The training set is used to train the model, and the validation set can be then used to test the model. The model can be used to predict the outcome of the dependent variable on the test data, and the predicted values can be compared to the actual known values in the test data. It is important to make sure that there is no data leakage between these two sets, or otherwise the validation is compromised.

In the workshop we use random forests and the caret package to train regression and classification models. The models will predict continuous butyrate concentration or discretized class (high/low butyrate) based on the microbiome composition (why butyrate?).

8.1 Data curation

We first make a data frame which includes only the butyrate concentration and the transformed genus-level microbiome data.

Creating a dataframe for modeling butyrate levels:

```
butyrate_df <- data.frame(cbind(y, x))
butyrate_df <- butyrate_df[,which(colnames(butyrate_df) %in% c("Butyrate", colnames(x)))]
```

A function in the caret package is used to divide the data once to 80% train and 20% test (validation) sets. This is to prevent data leakage and overestimation of model performance. The 20% test set is only used to conduct the final validation of the models. The number of samples in this case is low (n = 40), but as we can see later on, even 8 samples is sufficient for estimation of the performance. Note that the data is stratified to include a representative distribution of butyrate concentrations on both sides of the split. There is some randomness inherent in the splitting, so set.seed() needs to be used.

```
library(caret)
set.seed(42)
trainIndex <- createDataPartition(butyrate_df$Butyrate, p = .8, list = FALSE, times = 1)
butyrate_df_train <- butyrate_df[trainIndex,]
butyrate_df_test <- butyrate_df[-trainIndex,]
```

8.2 Regression with random forests

Random forests are a common and flexible ensemble learning method, which are a good starting point when choosing a machine learning model. We are using the ranger implementation of random forests, which runs quite fast compared to its alternatives in R. A wrapper train function from caret is used to conduct a 5-fold cross-validation, repeated 5 times with random partitions inside the training data (for further reading, see documentation). Because of the randomness, seed needs to be set again. **Note that the seed does *not* remain set if you re-run a function with a random component without calling set.seed() first!** Specific to random forests, we are also using an option to use permutation importance.

The train function is quite complex, and performs hyperparameter tuning while training the model with cross-validation. The final model included in the object is then trained on all input data and optimized hyperparameters.

```
set.seed(42)
fitControl <- trainControl(method = "repeatedcv", number = 5, repeats = 5)
rfFit1 <- train(Butyrate ~ ., data = butyrate_df_train,
                method = "ranger",
                trControl = fitControl,
                importance = "permutation")
```

Following the training, we can print out the resulting object, which shows details about the training. We also print out results of the final model.

```
print(rfFit1)
```

```
## Random Forest
##
## 32 samples
## 28 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 24, 26, 25, 28, 25, 25, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE       Rsquared   MAE
##    2    variance    0.8801800  0.5870278  0.6802341
##    2    extratrees  0.9039197  0.5888711  0.7113279
##   15    variance    0.8604439  0.5806824  0.6573050
##   15    extratrees  0.8678618  0.5773606  0.6682391
##   28    variance    0.8702166  0.5663926  0.6564932
```

```
##   28    extratrees  0.8569955  0.5883242  0.6589236
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 28, splitrule = extratrees
##  and min.node.size = 5.
```

```
print(rfFit1$finalModel)
```

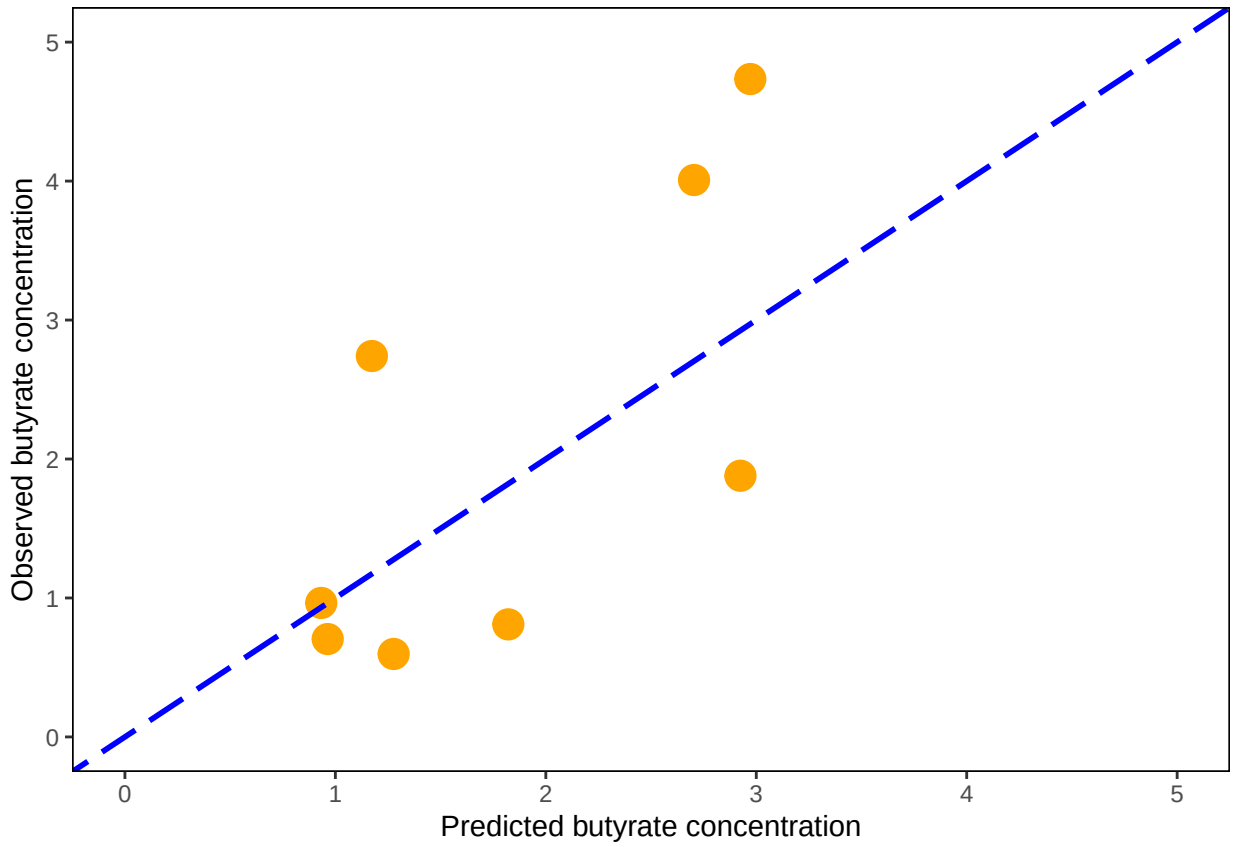
```
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = ".outcome", data = x,      mtry = min(param$mtry, ncol(x)), min.
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      32
## Number of independent variables:  28
## Mtry:                             28
## Target node size:                 5
## Variable importance mode:         permutation
## Splitrule:                        extratrees
## Number of random splits:          1
## OOB prediction error (MSE):       0.8650469
## R squared (OOB):                  0.4563803
```

We can then compare the metrics produced in training to actual validation metrics. Here, the final model predicts on unseen test data samples, and metrics are calculated against the observed (true) values. We can also plot the predicted values against the observed values, and include a line to show how a perfect model would predict.

```
test_predictions <- predict(rfFit1, newdata = butyrate_df_test)
print(postResample(test_predictions, butyrate_df_test$Butyrate))
```

```
##      RMSE  Rsquared       MAE
## 1.1129878 0.4973854 0.9576183
```

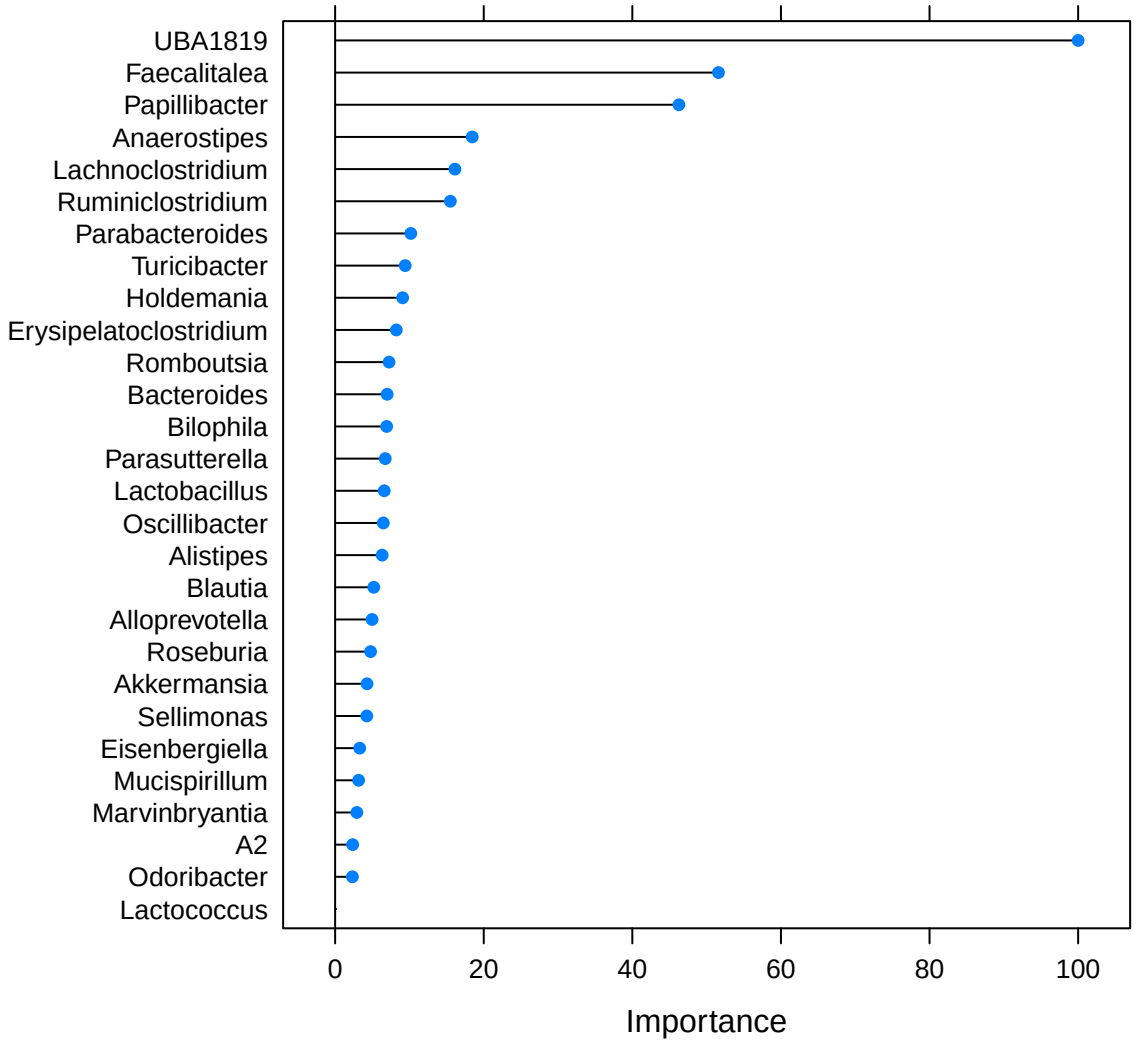
```
# Plot predicted vs observed
pred_obs <- data.frame(predicted = test_predictions, observed = butyrate_df_test$Butyrate)
ggplot(data = pred_obs, aes(x=predicted, y=observed)) + geom_point(size = 5, color = "orange") +
  xlab("Predicted butyrate concentration") + ylab("Observed butyrate concentration") +
  lims(x = c(0,5), y = c(0,5)) +
  geom_abline(linetype = 5, color = "blue", size = 1)+ # Plot a perfect fit line
    theme(panel.border = element_rect(colour = "black", fill = NA),
                            panel.background = element_blank())
```



All models are wrong, but some are useful. Luckily, our model seems to be somewhat useful. However, we are not yet done with the model. It is important to examine how our model actually works. Even though the random forest can look like a "black box", we can get much information out on how it ends up making specific predictions.

Feature importance can be used, quite literally, to see which features are important for model predictions. Permutation importance is a metric calculated by shuffling the values of individual feature columns. Permutation importance of a feature will be high if such corrupted data leads to bad predictions of the model. If shuffling a feature does not affect the model performance negatively, its permutation importance will be low.

```
plot(varImp(rfFit1))
```



We can see that some genera are highly important for model predictions. The importance values can be highly useful, and these are often used *e.g.,* for feature selection before conducting statistical (or other ML) tests. However, if we only conducted this supervised machine learning analysis, we would not know which features are positively and which ones negatively associated with butyrate levels.

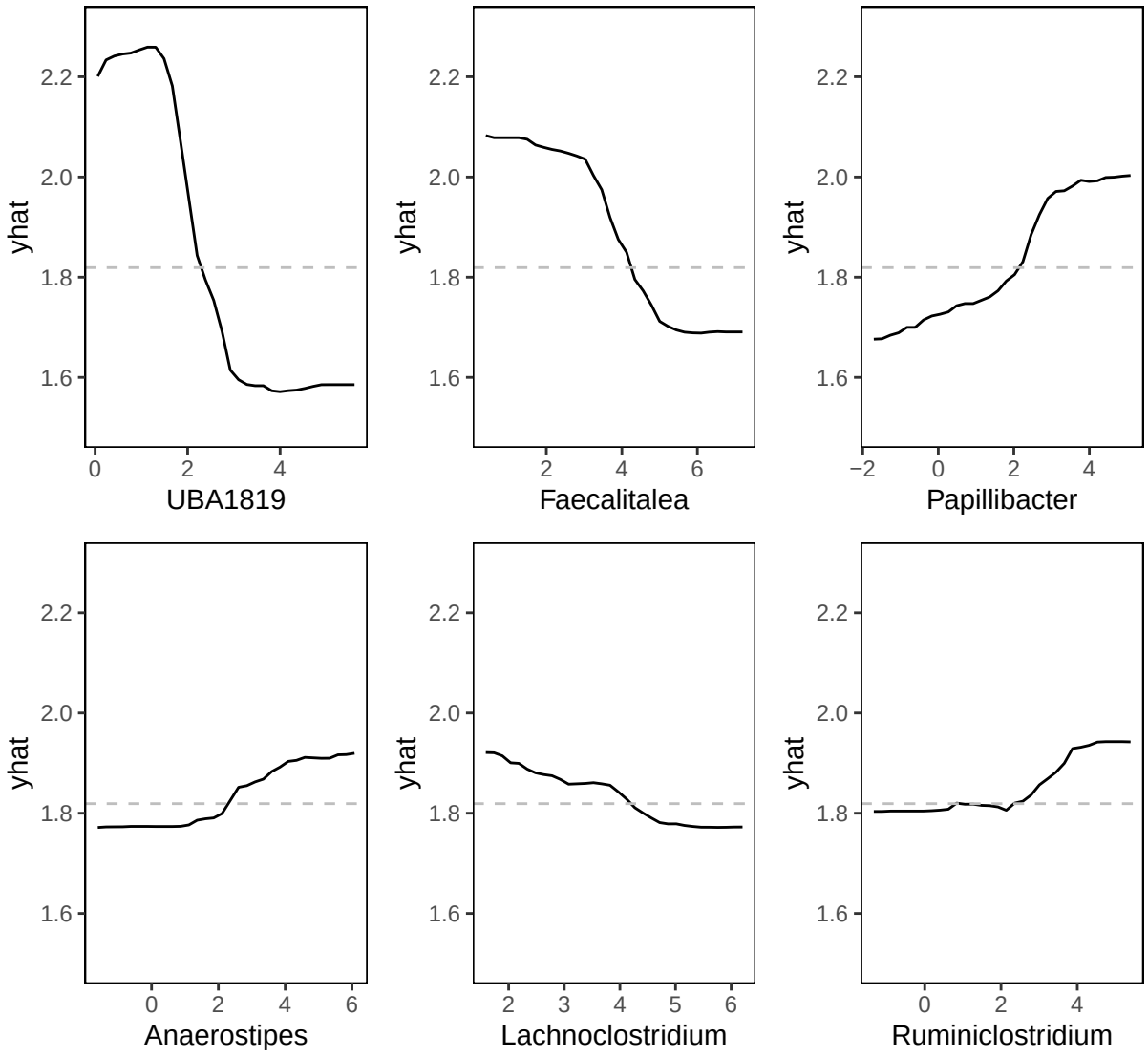
All black box models can however be examined through partial dependence plots. Similarly to validation, we can again utilize the fact that ML models are great at making new predictions. While the inner workings of the model are highly complex, we can assume that changing the values of an important feature affects the model prediction in a specific way. Briefly, partial dependence plots visualize the expected output of the model over the range of an individual input feature (up to 3 features). The pdp package is a versatile implementation for conducting these analyses in R and works directly on models fitted with caret::train().

```
library(patchwork)
library(pdp)
top_features <- rownames(varImp(rfFit1)$importance)[order(varImp(rfFit1)$importance[,"Overall"], decreasing =
```

```
pd_plots <- list(NULL)
for (feature in 1:length(top_features)) {
  pd_plots[[feature]] <- partial(rfFit1, pred.var = top_features[feature], rug = TRUE) %>% autoplot() +
    geom_hline(yintercept = mean(butyrate_df_train$Butyrate), linetype = 2, color = "gray") + # Show the mean
    scale_y_continuous(limits=c(1.5,2.3)) + # Harmonize the scale of yhat on all plots
      theme(panel.border = element_rect(colour = "black", fill = NA),
                            panel.background = element_blank())
  print(paste0("Partial dependence of ", top_features[feature]))
}
```

```
## [1] "Partial dependence of UBA1819"
## [1] "Partial dependence of Faecalitalea"
## [1] "Partial dependence of Papillibacter"
## [1] "Partial dependence of Anaerostipes"
## [1] "Partial dependence of Lachnoclostridium"
## [1] "Partial dependence of Ruminiclostridium"
```

```
wrap_plots(pd_plots)
```



8.3 Classification with random forests

In addition to regression, random forests can also be used for classification tasks. While the actual butyrate values might be important for some purposes, it can be expected that it is quite hard to model the concentration throughout its observed range accurately. As a demonstration, we can also discretize a constant variable, and handle the problem as a classification task. Here, we are using the median value of butyrate as a cutoff for high and low butyrate groups - which makes this a binary classification. Other ways to specify a cutoff for discretization are often more justified than just the mean or median. These are usually informed by previous studies and results.

```
butyrate_cutoff <- median(butyrate_df_test$Butyrate)
butyrate_df_test_2 <- butyrate_df_test
```

```
butyrate_df_train_2 <- butyrate_df_train
butyrate_df_test_2$Butyrate <- as.factor(ifelse(butyrate_df_test_2$Butyrate >= butyrate_cutoff, "High", "Low"
butyrate_df_train_2$Butyrate <- as.factor(ifelse(butyrate_df_train_2$Butyrate >= butyrate_cutoff, "High", "Lo
```

Training of the model is very similar to regression, but we want to define two options in the trainControl function sent to caret::train(). classProbs = TRUE is used to output classification probabilities instead of the classes themselves. This is required for calculating ROC-AUC values. Also, we are defining a summaryFunction which is used for evaluation and hyperparameter optimization.

```
set.seed(42)
fitControl <- trainControl(method = "repeatedcv", number = 5, repeats = 10, classProbs = TRUE, summaryFunctio
rfFit2 <- train(Butyrate ~ ., data = butyrate_df_train_2,
                method = "ranger",
                trControl = fitControl,
                importance = "permutation")
```

We then measure performance with test data. Here, we need to construct a data frame with specific dimensions as an input for twoClassSummary().

```
test_predictions_2 <- data.frame(obs = butyrate_df_test_2$Butyrate,
                                 pred = predict(rfFit2, newdata = butyrate_df_test_2),
                                 predict(rfFit2, newdata = butyrate_df_test_2, type = "prob"))

#Print out the metrics
print(rfFit2)
```

```
## Random Forest
##
## 32 samples
## 28 predictors
##  2 classes: 'High', 'Low'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 25, 25, 26, 26, 26, 26, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   ROC        Sens       Spec
##    2    gini        0.8622222  0.7733333  0.8433333
##    2    extratrees  0.8877778  0.8133333  0.8450000
##   15    gini        0.8261111  0.7866667  0.7950000
##   15    extratrees  0.8561111  0.8266667  0.8116667
##   28    gini        0.8150000  0.7733333  0.7950000
##   28    extratrees  0.8488889  0.8200000  0.8016667
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = extratrees
##  and min.node.size = 1.
```

```
print(rfFit2$finalModel)
```

```
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = ".outcome", data = x,      mtry = min(param$mtry, ncol(x)), min.
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      32
## Number of independent variables:  28
## Mtry:                             2
## Target node size:                 1
## Variable importance mode:         permutation
## Splitrule:                        extratrees
## Number of random splits:          1
## OOB prediction error (Brier s.):  0.163163
```

```
print(twoClassSummary(test_predictions_2, lev = c("High", "Low")))
```

```
##   ROC  Sens  Spec
## 0.875 0.750 0.750
```

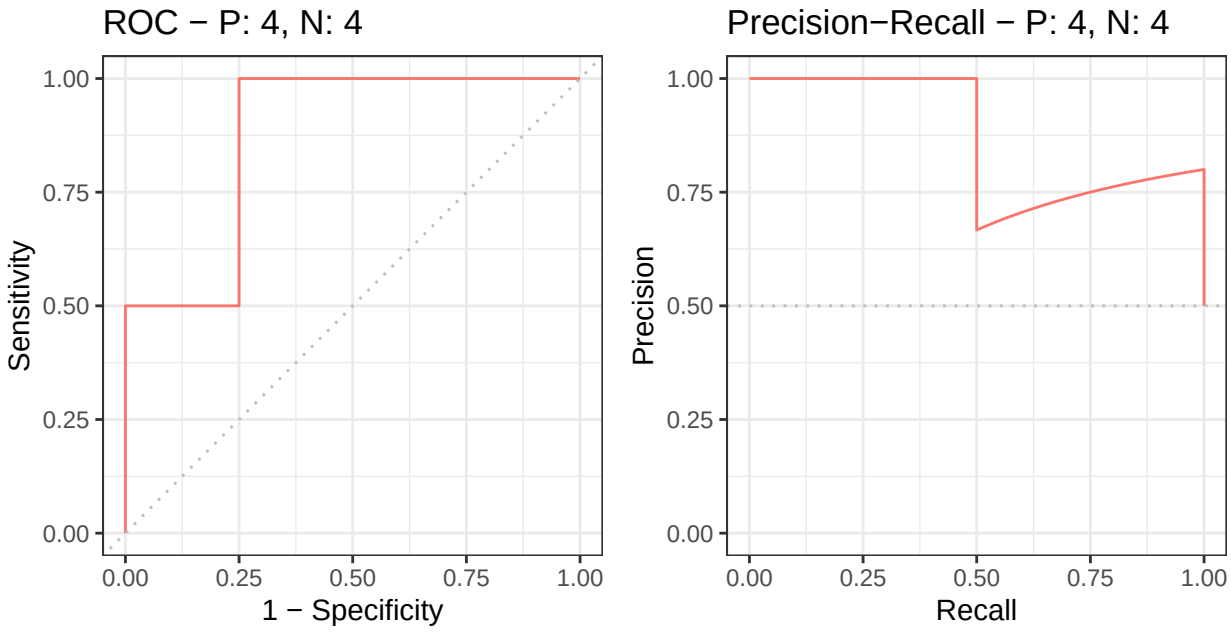
Often just the ROC-AUC value calculated above can suffice, for example for model comparisons. However, if class distribution is skewed, area under the precision-recall curve (AUPRC) should be used instead (see Fu et al., 2018).

Here is one way to calculate both with the package precrec and plot the curves.

```
library(MLmetrics)
library(precrec)
aucs <- evalmod(scores = test_predictions_2$Low, labels = test_predictions_2$obs)
print(aucs)
```

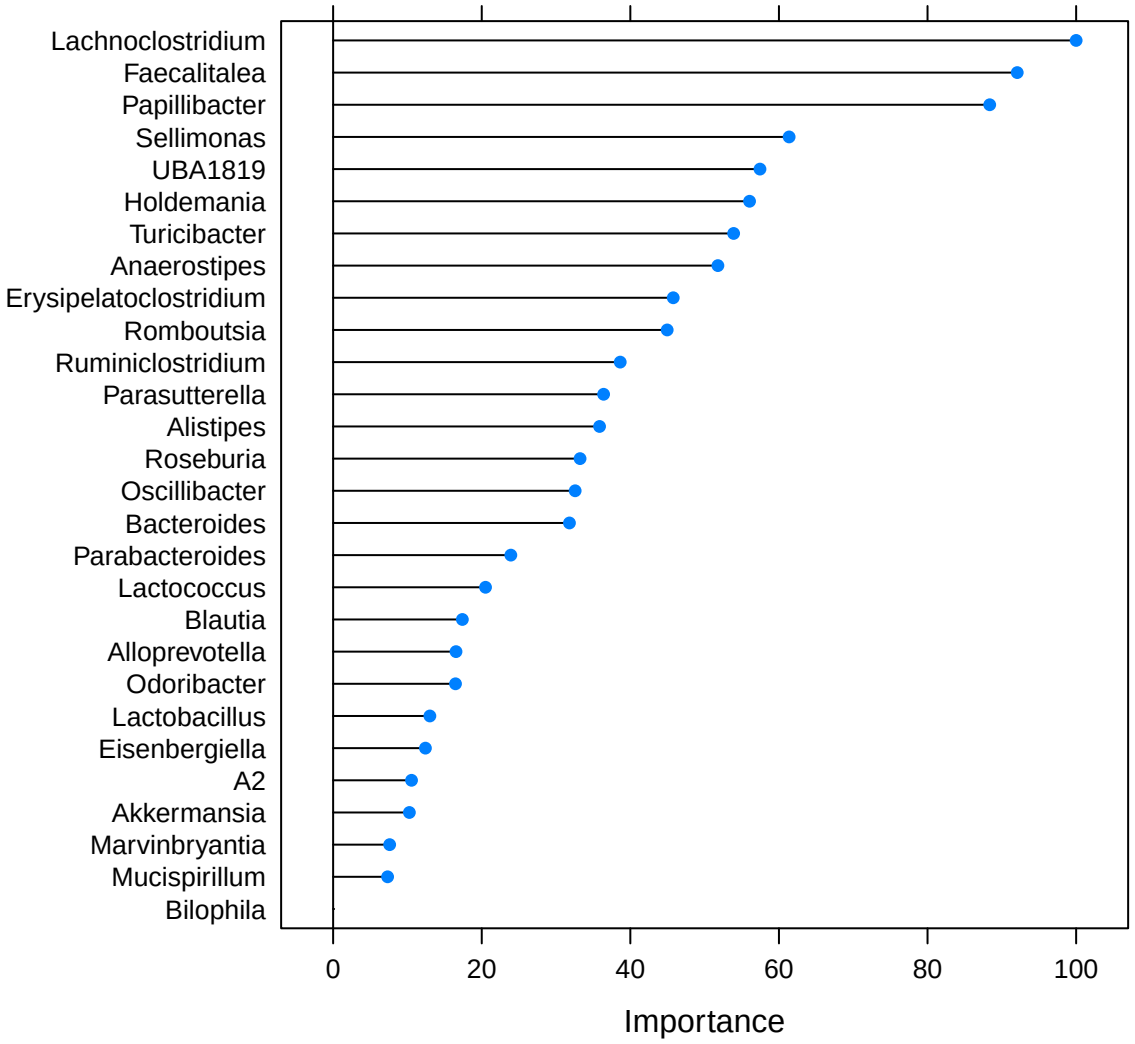
```
##
##     === AUCs ===
##
##      Model name Dataset ID Curve type       AUC
##    1         m1          1        ROC 0.8750000
##    2         m1          1        PRC 0.8722936
##
##
##     === Input data ===
##
##      Model name Dataset ID # of negatives # of positives
##    1         m1          1              4              4
```

```
autoplot(aucs)
```



Finally, we can also extract and plot the feature importance from the binary classification model. It is interesting to compare this result to the importances of the same features in the regression model.

```
plot(varImp(rfFit2))
```


Lachnoclostridium
Faecalitalea
Papillibacter
Sellimonas
UBA1819
Holdemania
Turicibacter
Anaerostipes
Erysipelatoclostridium
Romboutsia
Ruminiclostridium
Parasutterella
Alistipes
Roseburia
Oscillibacter
Bacteroides
Parabacteroides
Lactococcus
Blautia
Alloprevotella
Odoribacter
Lactobacillus
Eisenbergiella
A2
Akkermansia
Marvinbryantia
Mucispirillum
Bilophila
0
20
40
60
80
100
Importance

Bibliography

Borman, T., Eckermann, H., Benchraka, C., Ruuskanen, M., and Lahti, L. (2022). *Introduction to multi-omics data analysis*.