PLSDA-batch: a multivariate framework to correct for batch effects in microbiome data

Yiwen Wang[1,2], Kim-Anh Lê Cao[2]
[1]Agricultural Genomics Institute at Shenzhen, Chinese Academy of Agricultural Sciences, Shenzhen, Guangdong, China
[2]Melbourne Integrative Genomics, School of Mathematics and Statistics, The University of Melbourne, Parkville, VIC, Australia

2023-04-11

Contents

Chapter 1

Batch Effects Management in Case Studies

1.1 Introduction

Investigating the link between microbial composition and phenotypes has become the limelight of research in recent years, as microorganisms play a key role in extensive fields including agriculture, healthcare, food production, industry and climate change [Wang et al., 2020, Ray et al., 2020, Fan and Pedersen, 2020, Poirier et al., 2020]. The collection of microorganisms and their genomes within a specific environment is referred to as the *microbiome* [Marchesi and Ravel, 2015]. The microbiome can be profiled using 16S rRNA gene sequencing or whole-genome shotgun sequencing. The microbiome data is displayed as an abundance table of counts per sample for each taxon. This type of data have their inherent characteristics, including zero inflation, uneven library sizes, compositional structure, and multivariate nature, which limit statistical analysis.

Microbiome research faces the challenge of results reproducibility across studies. The potential reasons are poor experimental design and lack of rigorous operating procedures, which introduce variation in the data (batch effects) that obscure the effect of interest. Microbiome data are highly susceptible to batch effects because of the dynamic nature of microbial communities [Wang and LêCao, 2020]. Numerous studies have reported batch effects introduced by sequencing batches [Hieken et al., 2016], the inclusion of independent studies [Duvallet et al., 2017], geography, age, sex, health status, stress and diet [Gibson et al., 2004, Lozupone et al., 2013, Haro et al., 2016, Kim et al., 2017]. Attempts have been made to alleviate batch effects through standardised designs, but batch effects remain unavoidable in practice. For example, cage effects in murine experiments, age, sex and diet effects in human experiments, sequencing batches are ubiquitous in microbiome studies. So far, methods to manage batch effects in microbiome studies have been lacking, thus limiting microbiome researchers in their

ability to analyse their data.

Several methods to handle batch effects have been proposed. However, these methods 1/ were primarily developed for gene expression data, thus do not address the inherent characteristics of microbiome data or 2/ are limited to differential abundance analysis, thus limiting the breadth of statistical analysis that can be performed to answer biological questions for microbiome researchers.

I developed a new batch effect correction method based on Projection to Latent Structures Discriminant Analysis named "PLSDA-batch" to correct data prior to any downstream analysis. PLSDA-batch estimates latent components related to treatment and batch effects to remove batch variation. The method is multivariate, non-parametric and performs dimension reduction. Combined with centered log ratio transformation for addressing uneven library sizes and compositional structure, PLSDA-batch addresses all characteristics of microbiome data that existing correction methods have ignored so far. I also developed two variants for 1/ unbalanced batch x treatment designs that are commonly encountered in studies with small sample size, and for 2/ selection of discriminative variables to avoid overfitting in classification problems. These two variants have widened the scope of applicability of PLSDA-batch to different data settings [Wang and Lê Cao, 2020].

My package includes both the new method for batch effect correction, along with a comprehensive standardised framework for batch effect management including the application of existing methods ranging from accounting for batch effects (e.g. with linear models) to correcting for batch effects (e.g removeBatchEffect from the *limma* package, and ComBat from *sva*) and my proposed method for microbiome data. My package will benefit researchers who have already generated their data - despite a poor experimental design, to analyse their data appropriately. My package will also benefit well-designed studies to detect batch effects as a quality check to ensure reliable downstream results.

The framework includes microbiome data pre-filtering, transformation and batch effect detection, visualisation, accounting for or correcting for batch effects and assessing batch effect removal and variable selection after batch effect correction. We illustrated our framework with data sequenced using 16S rRNA gene sequencing, but shotgun sequencing data can also be analysed. Our framework guides data analysts in choosing the appropriate analytic method to either account for or correct for batch effects. In addition, batch effects can vary in sources, designs and scale of influence on microbial variables, and different methods for batch effect management have different assumptions. We have illustrated each step with exemplar studies and provided all the functions and packages for reproducibility (see Table 2). The R packages are from the open-source CRAN, Bioconductor projects or Github repositories, therefore facilitates the analysis of any microbial studies. The visualisation can be customised to apply to different results from multiple methods.

1.2 Case studies description

We considered four case studies with different data settings. We illustrated the complete analysis of only one study, and specific analysis steps for the other studies. The complete analyses of the other studies can be found in the next chapter.

The datasets from four case studies are stored internally in our R package PLSDAbatch. Their basic information can be found in Table 1.

Anaerobic digestion. This study explored the microbial indicators that could improve the efficacy of anaerobic digestion (AD) bioprocess and prevent its failure [Chapleur et al., 2016]. The samples were treated with two different ranges of phenol concentration (effect of interest) and processed at five different dates (batch effect). This study included a clear and strong batch effect with an approx. balanced batch x treatment design. Using this study, we illustrated the complete analysis of batch effect management.

Sponge *A. aerophoba*. This study investigated the relationship between metabolite concentration and microbial abundance of specific sponge tissues [Sacristán-Soriano et al., 2011]. The samples were collected from two types of tissues (Ectosome vs. Choanosome) and processed on two separate denaturing gradient gels in electrophoresis. This study included relative abundance data only and a completely balanced batch x treatment design.

High fat high sugar diet. This study aimed to investigate the effect of high fat high sugar (HFHS) diet on the mouse microbiome [Susin et al., 2020]. The samples were collected at different days from the mice treated with two types of diets (HFHS vs. normal) and housed in different cages. This study included a extremely unbalanced (nested) batch x treatment design between cages and diets and an approx. balanced design between days and diets. With this study we illustrated how to deal with weak batch effects.

Mice models with Huntington's disease. This study explored differences in microbial composition between Huntington's disease (HD) and wild-type (WT) mice [Kong et al., 2020]. The samples were collected from the mice with different genotypes and housed in different cages. We illustrated how to manage batch effects with a nested batch x treatment design.

	AD data			Sponge data			HFHS data						HD data		
No. of samples*	75			32			250						30		
No. of OTUs*	567			24			4524						368		
Effects of interest	Phenol concentration			Tissue types			Diet types						Genotypes		
Batch effects	Sample processed dates			Gels in electrophoresis			Sample collected days			Mice cages			Mice cages		
Design type	Approx. balanced			Completely balanced			Approx. balanced			Nested			Nested		
		0-0.5	1-2		Choanosome	Ectosome		HFHS	Normal		HFHS	Normal		HD	WT
	09/04/2015	4	5	Gel1	8	8	A	23	23	Cage1	0	30	CageA	2	0
	14/04/2016	4	12	Gel2	8	8	Day0	29	26	Cage2	33	0	CageB	3	0
	01/07/2016	8	13				Day1	23	24	Cage3	0	30	CageC	2	0
	14/11/2016	8	9				Day4	31	24	Cage4	35	0	CageD	0	4
Batch x treatment design	21/09/2017	2	10				Day7	23	24	Cage5	0	30	CageE	0	4
										Cage6	27	0	CageF	0	3
										Cage7	0	31	CageG	3	0
										Cage8	34	0	CageH	3	0
													CageI	2	0
													CageJ	0	4

Table 1: Overview of case studies with batch effects and their experimental designs. We considered microbial studies for Anaerobic Digestion (AD data), sponge

A. aerophoba (sponge data), mice models with High Fat High Sugar diets (HFHS data) and Huntington's Disease (HD data).

1.3 Packages installation and loading

First, we load the packages necessary for analysis, and check the version of each package.

```
# CRAN
cran.pkgs <- c('pheatmap', 'vegan', 'ruv', 'UpSetR', 'gplots',
               'ggplot2', 'gridExtra', 'performance', 'Rdpack', 'pROC')
# Bioconductor
bioc.pkgs <- c('mixOmics', 'sva', 'limma', 'Biobase', 'metagenomeSeq')
# GitHub
github.pkg <- 'PLSDAbatch'
# devtools::install_github("https://github.com/EvaYiwenWang/PLSDAbatch")

# built-in functions
source(file = './built_in_functions/data_simulation.R')

# load packages
sapply(c(cran.pkgs, bioc.pkgs, github.pkg), require, character.only = TRUE)
```

```
##      pheatmap         vegan           ruv        UpSetR        gplots
##          TRUE          TRUE          TRUE          TRUE          TRUE
##       ggplot2     gridExtra   performance        Rdpack          pROC
##          TRUE          TRUE          TRUE          TRUE          TRUE
##      mixOmics           sva         limma       Biobase metagenomeSeq
##          TRUE          TRUE          TRUE          TRUE          TRUE
##    PLSDAbatch
##          TRUE
```

```
# print package versions
sapply(c(cran.pkgs, bioc.pkgs, github.pkg), package.version)
```

```
##      pheatmap         vegan           ruv        UpSetR        gplots
##      "1.0.12"       "2.6-4"     "0.9.7.1"       "1.4.0"       "3.1.3"
##       ggplot2     gridExtra   performance        Rdpack          pROC
##       "3.3.6"         "2.3"       "0.9.2"         "2.4"      "1.18.0"
##      mixOmics           sva         limma       Biobase metagenomeSeq
##      "6.20.0"      "3.44.0"      "3.52.2"      "2.56.0"      "1.38.0"
##    PLSDAbatch
##       "0.2.3"
```

1.4 Data pre-processing

1.4.1 Pre-filtering

We load the AD data stored internally with function `data()`.

```
# AD data
data('AD_data')
ad.count <- AD_data$FullData$X.count
dim(ad.count)
```

```
## [1]  75 567
```

The raw AD data include 567 OTUs and 75 samples. We then use the function `PreFL()` from our PLSDAbatch R package to filter the data.

```
ad.filter.res <- PreFL(data = ad.count)
ad.filter <- ad.filter.res$data.filter
dim(ad.filter)
```

```
## [1]  75 231
```

```
# zero proportion before filtering
ad.filter.res$zero.prob
```

```
## [1] 0.6328042
```

```
# zero proportion after filtering
sum(ad.filter == 0)/(nrow(ad.filter) * ncol(ad.filter))
```

```
## [1] 0.3806638
```

After filtering, 231 OTUs remained, and the proportion of zeroes decreased from 63% to 38%.

We can also load the other data, such as the sponge data.

```
# Sponge data
data('sponge_data')
sponge.tss <- sponge_data$X.tss
dim(sponge.tss)
```

```
## [1] 32 24
```

```
# zero proportion
sum(sponge.tss == 0)/(nrow(sponge.tss) * ncol(sponge.tss))
```

```
## [1] 0.5455729
```

The sponge data include the relative abundance of 24 OTUs and 32 samples. Given the small number of OTUs, we advise not to pre-filter the data.

Note: The `PreFL()` function is only dedicated to raw counts, rather than relative abundance data. We also recommend to start the pre-filtering on raw counts, rather than relative abundance data to mitigate the compositionality issue.

1.4.2 Transformation

Prior to CLR transformation, we recommend adding 1 as the offset for the AD data - that are raw count data, and 0.01 as the offset for the sponge data - that are relative abundance data. We use `logratio.transfo()` function in mixOmics package to CLR transform the data.

```
ad.clr <- logratio.transfo(X = ad.filter, logratio = 'CLR', offset = 1)
class(ad.clr) = 'matrix'

sponge.clr <- logratio.transfo(X = sponge.tss, logratio = 'CLR', offset = 0.01)
class(sponge.clr) = 'matrix'
```

1.5 Batch effect detection

1.5.1 PCA

We apply `pca()` function from mixOmics package to the AD data and `Scatter_Density()` function from PLSDAbatch to represent the PCA sample plot with densities.

```
# AD data
ad.pca.before <- pca(ad.clr, ncomp = 3, scale = TRUE)

ad.metadata <- AD_data$FullData$metadata
ad.batch = factor(ad.metadata$sequencing_run_date,
                  levels = unique(ad.metadata$sequencing_run_date))
ad.trt = as.factor(ad.metadata$initial_phenol_concentration.regroup)
names(ad.batch) <- names(ad.trt) <- rownames(ad.metadata)

Scatter_Density(object = ad.pca.before, batch = ad.batch, trt = ad.trt,
                title = 'AD data', trt.legend.title = 'Phenol conc.')
```

In the above figure, we observed 1) the distinction between samples treated with different phenol concentrations and 2) the differences between samples sequenced at "14/04/2016", "21/09/2017" and the other dates. Therefore, the batch effect related to dates needs to be removed.

1.5.2 Boxplots and density plots

We first identify the top OTU driving the major variance in PCA using `selectVar()` in mixOmics package. Each identified OTU can then be plotted as boxplots and density plots using `box_plot()` and `density_plot()` in PLSDAbatch.

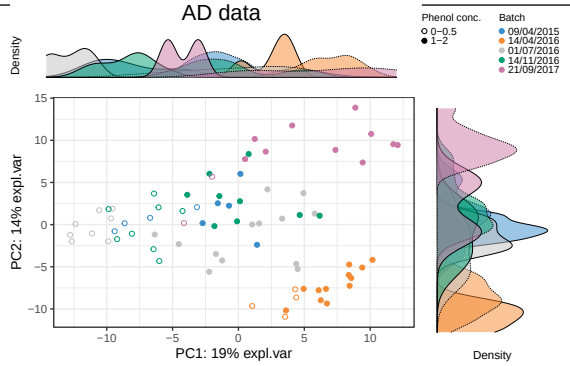

Figure 1.1: The PCA sample plot with densities in the AD data.

```
ad.OTU.name <- selectVar(ad.pca.before, comp = 1)$name[1]
ad.OTU_batch <- data.frame(value = ad.clr[,ad.OTU.name], batch = ad.batch)
box_plot(df = ad.OTU_batch, title = paste(ad.OTU.name, '(AD data)'),
         x.angle = 30)
```

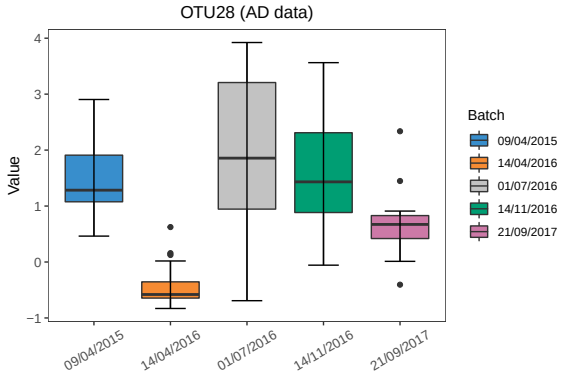

Figure 1.2: Boxplots of sample values in ”OTU28” before batch effect correction in the AD data.

```
density_plot(df = ad.OTU_batch, title = paste(ad.OTU.name, '(AD data)'))
```

The boxplot and density plot indicated a strong date batch effect because of the differences between “14/04/2016”, “21/09/2017” and the other dates in the “OTU28”.

We also apply a linear regression model to the “OTU28” using `linear_regres()` from PLSDAbatch with batch and treatment effects as covariates. We set “14/04/2016” and “21/09/2017” as the reference batch respectively with `relevel()` from stats.

```
# reference batch: 14/04/2016
ad.batch <- relevel(x = ad.batch, ref = '14/04/2016')
```

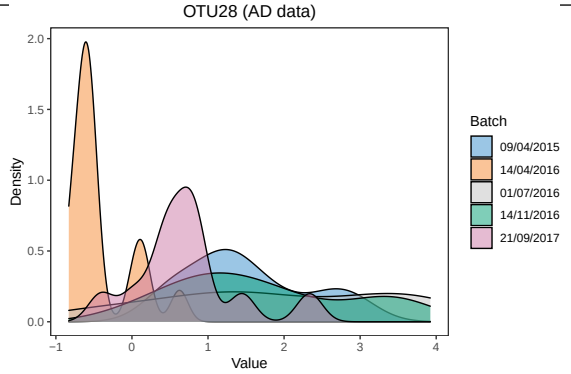

Figure 1.3: Density plots of sample values in ”OTU28” before batch effect correction in the AD data.

```
ad.OTU.lm <- linear_regres(data = ad.clr[,ad.OTU.name],
                           trt = ad.trt, batch.fix = ad.batch,
                           type = 'linear model')
summary(ad.OTU.lm$model$data)
```

```
##
## Call:
## lm(formula = data[, i] ~ trt + batch.fix)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9384 -0.3279  0.1635  0.3849  0.9887
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)            0.8501     0.2196   3.871 0.000243 ***
## trt1-2                -1.6871     0.1754  -9.617 2.27e-14 ***
## batch.fix09/04/2015    1.5963     0.2950   5.410 8.55e-07 ***
## batch.fix01/07/2016    2.0839     0.2345   8.886 4.82e-13 ***
## batch.fix14/11/2016    1.7405     0.2480   7.018 1.24e-09 ***
## batch.fix21/09/2017    1.2646     0.2690   4.701 1.28e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7033 on 69 degrees of freedom
## Multiple R-squared:  0.7546, Adjusted R-squared:  0.7368
## F-statistic: 42.44 on 5 and 69 DF,  p-value: < 2.2e-16
```

```
# reference batch: 21/09/2017
ad.batch <- relevel(x = ad.batch, ref = '21/09/2017')

ad.OTU.lm <- linear_regres(data = ad.clr[,ad.OTU.name],
                           trt = ad.trt, batch.fix = ad.batch,
                           type = 'linear model')
summary(ad.OTU.lm$model$data)
```

```
##
## Call:
## lm(formula = data[, i] ~ trt + batch.fix)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9384 -0.3279  0.1635  0.3849  0.9887
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)            2.1147     0.2502   8.453 2.97e-12 ***
## trt1-2                -1.6871     0.1754  -9.617 2.27e-14 ***
## batch.fix14/04/2016   -1.2646     0.2690  -4.701 1.28e-05 ***
## batch.fix09/04/2015    0.3317     0.3139   1.056  0.29446
## batch.fix01/07/2016    0.8193     0.2573   3.185  0.00218 **
## batch.fix14/11/2016    0.4759     0.2705   1.760  0.08292 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7033 on 69 degrees of freedom
## Multiple R-squared:  0.7546, Adjusted R-squared:  0.7368
## F-statistic: 42.44 on 5 and 69 DF,  p-value: < 2.2e-16
```

From the results of linear regression, we observed $P < 0.001$ for the regression coefficients associated with all the other batches when the reference batch was "14/04/2016", which confirmed the difference between the samples from batch "14/04/2016" and the other samples as observed from previous plots. When the reference batch was "21/09/2017", we also observed significant differences between batch "21/09/2017" and "14/04/2016", between "21/09/2017" and "01/07/2016". Therefore, the batch effect because of "21/09/2017" also exists.

1.5.3 Heatmap

We produce a heatmap using pheatmap package. The data first need to be scaled on both OTUs and samples.

```
# scale the clr data on both OTUs and samples
ad.clr.s <- scale(ad.clr, center = TRUE, scale = TRUE)
```

```
ad.clr.ss <- scale(t(ad.clr.s), center = TRUE, scale = TRUE)

ad.anno_col <- data.frame(Batch = ad.batch, Treatment = ad.trt)
ad.anno_colors <- list(Batch = color.mixo(seq_len(5)),
                       Treatment = pb_color(seq_len(2)))
names(ad.anno_colors$Batch) = levels(ad.batch)
names(ad.anno_colors$Treatment) = levels(ad.trt)

pheatmap(ad.clr.ss,
         cluster_rows = FALSE,
         fontsize_row = 4,
         fontsize_col = 6,
         fontsize = 8,
         clustering_distance_rows = 'euclidean',
         clustering_method = 'ward.D',
         treeheight_row = 30,
         annotation_col = ad.anno_col,
         annotation_colors = ad.anno_colors,
         border_color = 'NA',
         main = 'AD data - Scaled')
```

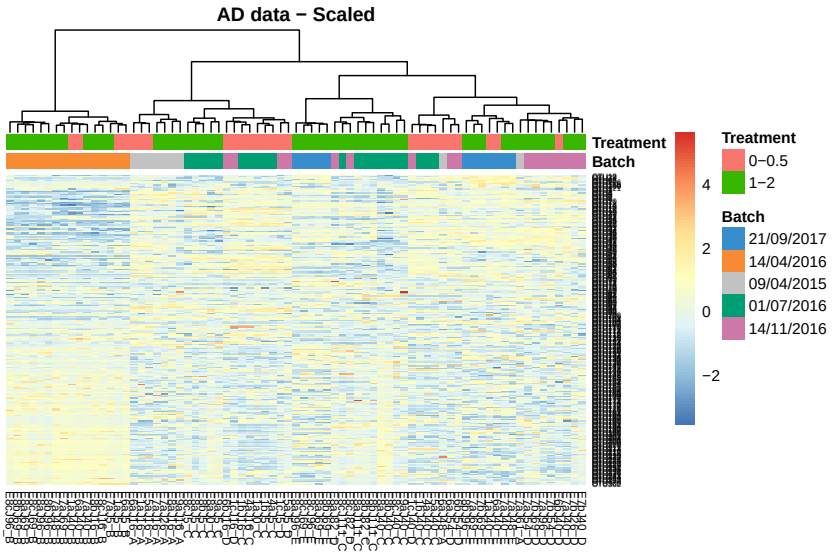

Figure 1.4: Hierarchical clustering for samples in the AD data.

In the heatmap, samples in the AD data from batch dated "14/04/2016" were clustered and distinct from other samples, indicating a batch effect.

1.5.4 pRDA

We apply pRDA with `varpart()` function from vegan R package.

```
# AD data
ad.factors.df <- data.frame(trt = ad.trt, batch = ad.batch)
class(ad.clr) <- 'matrix'
ad.rda.before <- varpart(ad.clr, ~ trt, ~ batch,
                         data = ad.factors.df, scale = TRUE)
ad.rda.before$part$indfract
```

```
##                  Df R.squared Adj.R.squared Testable
## [a] = X1|X2       1        NA    0.08943682     TRUE
## [b] = X2|X1       4        NA    0.26604420     TRUE
## [c]               0        NA    0.01296248    FALSE
## [d] = Residuals NA         NA    0.63155651    FALSE
```

In the result, `X1` and `X2` represent the first and second covariates fitted in the model. `[a]`, `[b]` represent the independent proportion of variance explained by `X1` and `X2` respectively, and `[c]` represents the intersection variance shared between `X1` and `X2`. In the AD data, batch variance (`X2`) was larger than treatment variance (`X1`) with some interaction proportion (indicated in line `[c]`, Adj.R.squared = 0.013). The greater the intersection variance, the more unbalanced batch x treatment design is. In this study, we considered the design as approx. balanced.

```
# Sponge data
sponge.batch <- sponge_data$Y.bat
sponge.trt <- sponge_data$Y.trt

sponge.factors.df <- data.frame(trt = sponge.trt, batch = sponge.batch)
class(sponge.clr) <- 'matrix'
sponge.rda.before <- varpart(sponge.clr, ~ trt, ~ batch,
                             data = sponge.factors.df, scale = TRUE)
sponge.rda.before$part$indfract
```

```
##                  Df R.squared Adj.R.squared Testable
## [a] = X1|X2       1        NA    0.16572246     TRUE
## [b] = X2|X1       1        NA    0.16396277     TRUE
## [c]               0        NA   -0.01063501    FALSE
## [d] = Residuals NA         NA    0.68094977    FALSE
```

The sponge data have a completely balanced batch x treatment design, so there was no intersection variance (indicated in line `[c]`, Adj.R.squared = -0.01) detected. The proportion of treatment and batch variance is nearly equal as indicated in lines `[a]` and `[b]`.

```
# HD data
data('HD_data')
hd.clr <- HD_data$EgData$X.clr
```

```
hd.trt <- HD_data$EgData$Y.trt
hd.batch <- HD_data$EgData$Y.bat

hd.factors.df <- data.frame(trt = hd.trt, batch = hd.batch)
class(hd.clr) <- 'matrix'
hd.rda.before <- varpart(hd.clr, ~ trt, ~ batch,
                         data = hd.factors.df, scale = TRUE)
hd.rda.before$part$indfract
```

```
##                   Df R.squared Adj.R.squared Testable
## [a] = X1|X2        0        NA -2.220446e-16    FALSE
## [b] = X2|X1        8        NA  1.608205e-01     TRUE
## [c]                0        NA  9.730583e-02    FALSE
## [d] = Residuals   NA        NA  7.418737e-01    FALSE
```

Collinearity was detected in the HD data between treatment and batch as indicated by lines `[a]` (Adj.R.squared = 0) and `[c]` (Adj.R.squared = 0.097) that all treatment variance was explained as intersection variance, because the batch x treatment design is nested. The intersection variance in such a design is usually considerable. As the intersection is shared between batch and treatment effects, the batch variance in the HD data was larger than the treatment variance.

1.6 Managing batch effects

1.6.1 Accounting for batch effects

The methods that we use to account for batch effects include the methods designed for microbiome data: zero-inflated Gaussian (ZIG) mixture model and the methods adapted for microbiome data: linear regression, SVA and RUV4. Among them, SVA and RUV4 were designed for unknown batch effects.

1.6.1.1 Methods designed for microbiome data

Zero-inflated Gaussian mixture model To use the ZIG model, we first create a `MRexperiment` object applying `newMRexperiment()` (from metagenomeSeq package) to microbiome counts and annotated data frames with metadata and taxonomic information generated with `AnnotatedDataFrame()` from Biobase package.

```
# Creating a MRexperiment object (make sure no NA in metadata)
AD.phenotypeData = AnnotatedDataFrame(data = AD_data$FullData$metadata)
AD.taxaData = AnnotatedDataFrame(data = AD_data$FullData$taxa)
AD.obj = newMRexperiment(counts = t(AD_data$FullData$X.count),
                         phenoData = AD.phenotypeData,
                         featureData = AD.taxaData)
AD.obj
```

```
## MRexperiment (storageMode: environment)
## assayData: 567 features, 75 samples
##   element names: counts
## protocolData: none
## phenoData
##   sampleNames: E1aJ16_A E5aJ16_A ... E8cJ96_E (75 total)
##   varLabels: sample_name.data.extraction analysis_name ...
##     initial_phenol_concentration.regroup (7 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: OTU12 OTU29 ... OTU17710 (567 total)
##   fvarLabels: Kingdom Phylum ... Species (7 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

The AD count data are then filtered with `filterData()` function (from metagenomeSeq). We can use `MRcounts()` to extract the count data from the `MRexperiment` object.

```
# filtering data to maintain a threshold of minimum depth or OTU presence
dim(MRcounts(AD.obj))
```

```
## [1] 567  75
```

```
AD.obj = filterData(obj = AD.obj, present = 20, depth = 5)
dim(MRcounts(AD.obj))
```

```
## [1] 289  75
```

After filtering, the AD count data were reduced to 289 OTUs and 75 samples.

We calculate the percentile for CSS normalisation with `cumNormStatFast()` function (from metagenomeSeq package). The CSS normalisation is applied with `cumNorm()` and the normalised data can be exported using `MRcounts()` with `norm = TRUE`. The normalisation scaling factors for each sample, which are the sum of counts up to the calculated percentile, can be accessed through `normFactors()`. We calculate the log transfomed scaling factors by diving them with their median, which are better than the default scaling factors that are divided by 1000 (`log2(normFactors(obj)/1000 + 1)`).

```
# calculate the percentile for CSS normalisation
AD.pctl = cumNormStatFast(obj = AD.obj)
# CSS normalisation
AD.obj <- cumNorm(obj = AD.obj, p = AD.pctl)
# export normalised data
AD.norm.data <- MRcounts(obj = AD.obj, norm = TRUE)

# normalisation scaling factors for each sample
AD.normFactor = normFactors(object = AD.obj)
```

```
AD.normFactor = log2(AD.normFactor/median(AD.normFactor) + 1)
```

We create a design matrix with treatment variable (`phenol_conc`), batch variable (`seq_run`) and the log transformed scaling factors using `model.matrix()`, and then apply the ZIG model by `fitZig()` function. We set `useCSSoffset = FALSE` to avoid using the default scaling factors as we have already included our customised scaling factor (`AD.normFactor`) in the design matrix.

```
# treatment variable
phenol_conc = pData(object = AD.obj)$initial_phenol_concentration.regroup
# batch variable
seq_run = pData(object = AD.obj)$sequencing_run_date

# build a design matrix
AD.mod.full = model.matrix(~ phenol_conc + seq_run + AD.normFactor)

# settings for the fitZig() function
AD.settings <- zigControl(maxit = 10, verbose = TRUE)

# apply the ZIG model
ADfit <- fitZig(obj = AD.obj, mod = AD.mod.full,
                useCSSoffset = FALSE, control = AD.settings)
```

```
## it= 0, nll=123.44, log10(eps+1)=Inf, stillActive=289
## it= 1, nll=134.33, log10(eps+1)=0.04, stillActive=10
## it= 2, nll=134.64, log10(eps+1)=0.01, stillActive=1
## it= 3, nll=134.80, log10(eps+1)=0.00, stillActive=0
```

The OTUs with the top 50 smallest p values are extracted using `MRcoefs()`. We set `eff = 0.5`, so only the OTUs with at least "0.5" quantile (50%) number of effective samples (positive samples + estimated undersampling zeroes) are extracted.

```
ADcoefs <- MRcoefs(ADfit, coef = 2, group = 3, number = 50, eff = 0.5)
head(ADcoefs)
```

```
##       phenol_conc1-2      pvalues   adjPvalues
## OTU68      -2.901041 2.467451e-15 4.081959e-13
## OTU46      -2.847212 2.824885e-15 4.081959e-13
## OTU28      -2.449290 4.012023e-14 3.864915e-12
## OTU24      -2.846452 6.397959e-14 4.622526e-12
## OTU59      -1.815250 3.784889e-13 2.187666e-11
## OTU65      -2.141183 3.342621e-11 1.610029e-09
```

1.6.1.2 Other methods adapted for microbiome data

Linear regression Linear regression is conducted with `linear_regres()` function in PLSDAbatch. We integrated the performance package that assesses performance of regression models into our function `linear_regres()`. Therefore, we can apply

`check_model()` from performance to the outputs from `linear_regres()` to diagnose the validity of the model fitted with treatment and batch effects for each variable [LÃŒdecke et al., 2020]. We can extract performance measurements such as adjusted R2, RMSE, RSE, AIC and BIC for the models fitted with and without batch effects, which are also the outputs of `linear_regres()`.

We apply `type = "linear model"` to the AD data because of the balanced batch x treatment design.

```
# AD data
ad.clr <- ad.clr[seq_len(nrow(ad.clr)), seq_len(ncol(ad.clr))]
ad.lm <- linear_regres(data = ad.clr, trt = ad.trt,
                       batch.fix = ad.batch, type = 'linear model')

ad.p <- sapply(ad.lm$lm.table, function(x){x$coefficients[2,4]})
ad.p.adj <- p.adjust(p = ad.p, method = 'fdr')

check_model(ad.lm$model$OTU12)
```

To diagnose the validity of the model fitted with both treatment and batch effects, we use different plots to check the assumptions of each microbial variable. For example, the diagnostic plots of "OTU12" are shown in the above figure panel. The simulated data under the fitted model were close to the real data (shown as green line), indicating good model fitness (top panel). The linearity (or homoscedasticity) and homogeneity of variance were not satisfied. The correlation between batch (`batch.fix`) and treatment (`trt`) effects was very low, indicating a good model with low collinearity. Some samples could be classified as outliers with a Cook's distance larger than or equal to 0.5, for example, "57", "39", "47", "44" and "16" (middle panel). The distribution of residuals was very close to normal (bottom panel). For the microbial variables with some assumptions not met, we should be careful about their results.

For performance measurements of models fitted with or without batch effects, We show an example of the results for some variables.

```
head(ad.lm$adj.R2)
```

```
##            trt.only trt.batch
## OTU12   0.008445545 0.4785532
## OTU29  -0.012966020 0.7808945
## OTU77  -0.013343146 0.4779754
## OTU86   0.183743803 0.2850153
## OTU97  -0.008989284 0.4502563
## OTU106  0.072963581 0.8745676
```

The adjusted R^2 of the model with both treatment and batch effects for all the OTUs listed was larger than the model with treatment effects only, suggesting that the model fitted with batch effects explained more data variance, and was thus better than the model without batch effects.

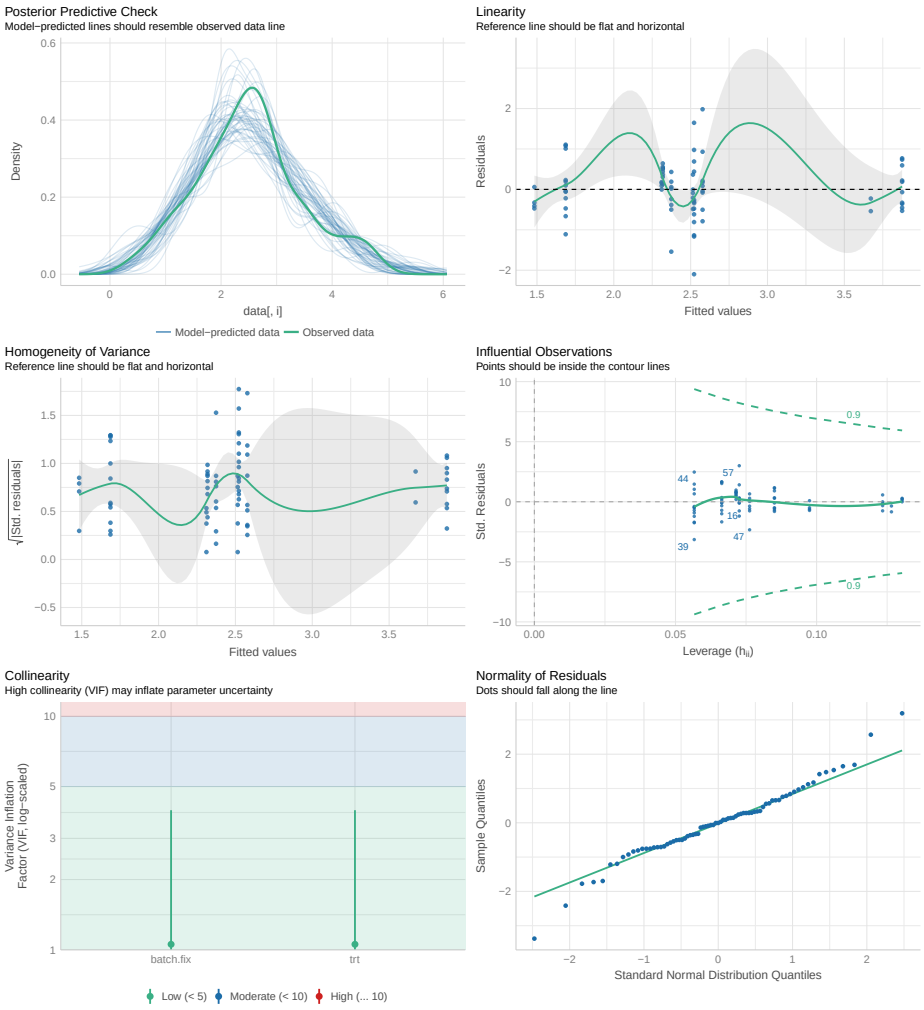

Figure 1.5: Diagnostic plots for the model fitted with batch effects of ”OTU12” in the AD data.

We next look at the AIC of models fitted with or without batch effects.

```
head(ad.lm$AIC)
```

```
##         trt.only trt.batch
## OTU12  208.5626 164.13617
## OTU29  266.0485 154.99072
## OTU77  180.7800 134.80638
## OTU86  212.2987 206.13722
## OTU97  161.6715 119.90108
## OTU106 205.4135  59.17005
```

A lower AIC indicates a better fit, here for a model fitted with batch effects for all the OTUs.

Both results strongly indicated that a batch effect should be fitted in the linear model.

We apply a `"linear mixed model"` to the HD data because of the nested batch x treatment design.

```
# HD data
hd.lmm <- linear_regres(data = hd.clr, trt = hd.trt,
                        batch.random = hd.batch,
                        type = 'linear mixed model')
```

```
hd.p <- sapply(hd.lmm$lmm.table, function(x){x$coefficients[2,5]})
hd.p.adj <- p.adjust(p = hd.p, method = 'fdr')

check_model(hd.lmm$model$OTU1)
```

According to the diagnostic plots of "OTU1" as shown in the above figure panel, the simulated data under the fitted model were not very close to the real data (shown as green line), indicating an imperfect fitness (top panel). The linearity (or homoscedasticity) and homogeneity of variance were not satisfied. Some samples could be classified as outliers with a Cook's distance larger than or equal to 0.5, for example, "12", "21", "20", "11" and "6" (middle panel). The distribution of residuals was not normal, while the one of random effects was very close to normal (bottom panel).

We then look at the AIC of models fitted with or without batch effects.

```
head(hd.lmm$AIC)
```

```
##       trt.only trt.batch
## OTU1  60.00637  65.46410
## OTU2 130.14167 130.96344
## OTU3  71.76985  75.58915
## OTU4 134.59992 134.04812
## OTU5 150.25059 148.30644
## OTU6  99.92864  99.05391
```

For some OTUs, the model fitted without batch effects was better with a lower AIC,

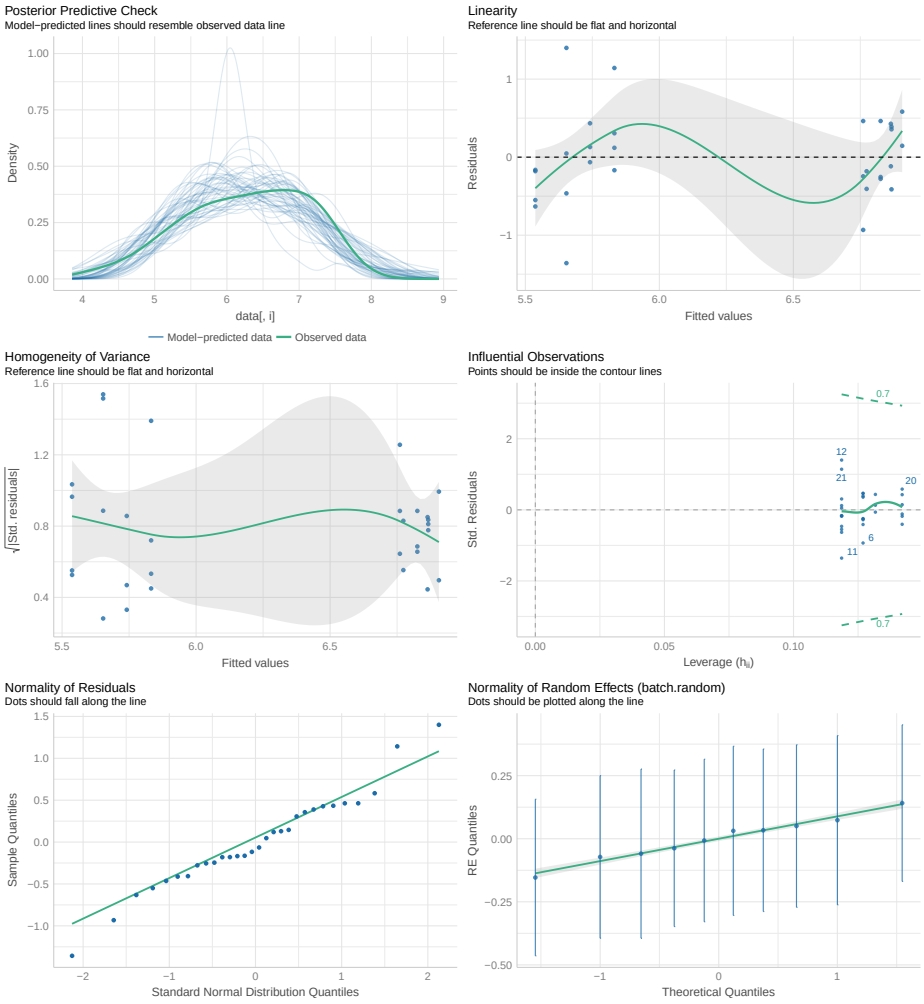

Figure 1.6: Diagnostic plots for the model fitted with batch effects of ”OTU1” in the HD data.

e.g., "OTU1", "OTU2" and "OTU3". "OTU4", "OTU5" and "OTU6" were more appropriate within a model with batch effects.

Since we apply a `"linear mixed model"` to the HD data, this type of model can only output conditional R^2 that includes the variance of both fixed and random effects (treatment, fixed and random batch effects) and marginal R^2 that includes only the variance of fixed effects (treatment and fixed batch effects) [Nakagawa and Schielzeth, 2013]. The marginal R^2s of the models with and without the batch effect in the HD data should be the same theoretically, as marginal R^2 only includes fixed effects and the treatment effect was the only fixed effect here. In the actual calculation, the results of these two models were not the same, because part of the variance was explained by the random effect in the model fitted with the batch effect.

```
head(hd.lmm$cond.R2)
```

```
##        trt.only trt.batch
## OTU1 0.47917775 0.5161317
## OTU2 0.39223824 0.4729848
## OTU3 0.02091376 0.2636432
## OTU4 0.37200835 0.5672392
## OTU5 0.07858373 0.2938212
## OTU6 0.41388250 0.6143134
```

```
head(hd.lmm$marg.R2)
```

```
##        trt.only  trt.batch
## OTU1 0.47917775 0.46735273
## OTU2 0.39223824 0.38459672
## OTU3 0.02091376 0.01489608
## OTU4 0.37200835 0.36122754
## OTU5 0.07858373 0.06052938
## OTU6 0.41388250 0.40749149
```

We observed that some variables resulted in singular fits (`error message: Can't compute random effect variances. Some variance components equal zero. Your model may suffer from singulariy.`), which are expected to happen in linear mixed models when covariates are nested. We recommend noting the variables for which this error occurs, as it may lead to unreliable results.

SVA accounts for unknown batch effects. Here we assume that the batch grouping information in the AD data is unknown. We first build two design matrices with (`ad.mod`) and without (`ad.mod0`) treatment grouping information generated with `model.matrix()` function from stats. We then use `num.sv()` from sva package to determine the number of batch variables `n.sv` that is used to estimate batch effects in function `sva()`.

```
# estimate batch effects
ad.mod <- model.matrix( ~ ad.trt)
ad.mod0 <- model.matrix( ~ 1, data = ad.trt)
```

```
ad.sva.n <- num.sv(dat = t(ad.clr), mod = ad.mod, method = 'leek')
ad.sva <- sva(t(ad.clr), ad.mod, ad.mod0, n.sv = ad.sva.n)
```

```
## Number of significant surrogate variables is:  4
## Iteration (out of 5 ):1  2  3  4  5
```

The estimated batch effects are then applied to `f.pvalue()` to calculate the P-values of treatment effects. The estimated batch effects in SVA are assumed to be independent of the treatment effects. However, SVA considers some correlation between batch and treatment effects [Wang and LêCao, 2020].

```
# include estimated batch effects in the linear model
ad.mod.batch <- cbind(ad.mod, ad.sva$sv)
ad.mod0.batch <- cbind(ad.mod0, ad.sva$sv)
ad.sva.p <- f.pvalue(t(ad.clr), ad.mod.batch, ad.mod0.batch)
ad.sva.p.adj <- p.adjust(ad.sva.p, method = 'fdr')
```

RUV4 Before applying RUV4 (`RUV4()` from ruv package), we need to specify negative control variables and the number of batch variables to estimate. We can use the empirical negative controls that are not significantly differentially abundant (adjusted P > 0.05) from a linear regression with the treatment information as the only covariate.

We use a loop to fit a linear regression for each microbial variable and adjust P values of treatment effects for multiple comparisons with `p.adjust()` from stats. The empirical negative controls are then extracted according to the adjusted P values.

```
# empirical negative controls
ad.empir.p <- c()
for(e in seq_len(ncol(ad.clr))){
  ad.empir.lm <- lm(ad.clr[,e] ~ ad.trt)
  ad.empir.p[e] <- summary(ad.empir.lm)$coefficients[2,4]
}
ad.empir.p.adj <- p.adjust(p = ad.empir.p, method = 'fdr')
ad.nc <- ad.empir.p.adj > 0.05
```

The number of batch variables `k` can be determined using `getK()` function.

```
# estimate k
ad.k.res <- getK(Y = ad.clr, X = ad.trt, ctl = ad.nc)
ad.k <- ad.k.res$k
```

We then apply `RUV4()` with known treatment variables, estimated negative control variables and `k` batch variables. The calculated P values also need to be adjusted for multiple comparisons.

```
# RUV4
ad.ruv4 <- RUV4(Y = ad.clr, X = ad.trt, ctl = ad.nc, k = ad.k)
ad.ruv4.p <- ad.ruv4$p
ad.ruv4.p.adj <- p.adjust(ad.ruv4.p, method = "fdr")
```

Note: A package named RUVSeq has been developed for count data. It provides `RUVg()` using negative control variables, and also other functions `RUVs()` and `RUVr()` using sample replicates [Moskovicz et al., 2020] or residuals from the regression on treatment effects to estimate and then account for latent batch effects. However, for CLR-transformed data, we still recommend the standard ruv package.

1.6.2 Correcting for batch effects

The methods that we use to correct for batch effects include removeBatchEffect, ComBat, PLSDA-batch, sPLSDA-batch, Percentile Normalisation and RUVIII. Among them, RUVIII was designed for unknown batch effects. They were all applied to and illustrated with the AD data.

removeBatchEffect

The `removeBatchEffect()` function is implemented in limma package. The design matrix (`design`) with treatment grouping information can be generated with `model.matrix()` function from stats as shown in section **accounting for batch effects** method "SVA".

Here we use `removeBatchEffect()` function with batch grouping information (`batch`) and treatment design matrix (`design`) to calculate batch effect corrected data `ad.rBE`.

```
ad.rBE <- t(removeBatchEffect(t(ad.clr), batch = ad.batch,
                              design = ad.mod))
```

ComBat

The `ComBat()` function (from sva package) is implemented as parametric or non-parametric correction with option `par.prior`. Under a parametric adjustment, we can assess the model's validity with `prior.plots = T` [Leek et al., 2012].

Here we use a non-parametric correction (`par.prior = FALSE`) with batch grouping information (`batch`) and treatment design matrix (`mod`) to calculate batch effect corrected data `ad.ComBat`.

```
ad.ComBat <- t(ComBat(t(ad.clr), batch = ad.batch,
                      mod = ad.mod, par.prior = FALSE))
```

PLSDA-batch

The `PLSDA_batch()` function is implemented in PLSDAbatch package. To use this function, we need to specify the optimal number of components related to treatment (`ncomp.trt`) or batch effects (`ncomp.bat`).

Here in the AD data, we use `plsda()` from mixOmics with only treatment grouping information to estimate the optimal number of treatment components to preserve.

```
# estimate the number of treatment components
ad.trt.tune <- plsda(X = ad.clr, Y = ad.trt, ncomp = 5)
ad.trt.tune$prop_expl_var #1
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5
## 0.18619506 0.07873817 0.08257396 0.09263497 0.06594757
##
## $Y
##      comp1      comp2      comp3      comp4      comp5
## 1.00000000 0.33857374 0.17315267 0.10551296 0.08185822
```

We choose the number that explains 100% variance in the outcome matrix `Y`, thus from the result, 1 component was enough to preserve the treatment information.

We then use `PLSDA_batch()` function with both treatment and batch grouping information to estimate the optimal number of batch components to remove.

```
# estimate the number of batch components
ad.batch.tune <- PLSDA_batch(X = ad.clr,
                             Y.trt = ad.trt, Y.bat = ad.batch,
                             ncomp.trt = 1, ncomp.bat = 10)
ad.batch.tune$explained_variance.bat #4
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5      comp6      comp7
## 0.17470922 0.11481264 0.10122717 0.07507395 0.03946605 0.03399731 0.03784224
##      comp8      comp9     comp10
## 0.02313026 0.02395840 0.01389336
##
## $Y
##       comp1       comp2       comp3       comp4       comp5       comp6
## 0.247465374 0.261574081 0.230138238 0.260822307 0.230056326 0.246345583
##       comp7       comp8       comp9      comp10
## 0.267963500 0.252208822 0.003425768 0.240095146
```

```
sum(ad.batch.tune$explained_variance.bat$Y[seq_len(4)])
```

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

Using the same criterion as choosing treatment components, we choose the number of batch components that explains 100% variance in the outcome matrix of batch. According to the result, 4 components were required to remove batch effects.

We then can correct for batch effects applying `PLSDA_batch()` with treatment, batch grouping information and corresponding optimal number of related components.

```
ad.PLSDA_batch.res <- PLSDA_batch(X = ad.clr,
                                  Y.trt = ad.trt, Y.bat = ad.batch,
```

```
                                ncomp.trt = 1, ncomp.bat = 4)
ad.PLSDA_batch <- ad.PLSDA_batch.res$X.nobatch
```

sPLSDA-batch

We apply sPLSDA-batch using the same function `PLSDA_batch()`, but we specify the number of variables to select on each component (usually only treatment-related components `keepX.trt`). To determine the optimal number of variables to select, we use `tune.splsda()` function from mixOmics package [Rohart et al., 2017] with all possible numbers of variables to select for each component (`test.keepX`).

```
# estimate the number of variables to select per treatment component
set.seed(777)
ad.test.keepX = c(seq(1, 10, 1), seq(20, 100, 10),
                  seq(150, 231, 50), 231)
ad.trt.tune.v <- tune.splsda(X = ad.clr, Y = ad.trt,
                             ncomp = 1, test.keepX = ad.test.keepX,
                             validation = 'Mfold', folds = 4,
                             nrepeat = 50)
ad.trt.tune.v$choice.keepX #100
```

Here the optimal number of variables to select for the treatment component was 100. Since we have adjusted the amount of treatment variation to preserve, we need to re-choose the optimal number of components related to batch effects using the same criterion mentioned in section **correcting for batch effects** method "PLSDA-batch".

```
# estimate the number of batch components
ad.batch.tune <- PLSDA_batch(X = ad.clr,
                             Y.trt = ad.trt, Y.bat = ad.batch,
                             ncomp.trt = 1, keepX.trt = 100,
                             ncomp.bat = 10)
ad.batch.tune$explained_variance.bat #4
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5      comp6      comp7
## 0.17420018 0.11477097 0.09813477 0.07894965 0.02946813 0.03702255 0.03195744
##      comp8      comp9     comp10
## 0.04098959 0.01559967 0.01584227
##
## $Y
##      comp1      comp2      comp3      comp4      comp5      comp6      comp7
## 0.24747749 0.26067155 0.23010809 0.26174286 0.26062893 0.26282635 0.24125867
##      comp8      comp9     comp10
## 0.22162763 0.01365842 0.25965197
```

```
sum(ad.batch.tune$explained_variance.bat$Y[seq_len(4)])
```

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

According to the result, we needed 4 batch related components to remove batch variance from the data with function `PLSDA_batch()`.

```
ad.sPLSDA_batch.res <- PLSDA_batch(X = ad.clr,
                                   Y.trt = ad.trt, Y.bat = ad.batch,
                                   ncomp.trt = 1, keepX.trt = 100,
                                   ncomp.bat = 4)
ad.sPLSDA_batch <- ad.sPLSDA_batch.res$X.nobatch
```

Note: for unbalanced batch x treatment design (with the exception of the nested design), we can specify `balance = FALSE` in `PLSDA_batch()` function to apply weighted PLSDA-batch.

Percentile Normalisation

To apply `percentile_norm()` function from PLSDAbatch package, we need to indicate a control group (`ctrl.grp`).

```
ad.PN <- percentile_norm(data = ad.clr, batch = ad.batch,
                         trt = ad.trt, ctrl.grp = '0-0.5')
```

RUVIII

The `RUVIII()` function is from ruv package. Similar to `RUV4()`, we use empirical negative control variables and `getK()` to determine the number of batch variables (`k`) to estimate. We also need sample replicates which should be structured into a mapping matrix using `replicate.matrix()`. We can then obtain the batch effect corrected data applying `RUVIII()` with above elements.

```
ad.replicates <- ad.metadata$sample_name.data.extraction
ad.replicates.matrix <- replicate.matrix(ad.replicates)

ad.RUVIII <- RUVIII(Y = ad.clr, M = ad.replicates.matrix,
                    ctl = ad.nc, k = ad.k)
rownames(ad.RUVIII) <- rownames(ad.clr)
```

1.7 Assessing batch effect correction

We apply different visualisation and quantitative methods to assessing batch effect correction.

1.7.1 Methods that detect batch effects

PCA

In the AD data, we compared the PCA sample plots before and after batch effect correction with different methods.

```
ad.pca.before <- pca(ad.clr, ncomp = 3, scale = TRUE)
ad.pca.rBE <- pca(ad.rBE, ncomp = 3, scale = TRUE)
ad.pca.ComBat <- pca(ad.ComBat, ncomp = 3, scale = TRUE)
ad.pca.PLSDA_batch <- pca(ad.PLSDA_batch, ncomp = 3, scale = TRUE)
ad.pca.sPLSDA_batch <- pca(ad.sPLSDA_batch, ncomp = 3, scale = TRUE)
ad.pca.PN <- pca(ad.PN, ncomp = 3, scale = TRUE)
ad.pca.RUVIII <- pca(ad.RUVIII, ncomp = 3, scale = TRUE)
```

```
# order batches
ad.batch = factor(ad.metadata$sequencing_run_date,
                  levels = unique(ad.metadata$sequencing_run_date))

ad.pca.before.plot <- Scatter_Density(object = ad.pca.before,
                                      batch = ad.batch,
                                      trt = ad.trt,
                                      title = 'Before correction')
```

```
ad.pca.rBE.plot <- Scatter_Density(object = ad.pca.rBE,
                                   batch = ad.batch,
                                   trt = ad.trt,
                                   title = 'removeBatchEffect')
```

```
ad.pca.ComBat.plot <- Scatter_Density(object = ad.pca.ComBat,
                                      batch = ad.batch,
                                      trt = ad.trt,
                                      title = 'ComBat')
```

```
ad.pca.PLSDA_batch.plot <- Scatter_Density(object = ad.pca.PLSDA_batch,
                                           batch = ad.batch,
                                           trt = ad.trt,
                                           title = 'PLSDA-batch')
```

```
ad.pca.sPLSDA_batch.plot <- Scatter_Density(object = ad.pca.sPLSDA_batch,
                                            batch = ad.batch,
                                            trt = ad.trt,
                                            title = 'sPLSDA-batch')
```

```
ad.pca.PN.plot <- Scatter_Density(object = ad.pca.PN,
                                  batch = ad.batch,
                                  trt = ad.trt,
                                  title = 'Percentile Normalisation')
```

```
ad.pca.RUVIII.plot <- Scatter_Density(object = ad.pca.RUVIII,
                                      batch = ad.batch,
                                      trt = ad.trt,
                                      title = 'RUVIII')
```

As shown in the PCA sample plots, the differences between the samples sequenced at

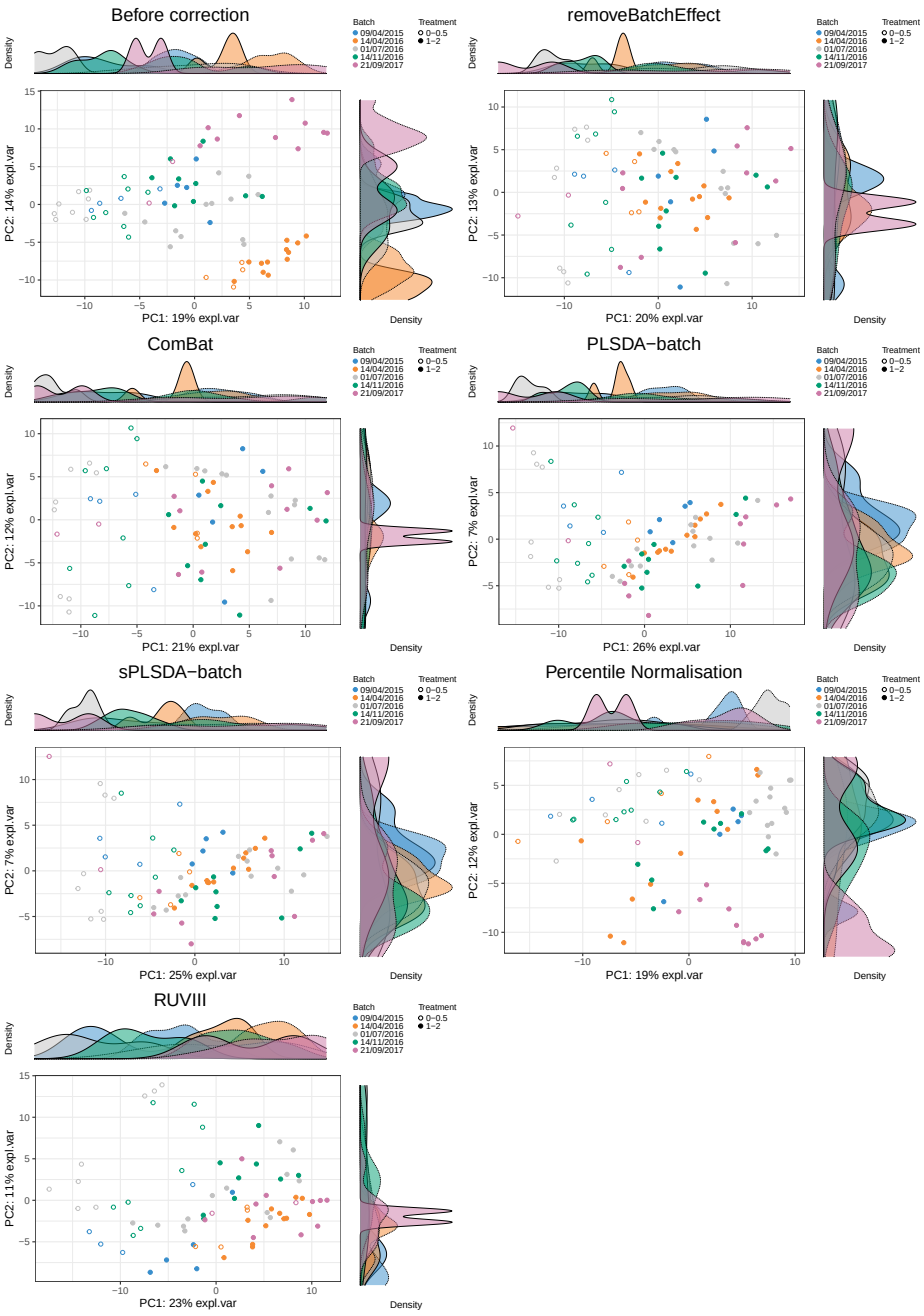

Figure 1.7: The PCA sample plots with densities before and after batch effect correction in the AD data.

"14/04/2016", "21/09/2017" and the other dates were removed after batch effect correction with most methods except percentile normalisation. The data corrected with PLSDA-batch included more treatment variation mostly on the first PC than other method-corrected data, as indicated on the x-axis label (26%). We can also compare the boxplots and density plots for key variables identified in PCA driving the major variance or heatmaps showing obvious patterns before and after batch effect correction (results not shown).

pRDA

We calculate the global explained variance across all microbial variables using pRDA. To achieve this, we create a loop for each variable from the original (uncorrected) and batch effect-corrected data. The final results are then displayed with `partVar_plot()` from PLSDAbatch package.

```
# AD data
ad.corrected.list <- list(`Before correction` = ad.clr,
                          removeBatchEffect = ad.rBE,
                          ComBat = ad.ComBat,
                          `PLSDA-batch` = ad.PLSDA_batch,
                          `sPLSDA-batch` = ad.sPLSDA_batch,
                          `Percentile Normalisation` = ad.PN,
                          RUVIII = ad.RUVIII)

ad.prop.df <- data.frame(Treatment = NA, Batch = NA,
                         Intersection = NA,
                         Residuals = NA)
for(i in seq_len(length(ad.corrected.list))){
  rda.res = varpart(ad.corrected.list[[i]], ~ trt, ~ batch,
                    data = ad.factors.df, scale = TRUE)
  ad.prop.df[i, ] <- rda.res$part$indfract$Adj.R.squared}

rownames(ad.prop.df) = names(ad.corrected.list)

ad.prop.df <- ad.prop.df[, c(1,3,2,4)]

ad.prop.df[ad.prop.df < 0] = 0
ad.prop.df <- as.data.frame(t(apply(ad.prop.df, 1,
                                    function(x){x/sum(x)})))

partVar_plot(prop.df = ad.prop.df)
```

As shown in the above figure, the intersection between batch and treatment variance was small (1.3%) for the AD data, which implies that the batch x treatment design is not highly unbalanced. Thus the unweighted PLSDA-batch and sPLSDA-batch were still applicable, and thus the weighted versions were not used. sPLSDA-batch corrected data led to the best performance with a slightly higher proportion of treatment variance

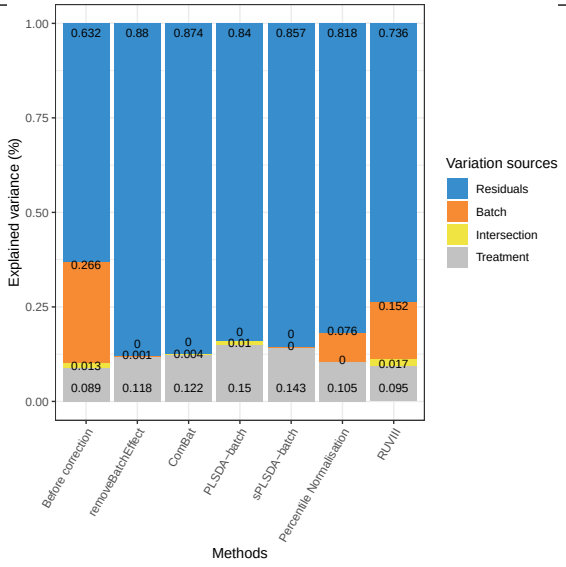

Figure 1.8: Global explained variance before and after batch effect correction for the AD data.

explained and undetectable batch and intersection variance compared to the other methods.

```
# HFHS data
data('HFHS_data')
hfhs.corrected.list <- HFHS_data$CorrectData$data

hfhs.trt <- HFHS_data$CorrectData$Y.trt
hfhs.batch <- HFHS_data$CorrectData$Y.bat
hfhs.factors.df <- data.frame(trt = hfhs.trt, batch = hfhs.batch)

hfhs.prop.df <- data.frame(Treatment = NA, Batch = NA,
                           Intersection = NA,
                           Residuals = NA)
for(i in seq_len(length(hfhs.corrected.list))){
  rda.res = varpart(hfhs.corrected.list[[i]], ~ trt, ~ batch,
                    data = hfhs.factors.df, scale = TRUE)
  hfhs.prop.df[i, ] <- rda.res$part$indfract$Adj.R.squared}

rownames(hfhs.prop.df) = names(hfhs.corrected.list)

hfhs.prop.df <- hfhs.prop.df[, c(1,3,2,4)]

hfhs.prop.df[hfhs.prop.df < 0] = 0
```

```
hfhs.prop.df <- as.data.frame(t(apply(hfhs.prop.df, 1,
                                      function(x){x/sum(x)})))


partVar_plot(prop.df = hfhs.prop.df)
```

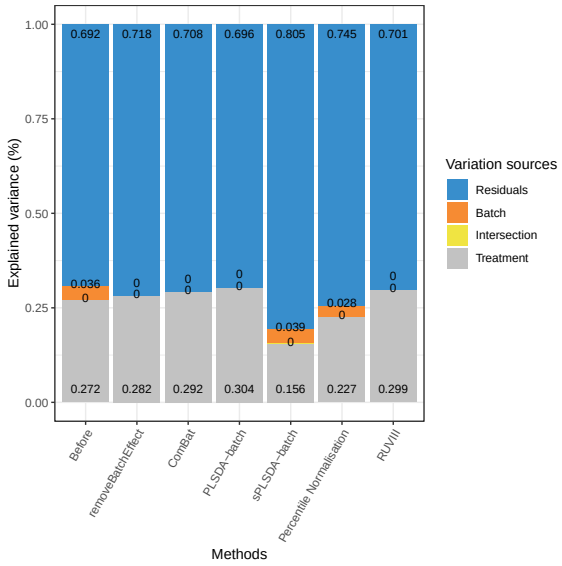

Figure 1.9: Global explained variance before and after batch effect correction for the HFHS data.

As shown in the above figure, a small amount of batch variance was observed (3.6%) for the HFHS data. PLSDA-batch achieved the best performance for preserving the largest treatment variance and completely removing batch variance compared to the other methods. The results also indicate that PLSDA-batch is more appropriate for weak batch effects, while sPLSDA-batch is more appropriate for strong batch effects. The RUVIII performed better in the HFHS data than the AD data because the sample replicates might capture more batch variation than in the AD data. Indeed, the sample replicates in HFHS data are across different day batches, while the replicates in AD data do not exist in all batches. Therefore, sample replicates play a critical role in RUVIII.

1.7.2 Other methods

R^2

The R^2 values for each variable are calculated with `lm()` from stats package. To compare the R^2 values among variables, we scale the corrected data before R^2 calculation. The results are displayed with `ggplot()` from ggplot2 R package.

```
# AD data
# scale
ad.corr_scale.list <- lapply(ad.corrected.list,
                             function(x){apply(x, 2, scale)})

ad.r_values.list <- list()
for(i in seq_len(length(ad.corr_scale.list))){
  ad.r_values <- data.frame(trt = NA, batch = NA)
  for(c in seq_len(ncol(ad.corr_scale.list[[i]]))){
    ad.fit.res.trt <- lm(ad.corr_scale.list[[i]][,c] ~ ad.trt)
    ad.r_values[c,1] <- summary(ad.fit.res.trt)$r.squared
    ad.fit.res.batch <- lm(ad.corr_scale.list[[i]][,c] ~ ad.batch)
    ad.r_values[c,2] <- summary(ad.fit.res.batch)$r.squared
  }
  ad.r_values.list[[i]] <- ad.r_values
}
names(ad.r_values.list) <- names(ad.corr_scale.list)

ad.boxp.list <- list()
for(i in seq_len(length(ad.r_values.list))){
  ad.boxp.list[[i]] <-
    data.frame(r2 = c(ad.r_values.list[[i]][ ,'trt'],
                      ad.r_values.list[[i]][ ,'batch']),
               Effects = as.factor(rep(c('Treatment','Batch'),
                                       each = 231)))
}
names(ad.boxp.list) <- names(ad.r_values.list)

ad.r2.boxp <- rbind(ad.boxp.list$`Before correction`,
                    ad.boxp.list$removeBatchEffect,
                    ad.boxp.list$ComBat,
                    ad.boxp.list$`PLSDA-batch`,
                    ad.boxp.list$`sPLSDA-batch`,
                    ad.boxp.list$`Percentile Normalisation`,
                    ad.boxp.list$RUVIII)

ad.r2.boxp$methods <- rep(c('Before correction', ' removeBatchEffect',
                            'ComBat','PLSDA-batch', 'sPLSDA-batch',
                            'Percentile Normalisation', 'RUVIII'), each = 462)

ad.r2.boxp$methods <- factor(ad.r2.boxp$methods,
                             levels = unique(ad.r2.boxp$methods))

ggplot(ad.r2.boxp, aes(x = Effects, y = r2, fill = Effects)) +
  geom_boxplot(alpha = 0.80) +
```

```
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 60, hjust = 1, size = 18),
        axis.text.y = element_text(size = 18),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid( ~ methods) +
  scale_fill_manual(values=pb_color(c(12,14)))
```

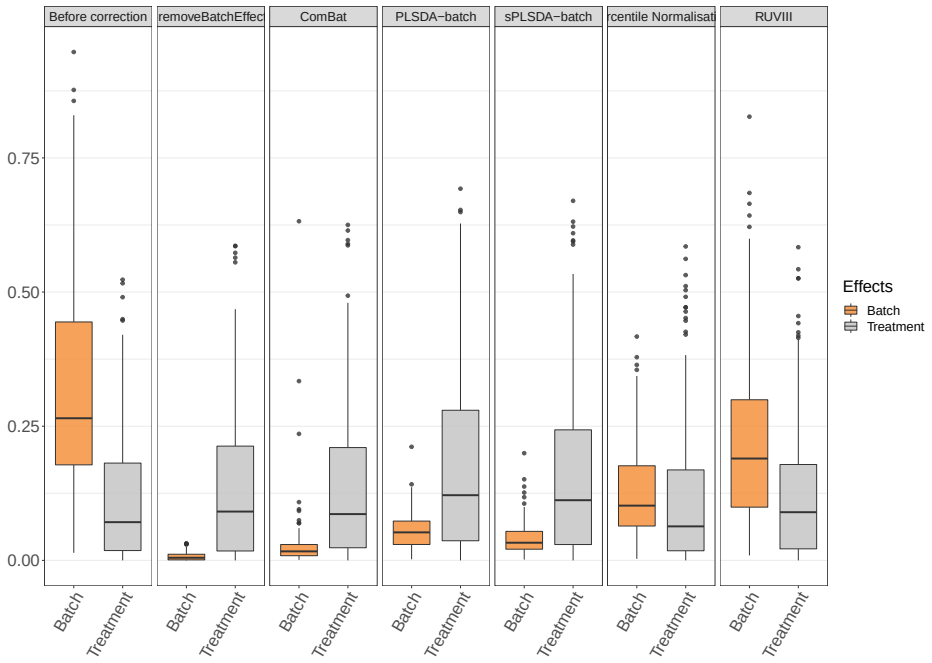

Figure 1.10: AD study: R^2 values for each microbial variable before and after batch effect correction.

The corrected data from ComBat still included a few variables with a large proportion of batch variance. A large number of variables from the data corrected by percentile normalisation and RUVIII still included considerable batch variance, especially RUVIII, which resulting data contained more proportion of batch variance than treatment variance.

```
##################################
ad.barp.list <- list()
for(i in seq_len(length(ad.r_values.list))){
  ad.barp.list[[i]] <- data.frame(r2 = c(sum(ad.r_values.list[[i]][ ,'trt']),
```

```
                                              sum(ad.r_values.list[[i]][ ,'batch'])),
                                    Effects = c('Treatment','Batch'))
}
names(ad.barp.list) <- names(ad.r_values.list)

ad.r2.barp <- rbind(ad.barp.list$`Before correction`,
                    ad.barp.list$removeBatchEffect,
                    ad.barp.list$ComBat,
                    ad.barp.list$`PLSDA-batch`,
                    ad.barp.list$`sPLSDA-batch`,
                    ad.barp.list$`Percentile Normalisation`,
                    ad.barp.list$RUVIII)


ad.r2.barp$methods <- rep(c('Before correction', ' removeBatchEffect',
                            'ComBat','PLSDA-batch', 'sPLSDA-batch',
                            'Percentile Normalisation', 'RUVIII'), each = 2)

ad.r2.barp$methods <- factor(ad.r2.barp$methods,
                             levels = unique(ad.r2.barp$methods))


ggplot(ad.r2.barp, aes(x = Effects, y = r2, fill = Effects)) +
  geom_bar(stat="identity") +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 60, hjust = 1, size = 18),
        axis.text.y = element_text(size = 18),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid( ~ methods) +
  scale_fill_manual(values=pb_color(c(12,14)))
```

The overall sum of R^2 values indicated that removeBatchEffect removed slightly more batch variance (removeBatchEffect: 1.70, PLSDA-batch: 12.40, sPLSDA-batch: 9.25) but preserved less treatment variance (removeBatchEffect: 31.75, PLSDA-batch: 40.00, sPLSDA-batch: 36.22) than our proposed approaches

Alignment scores

To use the `alignment_score()` function from PLSDAbatch, we need to specify the proportion of data variance to explain (`var`), the number of nearest neighbours (`k`) and the number of principal components to estimate (`ncomp`). We then use `ggplot()` function from ggplot2 to visualise the results.

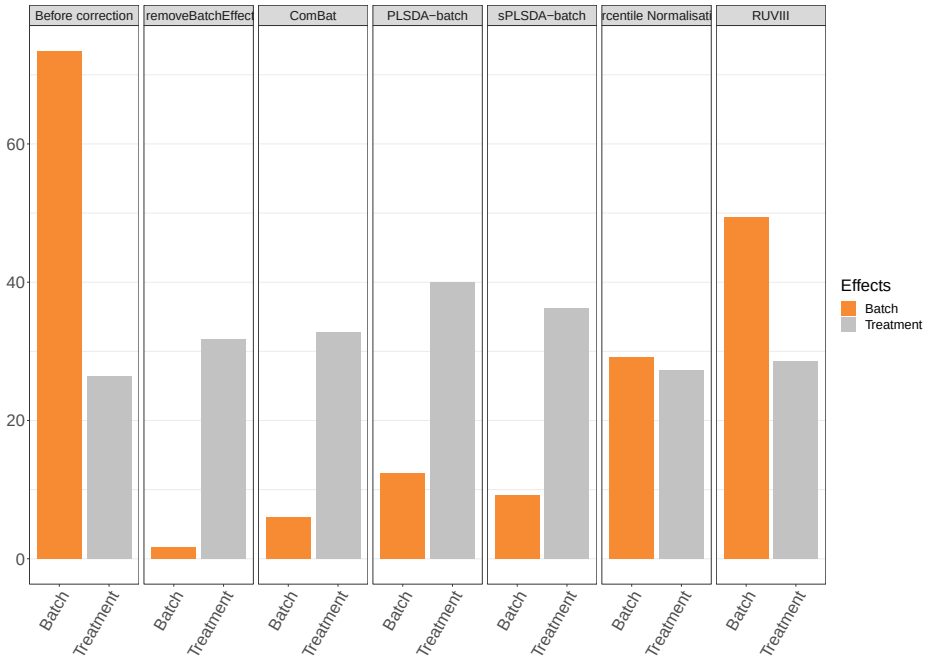

Figure 1.11: AD study: the sum of R^2 values for each microbial variable before and after batch effect correction.

```
# AD data
ad.scores <- c()
names(ad.batch) <- rownames(ad.clr)
for(i in seq_len(length(ad.corrected.list))){
  res <- alignment_score(data = ad.corrected.list[[i]],
                         batch = ad.batch,
                         var = 0.95,
                         k = 8,
                         ncomp = 50)
  ad.scores <- c(ad.scores, res)
}

ad.scores.df <- data.frame(scores = ad.scores,
                           methods = names(ad.corrected.list))

ad.scores.df$methods <- factor(ad.scores.df$methods,
                               levels = rev(names(ad.corrected.list)))


ggplot() + geom_col(aes(x = ad.scores.df$methods,
                        y = ad.scores.df$scores)) +
  geom_text(aes(x = ad.scores.df$methods,
                y = ad.scores.df$scores/2,
                label = round(ad.scores.df$scores, 3)),
            size = 3, col = 'white') +
  coord_flip() + theme_bw() + ylab('Alignment Scores') +
  xlab('') + ylim(0,0.85)
```

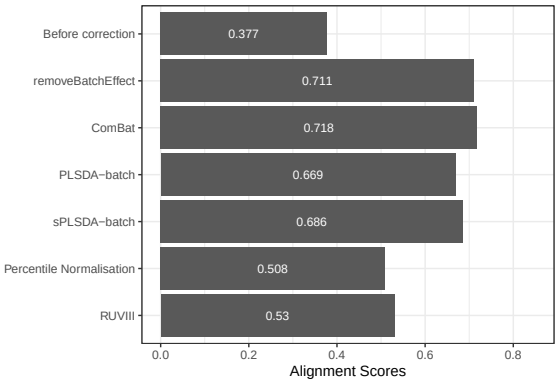

Figure 1.12: Comparison of alignment scores before and after batch effect correction using different methods for the AD data.

The alignment scores complement the PCA results, especially when batch effect re-

moval is difficult to assess on PCA sample plots. For example in previous PCA sample plots (**Figure 7**), we observed that the samples across different batches were better mixed after batch effect correction with different methods than before, whereas the performance of difference methods was difficult to compare. Since a higher alignment score indicates that samples are better mixed, as shown in the above bar plot, Combat gave a superior performance compared to the other methods. However, with the R^2 values the ComBat corrected data still included a few variables with a large proportion of batch variance (see section **other methods** method "R^2"). Therefore, it is important to compare different techniques and outputs for an unbiased assessment. In this example, the lower alignment scores of PLSDA-batch and sPLSDA-batch corrected data might result from the difference in the PCA sample projections of the batch effect corrected matrices. The data corrected with removeBatchEffect and ComBat had a large variance in their PCA projection, while PLSDA-batch and sPLSDA-batch corrected data had a small variance. A small variance projection results in a small alignment score, as it is easy to locate the samples from the same batch as nearest neighbours. In fact, pRDA presented above (**Figure 8**) quantitatively confirmed that both PLSDA-batch and sPLSDA-batch entirely removed the batch variance [Wang and Lê Cao, 2020].

1.8 Variable selection

We use `splsda()` from mixOmics to select the top 50 microbial variables that, in combination, discriminate the different treatment groups in the AD data. We apply `splsda()` to the different batch effect corrected data from all methods. Then we use `upset()` from UpSetR package [Lex et al., 2014] to visualise the concordance of variables selected.

In the code below, we first need to convert the list of variables selected from different method-corrected data into a data frame compatible with `upset()` using `fromList()`. We then assign different colour schemes for each variable selection.

```
ad.splsda.select <- list()
for(i in seq_len(length(ad.corrected.list))){
  splsda.res <- splsda(X = ad.corrected.list[[i]], Y = ad.trt,
                       ncomp = 3, keepX = rep(50,3))
  select.res <- selectVar(splsda.res, comp = 1)$name
  ad.splsda.select[[i]] <- select.res
}
names(ad.splsda.select) <- names(ad.corrected.list)

# can only visualise 5 methods
ad.splsda.select <- ad.splsda.select[seq_len(5)]

ad.splsda.upsetR <- fromList(ad.splsda.select)

upset(ad.splsda.upsetR, main.bar.color = 'gray36',
      sets.bar.color = pb_color(c(25:22,20)), matrix.color = 'gray36',
      order.by = 'freq', empty.intersections = 'on',
```

```
      queries = list(list(query = intersects,
                          params = list('Before correction'),
                          color = pb_color(20), active = TRUE),
                     list(query = intersects,
                          params = list('removeBatchEffect'),
                          color = pb_color(22), active = TRUE),
                     list(query = intersects,
                          params = list('ComBat'),
                          color = pb_color(23), active = TRUE),
                     list(query = intersects,
                          params = list('PLSDA-batch'),
                          color = pb_color(24), active = TRUE),
                     list(query = intersects,
                          params = list('sPLSDA-batch'),
                          color = pb_color(25), active = TRUE)))
```

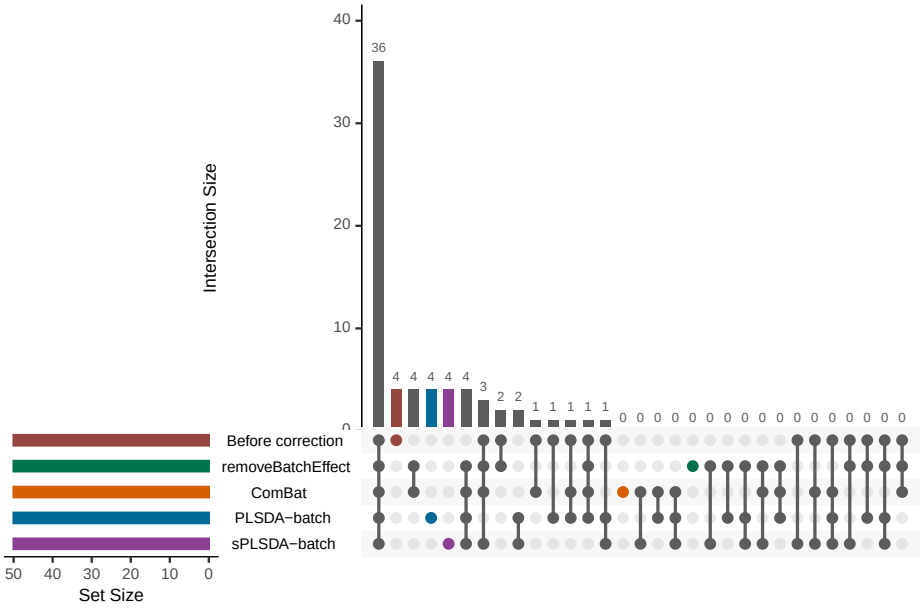

Figure 1.13: UpSet plot showing overlap between variables selected from different corrected data for the AD study.

In the above UpSet plot, the left bars indicate the number of variables selected from each data corrected with different methods. The right bar plot combined with the scatterplot show different intersection and their aggregates. We obtained a high overlap of 36 out of 50 selected variables between different corrected and uncorrected data. However, the data from each method still included unique variables that were not selected in the other corrected data, e.g., PLSDA-batch and sPLSDA-batch. As `upset()` can only include

five datasets at once, we only displayed the uncorrected data and four corrected data that had been more efficiently corrected for batch effects from our previous assessments compared to the other datasets.

We then extract each intersection of selected variables between these five corrected data with `venn()` function from gplots. We can also save the taxonomic information of these OTUs into a text file for further interpretation.

```
ad.splsda.select.overlap <- venn(ad.splsda.select, show.plot = FALSE)
ad.inters.splsda <- attr(ad.splsda.select.overlap, 'intersections')
ad.inters.splsda.taxa <-
  lapply(ad.inters.splsda,
         FUN = function(x){as.data.frame(AD_data$FullData$taxa[x, ])})
capture.output(ad.inters.splsda.taxa,
               file = "GeneratedData/ADselected_50_splsda.txt")
```

1.9 Summary

This vignette presents a complete framework for batch effect management in microbiome data. The first step includes pre-processing, with pre-filtering to remove low-count samples and variables, and data transformation to handle compositional data characteristics. For this step, we advise against using relative abundance data that are compositional, if possible. Visual tools, such as PCA, boxplots, density plots and heatmap, and quantitative approach pRDA are then applied to detect batch effects. When the batch effect is very weak, for example as informed by a very small proportion of variance explained by batch with pRDA, or difficult to visualise with PCA, then there may be no need to manage batch effects. However, when the batch effect is strong, two solutions are to either account for batch effects, or to correct for batch effects from the original data. For either solution, the batch x treatment design matters. If nested, we can only account for batch effects with a linear mixed model. If unbalanced but not nested, we can account for batch effects with any linear models or remove them using weighted PLSDA-batch. Regarding different batch effects, our proposed method PLSDA-batch is more appropriate for a weak batch effect, while sPLSDA-batch for a strong batch effect.

We usually assume batch grouping information is known. When the batch information is unknown, methods such as SVA estimate batch effects from the variables least affected by treatment effects, while RUV4 and RUVIII estimate batch effects from negative control variables or/and sample replicates. For the latter, negative control variables and sample replicates should capture the complete batch variation, if not, there may be a risk that batch effects cannot be completely considered or removed. We emphasised on this point in the AD and the HFHS studies. For these methods that can estimate batch effects, the estimated batch effects are independent of treatment effects. If a correlation between batch and treatment exists, this correlation cannot be estimated. Spurious correlation between batch and treatment effects may also be present, for example SVA, which estimates batch effects from the variables that may still include more or less

treatment effects.

In addition, most methods assume systematic batch effects across the whole dataset. This is the case of ComBat, where we recommend assessing the model's validity first.

The next critical step is to assess the efficacy of batch effect management for a given method. Such assessment is often implicit in methods that account for batch effects. In contrast, the methods that remove batch effects can be assessed by comparing the data before and after correction. All methods we have presented for batch effect detection can be used for assessing batch effect correction. We can also calculate the explained variance R^2 of each microbial variable for batch and treatment covariate respectively, as well as alignment scores measuring the performance of mixing samples across batches. We also have noted that some methods may not be very sensitive. For example, PCA plots rely on somewhat subjective evaluation of batch and treatment variation visualisation, and alignment scores only measure the performance of mixing samples. Therefore, a reliable conclusion should be made based on multiple assessment methods.

Once batch effects are removed, downstream analysis such as multivariate discriminant analysis, or univariate differential analysis can be performed. Multivariate methods may be more suitable for microbiome data analysis since microbial variables are naturally correlated because of mutual interactions.

Section		Applied data	Required function	Package	Repository
Data pre-processing	Pre-filtering	AD data, HFHS data, HD data	PreFL()	PLSDAbatch	GitHub
	Transformation	AD data, sponge data, HFHS data, HD data	logratio.transfo()	mixOmics	Bioconductor
Batch effect detection	PCA	AD data, sponge data, HFHS data, HD data	pca()	mixOmics	Bioconductor
			Scatter_Density()	PLSDAbatch	GitHub
	Boxplots and density plots	AD data, sponge data, HFHS data, HD data	selectVar()	mixOmics	Bioconductor
			box_plot()	PLSDAbatch	GitHub
			density_plot()	PLSDAbatch	GitHub
			linear_regres()	PLSDAbatch	GitHub
			relevel()	stats	
	Heatmap	AD data, sponge data, HFHS data, HD data	pheatmap()	pheatmap	CRAN
	pRDA	AD data, sponge data, HFHS data, HD data	varpart()	vegan	CRAN
Accounting for batch effects	Zero-inflated Gaussian mixture model	AD data, sponge data, HFHS data	newMRexperiment()	metagenomeSeq	Bioconductor
			AnnotatedDataFrame()	Biobase	Bioconductor
			filterData()	metagenomeSeq	Bioconductor
			MRcounts()	metagenomeSeq	Bioconductor
			cumNormStatFast()	metagenomeSeq	Bioconductor
			cumNorm()	metagenomeSeq	Bioconductor
			normFactors()	metagenomeSeq	Bioconductor
			model.matrix()	stats	
			fitZig()	metagenomeSeq	Bioconductor
			MRcoefs())	metagenomeSeq	Bioconductor
	Linear regression	AD data, sponge data, HFHS data, HD data	linear_regres()	PLSDAbatch	GitHub
			p.adjust()	stats	
			check_model()	perfaormance	CRAN
	SVA	AD data, sponge data, HFHS data	model.matrix()	stats	
			num.sv()	sva	Bioconductor
			sva()	sva	Bioconductor
			f.pvalue()	sva	Bioconductor
	RUV4	AD data, sponge data, HFHS data	getK()	ruv	CRAN
			lm()	stats	
			RUV4()	ruv	CRAN
Correcting for batch effects	removeBatchEffect	AD data, sponge data, HFHS data	model.matrix()	stats	
			removeBatchEffect()	limma	Bioconductor
	ComBat	AD data, sponge data, HFHS data	ComBat()	sva	Bioconductor
	PLSDA-batch	AD data, sponge data, HFHS data	plsda()	mixOmics	Bioconductor
			PLSDA_batch()	PLSDAbatch	GitHub
	sPLSDA-batch	AD data, sponge data, HFHS data	tune.splsda()	mixOmics	Bioconductor
			PLSDA_batch()	PLSDAbatch	GitHub
	Percentile Normalisation	AD data, sponge data, HFHS data	percentile_norm()	PLSDAbatch	GitHub
	RUVIII	AD data, HFHS data	getK()	ruv	CRAN
			replicate.matrix()	ruv	CRAN
			RUVIII()	ruv	CRAN
Assessing batch effect correction	pRDA	AD data, sponge data, HFHS data	varpart()	vegan	CRAN
			partVar_plot()	PLSDAbatch	GitHub
	R2	AD data, sponge data, HFHS data	lm()	stats	
			ggplot()	ggplot2	CRAN
	Alignment scores	AD data, sponge data, HFHS data	alignment_score()	PLSDAbatch	GitHub
			ggplot()	ggplot2	CRAN
Variable selection	Variable selection	AD data, sponge data, HFHS data	splsda()	mixOmics	Bioconductor
			selectVar()	mixOmics	Bioconductor
			fromList()	UpSetR	CRAN
			upset()	UpSetR	CRAN
			venn()	gplots	CRAN
			capture.output()	utils	

Table 2: A summary of all functions in each section.

1.10 References

Chapter 2

Supplementary Materials for Batch Effects Management in Case Studies

2.1 Sponge data

2.1.1 Data pre-processing

2.1.1.1 Pre-filtering

We load the sponge data stored internally with function `data()`.

```
# Sponge data
data('sponge_data')
sponge.tss <- sponge_data$X.tss
dim(sponge.tss)
```

```
## [1] 32 24
```

```
# zero proportion
mean(sponge.tss == 0)
```

```
## [1] 0.5455729
```

The sponge data include the relative abundance of 24 OTUs and 32 samples. Given the small number of OTUs we advise not to pre-filter the data.

2.1.1.2 Transformation

Prior to CLR transformation, we recommend adding 0.01 as the offset for the sponge data - that are relative abundance data. We use `logratio.transfo()` function in

mixOmics package to CLR transform the data.

```
sponge.clr <- logratio.transfo(X = sponge.tss, logratio = 'CLR', offset = 0.01)

class(sponge.clr) <- 'matrix'
```

2.1.2 Batch effect detection

2.1.2.1 PCA

We apply `pca()` function from mixOmics package to the sponge data and `Scatter_Density()` function from PLSDAbatch to represent the PCA sample plot with densities.

```
sponge.pca.before <- pca(sponge.clr, ncomp = 3, scale = TRUE)

sponge.batch <- sponge_data$Y.bat
sponge.trt <- sponge_data$Y.trt

Scatter_Density(object = sponge.pca.before,
                batch = sponge.batch,
                trt = sponge.trt, title = 'Sponge data',
                trt.legend.title = 'Tissue types')
```

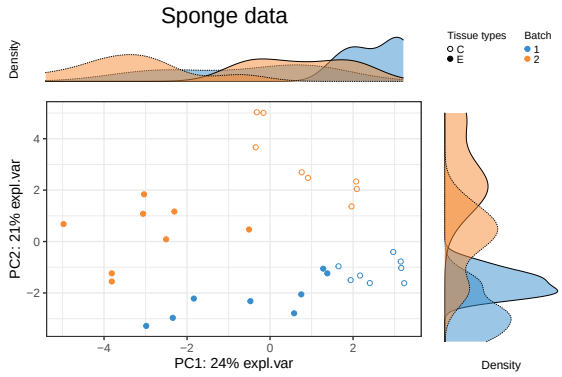

Figure 2.1: The PCA sample plot with densities in the sponge data.

In the above figure, the tissue variation (effects of interest) accounted for 24% of data variance on component 1, while the gel variation (batch effects) for 21% on component 2. Therefore, the batch effect is slightly weaker than the treatment effect in the sponge data.

2.1.2.2 Heatmap

We produce a heatmap using pheatmap package. The data first need to be scaled on both OTUs and samples.

```
# scale the clr data on both OTUs and samples
sponge.clr.s <- scale(sponge.clr, center = T, scale = T)
sponge.clr.ss <- scale(t(sponge.clr.s), center = T, scale = T)

sponge.anno_col <- data.frame(Batch = sponge.batch, Tissue = sponge.trt)
sponge.anno_colors <- list(Batch = color.mixo(1:2),
                           Tissue = pb_color(1:2))
names(sponge.anno_colors$Batch) = levels(sponge.batch)
names(sponge.anno_colors$Tissue) = levels(sponge.trt)

pheatmap(sponge.clr.ss,
         cluster_rows = F,
         fontsize_row = 4,
         fontsize_col = 6,
         fontsize = 8,
         clustering_distance_rows = 'euclidean',
         clustering_method = 'ward.D',
         treeheight_row = 30,
         annotation_col = sponge.anno_col,
         annotation_colors = sponge.anno_colors,
         border_color = 'NA',
         main = 'Sponge data - Scaled')
```

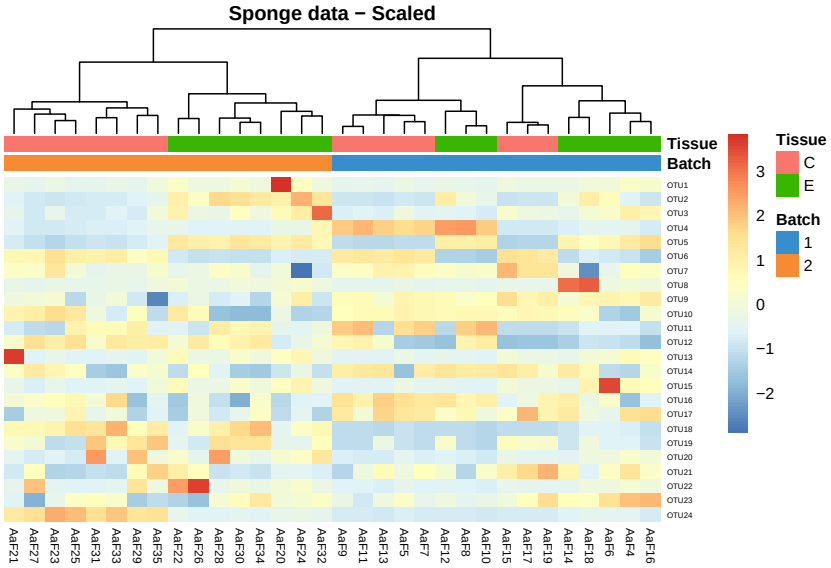

Figure 2.2: Hierarchical clustering for samples in the sponge data.

In the heatmap, samples in the sponge data were clustered or grouped by batches instead of tissues, indicating a batch effect.

2.1.2.3 pRDA

We apply pRDA with `varpart()` function from vegan R package.

```
sponge.factors.df <- data.frame(trt = sponge.trt,
                                batch = sponge.batch)
sponge.rda.before <- varpart(sponge.clr, ~ trt, ~ batch,
                             data = sponge.factors.df,
                             scale = T)
sponge.rda.before$part$indfract
```

```
##                  Df R.squared Adj.R.squared Testable
## [a] = X1|X2       1        NA    0.16572246     TRUE
## [b] = X2|X1       1        NA    0.16396277     TRUE
## [c]               0        NA   -0.01063501    FALSE
## [d] = Residuals NA         NA    0.68094977    FALSE
```

In the result, `X1` and `X2` represent the first and second covariates fitted in the model. `[a]`, `[b]` represent the independent proportion of variance explained by `X1` and `X2` respectively, and `[c]` represents the intersection variance shared between `X1` and `X2`. The sponge data have a completely balanced batch x treatment design, so there was no intersection variance (indicated in line `[c]`, Adj.R.squared = -0.01) detected. The proportion of treatment and batch variance was nearly equal as indicated in lines `[a]` and `[b]`.

2.1.3 Managing batch effects

2.1.3.1 Accounting for batch effects

The methods that we use to account for batch effects include the method designed for microbiome data: zero-inflated Gaussian (ZIG) mixture model and the methods adapted for microbiome data: linear regression, SVA and RUV4. Among them, SVA and RUV4 are designed for unknown batch effects.

As the ZIG model is applicable to counts data, the sponge data are not qualified for this model. We therefore only apply the methods that can be adapted for microbiome data.

Linear regression

Linear regression is conducted with `linear_regres()` function in PLSDAbatch. We integrated the performance package that assesses performance of regression models into our function `linear_regres()`. Therefore, we can apply `check_model()` from performance to the outputs from `linear_regres()` to diagnose the validity of the model fitted with treatment and batch effects for each variable [LÃŒdecke et al., 2020]. We can extract performance measurements such as adjusted R2, RMSE, RSE, AIC and BIC for the models fitted with and without batch effects, which are also the outputs of `linear_regres()`.

We apply `type = "linear model"` to the sponge data because of the balanced batch x treatment design.

```
sponge.clr <- sponge.clr[1:nrow(sponge.clr), 1:ncol(sponge.clr)]
sponge.lm <- linear_regres(data = sponge.clr,
                           trt = sponge.trt,
                           batch.fix = sponge.batch,
                           type = 'linear model')

sponge.p <- sapply(sponge.lm$lm.table, function(x){x$coefficients[2,4]})
sponge.p.adj <- p.adjust(p = sponge.p, method = 'fdr')

check_model(sponge.lm$model$OTU2)
```

To diagnose the validity of the model fitted with both treatment and batch effects, we use different plots to check the assumptions of each microbial variable. For example, the diagnostic plots of "OTU2" are shown in the above figure panel. The simulated data under the fitted model were not very close to the real data (shown as green line), indicating an imperfect fitness (top panel). The linearity (or homoscedasticity) and homogeneity of variance were not satisfied. The correlation between batch (`batch.fix`) and treatment (`trt`) effects was very low, indicating a good model with low collinearity. Some samples could be classified as outliers with a Cook's distance larger than or equal to 0.5, for example, "21", "15", "7", "1" and "13" (middle panel). The distribution of residuals was not close to normal (bottom panel). For the microbial variables with some assumptions not met, we should be careful about their results.

For performance measurements of models fitted with or without batch effects, we show an example of the results for some variables.

```
head(sponge.lm$adj.R2)
```

```
##          trt.only  trt.batch
## OTU1  0.06423228 0.05701977
## OTU2  0.60492094 0.67117014
## OTU3  0.19003076 0.18125188
## OTU4 -0.03327738 0.23731780
## OTU5  0.97483298 0.97643741
## OTU6  0.97496003 0.97634939
```

If the adjusted R^2 of the model with both treatment and batch effects is larger than the model with treatment effects only, e.g., OTU2, OTU4, OTU5 and OTU6, the model fitted with batch effects explains more data variance, and is thus better than the model without batch effects. Otherwise, the batch effect is not necessary to be added into the linear model.

We next look at the AIC of models fitted with or without batch effects.

```
head(sponge.lm$AIC)
```

```
##        trt.only  trt.batch
## OTU1 47.901506 49.0623531
```

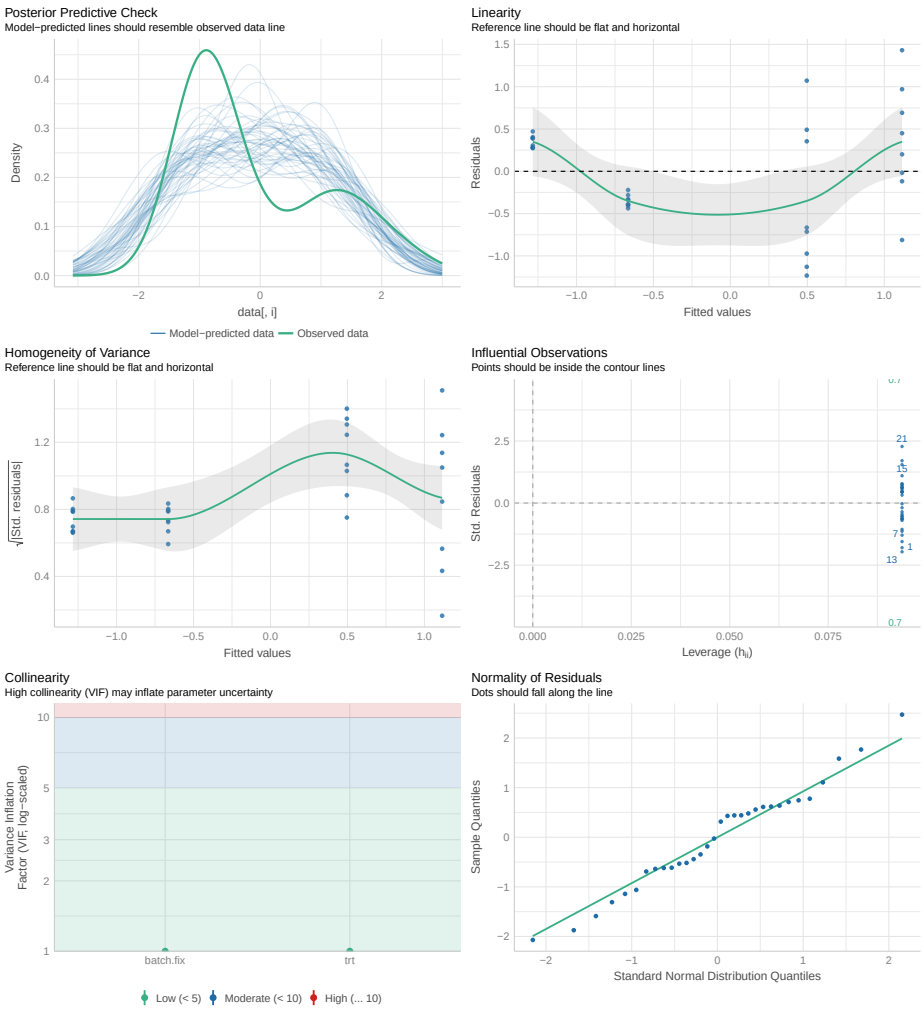

Figure 2.3: Diagnostic plots for the model fitted with batch effects of ”OTU2” in the sponge data.

```
## OTU2 74.001621 69.0433180
## OTU3 -5.630456 -4.3703385
## OTU4 90.699895 81.8982606
## OTU5 21.727333 20.5345128
## OTU6  1.196912  0.2853681
```

A lower AIC indicates a better fit. Both results of the adjusted R^2 and AIC were consistent on model selection for the listed OTUs.

SVA

SVA accounts for unknown batch effects. Here we assume that the batch grouping information in the sponge data is unknown. We first build two design matrices with (`sponge.mod`) and without (`sponge.mod0`) treatment grouping information generated with `model.matrix()` function from stats. We then use `num.sv()` from sva package to determine the number of batch variables `n.sv` that is used to estimate batch effects in function `sva()`.

```
# estimate batch matrix
sponge.mod <- model.matrix( ~ sponge.trt)
sponge.mod0 <- model.matrix( ~ 1, data = sponge.trt)
sponge.sva.n <- num.sv(dat = t(sponge.clr), mod = sponge.mod, method = 'leek')
sponge.sva <- sva(t(sponge.clr), sponge.mod, sponge.mod0, n.sv = sponge.sva.n)
```

```
## Number of significant surrogate variables is:  1
## Iteration (out of 5 ):1  2  3  4  5
```

The estimated batch effects are then applied to `f.pvalue()` to calculate the P-values of treatment effects. The estimated batch effects in SVA are assumed to be independent of the treatment effects. However, SVA considers some correlation between batch and treatment effects [Wang and LêCao, 2020].

```
# include estimated batch effects in the linear model
sponge.mod.batch <- cbind(sponge.mod, sponge.sva$sv)
sponge.mod0.batch <- cbind(sponge.mod0, sponge.sva$sv)
sponge.sva.p <- f.pvalue(t(sponge.clr), sponge.mod.batch, sponge.mod0.batch)
sponge.sva.p.adj <- p.adjust(sponge.sva.p, method = 'fdr')
```

RUV4

Before applying RUV4 (`RUV4()` from ruv package), we need to specify negative control variables and the number of batch variables to estimate. We can use the empirical negative controls that are not significantly differentially abundant (adjusted P > 0.05) from a linear regression with the treatment information as the only covariate.

We use a loop to fit a linear regression for each microbial variable and adjust P values of treatment effects for multiple comparisons with `p.adjust()` from stats. The empirical negative controls are then extracted according to the adjusted P values.

```
# empirical negative controls
sponge.empir.p <- c()
for(e in 1:ncol(sponge.clr)){
  sponge.empir.lm <- lm(sponge.clr[,e] ~ sponge.trt)
  sponge.empir.p[e] <- summary(sponge.empir.lm)$coefficients[2,4]
}
sponge.empir.p.adj <- p.adjust(p = sponge.empir.p, method = 'fdr')
sponge.nc <- sponge.empir.p.adj > 0.05
```

The number of batch variables `k` can be determined using `getK()` function.

```
# estimate k
sponge.k.res <- getK(Y = sponge.clr, X = sponge.trt, ctl = sponge.nc)
sponge.k <- sponge.k.res$k
sponge.k <- ifelse(sponge.k != 0, sponge.k, 1)
```

We then apply `RUV4()` with known treatment variables, estimated negative control variables and `k` batch variables. The calculated P values also need to be adjusted for multiple comparisons.

```
sponge.ruv4 <- RUV4(Y = sponge.clr, X = sponge.trt,
                    ctl = sponge.nc, k = sponge.k)
sponge.ruv4.p <- sponge.ruv4$p
sponge.ruv4.p.adj <- p.adjust(sponge.ruv4.p, method = "fdr")
```

Note: A package named RUVSeq has been developed for count data. It provides `RUVg()` using negative control variables, and also other functions `RUVs()` and `RUVr()` using sample replicates [Moskovicz et al., 2020] or residuals from the regression on treatment effects to estimate and then account for latent batch effects. However, for CLR-transformed data, we still recommend the standard ruv package.

2.1.3.2 Correcting for batch effects

The methods that we use to correct for batch effects include removeBatchEffect, ComBat, PLSDA-batch, sPLSDA-batch, Percentile Normalisation and RUVIII. Among them, RUVIII is designed for unknown batch effects. Except RUVIII that requires sample replicates which the sponge data do not have, these methods were all applied to and illustrated with the sponge data.

removeBatchEffect

The `removeBatchEffect()` function is implemented in limma package. The design matrix (`design`) with treatment grouping information can be generated with `model.matrix()` function from stats as shown in section **accounting for batch effects** method "SVA".

Here we use `removeBatchEffect()` function with batch grouping information (`batch`) and treatment design matrix (`design`) to calculate batch effect corrected data `sponge.rBE`.

```
sponge.rBE <- t(removeBatchEffect(t(sponge.clr), batch = sponge.batch,
                                  design = sponge.mod))
```

ComBat

The `ComBat()` function (from sva package) is implemented as parametric or non-parametric correction with option `par.prior`. Under a parametric adjustment, we can assess the model's validity with `prior.plots = T` [Leek et al., 2012].

Here we use a non-parametric correction (`par.prior = FALSE`) with batch grouping information (`batch`) and treatment design matrix (`mod`) to calculate batch effect corrected data `sponge.ComBat`.

```
sponge.ComBat <- t(ComBat(t(sponge.clr), batch = sponge.batch,
                          mod = sponge.mod, par.prior = F))
```

PLSDA-batch

The `PLSDA_batch()` function is implemented in PLSDAbatch package. To use this function, we need to specify the optimal number of components related to treatment (`ncomp.trt`) or batch effects (`ncomp.bat`).

Here in the sponge data, we use `plsda()` from mixOmics with only treatment grouping information to estimate the optimal number of treatment components to preserve.

```
# estimate the number of treatment components
sponge.trt.tune <- plsda(X = sponge.clr, Y = sponge.trt, ncomp = 5)
sponge.trt.tune$prop_expl_var #1
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5
## 0.22802218 0.10478658 0.16376238 0.06861581 0.04524292
##
## $Y
##      comp1      comp2      comp3      comp4      comp5
## 1.00000000 0.16584332 0.06024061 0.03278432 0.01144732
```

We choose the number that explains 100% variance in the outcome matrix `Y`, thus from the result, 1 component was enough to preserve the treatment information.

We then use `PLSDA_batch()` function with both treatment and batch grouping information to estimate the optimal number of batch components to remove.

```
# estimate the number of batch components
sponge.batch.tune <- PLSDA_batch(X = sponge.clr,
                                 Y.trt = sponge.trt,
                                 Y.bat = sponge.batch,
                                 ncomp.trt = 1,
                                 ncomp.bat = 5)
sponge.batch.tune$explained_variance.bat #1
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5
## 0.27660372 0.08286158 0.08187289 0.07026693 0.07081271
##
## $Y
## comp1 comp2 comp3 comp4 comp5
##     1     1     1     1     1
```

Using the same criterion as choosing treatment components, we choose the number of batch components that explains 100% variance in the outcome matrix of batch. According to the result, 1 component was required to remove batch effects.

We then can correct for batch effects applying `PLSDA_batch()` with treatment, batch grouping information and corresponding optimal number of related components.

```
sponge.PLSDA_batch.res <- PLSDA_batch(X = sponge.clr,
                                      Y.trt = sponge.trt,
                                      Y.bat = sponge.batch,
                                      ncomp.trt = 1,
                                      ncomp.bat = 1)
sponge.PLSDA_batch <- sponge.PLSDA_batch.res$X.nobatch
```

sPLSDA-batch

We apply sPLSDA-batch using the same function `PLSDA_batch()` but we specify the number of variables to select on each component (usually only treatment-related components `keepX.trt`). To determine the optimal number of variables to select, we use `tune.splsda()` function from mixOmics package [Rohart et al., 2017] with all possible numbers of variables to select for each component (`test.keepX`).

```
# estimate the number of variables to select per treatment component
set.seed(777)
sponge.test.keepX = seq(1, 24, 1)
sponge.trt.tune.v <- tune.splsda(X = sponge.clr,
                                 Y = sponge.trt,
                                 ncomp = 1,
                                 test.keepX = sponge.test.keepX,
                                 validation = 'Mfold',
                                 folds = 4, nrepeat = 50)
sponge.trt.tune.v$choice.keepX #1
```

```
## comp1
##     1
```

Here the optimal number of variables to select for the treatment component was 1. Since we have adjusted the amount of treatment variation to preserve, we need to re-choose the optimal number of components related to batch effects using the same criterion mentioned in section **correcting for batch effects** method “PLSDA-batch”.

```
# estimate the number of batch components
sponge.batch.tune <- PLSDA_batch(X = sponge.clr,
                                 Y.trt = sponge.trt,
                                 Y.bat = sponge.batch,
                                 ncomp.trt = 1,
                                 keepX.trt = 1,
                                 ncomp.bat = 5)
sponge.batch.tune$explained_variance.bat #1
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5
## 0.25672979 0.07716171 0.08936628 0.08434746 0.09422519
##
## $Y
## comp1 comp2 comp3 comp4 comp5
##     1     1     1     1     1
```

According to the result, we needed only 1 batch related component to remove batch variance from the data with function `PLSDA_batch()`.

```
sponge.sPLSDA_batch.res <- PLSDA_batch(X = sponge.clr,
                                       Y.trt = sponge.trt,
                                       Y.bat = sponge.batch,
                                       ncomp.trt = 1,
                                       keepX.trt = 1,
                                       ncomp.bat = 1)
sponge.sPLSDA_batch <- sponge.sPLSDA_batch.res$X.nobatch
```

Note: for unbalanced batch x treatment design (with the exception of the nested design), we can specify `balance = FALSE` in `PLSDA_batch()` function to apply weighted PLSDA-batch.

Percentile Normalisation

To apply `percentile_norm()` function from PLSDAbatch package, we need to indicate a control group (`ctrl.grp`).

```
sponge.PN <- percentile_norm(data = sponge.clr,
                             batch = sponge.batch,
                             trt = sponge.trt,
                             ctrl.grp = 'C')
```

2.1.4 Assessing batch effect correction

We apply different visualisation and quantitative methods to assessing batch effect correction.

2.1.4.1 Methods that detect batch effects

PCA

In the sponge data, we compared the PCA sample plots before and after batch effect correction with different methods.

```
sponge.pca.before <- pca(sponge.clr, ncomp = 3,
                         scale = TRUE)
sponge.pca.rBE <- pca(sponge.rBE, ncomp = 3,
                      scale = TRUE)
sponge.pca.ComBat <- pca(sponge.ComBat, ncomp = 3,
                         scale = TRUE)
sponge.pca.PLSDA_batch <- pca(sponge.PLSDA_batch, ncomp = 3,
                              scale = TRUE)
sponge.pca.sPLSDA_batch <- pca(sponge.sPLSDA_batch, ncomp = 3,
                               scale = TRUE)
sponge.pca.PN <- pca(sponge.PN, ncomp = 3,
                     scale = TRUE)
```

```
sponge.pca.before.plot <-
  Scatter_Density(object = sponge.pca.before,
                  batch = sponge.batch,
                  trt = sponge.trt,
                  title = 'Before',
                  trt.legend.title = 'Tissue')
```

```
sponge.pca.rBE.plot <-
  Scatter_Density(object = sponge.pca.rBE,
                  batch = sponge.batch,
                  trt = sponge.trt,
                  title = 'removeBatchEffect',
                  trt.legend.title = 'Tissue')
```

```
sponge.pca.ComBat.plot <-
  Scatter_Density(object = sponge.pca.ComBat,
                  batch = sponge.batch,
                  trt = sponge.trt,
                  title = 'ComBat',
                  trt.legend.title = 'Tissue')
```

```
sponge.pca.PLSDA_batch.plot <-
  Scatter_Density(object = sponge.pca.PLSDA_batch,
                  batch = sponge.batch,
                  trt = sponge.trt,
                  title = 'PLSDA-batch',
                  trt.legend.title = 'Tissue')
```

```
sponge.pca.sPLSDA_batch.plot <-
  Scatter_Density(object = sponge.pca.sPLSDA_batch,
                  batch = sponge.batch,
                  trt = sponge.trt,
                  title = 'sPLSDA-batch',
                  trt.legend.title = 'Tissue')
```

```
sponge.pca.PN.plot <-
  Scatter_Density(object = sponge.pca.PN,
                  batch = sponge.batch,
                  trt = sponge.trt,
                  title = 'Percentile Normalisation',
                  trt.legend.title = 'Tissue')
```

As shown in the PCA sample plots, the difference between the samples from two different batches was removed after batch effect correction with most methods except percentile normalisation. In the data corrected with percentile normalisation, the samples from tissue E still included a distinction between samples from two batches. We can also compare the boxplots and density plots for key variables identified in PCA driving the major variance or heatmaps showing obvious patterns before and after batch effect correction (results not shown).

pRDA

We calculate the global explained variance across all microbial variables using pRDA. To achieve this, we create a loop for each variable from the original (uncorrected) and batch effect-corrected data. The final results are then displayed with `partVar_plot()` from PLSDAbatch package.

```
sponge.corrected.list <- list(`Before correction` = sponge.clr,
                              removeBatchEffect = sponge.rBE,
                              ComBat = sponge.ComBat,
                              `PLSDA-batch` = sponge.PLSDA_batch,
                              `sPLSDA-batch` = sponge.sPLSDA_batch,
                              `Percentile Normalisation` = sponge.PN)

sponge.prop.df <- data.frame(Treatment = NA, Batch = NA,
                             Intersection = NA,
                             Residuals = NA)
for(i in 1:length(sponge.corrected.list)){
  rda.res = varpart(sponge.corrected.list[[i]], ~ trt, ~ batch,
                    data = sponge.factors.df, scale = T)
  sponge.prop.df[i, ] <- rda.res$part$indfract$Adj.R.squared}

rownames(sponge.prop.df) = names(sponge.corrected.list)

sponge.prop.df <- sponge.prop.df[, c(1,3,2,4)]
```

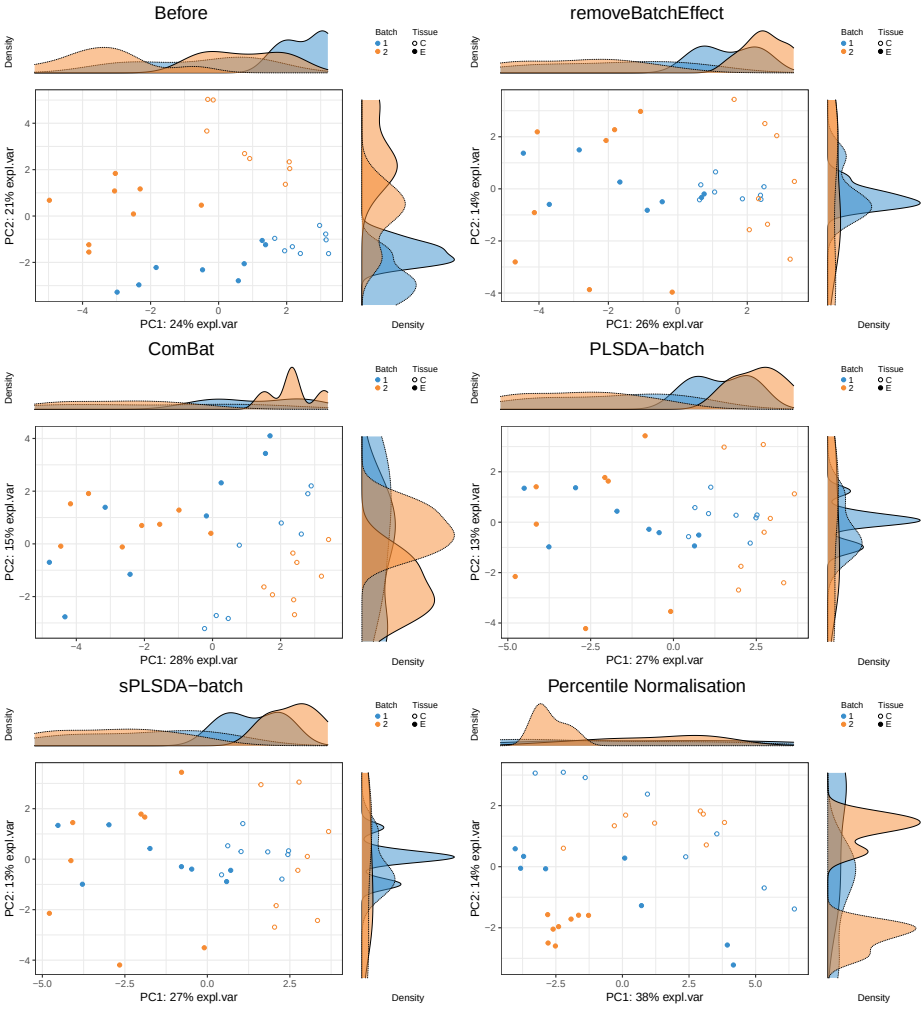

Figure 2.4: The PCA sample plots with densities before and after batch effect correction in the sponge data.

```
sponge.prop.df[sponge.prop.df < 0] = 0
sponge.prop.df <- as.data.frame(t(apply(sponge.prop.df, 1,
                                        function(x){x/sum(x)})))


partVar_plot(prop.df = sponge.prop.df)
```

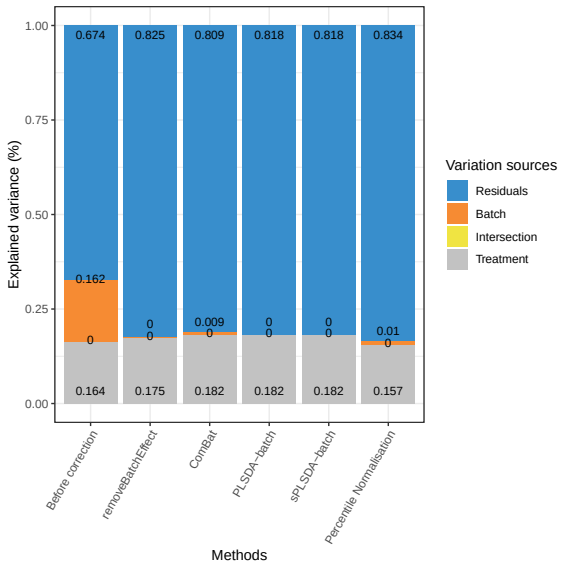

Figure 2.5: Global explained variance before and after batch effect correction for the sponge data.

As shown in the above figure, the intersection between batch and treatment variance was none for the sponge data, because the batch x treatment design is balanced. PLSDA-batch and sPLSDA-batch corrected data led to the best performance with a slightly higher proportion of explained treatment variance and no batch variance compared to the other methods.

2.1.4.2 Other methods

R^2

The R^2 values for each variable are calculated with `lm()` from stats package. To compare the R^2 values among variables, we scale the corrected data before R^2 calculation. The results are displayed with `ggplot()` from ggplot2 R package.

```
# scale
sponge.corr_scale.list <- lapply(sponge.corrected.list,
                                 function(x){apply(x, 2, scale)})
```

```
sponge.r_values.list <- list()
for(i in 1:length(sponge.corr_scale.list)){
  sponge.r_values <- data.frame(trt = NA, batch = NA)
  for(c in 1:ncol(sponge.corr_scale.list[[i]])){
    sponge.fit.res.trt <- lm(sponge.corr_scale.list[[i]][,c] ~ sponge.trt)
    sponge.r_values[c,1] <- summary(sponge.fit.res.trt)$r.squared
    sponge.fit.res.batch <- lm(sponge.corr_scale.list[[i]][,c] ~ sponge.batch)
    sponge.r_values[c,2] <- summary(sponge.fit.res.batch)$r.squared
  }
  sponge.r_values.list[[i]] <- sponge.r_values
}
names(sponge.r_values.list) <- names(sponge.corr_scale.list)

sponge.boxp.list <- list()
for(i in seq_len(length(sponge.r_values.list))){
  sponge.boxp.list[[i]] <-
    data.frame(r2 = c(sponge.r_values.list[[i]][ ,'trt'],
                      sponge.r_values.list[[i]][ ,'batch']),
               Effects = as.factor(rep(c('Treatment','Batch'),
                                       each = 24)))
}
names(sponge.boxp.list) <- names(sponge.r_values.list)

sponge.r2.boxp <- rbind(sponge.boxp.list$`Before correction`,
                        sponge.boxp.list$removeBatchEffect,
                        sponge.boxp.list$ComBat,
                        sponge.boxp.list$`PLSDA-batch`,
                        sponge.boxp.list$`sPLSDA-batch`,
                        sponge.boxp.list$`Percentile Normalisation`,
                        sponge.boxp.list$RUVIII)

sponge.r2.boxp$methods <- rep(c('Before correction', ' removeBatchEffect',
                                'ComBat','PLSDA-batch', 'sPLSDA-batch',
                                'Percentile Normalisation'), each = 48)

sponge.r2.boxp$methods <- factor(sponge.r2.boxp$methods,
                                 levels = unique(sponge.r2.boxp$methods))

ggplot(sponge.r2.boxp, aes(x = Effects, y = r2, fill = Effects)) +
  geom_boxplot(alpha = 0.80) +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 60, hjust = 1, size = 18),
```

```
        axis.text.y = element_text(size = 18),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid( ~ methods) +
  scale_fill_manual(values=pb_color(c(12,14)))
```

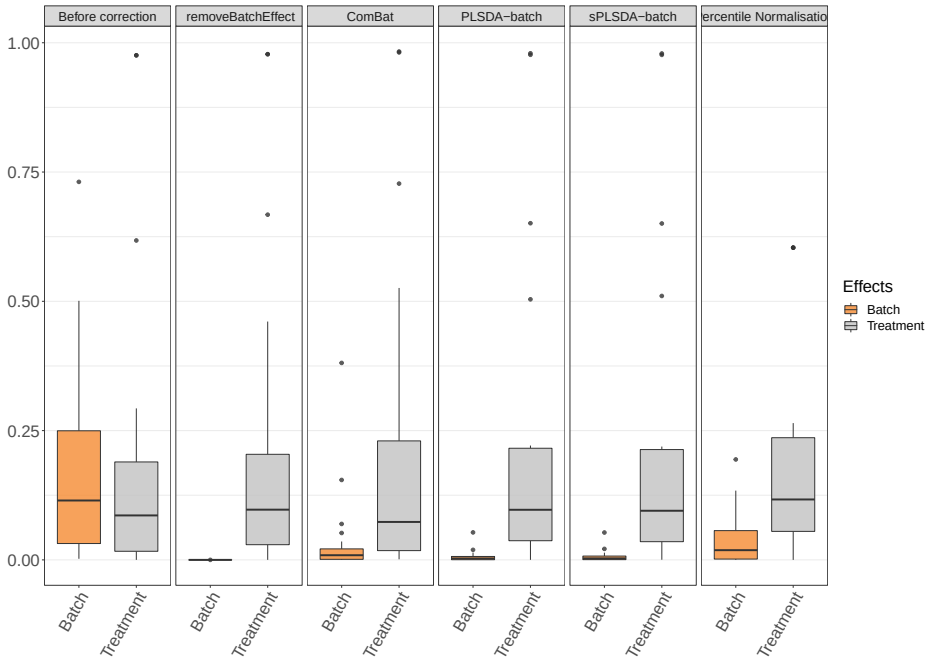

Figure 2.6: Sponge study: R^2 values for each microbial variable before and after batch effect correction.

We observed that the data corrected by ComBat, percentile normalisation still included a few variables with a large proportion of batch variance as shown in the above figures.

```
###################################
sponge.barp.list <- list()
for(i in seq_len(length(sponge.r_values.list))){
  sponge.barp.list[[i]] <-
    data.frame(r2 = c(sum(sponge.r_values.list[[i]][ ,'trt']),
                      sum(sponge.r_values.list[[i]][ ,'batch'])),
               Effects = c('Treatment','Batch'))
}
names(sponge.barp.list) <- names(sponge.r_values.list)

sponge.r2.barp <- rbind(sponge.barp.list$`Before correction`,
                        sponge.barp.list$removeBatchEffect,
```

```
                        sponge.barp.list$ComBat,
                        sponge.barp.list$`PLSDA-batch`,
                        sponge.barp.list$`sPLSDA-batch`,
                        sponge.barp.list$`Percentile Normalisation`,
                        sponge.barp.list$RUVIII)


sponge.r2.barp$methods <- rep(c('Before correction', ' removeBatchEffect',
                                'ComBat','PLSDA-batch', 'sPLSDA-batch',
                                'Percentile Normalisation'), each = 2)

sponge.r2.barp$methods <- factor(sponge.r2.barp$methods,
                                 levels = unique(sponge.r2.barp$methods))


ggplot(sponge.r2.barp, aes(x = Effects, y = r2, fill = Effects)) +
  geom_bar(stat="identity") +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 60, hjust = 1, size = 18),
        axis.text.y = element_text(size = 18),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid( ~ methods) +
  scale_fill_manual(values=pb_color(c(12,14)))
```

When considering the sum of all variables, the remaining batch variance of corrected data from sPLSDA-batch was similar as PLSDA-batch, which was greater than removeBatchEffect. The preserved treatment variance of corrected data from sPLSDA-batch was also similar as PLSDA-batch which was greater than removeBatchEffect.

Alignment scores

To use the `alignment_score()` function from PLSDAbatch, we need to specify the proportion of data variance to explain (`var`), the number of nearest neighbours (`k`) and the number of principal components to estimate (`ncomp`). We then use `ggplot()` function from ggplot2 to visualise the results.

```
sponge.var = 0.95
sponge.k = 3

sponge.scores <- c()
for(i in 1:length(sponge.corrected.list)){
  res <- alignment_score(data = sponge.corrected.list[[i]],
                         batch = sponge.batch,
```

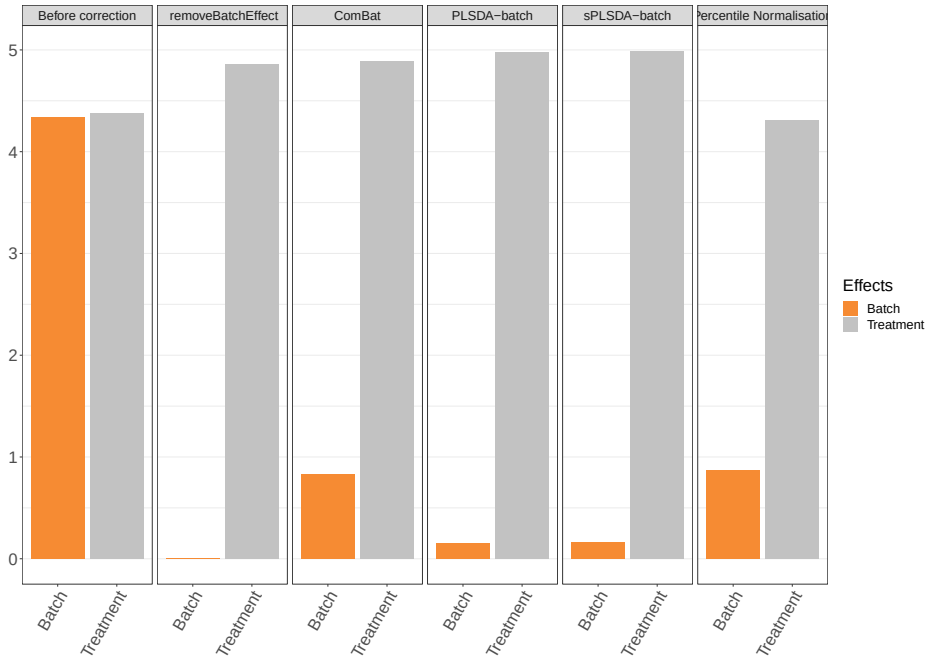

Figure 2.7: Sponge study: the sum of R^2 values for each microbial variable before and after batch effect correction.

```
                          var = sponge.var,
                          k = sponge.k,
                          ncomp = 20)
  sponge.scores <- c(sponge.scores, res)
}

sponge.scores.df <- data.frame(scores = sponge.scores,
                               methods = names(sponge.corrected.list))

sponge.scores.df$methods <- factor(sponge.scores.df$methods,
                                   levels = rev(names(sponge.corrected.list)))


ggplot() + geom_col(aes(x = sponge.scores.df$methods,
                        y = sponge.scores.df$scores)) +
  geom_text(aes(x = sponge.scores.df$methods,
                y = sponge.scores.df$scores/2,
                label = round(sponge.scores.df$scores, 3)),
            size = 3, col = 'white') +
  coord_flip() + theme_bw() + ylab('Alignment Scores') +
  xlab('') + ylim(0,0.4)
```

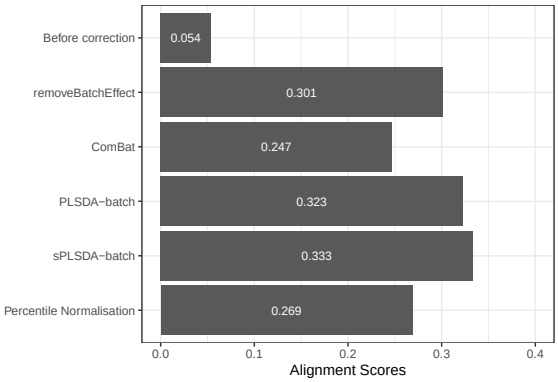

Figure 2.8: Comparison of alignment scores before and after batch effect correction using different methods for the sponge data.

The alignment scores complement the PCA results, especially when batch effect removal is difficult to assess on PCA sample plots. Since a higher alignment score indicates that samples are better mixed, as shown in the above figure, sPLSDA-batch gave a superior performance compared to the other methods.

Variable selection

We use `splsda()` from mixOmics to select the top 5 microbial variables that, in combination, discriminate the different treatment groups in the sponge data. We apply

`splsda()` to the different batch effect corrected data from all methods. Then we use `upset()` from UpSetR package [Lex et al., 2014] to visualise the concordance of variables selected.

In the code below, we first need to convert the list of variables selected from different method-corrected data into a data frame compatible with `upset()` using `fromList()`. We then assign different colour schemes for each variable selection.

```
sponge.splsda.select <- list()
for(i in 1:length(sponge.corrected.list)){
  splsda.res <- splsda(X = sponge.corrected.list[[i]],
                       Y = sponge.trt,
                       ncomp = 3, keepX = rep(5,3))
  select.res <- selectVar(splsda.res, comp = 1)$name
  sponge.splsda.select[[i]] <- select.res
}
names(sponge.splsda.select) <- names(sponge.corrected.list)

# can only visualise 5 methods
sponge.splsda.select <- sponge.splsda.select[c(1:5)]

sponge.splsda.upsetR <- fromList(sponge.splsda.select)

upset(sponge.splsda.upsetR, main.bar.color = 'gray36',
      sets.bar.color = pb_color(c(25:22,20)), matrix.color = 'gray36',
      order.by = 'freq', empty.intersections = 'on',
      queries =
        list(list(query = intersects, params = list('Before correction'),
                  color = pb_color(20), active = T),
             list(query = intersects, params = list('removeBatchEffect'),
                  color = pb_color(22), active = T),
             list(query = intersects, params = list('ComBat'),
                  color = pb_color(23), active = T),
             list(query = intersects, params = list('PLSDA-batch'),
                  color = pb_color(24), active = T),
             list(query = intersects, params = list('sPLSDA-batch'),
                  color = pb_color(25), active = T)))
```

In the above figure, the left bars indicate the number of variables selected from each data corrected with different methods. The right bar plot combined with the scatterplot show different intersection and their aggregates. We obtained an overlap of 4 out of 5 selected variables between different corrected and uncorrected data. However, the data from each method still included unique variables that were not selected in the other corrected data, e.g., ComBat and PLSDA-batch. As `upset()` can only include five datasets at once, we only displayed the uncorrected data and four corrected data that had been more efficiently corrected for batch effects from our previous assessments compared to the other datasets.

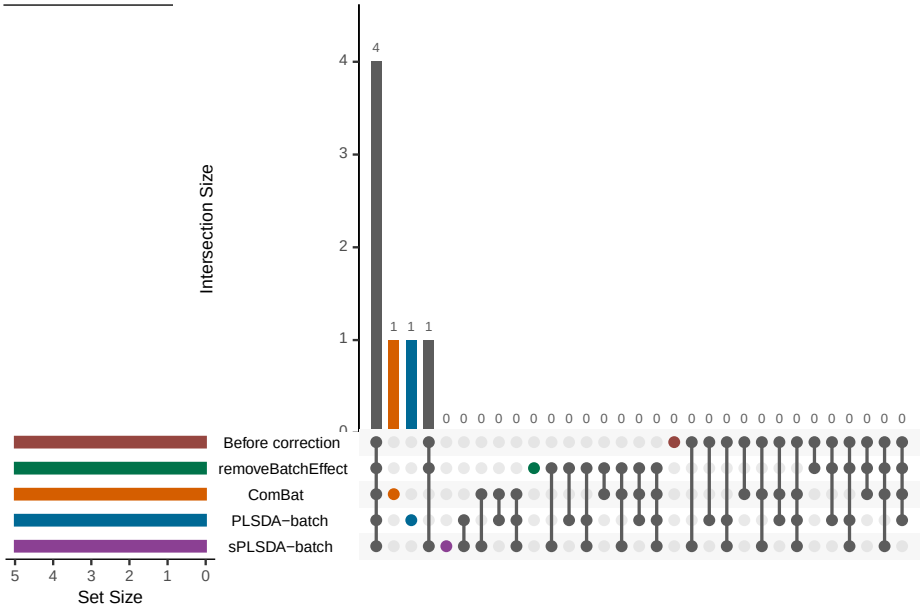

Figure 2.9: UpSet plot showing overlap between variables selected from different corrected data for the sponge study.

2.2 HFHS data

2.2.1 Data pre-processing

2.2.1.1 Prefiltering

We load the HFHS data stored internally with function `data()`.

```
data('HFHS_data')
hfhs.count <- HFHS_data$FullData$X.count
dim(hfhs.count)
```

```
## [1]  250 4524
```

```
hfhs.filter.res <- PreFL(data = hfhs.count)
hfhs.filter <- hfhs.filter.res$data.filter
dim(hfhs.filter)
```

```
## [1] 250 515
```

```
# zero proportion before filtering
hfhs.filter.res$zero.prob
```

```
## [1] 0.8490805
```

```
# zero proportion after filtering
sum(hfhs.filter == 0)/(nrow(hfhs.filter) * ncol(hfhs.filter))
```

```
## [1] 0.2955184
```

The raw HFHS data include 4524 OTUs and 250 samples. We use the function `PreFL()` from our PLSDAbatch R package to filter the data. After filtering, the HFHS data were reduced to 515 OTUs and 250 samples. The proportion of zeroes was reduced from 85% to 30%.

2.2.1.2 Transformation

Prior to CLR transformation, we recommend adding 1 as the offset for the HFHS data - that are raw count data. We use `logratio.transfo()` function in mixOmics package to CLR transform the data.

```
hfhs.clr <- logratio.transfo(X = hfhs.filter, logratio = 'CLR', offset = 1)

class(hfhs.clr) <- 'matrix'
```

2.2.2 Batch effect detection

2.2.2.1 PCA

We apply `pca()` function from mixOmics package to the HFHS data and use `Scatter_Density()` function from PLSDAbatch to represent the PCA sample plot with densities.

```
hfhs.pca.before <- pca(hfhs.clr, ncomp = 3, scale = TRUE)

hfhs.metadata <- HFHS_data$FullData$metadata
rownames(hfhs.metadata) <- hfhs.metadata$SampleID
hfhs.metadata <- hfhs.metadata[rownames(hfhs.clr),]

hfhs.cage <- as.factor(hfhs.metadata$DietCage)
hfhs.day <- as.factor(hfhs.metadata$Day)
hfhs.trt <- as.factor(hfhs.metadata$Diet)
names(hfhs.cage) = names(hfhs.day) = names(hfhs.trt) = hfhs.metadata$SampleID

hfhs.pca.before.cage.plot <-
  Scatter_Density(object = hfhs.pca.before,
                  batch = hfhs.cage,
                  trt = hfhs.trt,
                  title = 'HFHS data',
                  trt.legend.title = 'Diet',
                  batch.legend.title = 'Cage')
```

In the above figure, part of the samples with different diets were mixed together, while

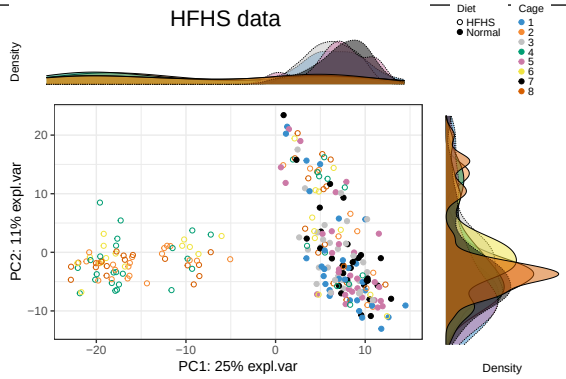

Figure 2.10: The PCA sample plot with densities coloured by cages in the HFHS data.

all the samples from different batches (i.e., cages) were well mixed and not distinct. This result indicates no cage effect for the whole data and no treatment effect for part of samples.

```
hfhs.pca.before.day.plot <-
  Scatter_Density(object = hfhs.pca.before,
                  batch = hfhs.day,
                  trt = hfhs.trt,
                  title = 'HFHS data',
                  trt.legend.title = 'Diet',
                  batch.legend.title = 'Day')
```

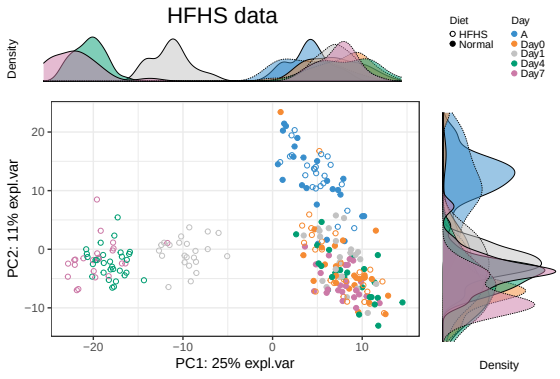

Figure 2.11: Figure 11: The PCA sample plot with densities coloured by days in the HFHS data.

As shown in the above figure, we did not observe a difference between samples with different diets on arrival day “A” and “Day0”. This result was consistent with the experimental design, as mice were not treated with different diets on these two days. We

thus removed the samples from these two days.

```
# remove samples from arrival day and day0
hfhs.dA0.idx <- c(which(hfhs.day == 'A'), which(hfhs.day == 'Day0'))
hfhs.clr.less <- hfhs.clr[-hfhs.dA0.idx,]
hfhs.metadata.less <- hfhs.metadata[-hfhs.dA0.idx,]

hfhs.cage.less <- as.factor(hfhs.metadata.less$DietCage)
hfhs.day.less <- as.factor(hfhs.metadata.less$Day)
hfhs.trt.less <- as.factor(hfhs.metadata.less$Diet)
names(hfhs.cage.less) = names(hfhs.day.less) =
  names(hfhs.trt.less) = hfhs.metadata.less$SampleID

hfhs.less.pca.before <- pca(hfhs.clr.less, ncomp = 3, scale = TRUE)

hfhs.less.pca.before.cage.plot <-
  Scatter_Density(object = hfhs.less.pca.before,
                  batch = hfhs.cage.less,
                  trt = hfhs.trt.less,
                  title = 'HFHS data',
                  trt.legend.title = 'Diet',
                  batch.legend.title = 'Cage')
```

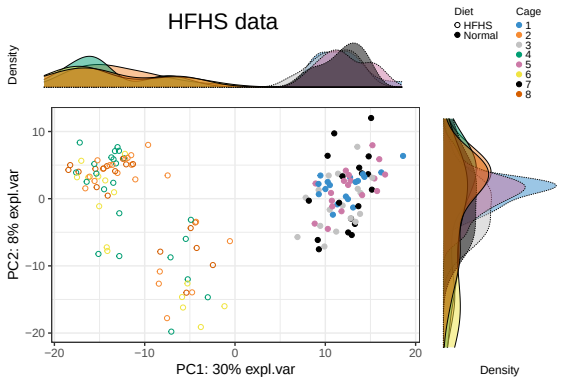

Figure 2.12: The PCA sample plot with densities coloured by cages in the reduced HFHS data.

```
hfhs.less.pca.before.day.plot <-
  Scatter_Density(object = hfhs.less.pca.before,
                  batch = hfhs.day.less,
                  trt = hfhs.trt.less,
                  title = 'HFHS data',
                  trt.legend.title = 'Diet',
                  batch.legend.title = 'Day')
```

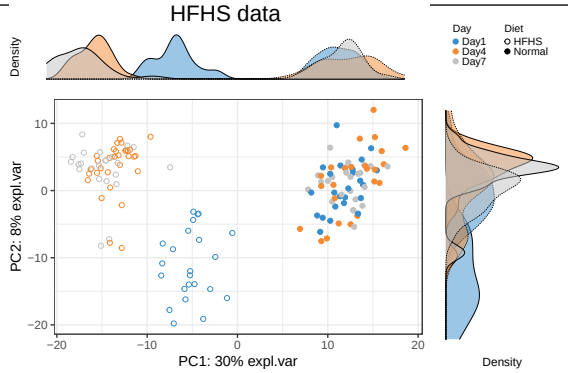

Figure 2.13: The PCA sample plot with densities coloured by days in the reduced HFHS data.

We observed a substantial diet variation on component 1, no cage variation, and a difference between samples collected on "Day1" and the other days with the HFHS diet as shown in the above figures.

2.2.2.2 Heatmap

We produce a heatmap using pheatmap package. The data first need to be scaled on both OTUs and samples.

```
# scale the clr data on both OTUs and samples
hfhs.clr.s <- scale(hfhs.clr.less, center = T, scale = T)
hfhs.clr.ss <- scale(t(hfhs.clr.s), center = T, scale = T)

# The cage effect
hfhs.anno_col <- data.frame(Cage = hfhs.cage.less,
                            Day = hfhs.day.less,
                            Treatment = hfhs.trt.less)
hfhs.anno_colors <- list(Cage = color.mixo(1:8),
                         Day = color.mixo(1:3),
                         Treatment = pb_color(1:2))
names(hfhs.anno_colors$Cage) = levels(hfhs.cage.less)
names(hfhs.anno_colors$Day) = levels(hfhs.day.less)
names(hfhs.anno_colors$Treatment) = levels(hfhs.trt.less)

pheatmap(hfhs.clr.ss,
         cluster_rows = F,
         fontsize_row = 4,
         fontsize_col = 6,
         fontsize = 8,
         clustering_distance_rows = 'euclidean',
```

```
        clustering_method = 'ward.D',
        treeheight_row = 30,
        annotation_col = hfhs.anno_col,
        annotation_colors = hfhs.anno_colors,
        border_color = 'NA',
        main = 'HFHS data - Scaled')
```

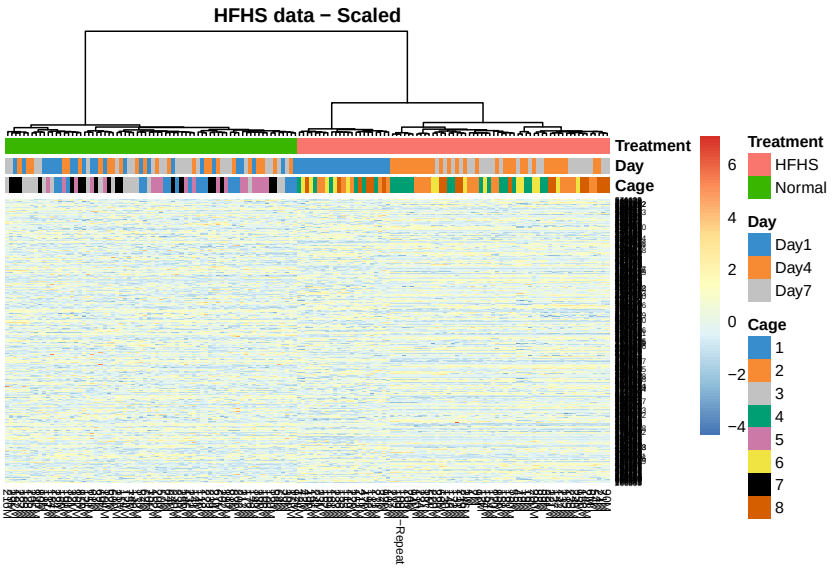

Figure 2.14: Hierarchical clustering for samples in the HFHS data.

In the above figure, samples in the HFHS data were clustered according to the treatments not cages, indicating no cage effect and a strong treatment effect. Samples from the batch "Day1" were clustered and distinct from the other batches with diet HFHS but not the normal diet, indicating a day effect but very weak.

2.2.2.3 pRDA

We apply pRDA with `varpart()` function from vegan R package.

```
hfhs.factors.df <- data.frame(trt = hfhs.trt.less,
                              cage = hfhs.cage.less,
                              day = hfhs.day.less)
hfhs.rda.cage.before <- varpart(hfhs.clr.less, ~ trt, ~ cage,
                                data = hfhs.factors.df, scale = T)
hfhs.rda.cage.before$part$indfract
```

```
##                 Df R.squared Adj.R.squared Testable
## [a] = X1|X2      0        NA  7.771561e-16    FALSE
## [b] = X2|X1      6        NA  1.277627e-02     TRUE
```

```
## [c]                0         NA  2.713111e-01    FALSE
## [d] = Residuals NA           NA  7.159126e-01    FALSE
```

In the result, `X1` and `X2` represent the first and second covariates fitted in the model. `[a]`, `[b]` represent the independent proportion of variance explained by `X1` and `X2` respectively, and `[c]` represents the intersection variance shared between `X1` and `X2`. In the HFHS data, the cage and diet effects have a nested batch x treatment design that is extremely unbalanced. We didn't detect any treatment variance (indicated in line `[a]`, Adj.R.squared = 0) but considerable intersection variance (indicated in line `[c]`, Adj.R.squared = 0.271). Thus, all the treatment variance was explained by the intersection variance shared with batch variance. It means batch and treatment effects are collinear. Therefore, the cage effect in the HFHS data can only be accounted for with a linear mixed model, as detailed in section **accounting for batch effects** method "Linear regression".

```
hfhs.rda.day.before <- varpart(hfhs.clr.less, ~ trt, ~ day,
                               data = hfhs.factors.df, scale = T)
hfhs.rda.day.before$part$indfract
```

```
##                 Df R.squared Adj.R.squared Testable
## [a] = X1|X2      1         NA  0.2721810304     TRUE
## [b] = X2|X1      2         NA  0.0361767632     TRUE
## [c]              0         NA -0.0008699296    FALSE
## [d] = Residuals NA         NA  0.6925121361    FALSE
```

For the day and diet effects, the batch x treatment design is balanced. There was no intersection variance (indicated in line `[c]`, Adj.R.squared = 0). The proportion of treatment variance was much higher than batch variance as indicated in lines `[a]` (Adj.R.squared = 0.272) and `[b]` (Adj.R.squared = 0.036).

2.2.3 Managing batch effects

2.2.3.1 Accounting for batch effects

The methods that we use to account for batch effects include the methods designed for microbiome data: zero-inflated Gaussian (ZIG) mixture model and the methods adapted for microbiome data: linear regression, SVA and RUV4. Among them, SVA and RUV4 are designed for unknown batch effects.

Methods designed for microbiome data

Zero-inflated Gaussian mixture model To use the ZIG model, we first create a `MRexperiment` object applying `newMRexperiment()` (from metagenomeSeq package) to microbiome counts and annotated data frames with metadata and taxonomic information generated with `AnnotatedDataFrame()` from Biobase package.

```
# Creating a MRexperiment object (make sure no NA in metadata)
rownames(HFHS_data$FullData$metadata) <- rownames(HFHS_data$FullData$X.count)
HFHS.phenotypeData =
```

```
  AnnotatedDataFrame(data = HFHS_data$FullData$metadata[-hfhs.dA0.idx,])
HFHS.taxaData =
  AnnotatedDataFrame(data = as.data.frame(HFHS_data$FullData$taxa))
HFHS.obj = newMRexperiment(counts =
                             t(HFHS_data$FullData$X.count[-hfhs.dA0.idx,]),
                           phenoData = HFHS.phenotypeData,
                           featureData = HFHS.taxaData)
HFHS.obj
```

```
## MRexperiment (storageMode: environment)
## assayData: 4524 features, 149 samples
##   element names: counts
## protocolData: none
## phenoData
##   sampleNames: 109M 131M ... 39M (149 total)
##   varLabels: SampleID MouseID ... Replicated (14 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: 4333897 541135 ... 228232 (4524 total)
##   fvarLabels: Kingdom Phylum ... Species (7 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

The HFHS data are then filtered with `filterData()` function (from metagenomeSeq). We can use `MRcounts()` to extract the count data from the `MRexperiment` object.

```
# filtering data to maintain a threshold of minimum depth or OTU presence
dim(MRcounts(HFHS.obj))
```

```
## [1] 4524  149
```

```
HFHS.obj = filterData(obj = HFHS.obj, present = 20, depth = 5)
dim(MRcounts(HFHS.obj))
```

```
## [1] 1255  149
```

After filtering, the HFHS data were reduced to 1255 OTUs and 149 samples.

We calculate the percentile for CSS normalisation with `cumNormStatFast()` function (from metagenomeSeq package). The CSS normalisation is applied with `cumNorm()` and the normalised data can be exported using `MRcounts()` with `norm = TRUE`. The normalisation scaling factors for each sample, which are the sum of counts up to the calculated percentile, can be accessed through `normFactors()`. We calculate the log transformed scaling factors by diving them with their median, which are better than the default scaling factors that are divided by 1000 (`log2(normFactors(obj)/1000 + 1)`).

```
# calculate the percentile for CSS normalisation
HFHS.pctl = cumNormStatFast(obj = HFHS.obj)
# CSS normalisation
HFHS.obj <- cumNorm(obj = HFHS.obj, p = HFHS.pctl)
# export normalised data
HFHS.norm.data <- MRcounts(obj = HFHS.obj, norm = TRUE)

# normalisation scaling factors for each sample
HFHS.normFactor = normFactors(object = HFHS.obj)
HFHS.normFactor = log2(HFHS.normFactor/median(HFHS.normFactor) + 1)
```

As the ZIG model applies a linear model, the cage effect cannot be fitted. Moreover, the cage effect is very weak which only accounted for 1.3% of the total data variance calculated with pRDA. Thus we only fit and account for the day effect. We create a design matrix with treatment variable (`Diet_effect`), batch variable (`Day_effect`) and the log transformed scaling factors by using `model.matrix()`, and then apply the ZIG model by `fitZig()` function. We set `useCSSoffset = FALSE` to avoid using the default scaling factors as we have already included our customised scaling factor (`HFHS.normFactor`) in the design matrix.

```
# treatment variable
Diet_effect = pData(object = HFHS.obj)$Diet

# batch variable
Day_effect = pData(object = HFHS.obj)$Day

# build a design matrix
HFHS.mod.full = model.matrix(~ Diet_effect + Day_effect + HFHS.normFactor)

# settings for the fitZig() function
HFHS.settings <- zigControl(maxit = 10, verbose = TRUE)

# apply the ZIG model
HFHSfit <- fitZig(obj = HFHS.obj, mod = HFHS.mod.full,
                  useCSSoffset = FALSE, control = HFHS.settings)
```

```
## it= 0, nll=236.34, log10(eps+1)=Inf, stillActive=1255
## it= 1, nll=256.75, log10(eps+1)=0.06, stillActive=203
## it= 2, nll=255.94, log10(eps+1)=0.04, stillActive=153
## it= 3, nll=256.56, log10(eps+1)=0.04, stillActive=47
## it= 4, nll=257.05, log10(eps+1)=0.01, stillActive=11
## it= 5, nll=257.28, log10(eps+1)=0.01, stillActive=3
## it= 6, nll=257.39, log10(eps+1)=0.02, stillActive=2
## it= 7, nll=257.46, log10(eps+1)=0.00, stillActive=0
```

The OTUs with the top 50 smallest p values are extracted using `MRcoefs()`. We set `eff = 0.5`, so only the OTUs with at least "0.5" quantile (50%) number of effective

samples (positive samples + estimated undersampling zeroes) are extracted.

```
HFHScoefs <- MRcoefs(HFHSfit, coef = 2, group = 3, number = 50, eff = 0.5)
head(HFHScoefs)
```

```
##         Diet_effectNormal      pvalues   adjPvalues
## 324926          1.0156067 7.360015e-11 9.236819e-08
## 337550          1.1730238 1.110138e-09 4.362904e-07
## 4353745        -0.7058975 1.212492e-09 4.362904e-07
## 357471          0.9636651 1.390567e-09 4.362904e-07
## 313499         -0.6093954 2.160477e-09 5.422797e-07
## 3940440        -0.7199750 5.150634e-09 1.077341e-06
```

Other methods adapted for microbiome data

Linear regression Linear regression is conducted with `linear_regres()` function in PLSDAbatch. We integrated the performance package that assesses performance of regression models into our function `linear_regres()`. Therefore, we can apply `check_model()` from performance to the outputs from `linear_regres()` to diagnose the validity of the model fitted with treatment and batch effects for each variable [LÃŒdecke et al., 2020]. We can extract performance measurements such as adjusted R2, RMSE, RSE, AIC and BIC for the models fitted with and without batch effects, which are also the outputs of `linear_regres()`.

To deal with the day effect only, we apply `type = "linear model"` because of the balanced batch x treatment design.

```
hfhs.lm <- linear_regres(data = hfhs.clr.less,
                         trt = hfhs.trt.less,
                         batch.fix = hfhs.day.less,
                         type = 'linear model')

hfhs.p <- sapply(hfhs.lm$lm.table, function(x){x$coefficients[2,4]})
hfhs.p.adj <- p.adjust(p = hfhs.p, method = 'fdr')

check_model(hfhs.lm$model$`541135`)
```

To diagnose the validity of the model fitted with both treatment and batch effects, we use different plots to check the assumptions of each microbial variable. For example, the diagnostic plots of "OTU 541135" are shown in the above figure panel. The simulated data under the fitted model were not very close to the real data (shown as green line), indicating an imperfect fitness (top panel). The linearity (or homoscedasticity) and homogeneity of variance were not satisfied. The correlation between batch (`batch.fix`) and treatment (`trt`) effects was very low, indicating a good model with low collinearity. Some samples could be classified as outliers with a Cook's distance larger than or equal to 0.5, for example, "46", "35", "126", "125" and "16" (middle panel). The distribution of residuals was very close to normal (bottom panel). For the microbial variables with some assumptions not met, we should be careful about their results.

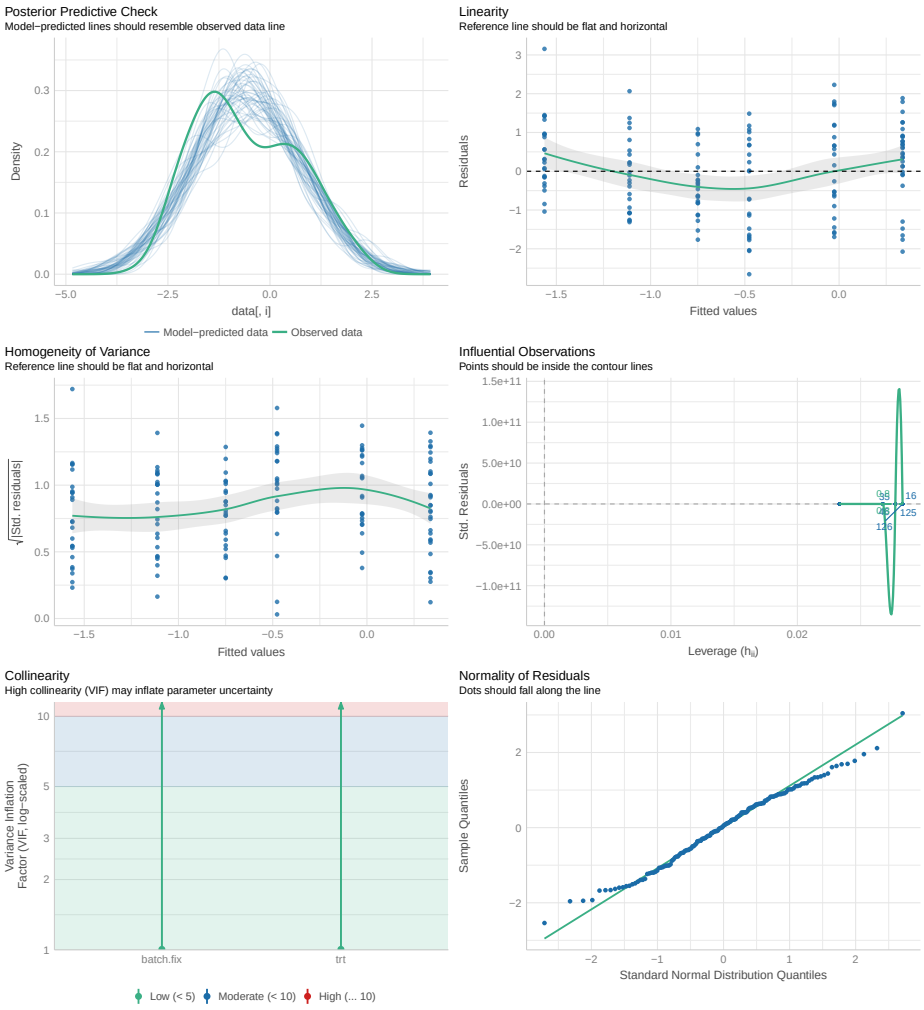

Figure 2.15: Diagnostic plots for the model fitted with the day effect of ”OTU 541135” in the HFHS data.

For performance measurements of models fitted with or without batch effects, We show an example of the results for some variables.

```
head(hfhs.lm$adj.R2)
```

```
##         trt.only trt.batch
## 541135 0.1969688 0.2590190
## 276629 0.6419160 0.6414949
## 196070 0.1618319 0.1530262
## 318732 0.1720195 0.1795344
## 339886 0.1542615 0.1779073
## 337724 0.2343320 0.2445358
```

If the adjusted R^2 of the model with both treatment and batch effects is larger than the model with treatment effects only, e.g., OTU 541135, the model fitted with batch effects explains more data variance, and is thus better than the model without batch effects. Otherwise, the batch effect is not necessary to be added into the linear model.

We next look at the AIC of models fitted with or without batch effects.

```
head(hfhs.lm$AIC)
```

```
##        trt.only trt.batch
## 541135 462.0799  452.0564
## 276629 294.7536  296.8876
## 196070 504.2549  507.7710
## 318732 462.8639  463.4643
## 339886 248.0616  245.7953
## 337724 424.2142  424.1741
```

A lower AIC indicates a better fit. Both results were mostly consistent on model selection, except "OTU318732". Based on adjusted R^2, we needed to include batch effects, while based on AIC, we should not. Therefore, we needed more measurements such as BIC, RSE.

To consider the cage effect only or the cage and day effects together, we apply `type = "linear mixed model"` to the HFHS data because of the nested batch x treatment design for the cage effect.

```
hfhs.lmm <- linear_regres(data = hfhs.clr.less,
                          trt = hfhs.trt.less,
                          batch.fix = hfhs.day.less,
                          batch.random = hfhs.cage.less,
                          type = 'linear mixed model')
```

```
hfhs.p <- sapply(hfhs.lmm$lmm.table, function(x){x$coefficients[2,5]})
hfhs.p.adj <- p.adjust(p = hfhs.p, method = 'fdr')

check_model(hfhs.lmm$model$`541135`)
```

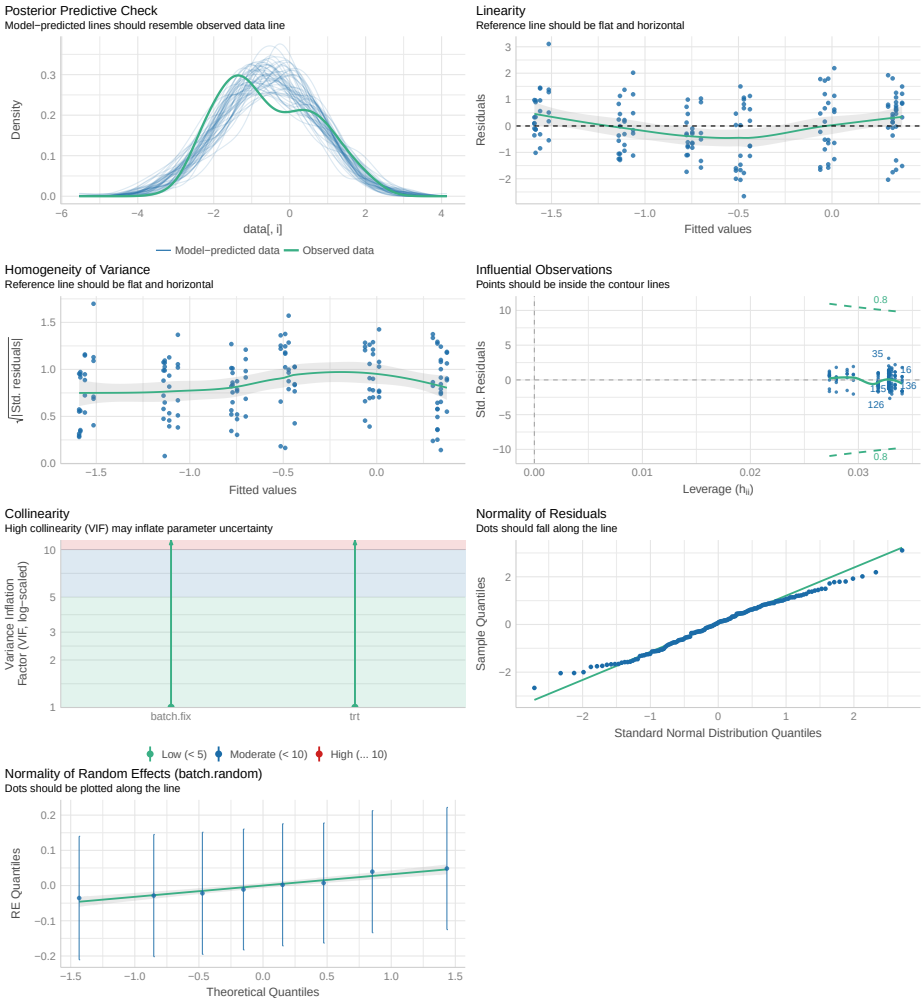

Figure 2.16: Diagnostic plots for the model fitted with the day and cage effects of ”OTU541135” in the HFHS data.

According to the diagnostic plots of "OTU541135" as shown in the above figure panel, The simulated data under the fitted model were not very close to the real data (shown as green line), indicating an imperfect fitness (top panel). The linearity (or homoscedasticity) and homogeneity of variance were not satisfied. The correlation between batch (`batch.fix`) and treatment (`trt`) effects was very low, indicating a good model with low collinearity. Some samples could be classified as outliers with a Cook's distance larger than or equal to 0.5, for example, "16", "35", "126", "125" and "136" (middle panel). The distribution of residuals and random effects was very close to normal (bottom panel).

For performance measurements of models fitted with or without batch effects, We show an example of the results for some variables.

```
head(hfhs.lmm$AIC)
```

```
##        trt.only trt.batch
## 541135 462.0799  461.4074
## 276629 294.7536  310.3455
## 196070 504.2549  515.0180
## 318732 462.8639  472.5556
## 339886 248.0616  260.7115
## 337724 424.2142  432.9937
```

For some OTUs, the model fitted without batch effects was better with a lower AIC. For example, "OTU276629", "OTU196070", "OTU318732", "OTU339886" and "OTU337724". "OTU541135" was more appropriate within a model with batch effects.

Since we apply a `"linear mixed model"` to the HFHS data, this type of model can only output conditional R^2 that includes the variance of both fixed and random effects (treatment, fixed and random batch effects) and marginal R^2 that includes only the variance of fixed effects (treatment and fixed batch effects) [Nakagawa and Schielzeth, 2013].

```
head(hfhs.lmm$cond.R2)
```

```
##         trt.only trt.batch
## 541135 0.2023947 0.2748381
## 276629 0.6443355 0.6475581
## 196070 0.1674952 0.1932495
## 318732 0.1776140        NA
## 339886 0.1599760 0.1954862
## 337724 0.2395055 0.2845443
```

```
head(hfhs.lmm$marg.R2)
```

```
##         trt.only trt.batch
## 541135 0.2023947 0.2692818
## 276629 0.6443355 0.6435223
```

```
## 196070 0.1674952 0.1678637
## 318732 0.1776140 0.1929564
## 339886 0.1599760 0.1916296
## 337724 0.2395055 0.2520680
```

We observed that some variables resulted in singular fits (`error message: Can't compute random effect variances. Some variance components equal zero. Your model may suffer from singulariy.`) and the conditional R^2s of these variables were "NA", which are expected to happen in linear mixed models when covariates are nested. We recommend noting the variables for which this error occurs, as it may lead to unreliable results.

SVA

SVA accounts for unknown batch effects. Here we assume that the batch grouping information in the HFHS data is unknown. We first build two design matrices with (`hfhs.mod`) and without (`hfhs.mod0`) treatment grouping information generated with `model.matrix()` function from stats. We then use `num.sv()` from sva package to determine the number of batch variables `n.sv` that will be used to estimate batch effects in function `sva()`.

```
# estimate batch matrix
hfhs.mod <- model.matrix( ~ hfhs.trt.less)
hfhs.mod0 <- model.matrix( ~ 1, data = hfhs.trt.less)
hfhs.sva.n <- num.sv(dat = t(hfhs.clr.less), mod = hfhs.mod,
                     method = 'leek')
hfhs.sva <- sva(t(hfhs.clr.less), hfhs.mod, hfhs.mod0, n.sv = hfhs.sva.n)
```

```
## Number of significant surrogate variables is:  1
## Iteration (out of 5 ):1  2  3  4  5
```

The estimated batch effects are then applied to `f.pvalue()` to calculate the P-values of treatment effects. The estimated batch effects in SVA are assumed to be independent of the treatment effects. However, SVA considers some correlation between batch and treatment effects [Wang and LêCao, 2020].

```
# include estimated batch effects in the linear model
hfhs.mod.batch <- cbind(hfhs.mod, hfhs.sva$sv)
hfhs.mod0.batch <- cbind(hfhs.mod0, hfhs.sva$sv)
hfhs.sva.p <- f.pvalue(t(hfhs.clr.less), hfhs.mod.batch, hfhs.mod0.batch)
hfhs.sva.p.adj <- p.adjust(hfhs.sva.p, method = 'fdr')
```

RUV4

Before applying RUV4 (`RUV4()` from ruv, we need to specify negative control variables and the number of batch variables to estimate. We can use the empirical negative controls that are not significantly differentially abundant (adjusted P > 0.05) from a linear regression with the treatment information as the only covariate.

We use a loop to fit a linear regression for each microbial variable and adjust P values of

treatment effects for multiple comparisons with `p.adjust()` from stats. The empirical negative controls are then extracted according to the adjusted P values.

```
# empirical negative controls
hfhs.empir.p <- c()
for(e in 1:ncol(hfhs.clr.less)){
  hfhs.empir.lm <- lm(hfhs.clr.less[,e] ~ hfhs.trt.less)
  hfhs.empir.p[e] <- summary(hfhs.empir.lm)$coefficients[2,4]
}
hfhs.empir.p.adj <- p.adjust(p = hfhs.empir.p, method = 'fdr')
hfhs.nc <- hfhs.empir.p.adj > 0.05
```

The number of batch variables `k` can be determined using `getK()` function.

```
# estimate k
hfhs.k.res <- getK(Y = hfhs.clr.less, X = hfhs.trt.less, ctl = hfhs.nc)
hfhs.k <- hfhs.k.res$k
hfhs.k <- ifelse(hfhs.k != 0, hfhs.k, 1)
```

We then apply `RUV4()` with known treatment variables, estimated negative control variables and `k` batch variables. The calculated P values also need to be adjusted for multiple comparisons.

```
hfhs.ruv4 <- RUV4(Y = hfhs.clr.less, X = hfhs.trt.less,
                  ctl = hfhs.nc, k = hfhs.k)
hfhs.ruv4.p <- hfhs.ruv4$p
hfhs.ruv4.p.adj <- p.adjust(hfhs.ruv4.p, method = "fdr")
```

Both SVA and RUV4 can account for both the cage and day effects, but not efficient at accounting for the cage effect, as the estimated surrogate batch effect in SVA and negative controls in RUV4 may not capture all the batch variation because of the nested batch x treatment design of the cage effect in the HFHS data.

2.2.3.2 Correcting for batch effects

The methods that we use to correct for batch effects include removeBatchEffect, ComBat, PLSDA-batch, sPLSDA-batch, Percentile Normalisation and RUVIII. Among them, RUVIII is designed for unknown batch effects.

Since the cage effect cannot be corrected for because of the collinearity between the cage and treatment effects and the cage effect only accounted for a small amount of the total data variance (1.3% calculated with pRDA), so we only correct for the day effect in the HFHS data.

removeBatchEffect

The `removeBatchEffect()` function is implemented in limma package. The design matrix (`design`) with treatment grouping information can be generated with `model.matrix()` function from stats as shown in section **accounting for batch effects** method "SVA".

Here we use `removeBatchEffect()` function with batch grouping information (`batch`) and treatment design matrix (`design`) to calculate batch effect corrected data `hfhs.rBE`.

```
hfhs.rBE <- t(removeBatchEffect(t(hfhs.clr.less), batch = hfhs.day.less,
                                design = hfhs.mod))
```

ComBat

The `ComBat()` function (from sva package) is implemented as parametric or non-parametric correction with option `par.prior`. Under a parametric adjustment, we can assess the model's validity with `prior.plots = T` [Leek et al., 2012].

Here we use a non-parametric correction (`par.prior = F`) with batch grouping information (`batch`) and treatment design matrix (`mod`) to calculate batch effect corrected data `hfhs.ComBat`.

```
hfhs.ComBat <- t(ComBat(t(hfhs.clr.less), batch = hfhs.day.less,
                        mod = hfhs.mod, par.prior = F))
```

PLSDA-batch

The `PLSDA_batch()` function is implemented in PLSDAbatch package. To use this function, we need to specify the optimal number of components related to treatment (`ncomp.trt`) or batch effects (`ncomp.bat`).

Here in the HFHS data, we use `plsda()` from mixOmics with only treatment grouping information to estimate the optimal number of treatment components to preserve.

```
# estimate the number of treatment components
hfhs.trt.tune <- plsda(X = hfhs.clr.less, Y = hfhs.trt.less, ncomp = 5)
hfhs.trt.tune$prop_expl_var #1
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5
## 0.29946816 0.06913457 0.04242662 0.02681544 0.02189936
##
## $Y
##      comp1      comp2      comp3      comp4      comp5
## 1.00000000 0.07124125 0.03858356 0.02564995 0.01555355
```

We choose the number that explains 100% variance in the outcome matrix `Y`, thus from the result, 1 component is enough to preserve the treatment information.

We then use `PLSDA_batch()` function with both treatment and batch grouping information to estimate the optimal number of batch components to remove.

```
# estimate the number of batch components
hfhs.batch.tune <- PLSDA_batch(X = hfhs.clr.less,
                               Y.trt = hfhs.trt.less,
                               Y.bat = hfhs.day.less,
```

```
                                ncomp.trt = 1, ncomp.bat = 10)
hfhs.batch.tune$explained_variance.bat #2
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5      comp6      comp7
## 0.10753644 0.03608189 0.02988203 0.02638198 0.04335240 0.02931383 0.02430825
##      comp8      comp9     comp10
## 0.03177304 0.01553524 0.03029469
##
## $Y
##        comp1        comp2        comp3        comp4        comp5        comp6
## 0.4937952138 0.5062047862 0.5128452381 0.4869109170 0.0002438449 0.5069367566
##        comp7        comp8        comp9       comp10
## 0.4921936988 0.0008695446 0.5082371149 0.4740866865
```

```
sum(hfhs.batch.tune$explained_variance.bat$Y[1:2])
```

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

Using the same criterion as choosing treatment components, we choose the number of batch components that explains 100% variance in the outcome matrix of batch. According to the result, 2 components are required to remove batch effects.

We then can correct for batch effects applying `PLSDA_batch()` with treatment, batch grouping information and corresponding optimal number of related components.

```
##########
hfhs.PLSDA_batch.res <- PLSDA_batch(X = hfhs.clr.less,
                                    Y.trt = hfhs.trt.less,
                                    Y.bat = hfhs.day.less,
                                    ncomp.trt = 1, ncomp.bat = 2)
hfhs.PLSDA_batch <- hfhs.PLSDA_batch.res$X.nobatch
```

sPLSDA-batch

We apply sPLSDA-batch using the same function `PLSDA_batch()` but we specify the number of variables to select on each component (usually only treatment-related components `keepX.trt`). To determine the optimal number of variables to select, we use `tune.splsda()` function from mixOmics package [Rohart et al., 2017] with all possible numbers of variables to select for each component (`test.keepX`).

```
# estimate the number of variables to select per treatment component
set.seed(777)
hfhs.test.keepX = c(seq(1, 10, 1), seq(20, 200, 10),
                    seq(250, 500, 50), 515)
hfhs.trt.tune.v <- tune.splsda(X = hfhs.clr.less,
                               Y = hfhs.trt.less,
                               ncomp = 1,
                               test.keepX = hfhs.test.keepX,
```

```
                                validation = 'Mfold',
                                folds = 4, nrepeat = 50)
hfhs.trt.tune.v$choice.keepX # 2
```

```
## comp1
##     2
```

Here the optimal number of variables to select for the treatment component is 2. Since we adjust the amount of treatment variation to preserve, we need to re-choose the optimal number of components related to batch effects using the same criterion mentioned in section **correcting for batch effects** method "PLSDA-batch".

```
# estimate the number of batch components
hfhs.batch.tune <- PLSDA_batch(X = hfhs.clr.less,
                                Y.trt = hfhs.trt.less,
                                Y.bat = hfhs.day.less,
                                ncomp.trt = 1,
                                keepX.trt = 2,
                                ncomp.bat = 10)
hfhs.batch.tune$explained_variance.bat #2
```

```
## $X
##      comp1      comp2      comp3      comp4      comp5      comp6      comp7
## 0.11496719 0.03611134 0.02997732 0.02496738 0.03220821 0.03283139 0.04100050
##      comp8      comp9     comp10
## 0.02775988 0.02420573 0.01955264
##
## $Y
##        comp1        comp2        comp3        comp4        comp5        comp6
## 0.4938682259 0.5061317741 0.5120581093 0.4871878556 0.0007540351 0.5090445753
##        comp7        comp8        comp9       comp10
## 0.4896338693 0.0013215554 0.4942076677 0.4961940448
```

```
sum(hfhs.batch.tune$explained_variance.bat$Y[1:2])
```

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

According to the result, we need 2 batch related components to remove batch variance from the data with function `PLSDA_batch()`.

```
##########
hfhs.sPLSDA_batch.res <- PLSDA_batch(X = hfhs.clr.less,
                                     Y.trt = hfhs.trt.less,
                                     Y.bat = hfhs.day.less,
                                     ncomp.trt = 1,
                                     keepX.trt = 2,
                                     ncomp.bat = 2)
hfhs.sPLSDA_batch <- hfhs.sPLSDA_batch.res$X.nobatch
```

Percentile Normalisation

To apply `percentile_norm()` function from PLSDAbatch package, we need to indicate a control group (`ctrl.grp`).

```
# HFHS data
hfhs.PN <- percentile_norm(data = hfhs.clr.less,
                           batch = hfhs.day.less,
                           trt = hfhs.trt.less,
                           ctrl.grp = 'Normal')
```

RUVIII

The `RUVIII()` function is from ruv package. Similar to `RUV4()`, we use empirical negative control variables and `getK()` to determine the number of batch variables (`k`) to estimate. We also need sample replicates which should be structured into a mapping matrix using `replicate.matrix()`. We can then obtain batch effect corrected data applying `RUVIII()` with above elements.

```
hfhs.replicates <- hfhs.metadata.less$MouseID
hfhs.replicates.matrix <- replicate.matrix(hfhs.replicates)

hfhs.RUVIII <- RUVIII(Y = hfhs.clr.less,
                      M = hfhs.replicates.matrix,
                      ctl = hfhs.nc, k = hfhs.k)
rownames(hfhs.RUVIII) <- rownames(hfhs.clr.less)
```

Compared to the AD data, the HFHS data have more appropriate sample replicates that fully capture the difference between different time points. The results of RUVIII applied to HFHS data would be better than AD data.

2.2.4 Assessing batch effect correction

We apply different visualisation and quantitative methods to assessing batch effect correction.

2.2.4.1 Methods that detect batch effects

PCA

In the HFHS data, we compare the PCA sample plots before and after batch effect correction with different methods.

```
hfhs.pca.before <- pca(hfhs.clr.less, ncomp = 3,
                       scale = TRUE)
hfhs.pca.rBE <- pca(hfhs.rBE, ncomp = 3,
                    scale = TRUE)
hfhs.pca.ComBat <- pca(hfhs.ComBat, ncomp = 3,
                       scale = TRUE)
```

```
hfhs.pca.PLSDA_batch <- pca(hfhs.PLSDA_batch, ncomp = 3,
                            scale = TRUE)
hfhs.pca.sPLSDA_batch <- pca(hfhs.sPLSDA_batch, ncomp = 3,
                             scale = TRUE)
hfhs.pca.PN <- pca(hfhs.PN, ncomp = 3,
                   scale = TRUE)
hfhs.pca.RUVIII <- pca(hfhs.RUVIII, ncomp = 3,
                       scale = TRUE)
```

```
hfhs.pca.before.plot <-
  Scatter_Density(object = hfhs.pca.before,
                  batch = hfhs.day.less,
                  trt = hfhs.trt.less,
                  title = 'Before')
```

```
hfhs.pca.rBE.plot <-
  Scatter_Density(object = hfhs.pca.rBE,
                  batch = hfhs.day.less,
                  trt = hfhs.trt.less,
                  title = 'removeBatchEffect')
```

```
hfhs.pca.ComBat.plot <-
  Scatter_Density(object = hfhs.pca.ComBat,
                  batch = hfhs.day.less,
                  trt = hfhs.trt.less,
                  title = 'ComBat')
```

```
hfhs.pca.PLSDA_batch.plot <-
  Scatter_Density(object = hfhs.pca.PLSDA_batch,
                  batch = hfhs.day.less,
                  trt = hfhs.trt.less,
                  title = 'PLSDA-batch')
```

```
hfhs.pca.sPLSDA_batch.plot <-
  Scatter_Density(object = hfhs.pca.sPLSDA_batch,
                  batch = hfhs.day.less,
                  trt = hfhs.trt.less,
                  title = 'sPLSDA-batch')
```

```
hfhs.pca.PN.plot <-
  Scatter_Density(object = hfhs.pca.PN,
                  batch = hfhs.day.less,
                  trt = hfhs.trt.less,
                  title = 'Percentile Normalisation')
```

```
hfhs.pca.RUVIII.plot <-
  Scatter_Density(object = hfhs.pca.RUVIII,
                  batch = hfhs.day.less,
```

```
                  trt = hfhs.trt.less,
                  title = 'RUVIII')
```

In the above figures, the differences between the samples from different days were well removed after batch effect correction with PLSDA-batch and RUVIII. We can also compare the boxplots and density plots for key variables identified in PCA driving the major variance or in heatmaps showing obvious patterns before and after batch effect correction (results not shown).

pRDA

We calculate the global explained variance across all microbial variables using pRDA. To achieve this, we create a loop for each variable from the original (uncorrected) and batch effect-corrected data. The final results are then displayed with `partVar_plot()` from PLSDAbatch package.

```
# HFHS data
hfhs.corrected.list <- list(`Before correction` = hfhs.clr.less,
                            removeBatchEffect = hfhs.rBE,
                            ComBat = hfhs.ComBat,
                            `PLSDA-batch` = hfhs.PLSDA_batch,
                            `sPLSDA-batch` = hfhs.sPLSDA_batch,
                            `Percentile Normalisation` = hfhs.PN,
                            RUVIII = hfhs.RUVIII)

hfhs.prop.df <- data.frame(Treatment = NA, Batch = NA,
                           Intersection = NA,
                           Residuals = NA)
for(i in 1:length(hfhs.corrected.list)){
  rda.res = varpart(hfhs.corrected.list[[i]], ~ trt, ~ day,
                    data = hfhs.factors.df, scale = T)
  hfhs.prop.df[i, ] <- rda.res$part$indfract$Adj.R.squared}

rownames(hfhs.prop.df) = names(hfhs.corrected.list)

hfhs.prop.df <- hfhs.prop.df[, c(1,3,2,4)]

hfhs.prop.df[hfhs.prop.df < 0] = 0
hfhs.prop.df <- as.data.frame(t(apply(hfhs.prop.df, 1,
                                      function(x){x/sum(x)})))


partVar_plot(prop.df = hfhs.prop.df)
```

In the HFHS data, a small amount of batch variance was observed (3.6%) as shown in the above figure. PLSDA-batch achieved the best performance for preserving the largest treatment variance and completely removing batch variance compared to the

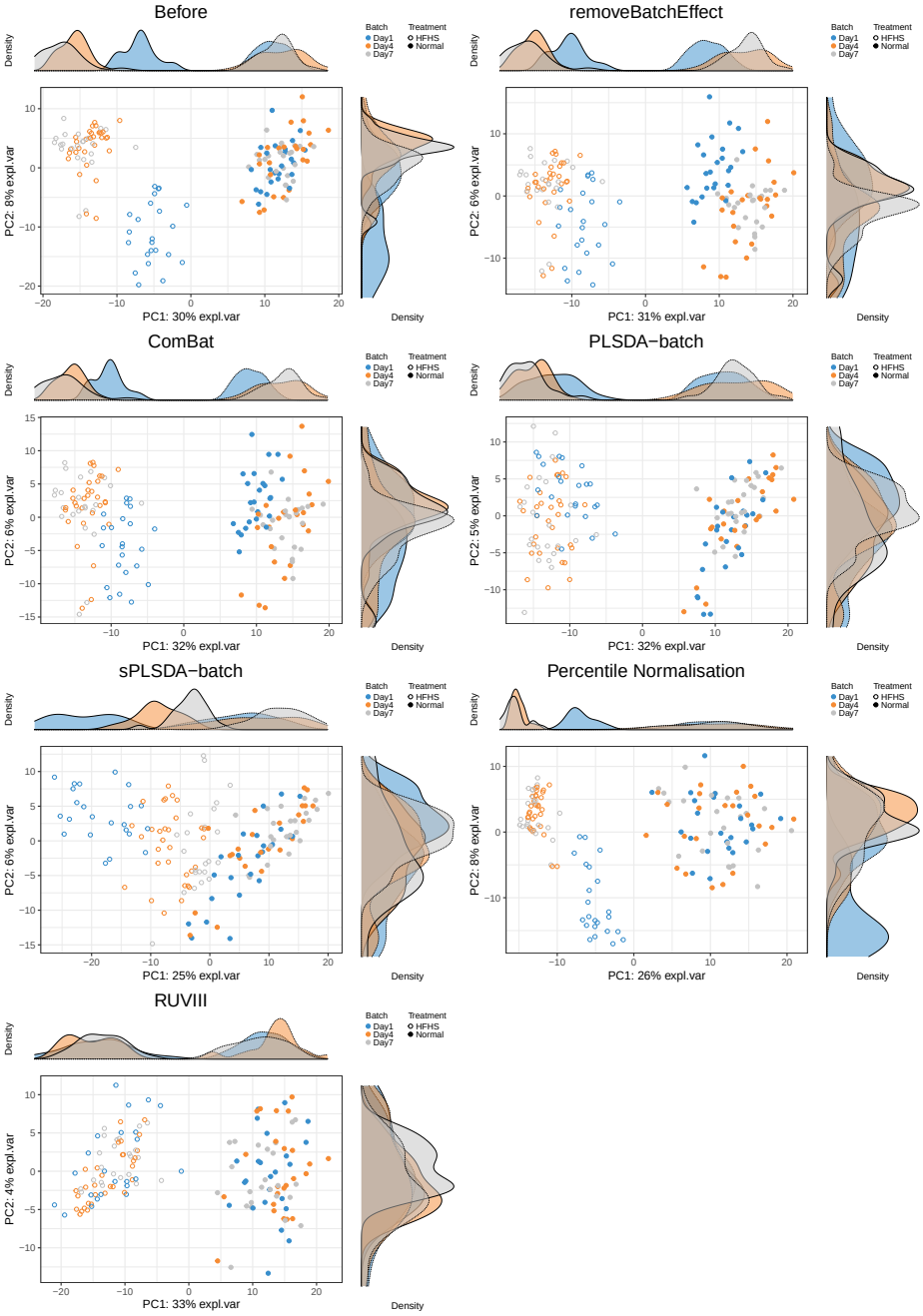

Figure 2.17: The PCA sample plots with densities before and after batch effect correction in the HFHS data.

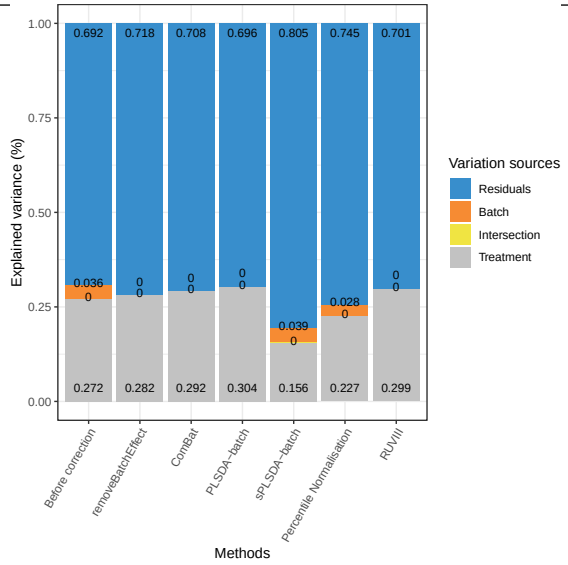

Figure 2.18: Global explained variance before and after batch effect correction for the HFHS data.

other methods. The results also indicate that PLSDA-batch is more appropriate for weak batch effects, while sPLSDA-batch is more appropriate for strong batch effects. The RUVIII performed better in the HFHS data than the AD data because the sample replicates may help capturing more batch variation than in the AD data. Indeed, the sample replicates in HFHS data are across different day batches, while the replicates in AD data do not exist in all batches. Therefore, sample replicates play a critical role in RUVIII.

2.2.4.2 Other methods

R^2

The R^2 values for each variable are calculated with `lm()` from stats package. To compare the R^2 values among variables, we scale the corrected data before R^2 calculation. The results are displayed with `ggplot()` from ggplot2 R package.

```
# scale
hfhs.corr_scale.list <- lapply(hfhs.corrected.list,
                               function(x){apply(x, 2, scale)})

# HFHS data
hfhs.r_values.list <- list()
for(i in 1:length(hfhs.corr_scale.list)){
  hfhs.r_values <- data.frame(trt = NA, batch = NA)
  for(c in 1:ncol(hfhs.corr_scale.list[[i]])){
```

```
    hfhs.fit.res.trt <- lm(hfhs.corr_scale.list[[i]][,c] ~ hfhs.trt.less)
    hfhs.r_values[c,1] <- summary(hfhs.fit.res.trt)$r.squared
    hfhs.fit.res.batch <- lm(hfhs.corr_scale.list[[i]][,c] ~ hfhs.day.less)
    hfhs.r_values[c,2] <- summary(hfhs.fit.res.batch)$r.squared
  }
  hfhs.r_values.list[[i]] <- hfhs.r_values
}
names(hfhs.r_values.list) <- names(hfhs.corr_scale.list)

hfhs.boxp.list <- list()
for(i in seq_len(length(hfhs.r_values.list))){
  hfhs.boxp.list[[i]] <-
    data.frame(r2 = c(hfhs.r_values.list[[i]][ ,'trt'],
                      hfhs.r_values.list[[i]][ ,'batch']),
               Effects = as.factor(rep(c('Treatment','Batch'),
                                       each = 515)))
}
names(hfhs.boxp.list) <- names(hfhs.r_values.list)

hfhs.r2.boxp <- rbind(hfhs.boxp.list$`Before correction`,
                      hfhs.boxp.list$removeBatchEffect,
                      hfhs.boxp.list$ComBat,
                      hfhs.boxp.list$`PLSDA-batch`,
                      hfhs.boxp.list$`sPLSDA-batch`,
                      hfhs.boxp.list$`Percentile Normalisation`,
                      hfhs.boxp.list$RUVIII)

hfhs.r2.boxp$methods <- rep(c('Before correction', ' removeBatchEffect',
                              'ComBat','PLSDA-batch', 'sPLSDA-batch',
                              'Percentile Normalisation', 'RUVIII'),
                            each = 1030)

hfhs.r2.boxp$methods <- factor(hfhs.r2.boxp$methods,
                               levels = unique(hfhs.r2.boxp$methods))

ggplot(hfhs.r2.boxp, aes(x = Effects, y = r2, fill = Effects)) +
  geom_boxplot(alpha = 0.80) +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 60, hjust = 1, size = 18),
        axis.text.y = element_text(size = 18),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
```

```
        legend.position = "right") + facet_grid( ~ methods) +
  scale_fill_manual(values=pb_color(c(12,14)))
```

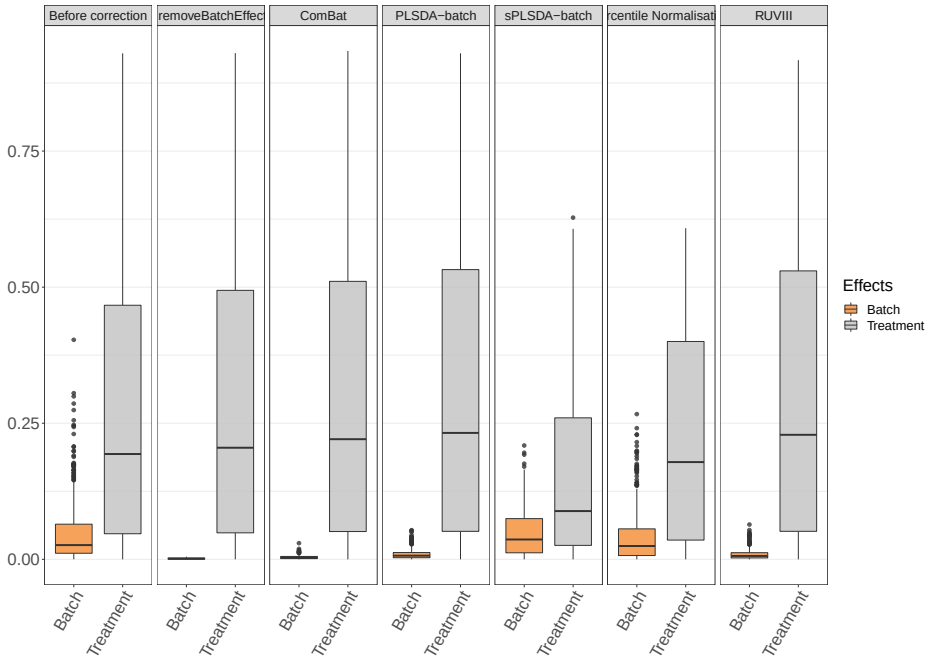

Figure 2.19: HFHS study: R^2 values for each microbial variable before and after batch effect correction.

We observed from these plots that the data corrected by sPLSDA-batch, percentile normalisation still included many variables with a large proportion of batch variance.

```
#################################
hfhs.barp.list <- list()
for(i in seq_len(length(hfhs.r_values.list))){
  hfhs.barp.list[[i]] <-
    data.frame(r2 = c(sum(hfhs.r_values.list[[i]][ ,'trt']),
                      sum(hfhs.r_values.list[[i]][ ,'batch'])),
               Effects = c('Treatment','Batch'))
}
names(hfhs.barp.list) <- names(hfhs.r_values.list)

hfhs.r2.barp <- rbind(hfhs.barp.list$`Before correction`,
                      hfhs.barp.list$removeBatchEffect,
                      hfhs.barp.list$ComBat,
                      hfhs.barp.list$`PLSDA-batch`,
                      hfhs.barp.list$`sPLSDA-batch`,
```

```
                      hfhs.barp.list$`Percentile Normalisation`,
                      hfhs.barp.list$RUVIII)


hfhs.r2.barp$methods <- rep(c('Before correction', ' removeBatchEffect',
                              'ComBat','PLSDA-batch', 'sPLSDA-batch',
                              'Percentile Normalisation', 'RUVIII'), each = 2)

hfhs.r2.barp$methods <- factor(hfhs.r2.barp$methods,
                               levels = unique(hfhs.r2.barp$methods))


ggplot(hfhs.r2.barp, aes(x = Effects, y = r2, fill = Effects)) +
  geom_bar(stat="identity") +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        axis.text.x = element_text(angle = 60, hjust = 1, size = 18),
        axis.text.y = element_text(size = 18),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid( ~ methods) +
  scale_fill_manual(values=pb_color(c(12,14)))
```

When considering the sum of all variables, the remaining batch variance of corrected data from PLSDA-batch is greater than removeBatchEffect, ComBat and RUVIII. The preserved treatment variance of corrected data from PLSDA-batch is the largest.

Alignment scores

We use the `alignment_score()` function from PLSDAbatch. We need to specify the proportion of data variance to explain (`var`), the number of nearest neighbours (`k`) and the number of principal components to estimate (`ncomp`). We then use `ggplot()` function from ggplot2 to visualise the results.

```
# HFHS data
hfhs.var = 0.95
hfhs.k = 15

hfhs.scores <- c()
for(i in 1:length(hfhs.corrected.list)){
  res <- alignment_score(data = hfhs.corrected.list[[i]],
                         batch = hfhs.day.less,
                         var = hfhs.var,
                         k = hfhs.k,
                         ncomp = 130)
```

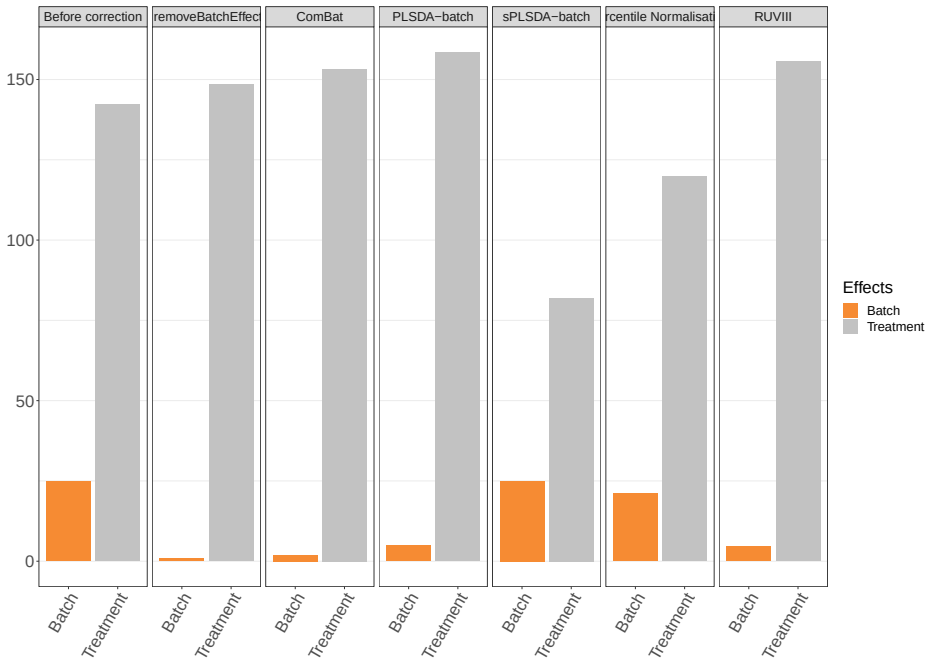

Figure 2.20: HFHS study: the sum of R^2 values for each microbial variable before and after batch effect correction.

```
  hfhs.scores <- c(hfhs.scores, res)
}

hfhs.scores.df <- data.frame(scores = hfhs.scores,
                             methods = names(hfhs.corrected.list))

hfhs.scores.df$methods <- factor(hfhs.scores.df$methods,
                                 levels = rev(names(hfhs.corrected.list)))


ggplot() + geom_col(aes(x = hfhs.scores.df$methods,
                        y = hfhs.scores.df$scores)) +
  geom_text(aes(x = hfhs.scores.df$methods,
                y = hfhs.scores.df$scores/2,
                label = round(hfhs.scores.df$scores, 3)),
            size = 3, col = 'white') +
  coord_flip() + theme_bw() + ylab('Alignment Scores') +
  xlab('') + ylim(0,0.85)
```

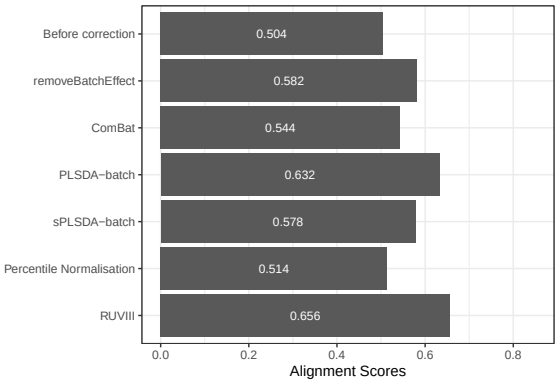

Figure 2.21: Comparison of alignment scores before and after batch effect correction using different methods for the HFHS data.

The alignment scores complement the PCA results, especially when batch effect removal is difficult to assess on PCA sample plots. For example in the PCA plots for the HFHS data, we observed that the samples across different batches were better mixed after batch effect correction with PLSDA-batch and RUVIII than before, while the performance of these two methods was difficult to compare. Since a higher alignment score indicates that samples are better mixed, as shown in the above figure, RUVIII gave a superior performance compared to the other methods.

We recommend the sparse version of PLSDA-batch for a very strong batch effect, for example, in the AD data. However, for a weak batch effect, we do not recommend removing batch effects in the HFHS data, but if we have to, we recommend using

PLSDA-batch. It is easier to lose treatment variation when the batch variation is very small. Therefore PLSDA-batch can better ensure the complete preservation of treatment variation than a sparse version.

Variable selection

We use `splsda()` from mixOmics to select the top 50 microbial variables that, in combination, discriminate the different treatment groups in the HFHS data. We apply `splsda()` to the different batch effect corrected data from all methods. Then we use `upset()` from UpSetR package [Lex et al., 2014] to visualise the concordance of variables selected.

In the code below, we first need to convert the list of variables selected from different method-corrected data into a data frame compatible with `upset()` using `fromList()`. We then assign different colour schemes for each variable selection.

```
hfhs.splsda.select <- list()
for(i in 1:length(hfhs.corrected.list)){
  splsda.res <- splsda(X = hfhs.corrected.list[[i]],
                       Y = hfhs.trt.less,
                       ncomp = 3, keepX = rep(50,3))
  select.res <- selectVar(splsda.res, comp = 1)$name
  hfhs.splsda.select[[i]] <- select.res
}
names(hfhs.splsda.select) <- names(hfhs.corrected.list)

# can only visualise 5 methods
hfhs.splsda.select <- hfhs.splsda.select[c(1:4,7)]

hfhs.splsda.upsetR <- fromList(hfhs.splsda.select)

upset(hfhs.splsda.upsetR, main.bar.color = 'gray36',
      sets.bar.color = pb_color(c(25:22,20)), matrix.color = 'gray36',
      order.by = 'freq', empty.intersections = 'on',
      queries =
        list(list(query = intersects, params = list('Before correction'),
                  color = pb_color(20), active = T),
             list(query = intersects, params = list('removeBatchEffect'),
                  color = pb_color(22), active = T),
             list(query = intersects, params = list('ComBat'),
                  color = pb_color(23), active = T),
             list(query = intersects, params = list('PLSDA-batch'),
                  color = pb_color(24), active = T),
             list(query = intersects, params = list('RUVIII'),
                  color = pb_color(25), active = T)))
```

In the above UpSet plot, the left bars indicate the number of variables selected from each data corrected with different methods. The right bar plot combined with the scatterplot

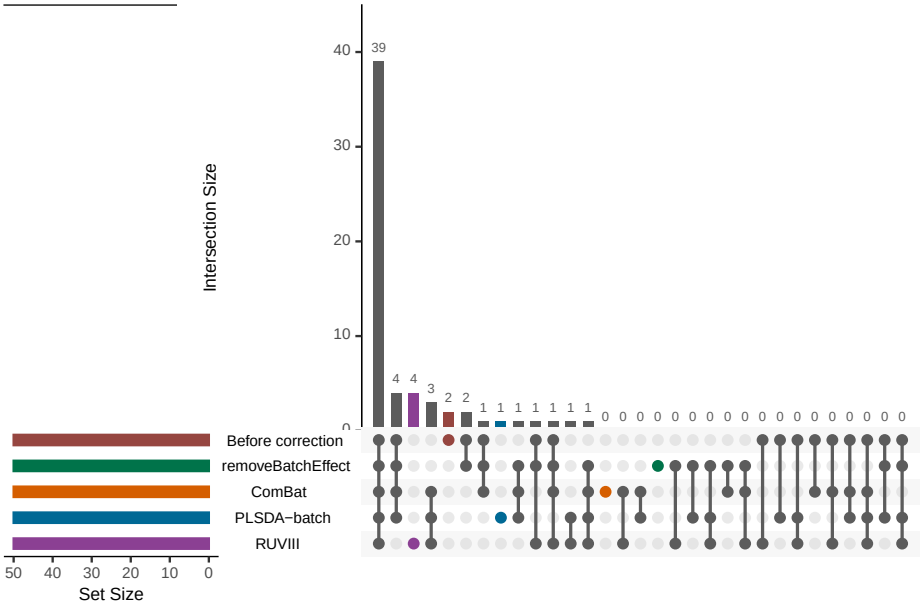

Figure 2.22: UpSet plot showing overlap between variables selected from different corrected data for the HFHS study.

show different intersection and their aggregates. We obtained an overlap of 39 out of 50 selected variables between different corrected and uncorrected data. However, the data from each method still included unique variables that were not selected in the other corrected data. As `upset()` can only include five datasets at once, we only display the uncorrected data and four corrected data that have been efficiently corrected for batch effects from our previous assessments.

```
hfhs.splsda.select.overlap <- venn(hfhs.splsda.select, show.plot = F)
hfhs.inters.splsda <- attr(hfhs.splsda.select.overlap, 'intersections')
hfhs.inters.splsda.taxa <-
  lapply(hfhs.inters.splsda,
         FUN = function(x){as.data.frame(HFHS_data$FullData$taxa[x, ])})
capture.output(hfhs.inters.splsda.taxa,
               file = "GeneratedData/HFHSselected_50_splsda.txt")
```

2.3 HD data

2.3.1 Data pre-processing

2.3.1.1 Prefiltering

We load the HD data stored internally with function `data()`.

```
data('HD_data')
hd.count <- HD_data$FullData$X.count
dim(hd.count)
```

```
## [1]  30 368
```

```
hd.filter.res <- PreFL(data = hd.count)
hd.filter <- hd.filter.res$data.filter
dim(hd.filter)
```

```
## [1]  30 269
```

```
# zero proportion before filtering
hd.filter.res$zero.prob
```

```
## [1] 0.4509058
```

```
# zero proportion after filtering
sum(hd.filter == 0)/(nrow(hd.filter) * ncol(hd.filter))
```

```
## [1] 0.3343247
```

The raw HD data include 368 OTUs and 30 samples. We use the function `PreFL()` from our PLSDAbatch R package to filter the data. After filtering, the HD data were reduced to 269 OTUs and 30 samples. The proportion of zeroes was reduced from 45% to 33%.

2.3.1.2 Transformation

Prior to CLR transformation, we add 1 as the offset for the HD data - that are raw count data. We use `logratio.transfo()` function in mixOmics package to CLR transform the data.

```
hd.clr <- logratio.transfo(X = hd.filter, logratio = 'CLR', offset = 1)

class(hd.clr) <- 'matrix'
```

2.3.2 Batch effect detection

2.3.2.1 PCA

We apply `pca()` function from mixOmics package to the HD data and use `Scatter_Density()` function from PLSDAbatch to represent the PCA sample plot with densities.

```
# HD data
hd.pca.before <- pca(hd.clr, ncomp = 3, scale = TRUE)

hd.metadata <- HD_data$FullData$metadata
```

```
hd.batch <- as.factor(hd.metadata$Cage)
hd.trt <- as.factor(hd.metadata$Genotype)
names(hd.batch) = names(hd.trt) = rownames(hd.metadata)

Scatter_Density(object = hd.pca.before,
                batch = hd.batch,
                trt = hd.trt,
                title = 'HD data',
                trt.legend.title = 'Genotype',
                batch.legend.title = 'Cage', legend.cex = 0.6)
```

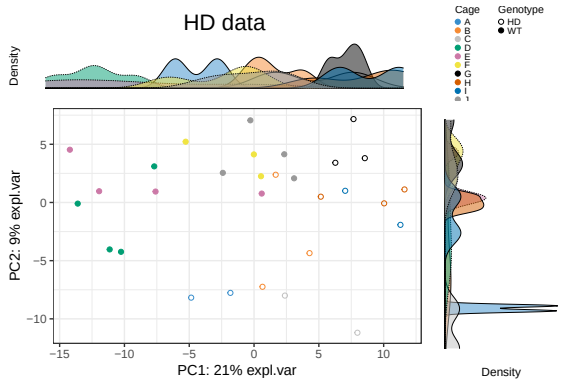

Figure 2.23: The PCA sample plot with densities in the HD data.

In the above figure, we observed a clear difference between genotypes but vague separation between cages on the PCA plot.

2.3.2.2 Heatmap

We produce a heatmap using pheatmap package. The data first need to be scaled on both OTUs and samples.

```
# scale the clr data on both OTUs and samples
hd.clr.s <- scale(hd.clr, center = T, scale = T)
hd.clr.ss <- scale(t(hd.clr.s), center = T, scale = T)

hd.anno_col <- data.frame(Cage = hd.batch, Genotype = hd.trt)
hd.anno_colors <- list(Cage = color.mixo(1:10),
                       Genotype = pb_color(1:2))
names(hd.anno_colors$Cage) = levels(hd.batch)
names(hd.anno_colors$Genotype) = levels(hd.trt)

pheatmap(hd.clr.ss,
         cluster_rows = F,
```

```
        fontsize_row = 4,
        fontsize_col = 6,
        fontsize = 8,
        clustering_distance_rows = 'euclidean',
        clustering_method = 'ward.D',
        treeheight_row = 30,
        annotation_col = hd.anno_col,
        annotation_colors = hd.anno_colors,
        border_color = 'NA',
        main = 'HD data - Scaled')
```

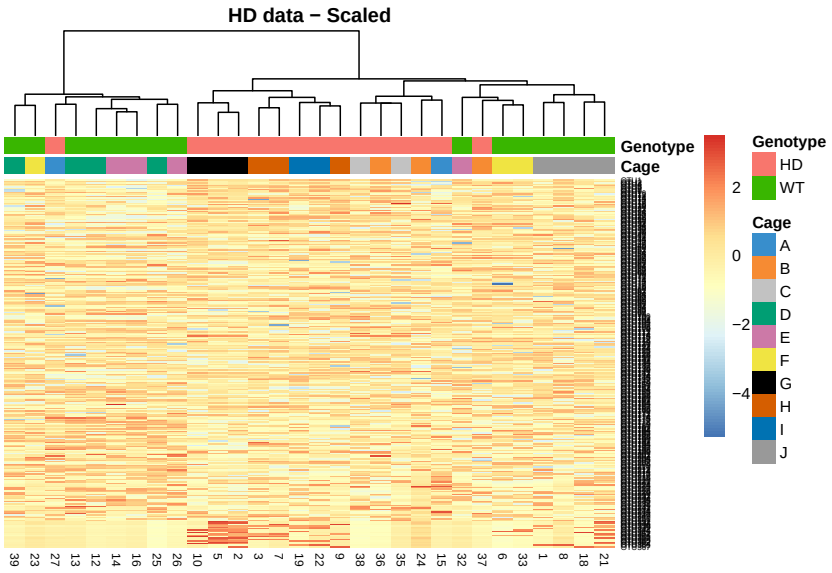

Figure 2.24: Hierarchical clustering for samples in the HD data.

In the heatmap, samples in the HD data were mostly grouped by genotypes instead of cages, indicating a weak cage effect.

2.3.2.3 pRDA

We apply pRDA with `varpart()` function from vegan R package.

```
hd.factors.df <- data.frame(trt = hd.trt, batch = hd.batch)
hd.rda.before <- varpart(hd.clr, ~ trt, ~ batch,
                         data = hd.factors.df, scale = T)
hd.rda.before$part$indfract
```

```
##                 Df R.squared Adj.R.squared Testable
## [a] = X1|X2      0        NA -2.220446e-16    FALSE
## [b] = X2|X1      8        NA  1.608205e-01     TRUE
```

```
## [c]                0         NA  9.730583e-02    FALSE
## [d] = Residuals NA           NA  7.418737e-01    FALSE
```

In the result, `X1` and `X2` represent the first and second covariates fitted in the model. `[a]`, `[b]` represent the independent proportion of variance explained by `X1` and `X2` respectively, and `[c]` represents the intersection variance shared between `X1` and `X2`. Collinearity was detected in the HD data between treatment and batch as indicated by lines `[a]` and `[c]` that all treatment variance (`[a]`: Adj.R.squared = 0, `[c]`: Adj.R.squared = 0.0973) was explained as intersection variance. The batch x treatment design in the HD data is nested, in such a design the intersection variance is usually considerable. As the intersection is shared between batch and treatment effects, the batch variance in the HD data is larger than the treatment variance.

2.3.3 Managing batch effects

Because of the nested batch x treatment design in the HD data, the only way to account for the batch effect (i.e., the cage effect) is to apply linear mixed model and include the batch effect as a random effect.

2.3.3.1 Accounting for batch effects

Linear regression

Linear regression is conducted with `linear_regres()` function in PLSDAbatch. We integrated the performance package that assesses performance of regression models into our function `linear_regres()`. Therefore, we can apply `check_model()` from performance to the outputs from `linear_regres()` to diagnose the validity of the model fitted with treatment and batch effects for each variable [LÃŒdecke et al., 2020]. We can extract performance measurements such as adjusted R2, RMSE, RSE, AIC and BIC for the models fitted with and without batch effects, which are also the outputs of `linear_regres()`.

We apply a `linear mixed model` and fit the batch effect as a random effect because the design of the HD data is nested. In such a design, the number of batches should be larger than treatment groups; otherwise, the linear mixed model cannot address the batch effect. Note: we assume that there is no interaction between batch and treatment effects on any microbial variable.

```
# HD data
hd.clr <- hd.clr[1:nrow(hd.clr), 1:ncol(hd.clr)]

hd.lmm <- linear_regres(data = hd.clr, trt = hd.trt,
                        batch.random = hd.batch,
                        type = 'linear mixed model')
```

```
hd.p <- sapply(hd.lmm$lmm.table, function(x){x$coefficients[2,5]})
hd.p.adj <- p.adjust(p = hd.p, method = 'fdr')
```

```
check_model(hd.lmm$model$OTU1)
```

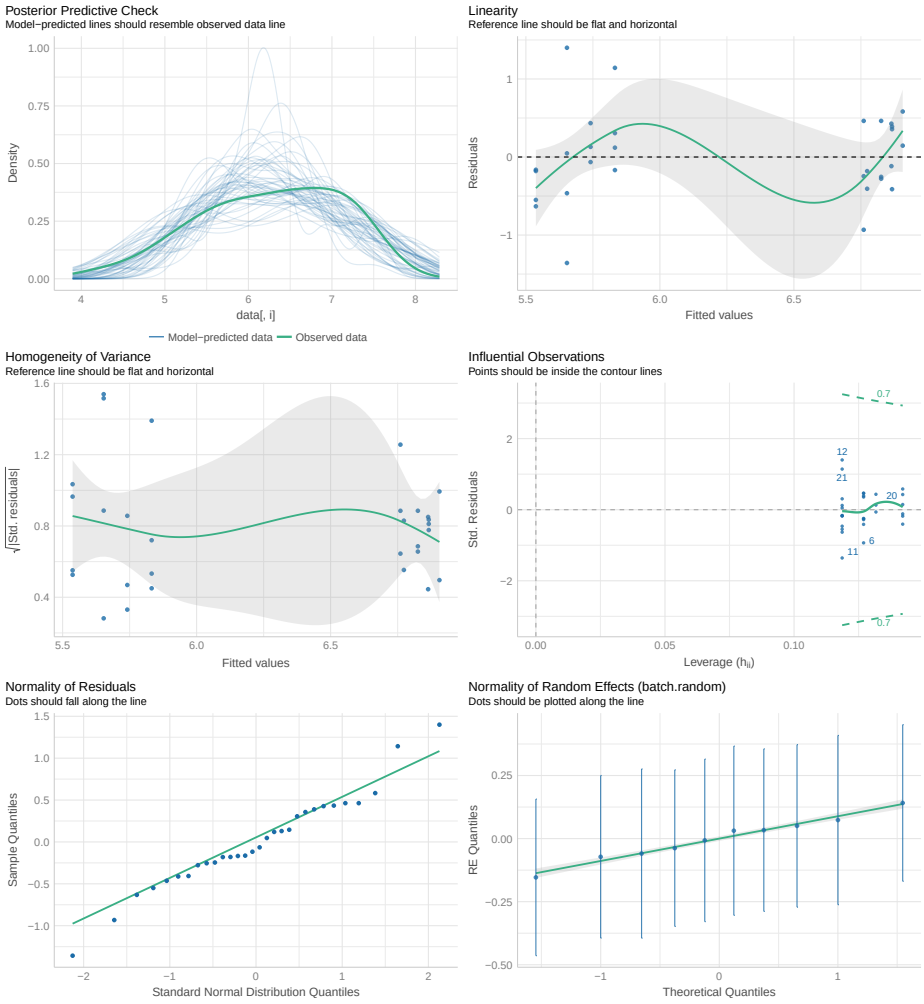

Figure 2.25: Diagnostic plots for the model fitted with the cage effect of "OTU1" in the HD data.

According to the diagnostic plots of "OTU1" as shown in the above figure panel, the simulated data under the fitted model were not very close to the real data (shown as green line), indicating an imperfect fitness (top panel). The linearity (or homoscedasticity) and homogeneity of variance were not satisfied. Some samples could be classified as outliers with a Cook's distance larger than or equal to 0.5, for example, "12", "21", "20", "11" and "6" (middle panel). The distribution of residuals was not normal, while the one of random effects was very close to normal (bottom panel).

We then look at the AIC of models fitted with or without batch effects.

```
head(hd.lmm$AIC)
```

```
##       trt.only trt.batch
## OTU1  60.00637  65.46410
## OTU2 130.14167 130.96344
## OTU3  71.76985  75.58915
## OTU4 134.59992 134.04812
## OTU5 150.25059 148.30644
## OTU6  99.92864  99.05391
```

For some OTUs, the model fitted without batch effects is better with a lower AIC. For example, "OTU1" and "OTU3". "OTU5" is more appropriate within a model with batch effects. The others are similar within either model.

Since we apply a `"linear mixed model"` to the HD data, this type of model can only output conditional R^2 that includes the variance of both fixed and random effects (treatment, fixed and random batch effects) and marginal R^2 that includes only the variance of fixed effects (treatment and fixed batch effects) [Nakagawa and Schielzeth, 2013]. The marginal R^2s of the models with and without the batch effect should be the same theoretically, as marginal R^2 only includes fixed effects and the treatment effect is the only fixed effect here. In the actual calculation, the results of these two models were not the same due to different ways of model fitting.

```
head(hd.lmm$cond.R2)
```

```
##        trt.only trt.batch
## OTU1 0.47917775 0.5161317
## OTU2 0.39223824 0.4729848
## OTU3 0.02091376 0.2636432
## OTU4 0.37200835 0.5672392
## OTU5 0.07858373 0.2938212
## OTU6 0.41388250 0.6143134
```

```
head(hd.lmm$marg.R2)
```

```
##        trt.only  trt.batch
## OTU1 0.47917775 0.46735273
## OTU2 0.39223824 0.38459672
## OTU3 0.02091376 0.01489608
## OTU4 0.37200835 0.36122754
## OTU5 0.07858373 0.06052938
## OTU6 0.41388250 0.40749149
```

We observed that some variables resulted in singular fits (`error message: Can't compute random effect variances. Some variance components equal zero. Your model may suffer from singulariy.`), which are expected to happen in linear mixed models when covariates are nested. We recommend noting the

variables for which this error occurs, as it may lead to unreliable results.

2.4 References

Chapter 3

Batch Effects Management in Simulation Studies (Gaussian Distribution)

3.1 Introduction

We adapted the simulation strategy that is component-based and multivariate from [Singh et al., 2019]. We assumed the input data after filtering follow a lognormal distribution inspired from [Weiss et al., 2017], thus after Centered Log Ratio (CLR) transformed follow a Gaussian distribution. Thus, we simulated components from a Gaussian distribution across all samples. The data matrix was generated based on the simulated components and corresponding loading vectors for each variable. Each simulated dataset included 300 variables and 40 samples grouped according to two treatments (trt1 and trt2) and two batches (batch1 and batch2). Different parameters including amount of batch and treatment variability among samples, number of variables with batch and/or treatment effects, balanced and unbalanced batch $\times$ treatment designs were considered and summarised in the table below.

Table 1: Summary of simulation scenarios (Gaussian distribution). For a given choice of parameters reported in this table, each simulation was repeated 50 times. $M^{(trt)}$, $M^{(batch)}$ and $M^{(trt\ \&\ batch)}$ represent the number of variables with treatment, batch, or both effects respectively. **Simu 6** includes parameters reflective of real data.

	$\mu_{(trt)}$	$\sigma'_{(trt)}$	$\mu_{(batch)}$	$\sigma'_{(batch)}$	$M^{(trt)}$	$M^{(batch)}$	$M^{(trt\ \&\ batch)}$
Simu 1	3	1	7	{1,4,8}	60	150	0
Simu 2	{3,5,7}	1	7	8	60	150	0

	$\mu_{(trt)}$	$\sigma'_{(trt)}$	$\mu_{(batch)}$	$\sigma'_{(batch)}$	$M^{(trt)}$	$M^{(batch)}$	$M^{(trt\ \&\ batch)}$
Simu 3	3	{1,2,4}	7	8	60	150	0
Simu 4	3	2	7	8	{30,60,100,150}	150	0
Simu 5	3	2	7	8	60	{30,60,100,150}	0
Simu 6	3	2	7	8	60	150	{0,18,30,42,60}

We first generated two base components $t^{(trt)}$ and $t^{(batch)}$ to represent the underlying treatment and batch variation across samples in the datasets. The samples with trt1 or trt2 in the component $t^{(trt)}$ were generated from $N(-\mu_{(trt)}, \sigma'^2_{(trt)})$ and $N(\mu_{(trt)}, \sigma'^2_{(trt)})$ respectively, where $\sigma'^2_{(trt)}$ refers to the variability of treatment effect among samples, and similarly for the batch component. We then sampled the corresponding loading vectors $\alpha^{(trt)}$ and $\alpha^{(batch)}$ from a uniform distribution $[-0.3, -0.2] \cup [0.2, 0.3]$ respectively and scaled them as an unit vector. We subsequently constructed the treatment relevant matrix as $X^{(trt)} = t^{(trt)}(\alpha^{(trt)})^\top$ and similarly for the batch relevant matrix.

We also generated background noise E ($E \in \mathbb{R}^{40\times300}$), where each element was randomly sampled from $N(0, 0.2^2)$. The final simulated dataset X_{result} was constructed based on the treatment, batch relevant matrices and background noise. Starting with $X_{result} = E$, we then added different types of variables, such that:

$$X_{result}[, \text{variables}^{(trt)}] = E[, \text{variables}^{(trt)}] + X^{(trt)}$$

$$X_{result}[, \text{variables}^{(batch)}] = X_{result}[, \text{variables}^{(batch)}] + X^{(batch)},$$

where variables with treatment or batch effects were randomly indexed in the data.

In addition to the data with batch effects, we also simulated a ground-truth dataset that only included the background noise and treatment but no batch effect to evaluate batch effect corrected datasets.

In these simulation scenarios, for PLSDA-batch we set $C-1$ (or $B-1$) components associated with treatment (or batch) effects (where C and B represent the total number of treatment and batch groups respectively) as $C-1$ ($B-1$) components are likely to explain 100% variance in Y. The number of variables with a true treatment effect ($M^{(trt)}$) is set as the optimal number to select on each treatment component in sPLSDA-batch.

To save space, here we only describe one scenario that we believe is a representative of real data ($M^{(trt\ \&\ batch)} = 30$, simu 6 in Table 1).

3.2 Simulations

3.2.1 Balanced batch × treatment design

The balanced batch × treatment experimental design included 10 samples from two batches respectively in each treatment group (see Table 2)

Table 2: Balanced batch × treatment design in the simulation study

	Trt1	Trt2
Batch1	10	10
Batch2	10	10

```
nitr <- 50
p_total <- 300
p_trt_relevant <- 60
p_bat_relevant <- 150

# global variance (RDA)
gvar.before <- gvar.clean <- gvar.rbe <- gvar.combat <-
  gvar.plsdab <- gvar.splsdab <- data.frame(treatment = NA, batch = NA,
                                            intersection = NA,
                                            residual = NA)

# individual variance (R2)
r2.trt.before <- r2.trt.clean <- r2.trt.rbe  <- r2.trt.combat <-
  r2.trt.plsdab <- r2.trt.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                       ncol = nitr))
r2.batch.before <- r2.batch.clean <- r2.batch.rbe  <- r2.batch.combat <-
  r2.batch.plsdab <- r2.batch.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                           ncol = nitr))

# precision & recall & F1 (ANOVA)
precision_limma <- recall_limma <- F1_limma <-
  data.frame(before = NA, clean = NA, rbe = NA, combat = NA,
             plsda_batch = NA, splsda_batch = NA)

# precision & recall & F1 (sPLSDA)
precision_splsda <- recall_splsda <- F1_splsda <-
  data.frame(before = NA, clean = NA, rbe = NA, combat = NA,
             plsda_batch = NA, splsda_batch = NA)

for(i in seq_len(nitr)){
  ### initial setup ###
  simulation <- simData_Gaussian(mean.batch = 7, sd.batch = 8,
```

```
                                mean.trt = 3, sd.trt = 2,
                                N = 40, p_total = 300,
                                p_trt_relevant = 60,
                                p_bat_relevant = 150,
                                percentage_overlap_samples = 0.5,
                                percentage_overlap_variables = 0.5, seeds = i)

X <- simulation$data
trt <- simulation$Y.trt
batch <- simulation$Y.bat
true.trt <- simulation$true.trt
true.batch <- simulation$true.batch

Batch_Trt.factors <- data.frame(Batch = batch, Treatment = trt)

### Before correction ###
# global variance (RDA)
rda.before = varpart(scale(X), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.before[i, ] <- rda.before$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.before <- c()
indiv.batch.before <- c()
for(c in seq_len(ncol(X))){
  fit.res1 <- lm(scale(X)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.before <- c(indiv.trt.before, fit.summary1$r.squared)
  indiv.batch.before <- c(indiv.batch.before, fit.summary2$r.squared)
}
r2.trt.before[ ,i] <-  indiv.trt.before
r2.batch.before[ ,i] <-  indiv.batch.before

# precision & recall & F1 (ANOVA)
fit.before <- lmFit(t(scale(X)), design = model.matrix( ~ 0 + as.factor(trt)))
fit.before <- contrasts.fit(fit.before, contrasts = c(1,-1))
fit.result.before <- topTable(eBayes(fit.before), number = p_total)
otu.sig.before <- rownames(fit.result.before)[fit.result.before$adj.P.Val <=
                                              0.05]

precision_limma.before <- length(intersect(true.trt, otu.sig.before))/
  length(otu.sig.before)
recall_limma.before <- length(intersect(true.trt, otu.sig.before))/
```

```
  length(true.trt)
F1_limma.before <- (2*precision_limma.before*recall_limma.before)/
  (precision_limma.before + recall_limma.before)

## replace NA value with 0
if(precision_limma.before == 'NaN'){
  precision_limma.before = 0
}
if(F1_limma.before == 'NaN'){
  F1_limma.before = 0
}

# precision & recall & F1 (sPLSDA)
fit.before_splsda <- splsda(X = X, Y = trt, ncomp = 1,
                            keepX = length(true.trt))
otu.dcn.before <- which(fit.before_splsda$loadings$X[ ,1] != 0)

precision_splsda.before <- length(intersect(true.trt, otu.dcn.before))/
  length(otu.dcn.before)
recall_splsda.before <- length(intersect(true.trt, otu.dcn.before))/
  length(true.trt)
F1_splsda.before <- (2*precision_splsda.before*recall_splsda.before)/
  (precision_splsda.before + recall_splsda.before)

## replace NA value with 0
if(F1_splsda.before == 'NaN'){
  F1_splsda.before = 0
}

#############################################################################
### Ground-truth data ###
X.clean <- simulation$cleanData
# global variance (RDA)
rda.clean = varpart(scale(X.clean), ~ Treatment, ~ Batch,
                    data = Batch_Trt.factors)
gvar.clean[i, ] <- rda.clean$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.clean <- c()
indiv.batch.clean <- c()
for(c in seq_len(ncol(X.clean))){
  fit.res1 <- lm(scale(X.clean)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.clean)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
```

```
  indiv.trt.clean <- c(indiv.trt.clean, fit.summary1$r.squared)
  indiv.batch.clean <- c(indiv.batch.clean, fit.summary2$r.squared)
}
r2.trt.clean[ ,i] <-  indiv.trt.clean
r2.batch.clean[ ,i] <-  indiv.batch.clean

# precision & recall & F1 (ANOVA)
fit.clean <- lmFit(t(scale(X.clean)),
                   design = model.matrix( ~ 0 + as.factor(trt)))
fit.clean <- contrasts.fit(fit.clean, contrasts = c(1,-1))
fit.result.clean <- topTable(eBayes(fit.clean), number = p_total)
otu.sig.clean <- rownames(fit.result.clean)[fit.result.clean$adj.P.Val <=
                                            0.05]

precision_limma.clean <- length(intersect(true.trt, otu.sig.clean))/
  length(otu.sig.clean)
recall_limma.clean <- length(intersect(true.trt, otu.sig.clean))/
  length(true.trt)
F1_limma.clean <- (2*precision_limma.clean*recall_limma.clean)/
  (precision_limma.clean + recall_limma.clean)

## replace NA value with 0
if(precision_limma.clean == 'NaN'){
  precision_limma.clean = 0
}
if(F1_limma.clean == 'NaN'){
  F1_limma.clean = 0
}

# precision & recall & F1 (sPLSDA)
fit.clean_splsda <- splsda(X = X.clean, Y = trt, ncomp = 1,
                           keepX = length(true.trt))
otu.dcn.clean <- which(fit.clean_splsda$loadings$X[ ,1] != 0)

precision_splsda.clean <- length(intersect(true.trt, otu.dcn.clean))/
  length(otu.dcn.clean)
recall_splsda.clean <- length(intersect(true.trt, otu.dcn.clean))/
  length(true.trt)
F1_splsda.clean <- (2*precision_splsda.clean*recall_splsda.clean)/
  (precision_splsda.clean + recall_splsda.clean)

## replace NA value with 0
if(F1_splsda.clean == 'NaN'){
  F1_splsda.clean = 0
}
```

```
##############################################################################
### removeBatchEffect corrected data ###
X.rbe <-t(removeBatchEffect(t(X), batch = batch,
                            design = model.matrix( ~ 0 + as.factor(trt))))

# global variance (RDA)
rda.rbe = varpart(scale(X.rbe), ~ Treatment, ~ Batch,
                  data = Batch_Trt.factors)
gvar.rbe[i, ] <- rda.rbe$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.rbe <- c()
indiv.batch.rbe <- c()
for(c in seq_len(ncol(X.rbe))){
  fit.res1 <- lm(scale(X.rbe)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.rbe)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.rbe <- c(indiv.trt.rbe, fit.summary1$r.squared)
  indiv.batch.rbe <- c(indiv.batch.rbe, fit.summary2$r.squared)
}
r2.trt.rbe[ ,i] <-  indiv.trt.rbe
r2.batch.rbe[ ,i] <-  indiv.batch.rbe

# precision & recall & F1 (ANOVA)
fit.rbe <- lmFit(t(scale(X.rbe)),
                 design = model.matrix( ~ 0 + as.factor(trt)))
fit.rbe <- contrasts.fit(fit.rbe, contrasts = c(1,-1))
fit.result.rbe <- topTable(eBayes(fit.rbe), number = p_total)
otu.sig.rbe <- rownames(fit.result.rbe)[fit.result.rbe$adj.P.Val <= 0.05]

precision_limma.rbe <- length(intersect(true.trt, otu.sig.rbe))/
  length(otu.sig.rbe)
recall_limma.rbe <- length(intersect(true.trt, otu.sig.rbe))/
  length(true.trt)
F1_limma.rbe <- (2*precision_limma.rbe*recall_limma.rbe)/
  (precision_limma.rbe + recall_limma.rbe)

## replace NA value with 0
if(precision_limma.rbe == 'NaN'){
  precision_limma.rbe = 0
}
if(F1_limma.rbe == 'NaN'){
  F1_limma.rbe = 0
}
```

```
# precision & recall & F1 (sPLSDA)
fit.rbe_splsda <- splsda(X = X.rbe, Y = trt, ncomp = 1,
                         keepX = length(true.trt))
otu.dcn.rbe <- which(fit.rbe_splsda$loadings$X[ ,1] != 0)

precision_splsda.rbe <- length(intersect(true.trt, otu.dcn.rbe))/
  length(otu.dcn.rbe)
recall_splsda.rbe <- length(intersect(true.trt, otu.dcn.rbe))/
  length(true.trt)
F1_splsda.rbe <- (2*precision_splsda.rbe*recall_splsda.rbe)/
  (precision_splsda.rbe + recall_splsda.rbe)

## replace NA value with 0
if(F1_splsda.rbe == 'NaN'){
  F1_splsda.rbe = 0
}

#############################################################################
### ComBat corrected data ###
X.combat <- t(ComBat(dat = t(X), batch = batch,
                     mod = model.matrix( ~ as.factor(trt))))

# global variance (RDA)
rda.combat = varpart(scale(X.combat), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.combat[i, ] <- rda.combat$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.combat <- c()
indiv.batch.combat <- c()
for(c in seq_len(ncol(X.combat))){
  fit.res1 <- lm(scale(X.combat)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.combat)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.combat <- c(indiv.trt.combat, fit.summary1$r.squared)
  indiv.batch.combat <- c(indiv.batch.combat, fit.summary2$r.squared)
}
r2.trt.combat[ ,i] <-  indiv.trt.combat
r2.batch.combat[ ,i] <-  indiv.batch.combat

# precision & recall & F1 (ANOVA)
fit.combat <- lmFit(t(scale(X.combat)),
                    design = model.matrix( ~ 0 + as.factor(trt)))
fit.combat <- contrasts.fit(fit.combat, contrasts = c(1,-1))
```

```
fit.result.combat <- topTable(eBayes(fit.combat), number = p_total)
otu.sig.combat <- rownames(fit.result.combat)[fit.result.combat$adj.P.Val <=
                                                0.05]

precision_limma.combat <- length(intersect(true.trt, otu.sig.combat))/
  length(otu.sig.combat)
recall_limma.combat <- length(intersect(true.trt, otu.sig.combat))/
  length(true.trt)
F1_limma.combat <- (2*precision_limma.combat*recall_limma.combat)/
  (precision_limma.combat + recall_limma.combat)

## replace NA value with 0
if(precision_limma.combat == 'NaN'){
  precision_limma.combat = 0
}
if(F1_limma.combat == 'NaN'){
  F1_limma.combat = 0
}

# precision & recall & F1 (sPLSDA)
fit.combat_splsda <- splsda(X = X.combat, Y = trt, ncomp = 1,
                            keepX = length(true.trt))
otu.dcn.combat <- which(fit.combat_splsda$loadings$X[ ,1] != 0)

precision_splsda.combat <- length(intersect(true.trt, otu.dcn.combat))/
  length(otu.dcn.combat)
recall_splsda.combat <- length(intersect(true.trt, otu.dcn.combat))/
  length(true.trt)
F1_splsda.combat <- (2*precision_splsda.combat*recall_splsda.combat)/
  (precision_splsda.combat + recall_splsda.combat)

## replace NA value with 0
if(F1_splsda.combat == 'NaN'){
  F1_splsda.combat = 0
}

################################################################################
### PLSDA-batch corrected data ###
X.plsda_batch.correct <- PLSDA_batch(X = X,
                                     Y.trt = trt, Y.bat = batch,
                                     ncomp.trt = 1, ncomp.bat = 1)
X.plsda_batch <- X.plsda_batch.correct$X.nobatch
colnames(X.plsda_batch) <- seq_len(p_total)

# global variance (RDA)
```

```
rda.plsda_batch = varpart(scale(X.plsda_batch), ~ Treatment, ~ Batch,
                          data = Batch_Trt.factors)
gvar.plsdab[i, ] <- rda.plsda_batch$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.plsda_batch <- c()
indiv.batch.plsda_batch <- c()
for(c in seq_len(ncol(X.plsda_batch))){
  fit.res1 <- lm(scale(X.plsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.plsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.plsda_batch <- c(indiv.trt.plsda_batch,
                             fit.summary1$r.squared)
  indiv.batch.plsda_batch <- c(indiv.batch.plsda_batch,
                               fit.summary2$r.squared)
}
r2.trt.plsdab[ ,i] <-  indiv.trt.plsda_batch
r2.batch.plsdab[ ,i] <-  indiv.batch.plsda_batch

# precision & recall & F1 (ANOVA)
fit.plsda_batch <- lmFit(t(scale(X.plsda_batch)),
                         design = model.matrix( ~ 0 + as.factor(trt)))
fit.plsda_batch <- contrasts.fit(fit.plsda_batch, contrasts = c(1,-1))
fit.result.plsda_batch <- topTable(eBayes(fit.plsda_batch), number = p_total)
otu.sig.plsda_batch <- rownames(fit.result.plsda_batch)[
  fit.result.plsda_batch$adj.P.Val <= 0.05]

precision_limma.plsda_batch <-
  length(intersect(true.trt, otu.sig.plsda_batch))/length(otu.sig.plsda_batch)
recall_limma.plsda_batch <-
  length(intersect(true.trt, otu.sig.plsda_batch))/length(true.trt)
F1_limma.plsda_batch <-
  (2*precision_limma.plsda_batch*recall_limma.plsda_batch)/
  (precision_limma.plsda_batch + recall_limma.plsda_batch)

## replace NA value with 0
if(precision_limma.plsda_batch == 'NaN'){
  precision_limma.plsda_batch = 0
}
if(F1_limma.plsda_batch == 'NaN'){
  F1_limma.plsda_batch = 0
}

# precision & recall & F1 (sPLSDA)
```

```
fit.plsda_batch_splsda <- splsda(X = X.plsda_batch, Y = trt, ncomp = 1,
                                 keepX = length(true.trt))
otu.dcn.plsda_batch <- which(fit.plsda_batch_splsda$loadings$X[ ,1] != 0)

precision_splsda.plsda_batch <-
  length(intersect(true.trt, otu.dcn.plsda_batch))/length(otu.dcn.plsda_batch)
recall_splsda.plsda_batch <-
  length(intersect(true.trt, otu.dcn.plsda_batch))/length(true.trt)
F1_splsda.plsda_batch <-
  (2*precision_splsda.plsda_batch*recall_splsda.plsda_batch)/
  (precision_splsda.plsda_batch + recall_splsda.plsda_batch)

## replace NA value with 0
if(F1_splsda.plsda_batch == 'NaN'){
  F1_splsda.plsda_batch = 0
}

#############################################################################
### sPLSDA-batch corrected data ###
X.splsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt,
                                      Y.bat = batch,
                                      ncomp.trt = 1,
                                      keepX.trt = length(true.trt),
                                      ncomp.bat = 1)
X.splsda_batch <- X.splsda_batch.correct$X.nobatch
colnames(X.splsda_batch) <- seq_len(p_total)

# global variance (RDA)
rda.splsda_batch = varpart(scale(X.splsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.splsdab[i, ] <- rda.splsda_batch$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.splsda_batch <- c()
indiv.batch.splsda_batch <- c()
for(c in seq_len(ncol(X.splsda_batch))){
  fit.res1 <- lm(scale(X.splsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.splsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.splsda_batch <- c(indiv.trt.splsda_batch,
                              fit.summary1$r.squared)
  indiv.batch.splsda_batch <- c(indiv.batch.splsda_batch,
                                fit.summary2$r.squared)
```

```
}
r2.trt.splsdab[ ,i] <-  indiv.trt.splsda_batch
r2.batch.splsdab[ ,i] <-  indiv.batch.splsda_batch

# precision & recall & F1 (ANOVA)
fit.splsda_batch <- lmFit(t(scale(X.splsda_batch)),
                          design = model.matrix( ~ 0 + as.factor(trt)))
fit.splsda_batch <- contrasts.fit(fit.splsda_batch, contrasts = c(1,-1))
fit.result.splsda_batch <- topTable(eBayes(fit.splsda_batch),
                                    number = p_total)
otu.sig.splsda_batch <- rownames(fit.result.splsda_batch)[
  fit.result.splsda_batch$adj.P.Val <= 0.05]

precision_limma.splsda_batch <-
  length(intersect(true.trt, otu.sig.splsda_batch))/
  length(otu.sig.splsda_batch)
recall_limma.splsda_batch <-
  length(intersect(true.trt, otu.sig.splsda_batch))/length(true.trt)
F1_limma.splsda_batch <-
  (2*precision_limma.splsda_batch*recall_limma.splsda_batch)/
  (precision_limma.splsda_batch + recall_limma.splsda_batch)

## replace NA value with 0
if(precision_limma.splsda_batch == 'NaN'){
  precision_limma.splsda_batch = 0
}
if(F1_limma.splsda_batch == 'NaN'){
  F1_limma.splsda_batch = 0
}

# precision & recall & F1 (sPLSDA)
fit.splsda_batch_splsda <- splsda(X = X.splsda_batch, Y = trt, ncomp = 1,
                                  keepX = length(true.trt))
otu.dcn.splsda_batch <- which(fit.splsda_batch_splsda$loadings$X[ ,1] != 0)

precision_splsda.splsda_batch <-
  length(intersect(true.trt, otu.dcn.splsda_batch))/
  length(otu.dcn.splsda_batch)
recall_splsda.splsda_batch <-
  length(intersect(true.trt, otu.dcn.splsda_batch))/length(true.trt)
F1_splsda.splsda_batch <-
  (2*precision_splsda.splsda_batch*recall_splsda.splsda_batch)/
  (precision_splsda.splsda_batch + recall_splsda.splsda_batch)

## replace NA value with 0
```

```
if(F1_splsda.splsda_batch == 'NaN'){
  F1_splsda.splsda_batch = 0
}

# summary
# precision & recall & F1 (ANOVA)
precision_limma[i, ] <- c(`Before correction` = precision_limma.before,
                          `Ground-truth data` = precision_limma.clean,
                          `removeBatchEffect` = precision_limma.rbe,
                          ComBat = precision_limma.combat,
                          `PLSDA-batch` = precision_limma.plsda_batch,
                          `sPLSDA-batch` = precision_limma.splsda_batch)

recall_limma[i, ] <- c(`Before correction` = recall_limma.before,
                       `Ground-truth data` = recall_limma.clean,
                       `removeBatchEffect` = recall_limma.rbe,
                       ComBat = recall_limma.combat,
                       `PLSDA-batch` = recall_limma.plsda_batch,
                       `sPLSDA-batch` = recall_limma.splsda_batch)

F1_limma[i, ] <- c(`Before correction` = F1_limma.before,
                   `Ground-truth data` = F1_limma.clean,
                   `removeBatchEffect` = F1_limma.rbe,
                   ComBat = F1_limma.combat,
                   `PLSDA-batch` = F1_limma.plsda_batch,
                   `sPLSDA-batch` = F1_limma.splsda_batch)


# precision & recall & F1 (sPLSDA)
precision_splsda[i, ] <- c(`Before correction` = precision_splsda.before,
                           `Ground-truth data` = precision_splsda.clean,
                           `removeBatchEffect` = precision_splsda.rbe,
                           ComBat = precision_splsda.combat,
                           `PLSDA-batch` = precision_splsda.plsda_batch,
                           `sPLSDA-batch` = precision_splsda.splsda_batch)

recall_splsda[i, ] <- c(`Before correction` = recall_splsda.before,
                        `Ground-truth data` = recall_splsda.clean,
                        `removeBatchEffect` = recall_splsda.rbe,
                        ComBat = recall_splsda.combat,
                        `PLSDA-batch` = recall_splsda.plsda_batch,
                        `sPLSDA-batch` = recall_splsda.splsda_batch)

F1_splsda[i, ] <- c(`Before correction` = F1_splsda.before,
                    `Ground-truth data` = F1_splsda.clean,
```

```
                      `removeBatchEffect` = F1_splsda.rbe,
                      ComBat = F1_splsda.combat,
                      `PLSDA-batch` = F1_splsda.plsda_batch,
                      `sPLSDA-batch` = F1_splsda.splsda_batch)

}
```

As the simulation step takes time, so we use the saved data into visulisation.

3.2.2 Figures

```
# global variance (RDA)
prop.gvar.all <- rbind(`Before correction` = colMeans(gvar.before),
                       `Ground-truth data` = colMeans(gvar.clean),
                       removeBatchEffect = colMeans(gvar.rbe),
                       ComBat = colMeans(gvar.combat),
                       `PLSDA-batch` = colMeans(gvar.plsdab),
                       `sPLSDA-batch` = colMeans(gvar.splsdab))

prop.gvar.all[prop.gvar.all < 0] = 0
prop.gvar.all <- t(apply(prop.gvar.all, 1, function(x){x/sum(x)}))
colnames(prop.gvar.all) <- c('Treatment', 'Intersection', 'Batch', 'Residuals')

partVar_plot(prop.df = prop.gvar.all)
```

We first considered the proportion of variance explained by treatment and batch effects before and after batch effect correction across all variables using pRDA. Efficient batch effect correction methods should generate data with a smaller proportion of batch associated variance and larger proportion of treatment variance compared to the original data. The figure shows that there was no intersection shared between treatment and batch variation with a balanced batch $\times$ treatment design. All methods successfully removed batch variation, but PLSDA-batch and sPLSDA-batch preserved more proportion of treatment variance than removeBatchEffect and ComBat. In addition, the data corrected by sPLSDA-batch included almost as much proportion of treatment variance as the ground-truth data.

```
##############################################################################
# individual variance (R2)
# color
gcolor <- c(rep(pb_color(16), p_trt_relevant),
            rep(pb_color(17), (p_total - p_trt_relevant)))
gcolor[intersect(true.trt, true.batch)] = pb_color(18)
gcolor[setdiff(seq_len(p_total), union(true.trt, true.batch))] = pb_color(15)

xlabs = 'R2(variable, treatment)'
ylabs = 'R2(variable, batch)'
```

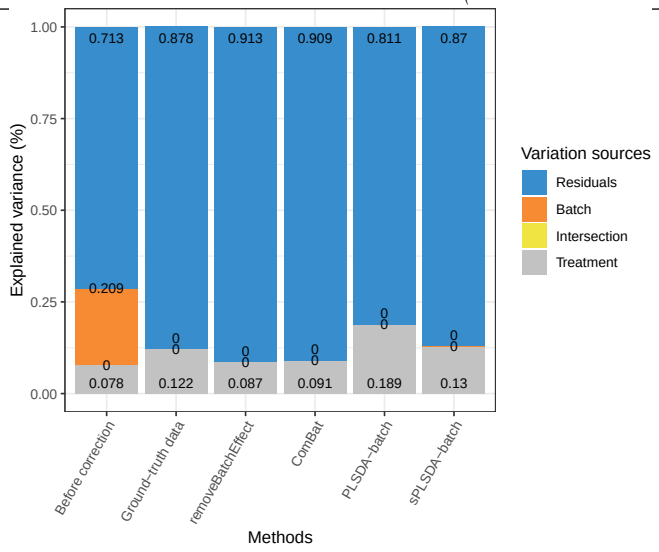

Figure 3.1: Figure 1: Simulation studies (Gaussian distribution): comparison of explained variance before and after batch effect correction for the balanced batch × treatment design.

```
edgex = 0.7
edgey = 0.5

# scatterplot
par(mfrow = c(3,2))
plot(rowMeans(r2.trt.before), rowMeans(r2.batch.before), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey),
     main = 'Before correction', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.clean), rowMeans(r2.batch.clean), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey),
     main = 'Ground-truth data', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.rbe), rowMeans(r2.batch.rbe), col = gcolor,
```

```
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey),
     main = 'removeBatchEffect', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.combat), rowMeans(r2.batch.combat), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey), main = 'ComBat', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.plsdab), rowMeans(r2.batch.plsdab), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey), main = 'PLSDA-batch', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.splsdab), rowMeans(r2.batch.splsdab), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey), main = 'sPLSDA-batch', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)
```

```
par(mfrow = c(1,1))
```

We also estimated the proportion of variance explained by treatment and batch effects for each variable respectively using the R^2 value. The variables assigned with both treatment and batch effects in the corrected data from removeBatchEffect and ComBat presented less proportion of treatment associated variance than in the ground truth data. This result agrees with the pRDA evaluation that these two methods do not preserve enough treatment variation. After PLSDA-batch correction, variables simulated with only batch effects displayed some amount of treatment variation, but only in the case where the batch effect variability among samples was high. sPLSDA-batch outperformed all methods, with results similar to the ground-truth data.

```
# precision & recall & F1 (ANOVA & sPLSDA)
## mean
acc_mean <- rbind(colMeans(precision_limma), colMeans(recall_limma),
                  colMeans(F1_limma), colMeans(precision_splsda))
rownames(acc_mean) <- c('Precision', 'Recall', 'F1', 'Multivariate selection')
colnames(acc_mean) <- c('Before correction', 'Ground-truth data',
```

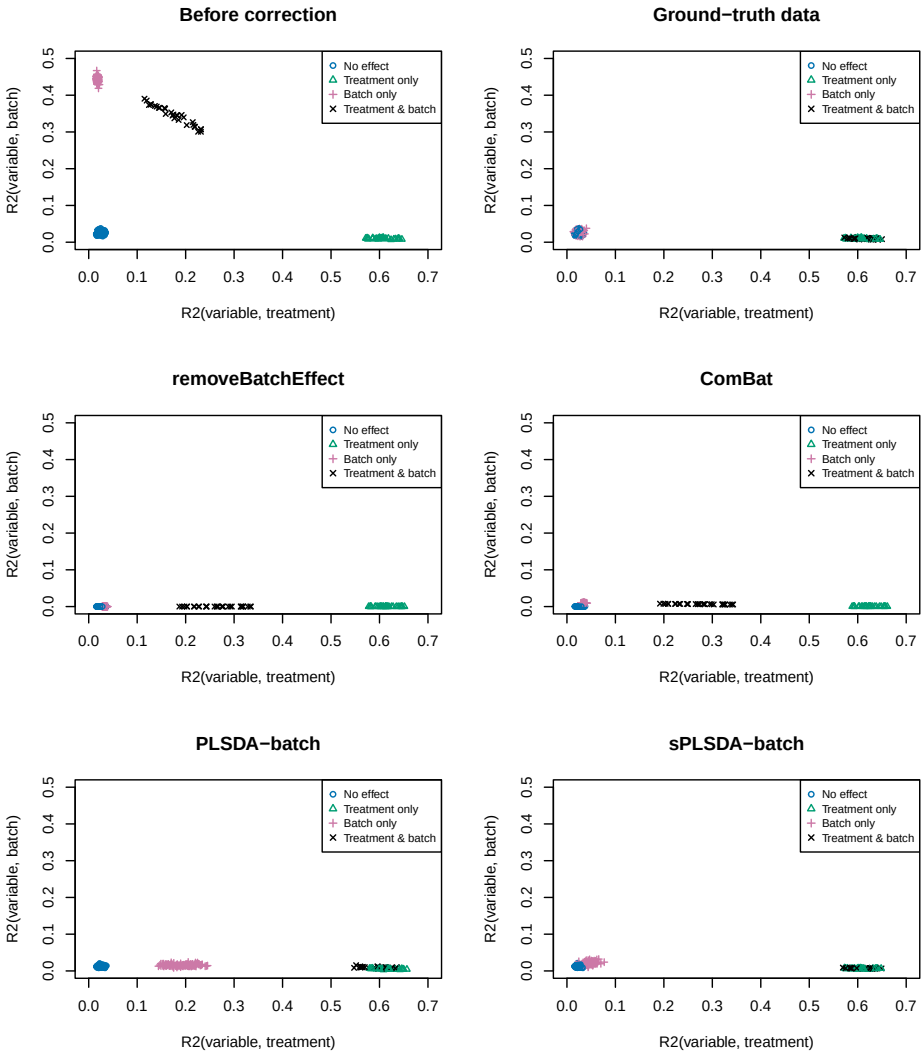

Figure 3.2: Figure 2: Simulation studies (Gaussian distribution): R2 values for each microbial variable before and after batch effect correction for the balanced batch × treatment design.

Table 3.3: Table 3: Simulation studies (Gaussian distribution): summary of accuracy measures before and after batch correction for the balanced batch × treatment design (mean).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	PLSDA-ba
Precision	0.98	0.95	0.94	0.93	0.56
Recall	0.74	1.00	0.87	0.88	1.00
F1	0.84	0.98	0.89	0.89	0.68
Multivariate selection	0.89	1.00	0.92	0.92	0.92

Table 3.4: Table 4: Simulation studies (Gaussian distribution): summary of accuracy measures before and after batch correction for the balanced batch × treatment design (standard deviation).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	PLSDA-ba
Precision	0.021	0.031	0.151	0.160	0.252
Recall	0.095	0.000	0.098	0.098	0.021
F1	0.065	0.016	0.118	0.124	0.200
Multivariate selection	0.061	0.000	0.068	0.069	0.123

```
                        'removeBatchEffect', 'ComBat',
                        'PLSDA-batch', 'sPLSDA-batch')
acc_mean <- format(acc_mean, digits = 2)
knitr::kable(acc_mean, caption = 'Table 3: Simulation studies (Gaussian distribution):
```

When considering the measures of accuracy with univariate one-way ANOVA, we observed that the corrected data from PLSDA-batch and sPLSDA-batch led to higher recall and lower precision than the data from removeBatchEffect and ComBat. However, the precision of sPLSDA-batch was competitive to removeBatchEffect and ComBat. Moreover, sPLSDA-batch achieved higher F1 scores and multivariate selection scores than removeBatchEffect and ComBat.

```
## sd
acc_sd <- rbind(apply(precision_limma, 2, sd), apply(recall_limma, 2, sd),
                apply(F1_limma, 2, sd), apply(precision_splsda, 2, sd))
rownames(acc_sd) <- c('Precision', 'Recall', 'F1', 'Multivariate selection')
colnames(acc_sd) <- c('Before correction', 'Ground-truth data',
                      'removeBatchEffect', 'ComBat',
                      'PLSDA-batch', 'sPLSDA-batch')
acc_sd <- format(acc_sd, digits = 1)
knitr::kable(acc_sd, caption = 'Table 4: Simulation studies (Gaussian distribution): su
```

The standard deviations of the multivariate selection scores were all smaller than the univariate selection scores for the different corrected data, indicating a better stability

of the variables selected by multivariate sPLSDA compared to the one-way ANOVA univariate selection.

3.2.3 Unbalanced batch × treatment design

The unbalanced design had 4 and 16 samples from batch1 and batch2 respectively in trt1, 16 and 4 samples from batch1 and batch2 in trt2 (see the table below).

Table 5: Unbalanced batch × treatment design in the simulation study

	Trt1	Trt2
Batch1	4	16
Batch2	16	4

```
nitr <- 50
p_total <- 300
p_trt_relevant <- 60
p_bat_relevant <- 150

# global variance (RDA)
gvar.before <- gvar.clean <- gvar.rbe <- gvar.combat <-
  gvar.wplsdab <- gvar.swplsdab <- gvar.plsdab <-
  gvar.splsdab <- data.frame(treatment = NA, batch = NA,
                             intersection = NA,
                             residual = NA)

# individual variance (R2)
r2.trt.before <- r2.trt.clean <- r2.trt.rbe  <- r2.trt.combat <-
  r2.trt.wplsdab <- r2.trt.swplsdab <- data.frame(matrix(NA, nrow = p_total,
                                                         ncol = nitr))
r2.batch.before <- r2.batch.clean <- r2.batch.rbe  <- r2.batch.combat <-
  r2.batch.wplsdab <- r2.batch.swplsdab <- data.frame(matrix(NA, nrow = p_total,
                                                             ncol = nitr))

# precision & recall & F1 (ANOVA)
precision_limma <- recall_limma <- F1_limma <-
  data.frame(before = NA, clean = NA, rbe = NA, combat = NA,
             wplsda_batch = NA, swplsda_batch = NA)

# precision & recall & F1 (sPLSDA)
precision_splsda <- recall_splsda <- F1_splsda <-
  data.frame(before = NA, clean = NA, rbe = NA, combat = NA,
             wplsda_batch = NA, swplsda_batch = NA)

for(i in seq_len(nitr)){
```

```
### initial setup ###
simulation <- simData_Gaussian(mean.batch = 7, sd.batch = 8,
                               mean.trt = 3, sd.trt = 2,
                               N = 40, p_total = 300,
                               p_trt_relevant = 60,
                               p_bat_relevant = 150,
                               percentage_overlap_samples = 0.2,
                               percentage_overlap_variables = 0.5, seeds = i)

X <- simulation$data
trt <- simulation$Y.trt
batch <- simulation$Y.bat
true.trt <- simulation$true.trt
true.batch <- simulation$true.batch

Batch_Trt.factors <- data.frame(Batch = batch, Treatment = trt)

### Before correction ###
# global variance (RDA)
rda.before = varpart(scale(X), ~ Treatment, ~ Batch, data = Batch_Trt.factors)
gvar.before[i, ] <- rda.before$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.before <- c()
indiv.batch.before <- c()
for(c in seq_len(ncol(X))){
  fit.res1 <- lm(scale(X)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.before <- c(indiv.trt.before, fit.summary1$r.squared)
  indiv.batch.before <- c(indiv.batch.before, fit.summary2$r.squared)
}
r2.trt.before[ ,i] <-  indiv.trt.before
r2.batch.before[ ,i] <-  indiv.batch.before

# precision & recall & F1 (ANOVA)
fit.before <- lmFit(t(scale(X)), design = model.matrix( ~ 0 + as.factor(trt)))
fit.before <- contrasts.fit(fit.before, contrasts = c(1,-1))
fit.result.before <- topTable(eBayes(fit.before), number = p_total)
otu.sig.before <-
  rownames(fit.result.before)[fit.result.before$adj.P.Val <= 0.05]

precision_limma.before <- length(intersect(true.trt, otu.sig.before))/
  length(otu.sig.before)
```

```
recall_limma.before <- length(intersect(true.trt, otu.sig.before))/
  length(true.trt)
F1_limma.before <- (2*precision_limma.before*recall_limma.before)/
  (precision_limma.before + recall_limma.before)

## replace NA value with 0
if(precision_limma.before == 'NaN'){
  precision_limma.before = 0
}
if(F1_limma.before == 'NaN'){
  F1_limma.before = 0
}

# precision & recall & F1 (sPLSDA)
fit.before_splsda <- splsda(X = X, Y = trt,
                            ncomp = 1, keepX = length(true.trt))
otu.dcn.before <- which(fit.before_splsda$loadings$X[,1] != 0)

precision_splsda.before <- length(intersect(true.trt, otu.dcn.before))/
  length(otu.dcn.before)
recall_splsda.before <- length(intersect(true.trt, otu.dcn.before))/
  length(true.trt)
F1_splsda.before <- (2*precision_splsda.before*recall_splsda.before)/
  (precision_splsda.before + recall_splsda.before)

## replace NA value with 0
if(F1_splsda.before == 'NaN'){
  F1_splsda.before = 0
}

#############################################################################
### Ground-truth data ###
X.clean <- simulation$cleanData
# global variance (RDA)
rda.clean = varpart(scale(X.clean), ~ Treatment, ~ Batch,
                    data = Batch_Trt.factors)
gvar.clean[i, ] <- rda.clean$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.clean <- c()
indiv.batch.clean <- c()
for(c in seq_len(ncol(X.clean))){
  fit.res1 <- lm(scale(X.clean)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.clean)[ ,c] ~ batch)
```

```
  fit.summary2 <- summary(fit.res2)
  indiv.trt.clean <- c(indiv.trt.clean, fit.summary1$r.squared)
  indiv.batch.clean <- c(indiv.batch.clean, fit.summary2$r.squared)
}
r2.trt.clean[ ,i] <-  indiv.trt.clean
r2.batch.clean[ ,i] <-  indiv.batch.clean

# precision & recall & F1 (ANOVA)
fit.clean <- lmFit(t(scale(X.clean)),
                   design = model.matrix( ~ 0 + as.factor(trt)))
fit.clean <- contrasts.fit(fit.clean, contrasts = c(1,-1))
fit.result.clean <- topTable(eBayes(fit.clean), number = p_total)
otu.sig.clean <-
  rownames(fit.result.clean)[fit.result.clean$adj.P.Val <= 0.05]

precision_limma.clean <- length(intersect(true.trt, otu.sig.clean))/
  length(otu.sig.clean)
recall_limma.clean <- length(intersect(true.trt, otu.sig.clean))/
  length(true.trt)
F1_limma.clean <- (2*precision_limma.clean*recall_limma.clean)/
  (precision_limma.clean + recall_limma.clean)

## replace NA value with 0
if(precision_limma.clean == 'NaN'){
  precision_limma.clean = 0
}
if(F1_limma.clean == 'NaN'){
  F1_limma.clean = 0
}

# precision & recall & F1 (sPLSDA)
fit.clean_splsda <- splsda(X = X.clean, Y = trt, ncomp = 1,
                           keepX = length(true.trt))
otu.dcn.clean <- which(fit.clean_splsda$loadings$X[ ,1] != 0)

precision_splsda.clean <- length(intersect(true.trt, otu.dcn.clean))/
  length(otu.dcn.clean)
recall_splsda.clean <- length(intersect(true.trt, otu.dcn.clean))/
  length(true.trt)
F1_splsda.clean <- (2*precision_splsda.clean*recall_splsda.clean)/
  (precision_splsda.clean + recall_splsda.clean)

## replace NA value with 0
if(F1_splsda.clean == 'NaN'){
  F1_splsda.clean = 0
```

```
}

#############################################################################
### removeBatchEffect corrected data ###
X.rbe <-t(removeBatchEffect(t(X), batch = batch,
                            design = model.matrix( ~ 0 + as.factor(trt))))

# global variance (RDA)
rda.rbe = varpart(scale(X.rbe), ~ Treatment, ~ Batch,
                  data = Batch_Trt.factors)
gvar.rbe[i, ] <- rda.rbe$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.rbe <- c()
indiv.batch.rbe <- c()
for(c in seq_len(ncol(X.rbe))){
  fit.res1 <- lm(scale(X.rbe)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.rbe)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.rbe <- c(indiv.trt.rbe, fit.summary1$r.squared)
  indiv.batch.rbe <- c(indiv.batch.rbe, fit.summary2$r.squared)
}
r2.trt.rbe[ ,i] <-  indiv.trt.rbe
r2.batch.rbe[ ,i] <-  indiv.batch.rbe

# precision & recall & F1 (ANOVA)
fit.rbe <- lmFit(t(scale(X.rbe)),
                 design = model.matrix( ~ 0 + as.factor(trt)))
fit.rbe <- contrasts.fit(fit.rbe, contrasts = c(1,-1))
fit.result.rbe <- topTable(eBayes(fit.rbe), number = p_total)
otu.sig.rbe <- rownames(fit.result.rbe)[fit.result.rbe$adj.P.Val <= 0.05]

precision_limma.rbe <- length(intersect(true.trt, otu.sig.rbe))/
  length(otu.sig.rbe)
recall_limma.rbe <- length(intersect(true.trt, otu.sig.rbe))/
  length(true.trt)
F1_limma.rbe <- (2*precision_limma.rbe*recall_limma.rbe)/
  (precision_limma.rbe + recall_limma.rbe)

## replace NA value with 0
if(precision_limma.rbe == 'NaN'){
  precision_limma.rbe = 0
}
if(F1_limma.rbe == 'NaN'){
```

```
  F1_limma.rbe = 0
}

# precision & recall & F1 (sPLSDA)
fit.rbe_splsda <- splsda(X = X.rbe, Y = trt,
                         ncomp = 1, keepX = length(true.trt))
otu.dcn.rbe <- which(fit.rbe_splsda$loadings$X[ ,1] != 0)

precision_splsda.rbe <- length(intersect(true.trt, otu.dcn.rbe))/
  length(otu.dcn.rbe)
recall_splsda.rbe <- length(intersect(true.trt, otu.dcn.rbe))/
  length(true.trt)
F1_splsda.rbe <- (2*precision_splsda.rbe*recall_splsda.rbe)/
  (precision_splsda.rbe + recall_splsda.rbe)

## replace NA value with 0
if(F1_splsda.rbe == 'NaN'){
  F1_splsda.rbe = 0
}

#############################################################################
### ComBat corrected data ###
X.combat <- t(ComBat(dat = t(X), batch = batch,
                     mod = model.matrix( ~ as.factor(trt))))

# global variance (RDA)
rda.combat = varpart(scale(X.combat), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.combat[i, ] <- rda.combat$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.combat <- c()
indiv.batch.combat <- c()
for(c in seq_len(ncol(X.combat))){
  fit.res1 <- lm(scale(X.combat)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.combat)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.combat <- c(indiv.trt.combat, fit.summary1$r.squared)
  indiv.batch.combat <- c(indiv.batch.combat, fit.summary2$r.squared)
}
r2.trt.combat[ ,i] <-  indiv.trt.combat
r2.batch.combat[ ,i] <-  indiv.batch.combat

# precision & recall & F1 (ANOVA)
```

```
fit.combat <- lmFit(t(scale(X.combat)),
                    design = model.matrix( ~ 0 + as.factor(trt)))
fit.combat <- contrasts.fit(fit.combat, contrasts = c(1,-1))
fit.result.combat <- topTable(eBayes(fit.combat), number = p_total)
otu.sig.combat <-
  rownames(fit.result.combat)[fit.result.combat$adj.P.Val <= 0.05]

precision_limma.combat <- length(intersect(true.trt, otu.sig.combat))/
  length(otu.sig.combat)
recall_limma.combat <- length(intersect(true.trt, otu.sig.combat))/
  length(true.trt)
F1_limma.combat <- (2*precision_limma.combat*recall_limma.combat)/
  (precision_limma.combat + recall_limma.combat)

## replace NA value with 0
if(precision_limma.combat == 'NaN'){
  precision_limma.combat = 0
}
if(F1_limma.combat == 'NaN'){
  F1_limma.combat = 0
}

# precision & recall & F1 (sPLSDA)
fit.combat_splsda <- splsda(X = X.combat, Y = trt, ncomp = 1,
                            keepX = length(true.trt))
otu.dcn.combat <- which(fit.combat_splsda$loadings$X[ ,1] != 0)

precision_splsda.combat <- length(intersect(true.trt, otu.dcn.combat))/
  length(otu.dcn.combat)
recall_splsda.combat <- length(intersect(true.trt, otu.dcn.combat))/
  length(true.trt)
F1_splsda.combat <- (2*precision_splsda.combat*recall_splsda.combat)/
  (precision_splsda.combat + recall_splsda.combat)

## replace NA value with 0
if(F1_splsda.combat == 'NaN'){
  F1_splsda.combat = 0
}

##############################################################################
### wPLSDA-batch corrected data ###
X.wplsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt, Y.bat = batch,
                                      ncomp.trt = 1, ncomp.bat = 1,
                                      balance = FALSE)
```

```
X.wplsda_batch <- X.wplsda_batch.correct$X.nobatch
colnames(X.wplsda_batch) <- seq_len(p_total)

# global variance (RDA)
rda.wplsda_batch = varpart(scale(X.wplsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.wplsdab[i, ] <- rda.wplsda_batch$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.wplsda_batch <- c()
indiv.batch.wplsda_batch <- c()
for(c in seq_len(ncol(X.wplsda_batch))){
  fit.res1 <- lm(scale(X.wplsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.wplsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.wplsda_batch <- c(indiv.trt.wplsda_batch, fit.summary1$r.squared)
  indiv.batch.wplsda_batch <- c(indiv.batch.wplsda_batch,
                                fit.summary2$r.squared)
}
r2.trt.wplsdab[ ,i] <-  indiv.trt.wplsda_batch
r2.batch.wplsdab[ ,i] <-  indiv.batch.wplsda_batch

# precision & recall & F1 (ANOVA)
fit.wplsda_batch <- lmFit(t(scale(X.wplsda_batch)),
                          design = model.matrix( ~ 0 + as.factor(trt)))
fit.wplsda_batch <- contrasts.fit(fit.wplsda_batch, contrasts = c(1,-1))
fit.result.wplsda_batch <- topTable(eBayes(fit.wplsda_batch),
                                    number = p_total)
otu.sig.wplsda_batch <- rownames(fit.result.wplsda_batch)[
  fit.result.wplsda_batch$adj.P.Val <= 0.05]

precision_limma.wplsda_batch <-
  length(intersect(true.trt, otu.sig.wplsda_batch))/
  length(otu.sig.wplsda_batch)
recall_limma.wplsda_batch <-
  length(intersect(true.trt, otu.sig.wplsda_batch))/
  length(true.trt)
F1_limma.wplsda_batch <-
  (2*precision_limma.wplsda_batch*recall_limma.wplsda_batch)/
  (precision_limma.wplsda_batch + recall_limma.wplsda_batch)

## replace NA value with 0
if(precision_limma.wplsda_batch == 'NaN'){
  precision_limma.wplsda_batch = 0
```

```
}
if(F1_limma.wplsda_batch == 'NaN'){
  F1_limma.wplsda_batch = 0
}

# precision & recall & F1 (sPLSDA)
fit.wplsda_batch_splsda <- splsda(X = X.wplsda_batch, Y = trt, ncomp = 1,
                                  keepX = length(true.trt))
otu.dcn.wplsda_batch <- which(fit.wplsda_batch_splsda$loadings$X[ ,1] != 0)

precision_splsda.wplsda_batch <-
  length(intersect(true.trt, otu.dcn.wplsda_batch))/
  length(otu.dcn.wplsda_batch)
recall_splsda.wplsda_batch <-
  length(intersect(true.trt, otu.dcn.wplsda_batch))/
  length(true.trt)
F1_splsda.wplsda_batch <-
  (2*precision_splsda.wplsda_batch*recall_splsda.wplsda_batch)/
  (precision_splsda.wplsda_batch + recall_splsda.wplsda_batch)

## replace NA value with 0
if(F1_splsda.wplsda_batch == 'NaN'){
  F1_splsda.wplsda_batch = 0
}

#############################################################################
### swPLSDA-batch corrected data ###
X.swplsda_batch.correct <- PLSDA_batch(X = X,
                                       Y.trt = trt, Y.bat = batch,
                                       ncomp.trt = 1,
                                       keepX.trt = length(true.trt),
                                       ncomp.bat = 1, balance = FALSE)
X.swplsda_batch <- X.swplsda_batch.correct$X.nobatch
colnames(X.swplsda_batch) <- seq_len(p_total)

# global variance (RDA)
rda.swplsda_batch = varpart(scale(X.swplsda_batch), ~ Treatment, ~ Batch,
                            data = Batch_Trt.factors)
gvar.swplsdab[i, ] <- rda.swplsda_batch$part$indfract$Adj.R.squared

# individual variance (R2)
indiv.trt.swplsda_batch <- c()
indiv.batch.swplsda_batch <- c()
for(c in seq_len(ncol(X.swplsda_batch))){
  fit.res1 <- lm(scale(X.swplsda_batch)[ ,c] ~ trt)
```

```
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.swplsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.swplsda_batch <- c(indiv.trt.swplsda_batch,
                               fit.summary1$r.squared)
  indiv.batch.swplsda_batch <- c(indiv.batch.swplsda_batch,
                                 fit.summary2$r.squared)
}
r2.trt.swplsdab[ ,i] <-  indiv.trt.swplsda_batch
r2.batch.swplsdab[ ,i] <-  indiv.batch.swplsda_batch

# precision & recall & F1 (ANOVA)
fit.swplsda_batch <- lmFit(t(scale(X.swplsda_batch)),
                           design = model.matrix( ~ 0 + as.factor(trt)))
fit.swplsda_batch <- contrasts.fit(fit.swplsda_batch, contrasts = c(1,-1))
fit.result.swplsda_batch <- topTable(eBayes(fit.swplsda_batch),
                                     number = p_total)
otu.sig.swplsda_batch <- rownames(fit.result.swplsda_batch)[
  fit.result.swplsda_batch$adj.P.Val <= 0.05]

precision_limma.swplsda_batch <-
  length(intersect(true.trt, otu.sig.swplsda_batch))/
  length(otu.sig.swplsda_batch)
recall_limma.swplsda_batch <-
  length(intersect(true.trt, otu.sig.swplsda_batch))/
  length(true.trt)
F1_limma.swplsda_batch <-
  (2*precision_limma.swplsda_batch*recall_limma.swplsda_batch)/
  (precision_limma.swplsda_batch + recall_limma.swplsda_batch)

## replace NA value with 0
if(precision_limma.swplsda_batch == 'NaN'){
  precision_limma.swplsda_batch = 0
}
if(F1_limma.swplsda_batch == 'NaN'){
  F1_limma.swplsda_batch = 0
}

# precision & recall & F1 (sPLSDA)
fit.swplsda_batch_splsda <- splsda(X = X.swplsda_batch, Y = trt, ncomp = 1,
                                   keepX = length(true.trt))
otu.dcn.swplsda_batch <- which(fit.swplsda_batch_splsda$loadings$X[ ,1] != 0)

precision_splsda.swplsda_batch <-
  length(intersect(true.trt, otu.dcn.swplsda_batch))/
```

```
  length(otu.dcn.swplsda_batch)
recall_splsda.swplsda_batch <-
  length(intersect(true.trt, otu.dcn.swplsda_batch))/
  length(true.trt)
F1_splsda.swplsda_batch <-
  (2*precision_splsda.swplsda_batch*recall_splsda.swplsda_batch)/
  (precision_splsda.swplsda_batch + recall_splsda.swplsda_batch)

## replace NA value with 0
if(F1_splsda.swplsda_batch == 'NaN'){
  F1_splsda.swplsda_batch = 0
}

#############################################################################
### PLSDA-batch corrected data ###
X.plsda_batch.correct <- PLSDA_batch(X = X,
                                     Y.trt = trt, Y.bat = batch,
                                     ncomp.trt = 1, ncomp.bat = 1)
X.plsda_batch <- X.plsda_batch.correct$X.nobatch
colnames(X.plsda_batch) <- seq_len(p_total)

# global variance (RDA)
rda.plsda_batch = varpart(scale(X.plsda_batch), ~ Treatment, ~ Batch,
                          data = Batch_Trt.factors)
gvar.plsdab[i, ] <- rda.plsda_batch$part$indfract$Adj.R.squared

#############################################################################
### sPLSDA-batch corrected data ###
X.splsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt, Y.bat = batch,
                                      ncomp.trt = 1,
                                      keepX.trt = length(true.trt),
                                      ncomp.bat = 1)
X.splsda_batch <- X.splsda_batch.correct$X.nobatch
colnames(X.splsda_batch) <- seq_len(p_total)

# global variance (RDA)
rda.splsda_batch = varpart(scale(X.splsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.splsdab[i, ] <- rda.splsda_batch$part$indfract$Adj.R.squared


# summary
# precision & recall & F1 (ANOVA)
precision_limma[i, ] <- c(`Before correction` = precision_limma.before,
```

```
                            `Ground-truth data` = precision_limma.clean,
                            `removeBatchEffect` = precision_limma.rbe,
                            ComBat = precision_limma.combat,
                            `wPLSDA-batch` = precision_limma.wplsda_batch,
                            `swPLSDA-batch` = precision_limma.swplsda_batch)

  recall_limma[i, ] <- c(`Before correction` = recall_limma.before,
                         `Ground-truth data` = recall_limma.clean,
                         `removeBatchEffect` = recall_limma.rbe,
                         ComBat = recall_limma.combat,
                         `wPLSDA-batch` = recall_limma.wplsda_batch,
                         `swPLSDA-batch` = recall_limma.swplsda_batch)

  F1_limma[i, ] <- c(`Before correction` = F1_limma.before,
                     `Ground-truth data` = F1_limma.clean,
                     `removeBatchEffect` = F1_limma.rbe,
                     ComBat = F1_limma.combat,
                     `wPLSDA-batch` = F1_limma.wplsda_batch,
                     `swPLSDA-batch` = F1_limma.swplsda_batch)


  # precision & recall & F1 (sPLSDA)
  precision_splsda[i, ] <- c(`Before correction` = precision_splsda.before,
                             `Ground-truth data` = precision_splsda.clean,
                             `removeBatchEffect` = precision_splsda.rbe,
                             ComBat = precision_splsda.combat,
                             `wPLSDA-batch` = precision_splsda.wplsda_batch,
                             `swPLSDA-batch` = precision_splsda.swplsda_batch)

  recall_splsda[i, ] <- c(`Before correction` = recall_splsda.before,
                          `Ground-truth data` = recall_splsda.clean,
                          `removeBatchEffect` = recall_splsda.rbe,
                          ComBat = recall_splsda.combat,
                          `wPLSDA-batch` = recall_splsda.wplsda_batch,
                          `swPLSDA-batch` = recall_splsda.swplsda_batch)

  F1_splsda[i, ] <- c(`Before correction` = F1_splsda.before,
                      `Ground-truth data` = F1_splsda.clean,
                      `removeBatchEffect` = F1_splsda.rbe,
                      ComBat = F1_splsda.combat,
                      `wPLSDA-batch` = F1_splsda.wplsda_batch,
                      `swPLSDA-batch` = F1_splsda.swplsda_batch)

}
```

3.2.4 Figures

```
# global variance (RDA)
prop.gvar.all <- rbind(`Before correction` = colMeans(gvar.before),
                       `Ground-truth data` = colMeans(gvar.clean),
                       removeBatchEffect = colMeans(gvar.rbe),
                       ComBat = colMeans(gvar.combat),
                       `wPLSDA-batch` = colMeans(gvar.wplsdab),
                       `swPLSDA-batch` = colMeans(gvar.swplsdab),
                       `PLSDA-batch` = colMeans(gvar.plsdab),
                       `sPLSDA-batch` = colMeans(gvar.splsdab))

prop.gvar.all[prop.gvar.all < 0] = 0
prop.gvar.all <- t(apply(prop.gvar.all, 1, function(x){x/sum(x)}))
colnames(prop.gvar.all) <- c('Treatment', 'Intersection', 'Batch', 'Residuals')

partVar_plot(prop.df = prop.gvar.all)
```

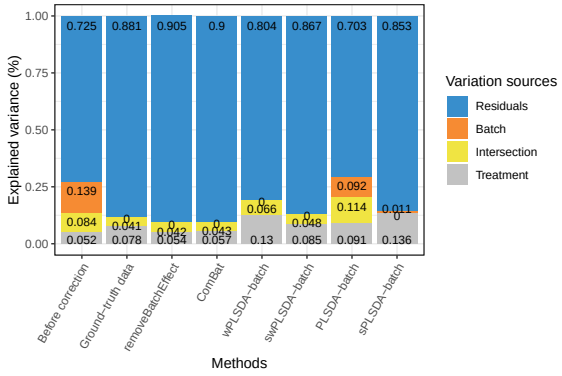

Figure 3.3: Figure 3: Simulation studies (Gaussian distribution): comparison of explained variance before and after batch effect correction for the unbalanced batch × treatment design.

With an unbalanced batch × treatment design, we observed that certain amount of variance was shared (intersection) and explained by both batch and treatment effects. Such intersectional variance should exist even in the ground-truth data with no batch effect, as it originates from treatment variation because of the unbalanced design. Unweighted PLSDA-batch and sPLSDA-batch failed in such design, as their corrected data still included a large amount of batch variation (PLSDA-batch) or not included intersectional variance (sPLSDA-batch), while the other methods removed batch variation successfully. The corrected data from removeBatchEffect and ComBat included less proportion of variance explained by treatment but more intersectional variance compared to the ground-truth data. Although wPLSDA-batch corrected data included the largest treatment variance, swPLSDA-batch outperformed all methods with results similar to

the ground-truth data.

```
##############################################################################
# individual variance (R2)
# color
gcolor <- c(rep(pb_color(16), p_trt_relevant),
            rep(pb_color(17), (p_total - p_trt_relevant)))
gcolor[intersect(true.trt, true.batch)] = pb_color(18)
gcolor[setdiff(seq_len(p_total), union(true.trt, true.batch))] = pb_color(15)

xlabs = 'R2(variable, treatment)'
ylabs = 'R2(variable, batch)'
edgex = 0.7
edgey = 0.6

# scatterplot
par(mfrow = c(3,2))
plot(rowMeans(r2.trt.before), rowMeans(r2.batch.before), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey),
     main = 'Before correction', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.clean), rowMeans(r2.batch.clean), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey),
     main = 'Ground-truth data', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.rbe), rowMeans(r2.batch.rbe), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey),
     main = 'removeBatchEffect', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.combat), rowMeans(r2.batch.combat), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey), main = 'ComBat', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
```

Table 3.6: Table 6: Simulation studies (Gaussian distribution): summary of accuracy measures before and after batch correction for the unbalanced batch × treatment design (mean).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	wPLSDA-batch	swPL
Precision	0.52	0.96	0.85	0.80	0.52	0.80
Recall	0.72	1.00	0.86	0.86	0.99	1.00
F1	0.55	0.98	0.84	0.81	0.65	0.88
Multivariate selection	0.73	1.00	0.88	0.87	0.89	0.99

```
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.wplsdab), rowMeans(r2.batch.wplsdab), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey), main = 'wPLSDA-batch', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)

plot(rowMeans(r2.trt.swplsdab), rowMeans(r2.batch.swplsdab), col = gcolor,
     xlab = xlabs, ylab = ylabs, pch = as.numeric(as.factor(gcolor)),
     xlim = c(0, edgex), ylim = c(0, edgey), main = 'swPLSDA-batch', cex = 0.7)
legend('topright', legend = c('No effect', 'Treatment only',
                              'Batch only', 'Treatment & batch'),
       col = pb_color(15:18), pch = seq_len(4), cex = 0.8)
```

```
par(mfrow = c(1,1))
```

Based on the R^2 values, variables assigned with both treatment and batch effects were segregated into two groups depending on whether their abundance increased or decreased consistently or not according to the two effects. We observed similar results to those obtained from the balanced design.

```
# precision & recall & F1 (ANOVA & sPLSDA)
## mean
acc_mean <- rbind(colMeans(precision_limma), colMeans(recall_limma),
                  colMeans(F1_limma), colMeans(precision_splsda))
rownames(acc_mean) <- c('Precision', 'Recall', 'F1', 'Multivariate selection')
colnames(acc_mean) <- c('Before correction', 'Ground-truth data',
                        'removeBatchEffect', 'ComBat',
                        'wPLSDA-batch', 'swPLSDA-batch')
acc_mean <- format(acc_mean, digits = 2)
knitr::kable(acc_mean, caption = 'Table 6: Simulation studies (Gaussian distribution): summary of
```

When considering the measures of accuracy with univariate one-way ANOVA, we

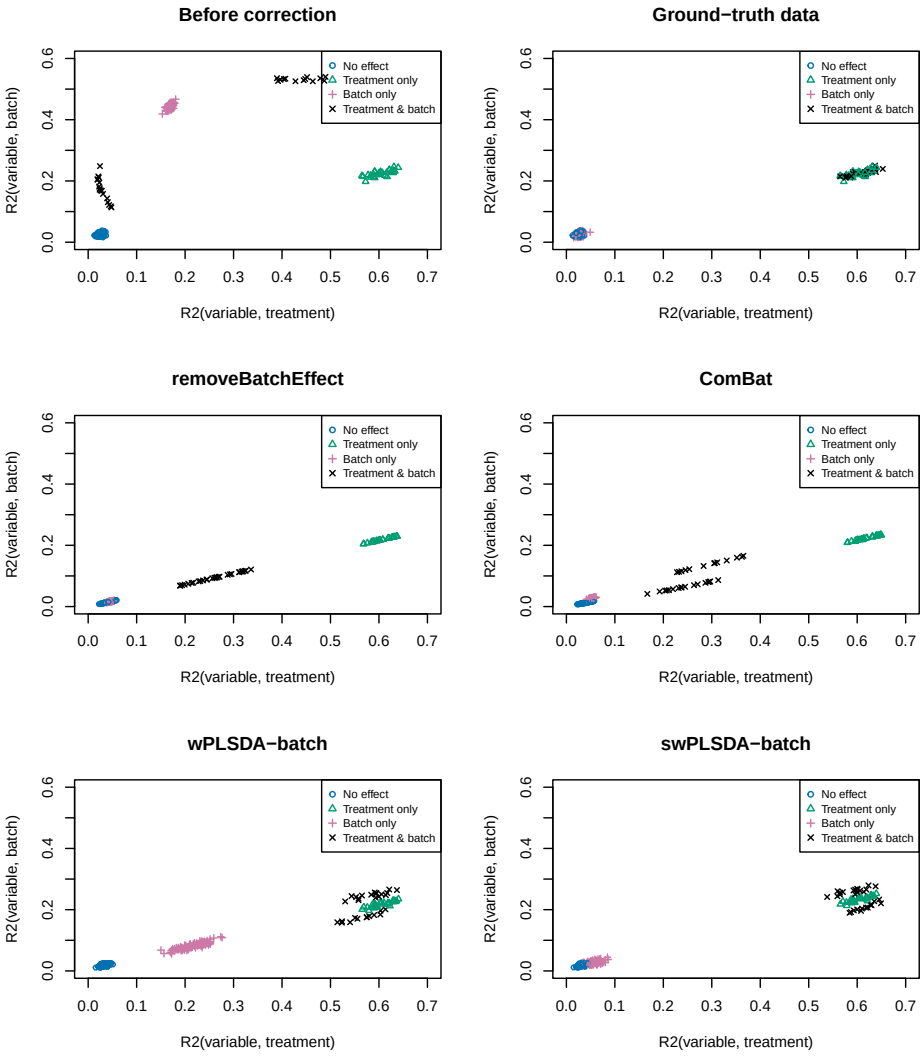

Figure 3.4: Figure 4: Simulation studies (Gaussian distribution): R2 values for each microbial variable before and after batch effect correction for the unbalanced batch × treatment design.

Table 3.7: Table 7: Simulation studies (Gaussian distribution): summary of accuracy measures before and after batch correction for the unbalanced batch × treatment design (standard deviation).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	wPLSDA-batch	swPL
Precision	0.325	0.031	0.176	0.231	0.232	0.137
Recall	0.042	0.000	0.104	0.100	0.029	0.000
F1	0.212	0.017	0.137	0.177	0.189	0.092
Multivariate selection	0.051	0.000	0.069	0.083	0.148	0.016

observed that the corrected data from wPLSDA-batch and swPLSDA-batch led to higher recall and lower precision than the data from removeBatchEffect and ComBat. However, the precision of swPLSDA-batch was competitive to removeBatchEffect and ComBat. Moreover, weighted sPLSDA-batch achieved higher F1 scores and multivariate selection scores than removeBatchEffect and ComBat.

```
## sd
acc_sd <- rbind(apply(precision_limma, 2, sd), apply(recall_limma, 2, sd),
                apply(F1_limma, 2, sd), apply(precision_splsda, 2, sd))
rownames(acc_sd) <- c('Precision', 'Recall', 'F1', 'Multivariate selection')
colnames(acc_sd) <- c('Before correction', 'Ground-truth data',
                      'removeBatchEffect', 'ComBat',
                      'wPLSDA-batch', 'swPLSDA-batch')
acc_sd <- format(acc_sd, digits = 2)
knitr::kable(acc_sd, caption = 'Table 7: Simulation studies (Gaussian distribution): summary of a
```

The standard deviations of the multivariate selection scores were all smaller than the univariate selection scores for the different corrected data, indicating a better stability of the variables selected by multivariate sPLSDA compared to the one-way ANOVA univariate selection.

3.3 References

Chapter 4

Batch Effects Management in Simulation Studies (Negative Binomial Distribution)

4.1 Introduction

Microbiome data are multivariate with inherent correlation structure between microbial variables. The data are over-dispersed with a distribution close to a negative binomial distribution [McMurdie and Holmes, 2014, Quinn et al., 2018]. Inspired by [Hawinkel et al., 2019], we simulated data from multivariate negative binomial distribution achieved with quantile-quantile transformation between multivariate normal and negative binomial distributions. To add treatment and batch effects, we used matrix factorisation to simulate the mean for modelling negative binomial distribution as a matrix

$$\Theta = \begin{bmatrix} \theta_{11} & ... & \theta_{1M} \\ ... & ... & ... \\ \theta_{1N} & ... & \theta_{NM} \end{bmatrix}$$

for N samples and M microbial variables as follows:

$$\Theta = exp(x_{(trt)}^{\top}\beta^{(trt)} + x_{(batch)}^{\top}\beta^{(batch)} + \epsilon)$$

where $x_{(trt)}$ and $x_{(batch)}$ represent the design vectors of treatment and batch effects respectively for each sample. $\beta^{(trt)}$ and $\beta^{(batch)}$ represent the regression coefficients of treatment and batch effects for each microbial variable, and $\beta_j^{(trt)} \in N(\mu_{(trt)}, \sigma_{(trt)}^2)$, $\beta_j^{(batch)} \in N(\mu_{(batch)}, \sigma_{(batch)}^2)$. ϵ contains the random noise that is independent and

identically distributed (i.i.d) and $\epsilon_{ij} \in N(0, \delta^2)$, in which $i = 1, 2, ..., N$ samples, $j = 1, 2, .., M$ variables.

The probability matrix

$$P = \begin{bmatrix} p_{11} & \cdots & p_{1M} \\ \cdots & \cdots & \cdots \\ p_{1N} & \cdots & p_{NM} \end{bmatrix}$$

for modelling negative binomial distribution is calculated as

$$p_{ij} = \frac{r}{r + \theta_{ij}}$$

where p_{ij} and θ_{ij} represent the probability of success in each trial and the mean for negative binomial distribution of sample i and microbial variable j, and r is the dispersion parameter representing the number of successes.

We then simulated a data matrix based on multivariate normal distribution with mean 0 and correlation matrix Σ:

$$X^{normal} = N(0, \Sigma)$$

where the correlation matrix Σ was simulated with the strategy adapted from [McGregor et al., 2020] as follows: We first generated a lower-triangular matrix L, in which the diagonal elements follow $Unif(1.5, 2.5)$, and the other elements $Unif(-1.5, 1.5)$. We randomly set the elements outside the diagonal of L to zero with probability 0.7. A precision matrix, which is the inverse of covariance matrix was created as $R^{-1} = LL^{\top}$. The corresponding correlation matrix Σ to R was then obtained. These parameters were set according to [McGregor et al., 2020].

Thereafter we used Cumulative Distribution Function (CDF) to achieve quantile-quantile transformation as:

$$CDF(x_{ij}^{normal}) = CDF(x_{ij}^{nb})$$

where $CDF(x_{ij}^{normal})$ represents the cumulative probability of x_{ij}^{normal} for sample i and variable j that belongs to matrix X^{normal} from multivariate normal distribution. $CDF(x_{ij}^{nb})$ represents the cumulative probability of each x_{ij}^{nb} in matrix X^{nb} from negative binomial distribution.

Based on the cumulative probability, we can simulate a data matrix X^{nb} with multivariate negative binomial distribution:

$$X^{nb} = NB(r, P, \Sigma)$$

where r represents the dispersion parameter, P represents the probability matrix and Σ the correlation matrix explaining the dependence structure between microbial variables.

We simulated datasets with different parameters including amount of batch and treatment effects ($\mu_{(batch)}$, $\mu_{(trt)}$) and variability among variables ($\sigma_{(batch)}$, $\sigma_{(trt)}$), number of variables with batch and/or treatment effects ($M^{(batch)}$, $M^{(trt)}$ and $M^{(trt\ \&\ batch)}$), balanced and unbalanced batch $\times$ treatment designs, as summarised in the table below.

Table 1: Summary of simulation scenarios (Negative Binomial distribution). For a given choice of parameters reported in this table, each simulation was repeated 50 times. $M^{(trt)}$, $M^{(batch)}$ and $M^{(trt\ \&\ batch)}$ represent the number of variables with treatment, batch, or both effects respectively. Simu 6 includes parameters likely to represent real data according to our experience in analysing microbiome datasets.

	$\mu_{(trt)}$	$\sigma_{(trt)}$	$\mu_{(batch)}$	$\sigma_{(batch)}$	$M^{(trt)}$	$M^{(batch)}$	$M^{(trt\ \&\ batch)}$
Simu 1	3	1	7	{1,4,8}	60	150	0
Simu 2	{3,5,7}	1	7	8	60	150	0
Simu 3	3	{1,2,4}	7	8	60	150	0
Simu 4	3	2	7	8	{30,60,100,150}	150	0
Simu 5	3	2	7	8	60	{30,60,100,150}	0
Simu 6	3	2	7	8	60	150	{0,18,30,42,60}

The microbial variables with treatment or batch effects were randomly indexed in the data with non-zero $\beta^{(trt)}$ or $\beta^{(batch)}$. The background noise ϵ_{ij} was randomly sampled from $N(0, 0.2^2)$, reflecting real microbiome datasets.

We also simulated datasets with different number of batch groups:

1. Two batch groups: Each dataset included 300 variables and 40 samples grouped according to two treatments (trt1 and trt2) and two batches (batch1 and batch2).

2. Three batch groups: Each dataset included 300 variables and 36 samples grouped according to two treatments (trt1 and trt2) and three batches (batch1, batch2 and batch3).

In addition, we simulated a ground-truth dataset that only included treatment effects and background noise without batch effects to evaluate batch effect correction methods.

Our simulations generate over-dispersed count data with batch and treatment effects as well as correlation structure among variables, but without any compositional structure. We therefore only applied natural log transformation to the simulated data prior to analysis.

In these simulation scenarios, for PLSDA-batch we set $C-1$ (or $B-1$) components associated with treatment (or batch) effects (where C and B represent the total number

of treatment and batch groups respectively) as $C-1$ ($B-1$) components are likely to explain 100% variance in Y. The number of variables with a true treatment effect ($M^{(trt)}$) is set as the optimal number to select on each treatment component in sPLSDA-batch.

4.2 Simulations (two batch groups)

4.2.1 Balanced batch $\times$ treatment design

The balanced batch $\times$ treatment experimental design included 10 samples from two batches respectively in each treatment group.

Table 2: Balanced batch $\times$ treatment design in the simulation study

	Trt1	Trt2
Batch1	10	10
Batch2	10	10

```
nitr <- 50
N = 40
p_total = 300
p_trt_relevant = 100
p_bat_relevant = 200

# global variance (RDA)
gvar.before <- gvar.clean <-
  gvar.rbe <- gvar.combat <-
  gvar.plsdab <- gvar.splsdab <- data.frame(treatment = NA, batch = NA,
                                            intersection = NA,
                                            residual = NA)

# individual variance (R2)
r2.trt.before <- r2.trt.clean <-
  r2.trt.rbe  <- r2.trt.combat <-
  r2.trt.plsdab <- r2.trt.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                       ncol = nitr))
r2.batch.before <- r2.batch.clean <-
  r2.batch.rbe  <- r2.batch.combat <-
  r2.batch.plsdab <- r2.batch.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                           ncol = nitr))

# precision & recall & F1 (ANOVA)
precision_limma <- recall_limma <- F1_limma <-
  data.frame(before = NA, clean = NA,
```

```
              rbe = NA, combat = NA,
              plsda_batch = NA, splsda_batch = NA,
              sva = NA)

# auc (splsda)
auc_splsda <-
  data.frame(before = NA, clean = NA,
              rbe = NA, combat = NA,
              plsda_batch = NA, splsda_batch = NA)


set.seed(70)
data.cor.res = corStruct(p = 300, zero_prob = 0.7)

for(i in 1: nitr){
  ### initial setup ###
  simulation <- simData_mnegbinom(batch.group = 2,
                                  mean.batch = 7,
                                  sd.batch = 8,
                                  mean.trt = 3,
                                  sd.trt = 2,
                                  mean.bg = 0,
                                  sd.bg = 0.2,
                                  N = 40,
                                  p_total = 300,
                                  p_trt_relevant = 100,
                                  p_bat_relevant = 200,
                                  percentage_overlap_samples = 0.5,
                                  percentage_overlap_variables = 0.5,
                                  data.cor = data.cor.res$data.cor,
                                  disp = 10, prob_zero = 0,
                                  seeds = i)

  set.seed(i)
  raw_count <- simulation$data
  raw_count_clean <- simulation$cleanData

  ## log transformation
  data_log <- log(raw_count + 1)
  data_log_clean <- log(raw_count_clean + 1)

  trt <- simulation$Y.trt
  batch <- simulation$Y.bat

  true.trt <- simulation$true.trt
```

```
true.batch <- simulation$true.batch

Batch_Trt.factors <- data.frame(Batch = batch, Treatment = trt)

### Original ###
X <- data_log

### Clean data ###
X.clean <- data_log_clean

#####
rownames(X) = rownames(X.clean) = names(trt) = names(batch) =
  paste0('sample', 1:N)

colnames(X) = colnames(X.clean) = paste0('otu', 1:p_total)

### Before correction ###
# global variance (RDA)
rda.before = varpart(scale(X), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.before[i,] <- rda.before$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.before <- lmFit(t(scale(X)), design = model.matrix(~ as.factor(trt)))
fit.result.before <- topTable(eBayes(fit.before), coef = 2, number = p_total)
otu.sig.before <-
  rownames(fit.result.before)[fit.result.before$adj.P.Val <= 0.05]

precision_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/
  length(otu.sig.before)
recall_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/length(true.trt)
F1_limma.before <-
  (2*precision_limma.before*recall_limma.before)/
  (precision_limma.before + recall_limma.before)

## replace NA value with 0
if(precision_limma.before == 'NaN'){
  precision_limma.before = 0
}
if(F1_limma.before == 'NaN'){
  F1_limma.before = 0
}
```

```
# individual variance (R2)
indiv.trt.before <- c()
indiv.batch.before <- c()
for(c in seq_len(ncol(X))){
  fit.res1 <- lm(scale(X)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.before <- c(indiv.trt.before, fit.summary1$r.squared)
  indiv.batch.before <- c(indiv.batch.before, fit.summary2$r.squared)
}
r2.trt.before[ ,i] <-  indiv.trt.before
r2.batch.before[ ,i] <-  indiv.batch.before


# auc (sPLSDA)
fit.before_plsda <- splsda(X = X, Y = trt, ncomp = 1)

true.response <- rep(0, p_total)
true.response[true.trt] = 1
before.predictor <- as.numeric(abs(fit.before_plsda$loadings$X))
roc.before_splsda <- roc(true.response, before.predictor, auc = TRUE)
auc.before_splsda <- roc.before_splsda$auc


#############################################################################
### Ground-truth data ###
# global variance (RDA)
rda.clean = varpart(scale(X.clean), ~ Treatment, ~ Batch,
                    data = Batch_Trt.factors)
gvar.clean[i, ] <- rda.clean$part$indfract$Adj.R.squared


# precision & recall & F1 (ANOVA)
fit.clean <- lmFit(t(scale(X.clean)), design = model.matrix(~ as.factor(trt)))
fit.result.clean <- topTable(eBayes(fit.clean), coef = 2, number = p_total)
otu.sig.clean <-
  rownames(fit.result.clean)[fit.result.clean$adj.P.Val <= 0.05]

precision_limma.clean<-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/
  length(otu.sig.clean)
recall_limma.clean<-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/
  length(true.trt)
```

```
F1_limma.clean <-
  (2*precision_limma.clean*recall_limma.clean)/
  (precision_limma.clean + recall_limma.clean)

## replace NA value with 0
if(precision_limma.clean == 'NaN'){
  precision_limma.clean = 0
}
if(F1_limma.clean == 'NaN'){
  F1_limma.clean = 0
}

# individual variance (R2)
indiv.trt.clean <- c()
indiv.batch.clean <- c()
for(c in seq_len(ncol(X.clean))){
  fit.res1 <- lm(scale(X.clean)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.clean)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.clean <- c(indiv.trt.clean, fit.summary1$r.squared)
  indiv.batch.clean <- c(indiv.batch.clean, fit.summary2$r.squared)
}
r2.trt.clean[ ,i] <-  indiv.trt.clean
r2.batch.clean[ ,i] <-  indiv.batch.clean

# auc (sPLSDA)
fit.clean_plsda <- splsda(X = X.clean, Y = trt, ncomp = 1)

clean.predictor <- as.numeric(abs(fit.clean_plsda$loadings$X))
roc.clean_splsda <- roc(true.response, clean.predictor, auc = TRUE)
auc.clean_splsda <- roc.clean_splsda$auc

##############################################################################
### removeBatchEffect corrected data ###
X.rbe <-t(removeBatchEffect(t(X), batch = batch,
                            design = model.matrix(~ as.factor(trt))))

# global variance (RDA)
rda.rbe = varpart(scale(X.rbe), ~ Treatment, ~ Batch,
                  data = Batch_Trt.factors)
gvar.rbe[i, ] <- rda.rbe$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.rbe <- lmFit(t(scale(X.rbe)),
```

```
                  design = model.matrix( ~ as.factor(trt)))
fit.result.rbe <- topTable(eBayes(fit.rbe), coef = 2, number = p_total)
otu.sig.rbe <- rownames(fit.result.rbe)[fit.result.rbe$adj.P.Val <= 0.05]

precision_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(otu.sig.rbe)
recall_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(true.trt)
F1_limma.rbe <- (2*precision_limma.rbe*recall_limma.rbe)/
  (precision_limma.rbe + recall_limma.rbe)

## replace NA value with 0
if(precision_limma.rbe == 'NaN'){
  precision_limma.rbe = 0
}
if(F1_limma.rbe == 'NaN'){
  F1_limma.rbe = 0
}

# individual variance (R2)
indiv.trt.rbe <- c()
indiv.batch.rbe <- c()
for(c in seq_len(ncol(X.rbe))){
  fit.res1 <- lm(scale(X.rbe)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.rbe)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.rbe <- c(indiv.trt.rbe, fit.summary1$r.squared)
  indiv.batch.rbe <- c(indiv.batch.rbe, fit.summary2$r.squared)
}
r2.trt.rbe[ ,i] <-  indiv.trt.rbe
r2.batch.rbe[ ,i] <-  indiv.batch.rbe


# auc (sPLSDA)
fit.rbe_plsda <- splsda(X = X.rbe, Y = trt, ncomp = 1)

rbe.predictor <- as.numeric(abs(fit.rbe_plsda$loadings$X))
roc.rbe_splsda <- roc(true.response, rbe.predictor, auc = TRUE)
auc.rbe_splsda <- roc.rbe_splsda$auc

##############################################################################
### ComBat corrected data ###
X.combat <- t(ComBat(dat = t(X), batch = batch,
                     mod = model.matrix( ~ as.factor(trt))))
```

```
# global variance (RDA)
rda.combat = varpart(scale(X.combat), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.combat[i, ] <- rda.combat$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.combat <- lmFit(t(scale(X.combat)),
                    design = model.matrix( ~ as.factor(trt)))
fit.result.combat <- topTable(eBayes(fit.combat), coef = 2, number = p_total)
otu.sig.combat <- rownames(fit.result.combat)[fit.result.combat$adj.P.Val <=
                                                0.05]

precision_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(otu.sig.combat)
recall_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(true.trt)
F1_limma.combat <- (2*precision_limma.combat*recall_limma.combat)/
  (precision_limma.combat + recall_limma.combat)

## replace NA value with 0
if(precision_limma.combat == 'NaN'){
  precision_limma.combat = 0
}
if(F1_limma.combat == 'NaN'){
  F1_limma.combat = 0
}

# individual variance (R2)
indiv.trt.combat <- c()
indiv.batch.combat <- c()
for(c in seq_len(ncol(X.combat))){
  fit.res1 <- lm(scale(X.combat)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.combat)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.combat <- c(indiv.trt.combat, fit.summary1$r.squared)
  indiv.batch.combat <- c(indiv.batch.combat, fit.summary2$r.squared)
}
r2.trt.combat[ ,i] <-  indiv.trt.combat
r2.batch.combat[ ,i] <-  indiv.batch.combat


# auc (sPLSDA)
```

```
fit.combat_plsda <- splsda(X = X.combat, Y = trt, ncomp = 1)

combat.predictor <- as.numeric(abs(fit.combat_plsda$loadings$X))
roc.combat_splsda <- roc(true.response, combat.predictor, auc = TRUE)
auc.combat_splsda <- roc.combat_splsda$auc


##############################################################################
### PLSDA-batch corrected data ###
X.plsda_batch.correct <- PLSDA_batch(X = X,
                                     Y.trt = trt, Y.bat = batch,
                                     ncomp.trt = 1, ncomp.bat = 1)
X.plsda_batch <- X.plsda_batch.correct$X.nobatch

# global variance (RDA)
rda.plsda_batch = varpart(scale(X.plsda_batch), ~ Treatment, ~ Batch,
                          data = Batch_Trt.factors)
gvar.plsdab[i, ] <- rda.plsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.plsda_batch <- lmFit(t(scale(X.plsda_batch)),
                         design = model.matrix( ~ as.factor(trt)))
fit.result.plsda_batch <- topTable(eBayes(fit.plsda_batch),
                                   coef = 2, number = p_total)
otu.sig.plsda_batch <- rownames(fit.result.plsda_batch)[
  fit.result.plsda_batch$adj.P.Val <= 0.05]

precision_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(otu.sig.plsda_batch)
recall_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(true.trt)
F1_limma.plsda_batch <-
  (2*precision_limma.plsda_batch*recall_limma.plsda_batch)/
  (precision_limma.plsda_batch + recall_limma.plsda_batch)

## replace NA value with 0
if(precision_limma.plsda_batch == 'NaN'){
  precision_limma.plsda_batch = 0
}
if(F1_limma.plsda_batch == 'NaN'){
  F1_limma.plsda_batch = 0
}
```

```
# individual variance (R2)
indiv.trt.plsda_batch <- c()
indiv.batch.plsda_batch <- c()
for(c in seq_len(ncol(X.plsda_batch))){
  fit.res1 <- lm(scale(X.plsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.plsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.plsda_batch <- c(indiv.trt.plsda_batch,
                             fit.summary1$r.squared)
  indiv.batch.plsda_batch <- c(indiv.batch.plsda_batch,
                               fit.summary2$r.squared)
}
r2.trt.plsdab[ ,i] <-  indiv.trt.plsda_batch
r2.batch.plsdab[ ,i] <-  indiv.batch.plsda_batch

# auc (sPLSDA)
fit.plsda_batch_plsda <- splsda(X = X.plsda_batch, Y = trt, ncomp = 1)

plsda_batch.predictor <- as.numeric(abs(fit.plsda_batch_plsda$loadings$X))
roc.plsda_batch_splsda <- roc(true.response,
                              plsda_batch.predictor, auc = TRUE)
auc.plsda_batch_splsda <- roc.plsda_batch_splsda$auc

################################################################################
### sPLSDA-batch corrected data ###
X.splsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt,
                                      Y.bat = batch,
                                      ncomp.trt = 1,
                                      keepX.trt = length(true.trt),
                                      ncomp.bat = 1)
X.splsda_batch <- X.splsda_batch.correct$X.nobatch

# global variance (RDA)
rda.splsda_batch = varpart(scale(X.splsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.splsdab[i, ] <- rda.splsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.splsda_batch <- lmFit(t(scale(X.splsda_batch)),
                          design = model.matrix( ~ as.factor(trt)))
fit.result.splsda_batch <- topTable(eBayes(fit.splsda_batch), coef = 2,
                                    number = p_total)
otu.sig.splsda_batch <- rownames(fit.result.splsda_batch)[
```

```
  fit.result.splsda_batch$adj.P.Val <= 0.05]

precision_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(otu.sig.splsda_batch)
recall_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(true.trt)
F1_limma.splsda_batch <-
  (2*precision_limma.splsda_batch*recall_limma.splsda_batch)/
  (precision_limma.splsda_batch + recall_limma.splsda_batch)

## replace NA value with 0
if(precision_limma.splsda_batch == 'NaN'){
  precision_limma.splsda_batch = 0
}
if(F1_limma.splsda_batch == 'NaN'){
  F1_limma.splsda_batch = 0
}


# individual variance (R2)
indiv.trt.splsda_batch <- c()
indiv.batch.splsda_batch <- c()
for(c in seq_len(ncol(X.splsda_batch))){
  fit.res1 <- lm(scale(X.splsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.splsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.splsda_batch <- c(indiv.trt.splsda_batch,
                              fit.summary1$r.squared)
  indiv.batch.splsda_batch <- c(indiv.batch.splsda_batch,
                                fit.summary2$r.squared)
}
r2.trt.splsdab[ ,i] <-  indiv.trt.splsda_batch
r2.batch.splsdab[ ,i] <-  indiv.batch.splsda_batch

# auc (sPLSDA)
fit.splsda_batch_plsda <- splsda(X = X.splsda_batch, Y = trt, ncomp = 1)

splsda_batch.predictor <- as.numeric(abs(fit.splsda_batch_plsda$loadings$X))
roc.splsda_batch_splsda <- roc(true.response,
                               splsda_batch.predictor, auc = TRUE)
auc.splsda_batch_splsda <- roc.splsda_batch_splsda$auc
```

```
##############################################################################
### SVA ###
X.mod <- model.matrix(~ as.factor(trt))
X.mod0 <- model.matrix(~ 1, data = as.factor(trt))
X.sva.n <- num.sv(dat = t(X), mod = X.mod, method = 'leek')
X.sva <- sva(t(X), X.mod, X.mod0, n.sv = X.sva.n)

X.mod.batch <- cbind(X.mod, X.sva$sv)
X.mod0.batch <- cbind(X.mod0, X.sva$sv)
X.sva.p <- f.pvalue(t(X), X.mod.batch, X.mod0.batch)
X.sva.p.adj <- p.adjust(X.sva.p, method = 'fdr')

otu.sig.sva <- which(X.sva.p.adj <= 0.05)

# precision & recall & F1 (ANOVA)
precision_limma.sva <-
  length(intersect(true.trt, otu.sig.sva))/length(otu.sig.sva)
recall_limma.sva <-
  length(intersect(true.trt, otu.sig.sva))/length(true.trt)
F1_limma.sva <-
  (2*precision_limma.sva*recall_limma.sva)/
  (precision_limma.sva + recall_limma.sva)

## replace NA value with 0
if(precision_limma.sva == 'NaN'){
  precision_limma.sva = 0
}
if(F1_limma.sva == 'NaN'){
  F1_limma.sva = 0
}


# summary
# precision & recall & F1 (ANOVA)
precision_limma[i, ] <- c(`Before correction` = precision_limma.before,
                          `Ground-truth data` = precision_limma.clean,
                          `removeBatchEffect` = precision_limma.rbe,
                          ComBat = precision_limma.combat,
                          `PLSDA-batch` = precision_limma.plsda_batch,
                          `sPLSDA-batch` = precision_limma.splsda_batch,
                          SVA = precision_limma.sva)

recall_limma[i, ] <- c(`Before correction` = recall_limma.before,
                       `Ground-truth data` = recall_limma.clean,
                       `removeBatchEffect` = recall_limma.rbe,
```

```
                        ComBat = recall_limma.combat,
                        `PLSDA-batch` = recall_limma.plsda_batch,
                        `sPLSDA-batch` = recall_limma.splsda_batch,
                        SVA = recall_limma.sva)

  F1_limma[i, ] <- c(`Before correction` = F1_limma.before,
                     `Ground-truth data` = F1_limma.clean,
                     `removeBatchEffect` = F1_limma.rbe,
                     ComBat = F1_limma.combat,
                     `PLSDA-batch` = F1_limma.plsda_batch,
                     `sPLSDA-batch` = F1_limma.splsda_batch,
                     SVA = F1_limma.sva)

  # auc (splsda)
  auc_splsda[i, ] <- c(`Before correction` = auc.before_splsda,
                       `Ground-truth data` = auc.clean_splsda,
                       `removeBatchEffect` = auc.rbe_splsda,
                       ComBat = auc.combat_splsda,
                       `PLSDA-batch` = auc.plsda_batch_splsda,
                       `sPLSDA-batch` = auc.splsda_batch_splsda)

  # print(i)

}
```

4.2.2 Figures

```
# global variance (RDA)
prop.gvar.all <- rbind(`Before correction` = colMeans(gvar.before),
                       `Ground-truth data` = colMeans(gvar.clean),
                       removeBatchEffect = colMeans(gvar.rbe),
                       ComBat = colMeans(gvar.combat),
                       `PLSDA-batch` = colMeans(gvar.plsdab),
                       `sPLSDA-batch` = colMeans(gvar.splsdab))

prop.gvar.all[prop.gvar.all < 0] = 0
prop.gvar.all <- t(apply(prop.gvar.all, 1, function(x){x/sum(x)}))
colnames(prop.gvar.all) <- c('Treatment', 'Intersection', 'Batch', 'Residuals')

partVar_plot(prop.df = prop.gvar.all)
```

Efficient batch effect correction methods should generate data with a null proportion of variance explained by batch effects, and a proportion of variance explained by treatment that is larger compared to the original data, as shown in the figure above original data and ground-truth data.

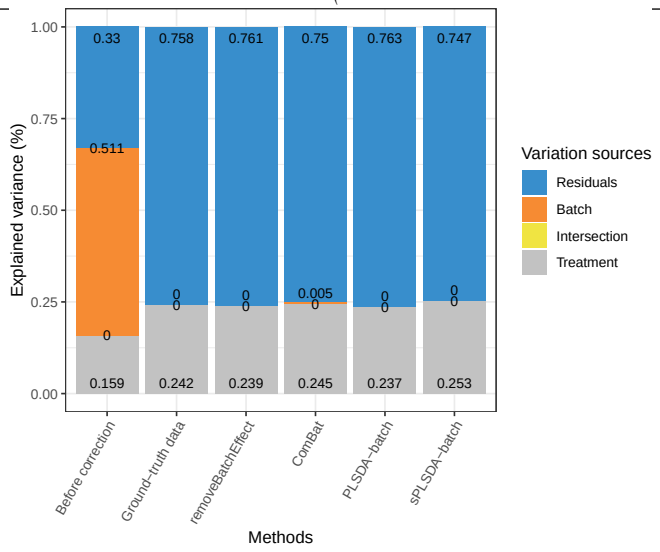

Figure 4.1: Figure 1: Simulation studies (two batch groups): comparison of explained variance before and after batch effect correction for the balanced batch × treatment design.

For a balanced batch × treatment design, we observed no intersection shared between treatment and batch variance, as expected. All methods successfully removed batch variance and preserved (or slightly increased) treatment variance (sPLSDA-batch), with the exception of ComBat where a very small amount of batch variance remained.

```
##############################################################################
# individual variance (R2)
## boxplot
# class
gclass <- c(rep('Treatment only', p_trt_relevant),
            rep('Batch only', (p_total - p_trt_relevant)))
gclass[intersect(true.trt, true.batch)] = 'Treatment & batch'
gclass[setdiff(1:p_total, union(true.trt, true.batch))] = 'No effect'

gclass <- factor(gclass, levels = c('Treatment & batch',
                                    'Treatment only',
                                    'Batch only',
                                    'No effect'))

before.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.before),
                                      rowMeans(r2.batch.before)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                    each = 300)),
```

```
                                  class = rep(gclass,2))
clean.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.clean),
                                     rowMeans(r2.batch.clean)),
                              type = as.factor(rep(c('Treatment','Batch'),
                                               each = 300)),
                              class = rep(gclass,2))
rbe.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.rbe),
                                   rowMeans(r2.batch.rbe)),
                            type = as.factor(rep(c('Treatment','Batch'),
                                             each = 300)),
                            class = rep(gclass,2))
combat.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.combat),
                                      rowMeans(r2.batch.combat)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                each = 300)),
                               class = rep(gclass,2))
plsda_batch.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.plsdab),
                                           rowMeans(r2.batch.plsdab)),
                                    type = as.factor(rep(c('Treatment','Batch'),
                                                     each = 300)),
                                    class = rep(gclass,2))
splsda_batch.r2.df.ggp <-
  data.frame(r2 = c(rowMeans(r2.trt.splsdab),
                    rowMeans(r2.batch.splsdab)),
             type = as.factor(rep(c('Treatment','Batch'),
                                  each = 300)),
             class = rep(gclass,2))

all.r2.df.ggp <- rbind(before.r2.df.ggp, clean.r2.df.ggp,
                       rbe.r2.df.ggp, combat.r2.df.ggp,
                       plsda_batch.r2.df.ggp, splsda_batch.r2.df.ggp)

all.r2.df.ggp$methods <- rep(c('Before correction',
                               'Ground-truth data',
                               'removeBatchEffect',
                               'ComBat',
                               'PLSDA-batch',
                               'sPLSDA-batch'), each = 600)

all.r2.df.ggp$methods <- factor(all.r2.df.ggp$methods,
                                levels = unique(all.r2.df.ggp$methods))

ggplot(all.r2.df.ggp, aes(x = type, y = r2, fill = class)) +
  geom_boxplot(alpha = 0.80) +
  theme_bw() +
```

```
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

Figure 4.2: Figure 2: Simulation studies (two batch groups): R2 values for each microbial variable before and after batch effect correction for the balanced batch × treatment design.

```
##############################################################################
## barplot
# class
before.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.before), gclass, sum),
                    tapply(rowMeans(r2.batch.before), gclass, sum)),
```

```
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

clean.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.clean), gclass, sum),
                    tapply(rowMeans(r2.batch.clean), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

rbe.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.rbe), gclass, sum),
                    tapply(rowMeans(r2.batch.rbe), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

combat.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.combat), gclass, sum),
                    tapply(rowMeans(r2.batch.combat), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

plsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.plsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.plsdab), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

splsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.splsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.splsdab), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))


all.r2.df.bp <- rbind(before.r2.df.bp, clean.r2.df.bp,
                      rbe.r2.df.bp, combat.r2.df.bp,
                      plsda_batch.r2.df.bp, splsda_batch.r2.df.bp)


all.r2.df.bp$methods <- rep(c('Before correction',
                              'Ground-truth data',
                              'removeBatchEffect', 'ComBat',
                              'PLSDA-batch', 'sPLSDA-batch'), each = 8)

all.r2.df.bp$methods <- factor(all.r2.df.bp$methods,
```

```
                                levels = unique(all.r2.df.bp$methods))

ggplot(all.r2.df.bp, aes(x = type, y = r2, fill = class)) +
  geom_bar(stat="identity") +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

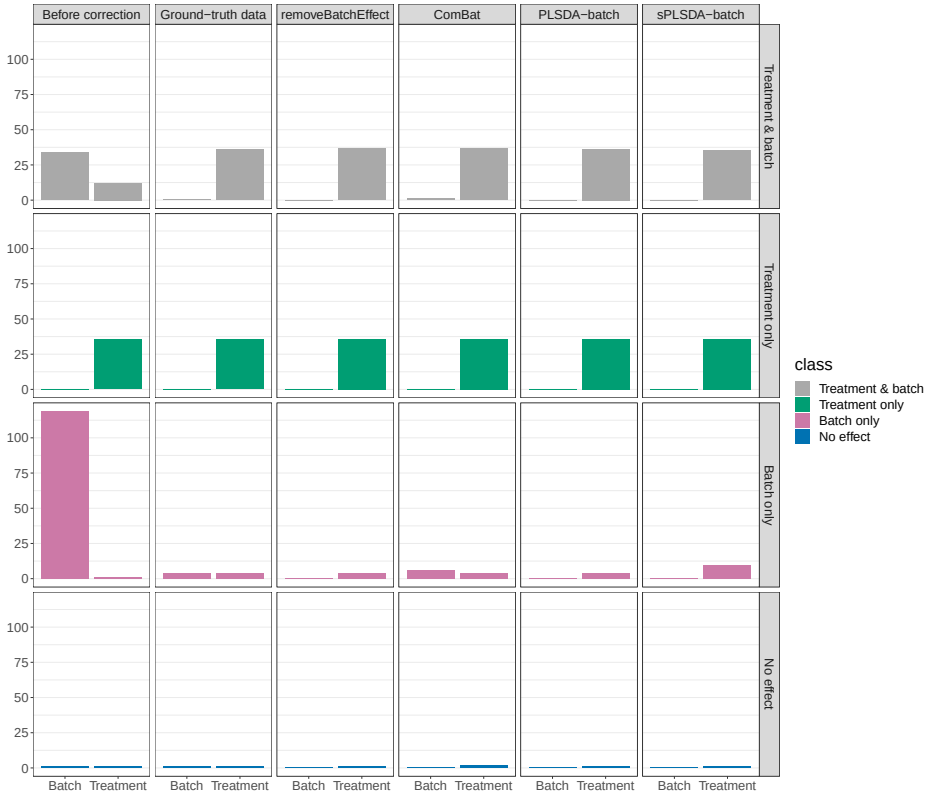

Figure 4.3: Figure 3: Simulation studies (two batch groups): the sum of R2 values for each microbial variable before and after batch effect correction for the balanced batch × treatment design.

We estimated the proportion of variance explained by treatment and batch effects for

Table 4.3: Table 3: Simulation studies (two batch groups): summary of accuracy measurements before and after batch effect correction for the balanced batch × treatment design (mean).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	PLSDA-batch	sPLSDA-batch	S
Precision	0.984	0.952	0.950	0.952	0.952	0.807	0.
Recall	0.674	0.900	0.910	0.911	0.910	0.910	0.
F1	0.799	0.923	0.927	0.929	0.929	0.851	0.
AUC	0.944	0.964	0.968	0.968	0.969	0.954	N

each variable using the R^2 value. removeBatchEffect and PLSDA-batch had the best performance, with results very similar to the ground-truth data. ComBat retained more batch variance of variables with batch effects only, and with both batch and treatment effects, indicating an incomplete removal of batch effects. This result is in agreement with the overall pRDA evaluation described earlier. For sPLSDA-batch, variables with no treatment effect (batch effects only) included a slight amount of (spurious) treatment variance. This was also observed in pRDA evaluation. However, sPLSDA-batch performed as well as PLSDA-batch when the simulated data did not include variables with both batch and treatment effects.

```
# precision & recall & F1 (ANOVA & sPLSDA)
## mean
acc_mean <- rbind(colMeans(precision_limma), colMeans(recall_limma),
                  colMeans(F1_limma), c(colMeans(auc_splsda), sva = NA))
rownames(acc_mean) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_mean) <- c('Before correction', 'Ground-truth data',
                        'removeBatchEffect', 'ComBat',
                        'PLSDA-batch', 'sPLSDA-batch', 'SVA')
acc_mean <- format(acc_mean, digits = 3)
knitr::kable(acc_mean, caption = 'Table 3: Simulation studies (two batch groups): summary of accu
```

The results from the accuracy measures combined with variable selection highlight the importance of removing batch effects as both F1 score and AUC largely improved compared to the original data.

Starting from the original data compared to the ground-truth data, selected variables had a higher precision, lower recall and lower AUC, indicating a smaller number of variables selected with an actual treatment effect. Combined with univariate one-way ANOVA, SVA performed best with the highest, and sometimes greater, accuracy measurements than the ground-truth data. The other methods led to similar performance with the exception of sPLSDA-batch, which selected more false positives than the other methods. PLSDA-batch led to a slightly better AUC than the other methods.

```
## sd
acc_sd <- rbind(apply(precision_limma, 2, sd), apply(recall_limma, 2, sd),
                apply(F1_limma, 2, sd), c(apply(auc_splsda, 2, sd), NA))
```

Table 4.4: Table 4: Simulation studies (two batch groups): summary of accuracy measurements before and after batch effect correction for the balanced batch × treatment design (standard deviation).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	PLSDA-batch	sPLSD
Precision	0.04	0.08	0.09	0.08	0.08	0.11
Recall	0.03	0.03	0.03	0.03	0.03	0.03
F1	0.02	0.05	0.05	0.05	0.05	0.06
AUC	0.02	0.02	0.02	0.02	0.01	0.02

```
rownames(acc_sd) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_sd) <- c('Before correction', 'Ground-truth data',
                      'removeBatchEffect', 'ComBat',
                      'PLSDA-batch', 'sPLSDA-batch', 'SVA')
acc_sd <- format(acc_sd, digits = 1)
knitr::kable(acc_sd, caption = 'Table 4: Simulation studies (two batch groups): summary
```

4.2.3 Unbalanced batch × treatment design

The unbalanced design included 4 and 16 samples from batch1 and batch2 respectively in trt1, 16 and 4 samples from batch1 and batch2 in trt2.

Table 5: Unbalanced batch × treatment design in the simulation study

	Trt1	Trt2
Batch1	4	16
Batch2	16	4

```
nitr <- 50
N = 40
p_total = 300
p_trt_relevant = 100
p_bat_relevant = 200

# global variance (RDA)
gvar.before <- gvar.clean <-
  gvar.rbe <- gvar.combat <-
  gvar.wplsdab <- gvar.swplsdab <-
  gvar.plsdab <- gvar.splsdab <- data.frame(treatment = NA, batch = NA,
                                            intersection = NA,
                                            residual = NA)

# individual variance (R2)
```

```
r2.trt.before <- r2.trt.clean <-
  r2.trt.rbe  <- r2.trt.combat <-
  r2.trt.wplsdab <- r2.trt.swplsdab <-
  r2.trt.plsdab <- r2.trt.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                       ncol = nitr))
r2.batch.before <- r2.batch.clean <-
  r2.batch.rbe  <- r2.batch.combat <-
  r2.batch.wplsdab <- r2.batch.swplsdab <-
  r2.batch.plsdab <- r2.batch.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                           ncol = nitr))

# precision & recall & F1 (ANOVA)
precision_limma <- recall_limma <- F1_limma <-
  data.frame(before = NA, clean = NA,
             rbe = NA, combat = NA,
             wplsda_batch = NA, swplsda_batch = NA,
             sva = NA)

# auc (splsda)
auc_splsda <-
  data.frame(before = NA, clean = NA,
             rbe = NA, combat = NA,
             wplsda_batch = NA, swplsda_batch = NA)


set.seed(70)
data.cor.res = corStruct(p = 300, zero_prob = 0.7)

for(i in 1: nitr){
  ### initial setup ###
  simulation <- simData_mnegbinom(batch.group = 2,
                                  mean.batch = 7,
                                  sd.batch = 8,
                                  mean.trt = 3,
                                  sd.trt = 2,
                                  mean.bg = 0,
                                  sd.bg = 0.2,
                                  N = 40,
                                  p_total = 300,
                                  p_trt_relevant = 100,
                                  p_bat_relevant = 200,
                                  percentage_overlap_samples = 0.2,
                                  percentage_overlap_variables = 0.5,
                                  data.cor = data.cor.res$data.cor,
                                  disp = 10, prob_zero = 0,
```

```
                                    seeds = i)

set.seed(i)
raw_count <- simulation$data
raw_count_clean <- simulation$cleanData

## log transformation
data_log <- log(raw_count + 1)
data_log_clean <- log(raw_count_clean + 1)

trt <- simulation$Y.trt
batch <- simulation$Y.bat

true.trt <- simulation$true.trt
true.batch <- simulation$true.batch

Batch_Trt.factors <- data.frame(Batch = batch, Treatment = trt)

### Original ###
X <- data_log

### Clean data ###
X.clean <- data_log_clean

#####
rownames(X) = rownames(X.clean) = names(trt) = names(batch) =
  paste0('sample', 1:N)

colnames(X) = colnames(X.clean) = paste0('otu', 1:p_total)

### Before correction ###
# global variance (RDA)
rda.before = varpart(scale(X), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.before[i,] <- rda.before$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.before <- lmFit(t(scale(X)), design = model.matrix(~ as.factor(trt)))
fit.result.before <- topTable(eBayes(fit.before), coef = 2, number = p_total)
otu.sig.before <-
  rownames(fit.result.before)[fit.result.before$adj.P.Val <= 0.05]

precision_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/
  length(otu.sig.before)
```

```
recall_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/
  length(true.trt)
F1_limma.before <-
  (2*precision_limma.before*recall_limma.before)/
  (precision_limma.before + recall_limma.before)

## replace NA value with 0
if(precision_limma.before == 'NaN'){
  precision_limma.before = 0
}
if(F1_limma.before == 'NaN'){
  F1_limma.before = 0
}

# individual variance (R2)
indiv.trt.before <- c()
indiv.batch.before <- c()
for(c in seq_len(ncol(X))){
  fit.res1 <- lm(scale(X)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.before <- c(indiv.trt.before, fit.summary1$r.squared)
  indiv.batch.before <- c(indiv.batch.before, fit.summary2$r.squared)
}
r2.trt.before[ ,i] <-  indiv.trt.before
r2.batch.before[ ,i] <-  indiv.batch.before


# auc (sPLSDA)
fit.before_plsda <- splsda(X = X, Y = trt, ncomp = 1)

true.response <- rep(0, p_total)
true.response[true.trt] = 1
before.predictor <- as.numeric(abs(fit.before_plsda$loadings$X))
roc.before_splsda <- roc(true.response, before.predictor, auc = TRUE)
auc.before_splsda <- roc.before_splsda$auc


################################################################################
### Ground-truth data ###
# global variance (RDA)
rda.clean = varpart(scale(X.clean), ~ Treatment, ~ Batch,
                    data = Batch_Trt.factors)
```

```
gvar.clean[i, ] <- rda.clean$part$indfract$Adj.R.squared


# precision & recall & F1 (ANOVA)
fit.clean <- lmFit(t(scale(X.clean)), design = model.matrix(~ as.factor(trt)))
fit.result.clean <- topTable(eBayes(fit.clean), coef = 2, number = p_total)
otu.sig.clean <-
  rownames(fit.result.clean)[fit.result.clean$adj.P.Val <= 0.05]

precision_limma.clean <-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/
  length(otu.sig.clean)
recall_limma.clean <-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/length(true.trt)
F1_limma.clean <-
  (2*precision_limma.clean*recall_limma.clean)/
  (precision_limma.clean + recall_limma.clean)

## replace NA value with 0
if(precision_limma.clean == 'NaN'){
  precision_limma.clean = 0
}
if(F1_limma.clean == 'NaN'){
  F1_limma.clean = 0
}

# individual variance (R2)
indiv.trt.clean <- c()
indiv.batch.clean <- c()
for(c in seq_len(ncol(X.clean))){
  fit.res1 <- lm(scale(X.clean)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.clean)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.clean <- c(indiv.trt.clean, fit.summary1$r.squared)
  indiv.batch.clean <- c(indiv.batch.clean, fit.summary2$r.squared)
}
r2.trt.clean[ ,i] <-  indiv.trt.clean
r2.batch.clean[ ,i] <-  indiv.batch.clean

# auc (sPLSDA)
fit.clean_plsda <- splsda(X = X.clean, Y = trt, ncomp = 1)

clean.predictor <- as.numeric(abs(fit.clean_plsda$loadings$X))
roc.clean_splsda <- roc(true.response, clean.predictor, auc = TRUE)
```

```
auc.clean_splsda <- roc.clean_splsda$auc

##############################################################################
### removeBatchEffect corrected data ###
X.rbe <-t(removeBatchEffect(t(X), batch = batch,
                            design = model.matrix(~ as.factor(trt))))

# global variance (RDA)
rda.rbe = varpart(scale(X.rbe), ~ Treatment, ~ Batch,
                  data = Batch_Trt.factors)
gvar.rbe[i, ] <- rda.rbe$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.rbe <- lmFit(t(scale(X.rbe)),
                 design = model.matrix( ~ as.factor(trt)))
fit.result.rbe <- topTable(eBayes(fit.rbe), coef = 2, number = p_total)
otu.sig.rbe <- rownames(fit.result.rbe)[fit.result.rbe$adj.P.Val <= 0.05]

precision_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(otu.sig.rbe)
recall_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(true.trt)
F1_limma.rbe <- (2*precision_limma.rbe*recall_limma.rbe)/
  (precision_limma.rbe + recall_limma.rbe)

## replace NA value with 0
if(precision_limma.rbe == 'NaN'){
  precision_limma.rbe = 0
}
if(F1_limma.rbe == 'NaN'){
  F1_limma.rbe = 0
}

# individual variance (R2)
indiv.trt.rbe <- c()
indiv.batch.rbe <- c()
for(c in seq_len(ncol(X.rbe))){
  fit.res1 <- lm(scale(X.rbe)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.rbe)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.rbe <- c(indiv.trt.rbe, fit.summary1$r.squared)
  indiv.batch.rbe <- c(indiv.batch.rbe, fit.summary2$r.squared)
}
r2.trt.rbe[ ,i] <-  indiv.trt.rbe
```

```
r2.batch.rbe[ ,i] <-  indiv.batch.rbe


# auc (sPLSDA)
fit.rbe_plsda <- splsda(X = X.rbe, Y = trt, ncomp = 1)

rbe.predictor <- as.numeric(abs(fit.rbe_plsda$loadings$X))
roc.rbe_splsda <- roc(true.response, rbe.predictor, auc = TRUE)
auc.rbe_splsda <- roc.rbe_splsda$auc

################################################################################
### ComBat corrected data ###
X.combat <- t(ComBat(dat = t(X), batch = batch,
                     mod = model.matrix( ~ as.factor(trt))))

# global variance (RDA)
rda.combat = varpart(scale(X.combat), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.combat[i, ] <- rda.combat$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.combat <- lmFit(t(scale(X.combat)),
                    design = model.matrix( ~ as.factor(trt)))
fit.result.combat <- topTable(eBayes(fit.combat), coef = 2, number = p_total)
otu.sig.combat <-
  rownames(fit.result.combat)[fit.result.combat$adj.P.Val <= 0.05]

precision_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(otu.sig.combat)
recall_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(true.trt)
F1_limma.combat <- (2*precision_limma.combat*recall_limma.combat)/
  (precision_limma.combat + recall_limma.combat)

## replace NA value with 0
if(precision_limma.combat == 'NaN'){
  precision_limma.combat = 0
}
if(F1_limma.combat == 'NaN'){
  F1_limma.combat = 0
}

# individual variance (R2)
```

```
indiv.trt.combat <- c()
indiv.batch.combat <- c()
for(c in seq_len(ncol(X.combat))){
  fit.res1 <- lm(scale(X.combat)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.combat)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.combat <- c(indiv.trt.combat, fit.summary1$r.squared)
  indiv.batch.combat <- c(indiv.batch.combat, fit.summary2$r.squared)
}
r2.trt.combat[ ,i] <-  indiv.trt.combat
r2.batch.combat[ ,i] <-  indiv.batch.combat


# auc (sPLSDA)
fit.combat_plsda <- splsda(X = X.combat, Y = trt, ncomp = 1)

combat.predictor <- as.numeric(abs(fit.combat_plsda$loadings$X))
roc.combat_splsda <- roc(true.response, combat.predictor, auc = TRUE)
auc.combat_splsda <- roc.combat_splsda$auc


##############################################################################
### wPLSDA-batch corrected data ###
X.wplsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt, Y.bat = batch,
                                      ncomp.trt = 1, ncomp.bat = 1,
                                      balance = FALSE)
X.wplsda_batch <- X.wplsda_batch.correct$X.nobatch

# global variance (RDA)
rda.wplsda_batch = varpart(scale(X.wplsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.wplsdab[i, ] <- rda.wplsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.wplsda_batch <- lmFit(t(scale(X.wplsda_batch)),
                          design = model.matrix( ~ as.factor(trt)))
fit.result.wplsda_batch <- topTable(eBayes(fit.wplsda_batch),
                                    coef = 2, number = p_total)
otu.sig.wplsda_batch <- rownames(fit.result.wplsda_batch)[
  fit.result.wplsda_batch$adj.P.Val <= 0.05]

precision_limma.wplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.wplsda_batch))/
```

```
  length(otu.sig.wplsda_batch)
recall_limma.wplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.wplsda_batch))/
  length(true.trt)
F1_limma.wplsda_batch <-
  (2*precision_limma.wplsda_batch*recall_limma.wplsda_batch)/
  (precision_limma.wplsda_batch + recall_limma.wplsda_batch)

## replace NA value with 0
if(precision_limma.wplsda_batch == 'NaN'){
  precision_limma.wplsda_batch = 0
}
if(F1_limma.wplsda_batch == 'NaN'){
  F1_limma.wplsda_batch = 0
}

# individual variance (R2)
indiv.trt.wplsda_batch <- c()
indiv.batch.wplsda_batch <- c()
for(c in seq_len(ncol(X.wplsda_batch))){
  fit.res1 <- lm(scale(X.wplsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.wplsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.wplsda_batch <- c(indiv.trt.wplsda_batch,
                              fit.summary1$r.squared)
  indiv.batch.wplsda_batch <- c(indiv.batch.wplsda_batch,
                                fit.summary2$r.squared)
}
r2.trt.wplsdab[ ,i] <-  indiv.trt.wplsda_batch
r2.batch.wplsdab[ ,i] <-  indiv.batch.wplsda_batch

# auc (sPLSDA)
fit.wplsda_batch_plsda <- splsda(X = X.wplsda_batch, Y = trt, ncomp = 1)

wplsda_batch.predictor <- as.numeric(abs(fit.wplsda_batch_plsda$loadings$X))
roc.wplsda_batch_splsda <- roc(true.response,
                               wplsda_batch.predictor, auc = TRUE)
auc.wplsda_batch_splsda <- roc.wplsda_batch_splsda$auc

##############################################################################
### sPLSDA-batch corrected data ###
X.swplsda_batch.correct <- PLSDA_batch(X = X,
                                       Y.trt = trt,
                                       Y.bat = batch,
```

```
                                        ncomp.trt = 1,
                                        keepX.trt = length(true.trt),
                                        ncomp.bat = 1,
                                        balance = FALSE)
X.swplsda_batch <- X.swplsda_batch.correct$X.nobatch

# global variance (RDA)
rda.swplsda_batch = varpart(scale(X.swplsda_batch), ~ Treatment, ~ Batch,
                            data = Batch_Trt.factors)
gvar.swplsdab[i, ] <- rda.swplsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.swplsda_batch <- lmFit(t(scale(X.swplsda_batch)),
                           design = model.matrix( ~ as.factor(trt)))
fit.result.swplsda_batch <- topTable(eBayes(fit.swplsda_batch), coef = 2,
                                     number = p_total)
otu.sig.swplsda_batch <- rownames(fit.result.swplsda_batch)[
  fit.result.swplsda_batch$adj.P.Val <= 0.05]

precision_limma.swplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.swplsda_batch))/
  length(otu.sig.swplsda_batch)
recall_limma.swplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.swplsda_batch))/
  length(true.trt)
F1_limma.swplsda_batch <-
  (2*precision_limma.swplsda_batch*recall_limma.swplsda_batch)/
  (precision_limma.swplsda_batch + recall_limma.swplsda_batch)

## replace NA value with 0
if(precision_limma.swplsda_batch == 'NaN'){
  precision_limma.swplsda_batch = 0
}
if(F1_limma.swplsda_batch == 'NaN'){
  F1_limma.swplsda_batch = 0
}


# individual variance (R2)
indiv.trt.swplsda_batch <- c()
indiv.batch.swplsda_batch <- c()
for(c in seq_len(ncol(X.swplsda_batch))){
  fit.res1 <- lm(scale(X.swplsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.swplsda_batch)[ ,c] ~ batch)
```

```
  fit.summary2 <- summary(fit.res2)
  indiv.trt.swplsda_batch <- c(indiv.trt.swplsda_batch,
                               fit.summary1$r.squared)
  indiv.batch.swplsda_batch <- c(indiv.batch.swplsda_batch,
                                 fit.summary2$r.squared)
}
r2.trt.swplsdab[ ,i] <-  indiv.trt.swplsda_batch
r2.batch.swplsdab[ ,i] <-  indiv.batch.swplsda_batch

# auc (sPLSDA)
fit.swplsda_batch_plsda <- splsda(X = X.swplsda_batch, Y = trt, ncomp = 1)

swplsda_batch.predictor <- as.numeric(abs(fit.swplsda_batch_plsda$loadings$X))
roc.swplsda_batch_splsda <- roc(true.response,
                                swplsda_batch.predictor, auc = TRUE)
auc.swplsda_batch_splsda <- roc.swplsda_batch_splsda$auc

##############################################################################
### PLSDA-batch corrected data ###
X.plsda_batch.correct <- PLSDA_batch(X = X,
                                     Y.trt = trt, Y.bat = batch,
                                     ncomp.trt = 1, ncomp.bat = 1)
X.plsda_batch <- X.plsda_batch.correct$X.nobatch

# global variance (RDA)
rda.plsda_batch = varpart(scale(X.plsda_batch), ~ Treatment, ~ Batch,
                          data = Batch_Trt.factors)
gvar.plsdab[i, ] <- rda.plsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.plsda_batch <- lmFit(t(scale(X.plsda_batch)),
                         design = model.matrix( ~ as.factor(trt)))
fit.result.plsda_batch <- topTable(eBayes(fit.plsda_batch),
                                   coef = 2, number = p_total)
otu.sig.plsda_batch <- rownames(fit.result.plsda_batch)[
  fit.result.plsda_batch$adj.P.Val <= 0.05]

precision_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(otu.sig.plsda_batch)
recall_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(true.trt)
F1_limma.plsda_batch <-
  (2*precision_limma.plsda_batch*recall_limma.plsda_batch)/
```

```
  (precision_limma.plsda_batch + recall_limma.plsda_batch)

## replace NA value with 0
if(precision_limma.plsda_batch == 'NaN'){
  precision_limma.plsda_batch = 0
}
if(F1_limma.plsda_batch == 'NaN'){
  F1_limma.plsda_batch = 0
}

# individual variance (R2)
indiv.trt.plsda_batch <- c()
indiv.batch.plsda_batch <- c()
for(c in seq_len(ncol(X.plsda_batch))){
  fit.res1 <- lm(scale(X.plsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.plsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.plsda_batch <- c(indiv.trt.plsda_batch,
                             fit.summary1$r.squared)
  indiv.batch.plsda_batch <- c(indiv.batch.plsda_batch,
                               fit.summary2$r.squared)
}
r2.trt.plsdab[ ,i] <-  indiv.trt.plsda_batch
r2.batch.plsdab[ ,i] <-  indiv.batch.plsda_batch

# auc (sPLSDA)
fit.plsda_batch_plsda <- splsda(X = X.plsda_batch, Y = trt, ncomp = 1)

plsda_batch.predictor <- as.numeric(abs(fit.plsda_batch_plsda$loadings$X))
roc.plsda_batch_splsda <- roc(true.response,
                              plsda_batch.predictor, auc = TRUE)
auc.plsda_batch_splsda <- roc.plsda_batch_splsda$auc

##############################################################################
### sPLSDA-batch corrected data ###
X.splsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt,
                                      Y.bat = batch,
                                      ncomp.trt = 1,
                                      keepX.trt = length(true.trt),
                                      ncomp.bat = 1)
X.splsda_batch <- X.splsda_batch.correct$X.nobatch

# global variance (RDA)
```

```
rda.splsda_batch = varpart(scale(X.splsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.splsdab[i, ] <- rda.splsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.splsda_batch <- lmFit(t(scale(X.splsda_batch)),
                          design = model.matrix( ~ as.factor(trt)))
fit.result.splsda_batch <- topTable(eBayes(fit.splsda_batch), coef = 2,
                                    number = p_total)
otu.sig.splsda_batch <- rownames(fit.result.splsda_batch)[
  fit.result.splsda_batch$adj.P.Val <= 0.05]

precision_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(otu.sig.splsda_batch)
recall_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(true.trt)
F1_limma.splsda_batch <-
  (2*precision_limma.splsda_batch*recall_limma.splsda_batch)/
  (precision_limma.splsda_batch + recall_limma.splsda_batch)

## replace NA value with 0
if(precision_limma.splsda_batch == 'NaN'){
  precision_limma.splsda_batch = 0
}
if(F1_limma.splsda_batch == 'NaN'){
  F1_limma.splsda_batch = 0
}


# individual variance (R2)
indiv.trt.splsda_batch <- c()
indiv.batch.splsda_batch <- c()
for(c in seq_len(ncol(X.splsda_batch))){
  fit.res1 <- lm(scale(X.splsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.splsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.splsda_batch <- c(indiv.trt.splsda_batch,
                              fit.summary1$r.squared)
  indiv.batch.splsda_batch <- c(indiv.batch.splsda_batch,
                                fit.summary2$r.squared)
}
r2.trt.splsdab[ ,i] <-  indiv.trt.splsda_batch
```

```
r2.batch.splsdab[ ,i] <-  indiv.batch.splsda_batch

# auc (sPLSDA)
fit.splsda_batch_plsda <- splsda(X = X.splsda_batch, Y = trt, ncomp = 1)

splsda_batch.predictor <- as.numeric(abs(fit.splsda_batch_plsda$loadings$X))
roc.splsda_batch_splsda <- roc(true.response,
                               splsda_batch.predictor, auc = TRUE)
auc.splsda_batch_splsda <- roc.splsda_batch_splsda$auc

##############################################################################
### SVA ###
X.mod <- model.matrix(~ as.factor(trt))
X.mod0 <- model.matrix(~ 1, data = as.factor(trt))
X.sva.n <- num.sv(dat = t(X), mod = X.mod, method = 'leek')
X.sva <- sva(t(X), X.mod, X.mod0, n.sv = X.sva.n)

X.mod.batch <- cbind(X.mod, X.sva$sv)
X.mod0.batch <- cbind(X.mod0, X.sva$sv)
X.sva.p <- f.pvalue(t(X), X.mod.batch, X.mod0.batch)
X.sva.p.adj <- p.adjust(X.sva.p, method = 'fdr')

otu.sig.sva <- which(X.sva.p.adj <= 0.05)

# precision & recall & F1 (ANOVA)
precision_limma.sva <-
  length(intersect(true.trt, otu.sig.sva))/length(otu.sig.sva)
recall_limma.sva <-
  length(intersect(true.trt, otu.sig.sva))/length(true.trt)
F1_limma.sva <-
  (2*precision_limma.sva*recall_limma.sva)/
  (precision_limma.sva + recall_limma.sva)

## replace NA value with 0
if(precision_limma.sva == 'NaN'){
  precision_limma.sva = 0
}
if(F1_limma.sva == 'NaN'){
  F1_limma.sva = 0
}


# summary
# precision & recall & F1 (ANOVA)
precision_limma[i, ] <- c(`Before correction` = precision_limma.before,
```

```
                            `Ground-truth data` = precision_limma.clean,
                            `removeBatchEffect` = precision_limma.rbe,
                            ComBat = precision_limma.combat,
                            `wPLSDA-batch` = precision_limma.wplsda_batch,
                            `swPLSDA-batch` = precision_limma.swplsda_batch,
                            SVA = precision_limma.sva)

  recall_limma[i, ] <- c(`Before correction` = recall_limma.before,
                         `Ground-truth data` = recall_limma.clean,
                         `removeBatchEffect` = recall_limma.rbe,
                         ComBat = recall_limma.combat,
                         `wPLSDA-batch` = recall_limma.wplsda_batch,
                         `swPLSDA-batch` = recall_limma.swplsda_batch,
                         SVA = recall_limma.sva)

  F1_limma[i, ] <- c(`Before correction` = F1_limma.before,
                     `Ground-truth data` = F1_limma.clean,
                     `removeBatchEffect` = F1_limma.rbe,
                     ComBat = F1_limma.combat,
                     `wPLSDA-batch` = F1_limma.wplsda_batch,
                     `swPLSDA-batch` = F1_limma.swplsda_batch,
                     SVA = F1_limma.sva)

  # auc (splsda)
  auc_splsda[i, ] <- c(`Before correction` = auc.before_splsda,
                       `Ground-truth data` = auc.clean_splsda,
                       `removeBatchEffect` = auc.rbe_splsda,
                       ComBat = auc.combat_splsda,
                       `wPLSDA-batch` = auc.wplsda_batch_splsda,
                       `swPLSDA-batch` = auc.swplsda_batch_splsda)

  #print(i)

}
```

4.2.4 Figures

```
# global variance (RDA)
prop.gvar.all <- rbind(`Before correction` = colMeans(gvar.before),
                       `Ground-truth data` = colMeans(gvar.clean),
                       removeBatchEffect = colMeans(gvar.rbe),
                       ComBat = colMeans(gvar.combat),
                       `wPLSDA-batch` = colMeans(gvar.wplsdab),
                       `swPLSDA-batch` = colMeans(gvar.swplsdab),
```

```
                      `PLSDA-batch` = colMeans(gvar.plsdab),
                      `sPLSDA-batch` = colMeans(gvar.splsdab))

prop.gvar.all[prop.gvar.all < 0] = 0
prop.gvar.all <- t(apply(prop.gvar.all, 1, function(x){x/sum(x)}))
colnames(prop.gvar.all) <- c('Treatment', 'Intersection', 'Batch', 'Residuals')

partVar_plot(prop.df = prop.gvar.all)
```

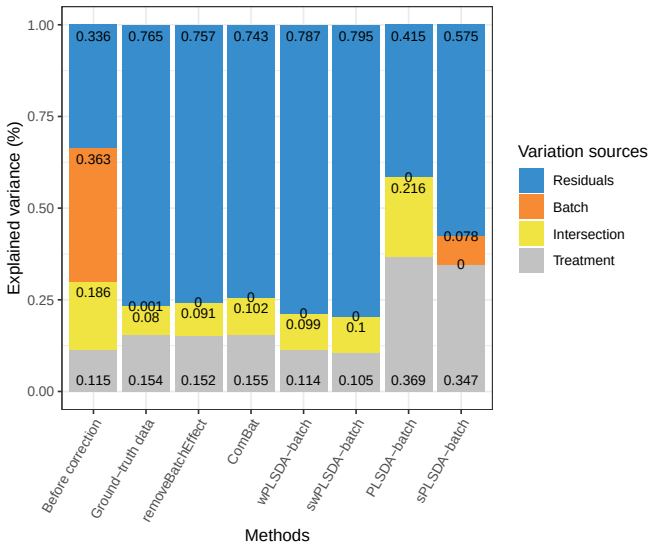

Figure 4.4: Figure 4: Simulation studies (two batch groups): comparison of explained variance before and after batch effect correction for the unbalanced batch × treatment design.

For a strong unbalanced batch × treatment design, we observed the presence of intersectional variance explained by both batch and treatment effects, as expected. This source of variance is also present in the ground-truth data but should be smaller compared to the uncorrected data. Both unweighted PLSDA-batch and sPLSDA-batch performed poorly for such design - for PLSDA-batch the intersectional variance increased while for sPLSDA-batch the batch variance was not entirely removed. The other methods were successful in removing batch variance. removeBatchEffect and ComBat explained a proportion of variance by treatment similar to the ground-truth data, while wPLSDA-batch and swPLSDA-batch explained slightly less treatment variance.

```
##############################################################################
# individual variance (R2)
## boxplot
# class
gclass <- c(rep('Treatment only', p_trt_relevant),
```

```
           rep('Batch only', (p_total - p_trt_relevant)))
gclass[intersect(true.trt, true.batch)] = 'Treatment & batch'
gclass[setdiff(1:p_total, union(true.trt, true.batch))] = 'No effect'

gclass <- factor(gclass, levels = c('Treatment & batch',
                                    'Treatment only',
                                    'Batch only',
                                    'No effect'))

before.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.before),
                                      rowMeans(r2.batch.before)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                    each = 300)),
                               class = rep(gclass,2))
clean.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.clean),
                                     rowMeans(r2.batch.clean)),
                              type = as.factor(rep(c('Treatment','Batch'),
                                                   each = 300)),
                              class = rep(gclass,2))
rbe.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.rbe),
                                   rowMeans(r2.batch.rbe)),
                            type = as.factor(rep(c('Treatment','Batch'),
                                                 each = 300)),
                            class = rep(gclass,2))
combat.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.combat),
                                      rowMeans(r2.batch.combat)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                    each = 300)),
                               class = rep(gclass,2))
wplsda_batch.r2.df.ggp <-
  data.frame(r2 = c(rowMeans(r2.trt.wplsdab),
                    rowMeans(r2.batch.wplsdab)),
             type = as.factor(rep(c('Treatment','Batch'),
                                  each = 300)),
             class = rep(gclass,2))
swplsda_batch.r2.df.ggp <-
  data.frame(r2 = c(rowMeans(r2.trt.swplsdab),
                    rowMeans(r2.batch.swplsdab)),
             type = as.factor(rep(c('Treatment','Batch'),
                                  each = 300)),
             class = rep(gclass,2))

all.r2.df.ggp <- rbind(before.r2.df.ggp, clean.r2.df.ggp,
                       rbe.r2.df.ggp, combat.r2.df.ggp,
                       wplsda_batch.r2.df.ggp, swplsda_batch.r2.df.ggp)
```

```
all.r2.df.ggp$methods <- rep(c('Before correction',
                               'Ground-truth data',
                               'removeBatchEffect',
                               'ComBat',
                               'wPLSDA-batch',
                               'swPLSDA-batch'), each = 600)

all.r2.df.ggp$methods <- factor(all.r2.df.ggp$methods,
                                levels = unique(all.r2.df.ggp$methods))

ggplot(all.r2.df.ggp, aes(x = type, y = r2, fill = class)) +
  geom_boxplot(alpha = 0.80) +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

```
##############################################################################
## barplot
# class
before.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.before), gclass, sum),
                    tapply(rowMeans(r2.batch.before), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

clean.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.clean), gclass, sum),
                    tapply(rowMeans(r2.batch.clean), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

rbe.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.rbe), gclass, sum),
                    tapply(rowMeans(r2.batch.rbe), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

combat.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.combat), gclass, sum),
```

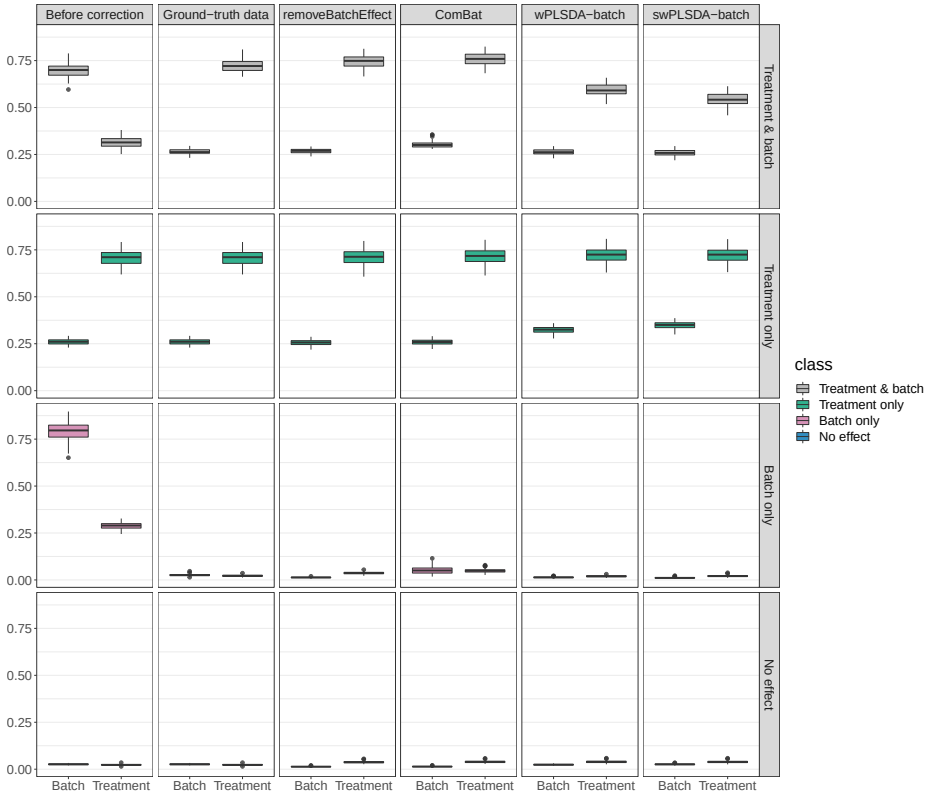

Figure 4.5: Figure 5: Simulation studies (two batch groups): R2 values for each microbial variable before and after batch effect correction for the unbalanced batch × treatment design.

```
                    tapply(rowMeans(r2.batch.combat), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

wplsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.wplsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.wplsdab), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

swplsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.swplsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.swplsdab), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))


all.r2.df.bp <- rbind(before.r2.df.bp, clean.r2.df.bp,
                      rbe.r2.df.bp, combat.r2.df.bp,
                      wplsda_batch.r2.df.bp, swplsda_batch.r2.df.bp)


all.r2.df.bp$methods <- rep(c('Before correction',
                              'Ground-truth data',
                              'removeBatchEffect',
                              'ComBat',
                              'wPLSDA-batch',
                              'swPLSDA-batch'), each = 8)

all.r2.df.bp$methods <- factor(all.r2.df.bp$methods,
                               levels = unique(all.r2.df.bp$methods))

ggplot(all.r2.df.bp, aes(x = type, y = r2, fill = class)) +
  geom_bar(stat="identity") +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

We observed similar performance for removeBatchEffect and ComBat for the unbal-

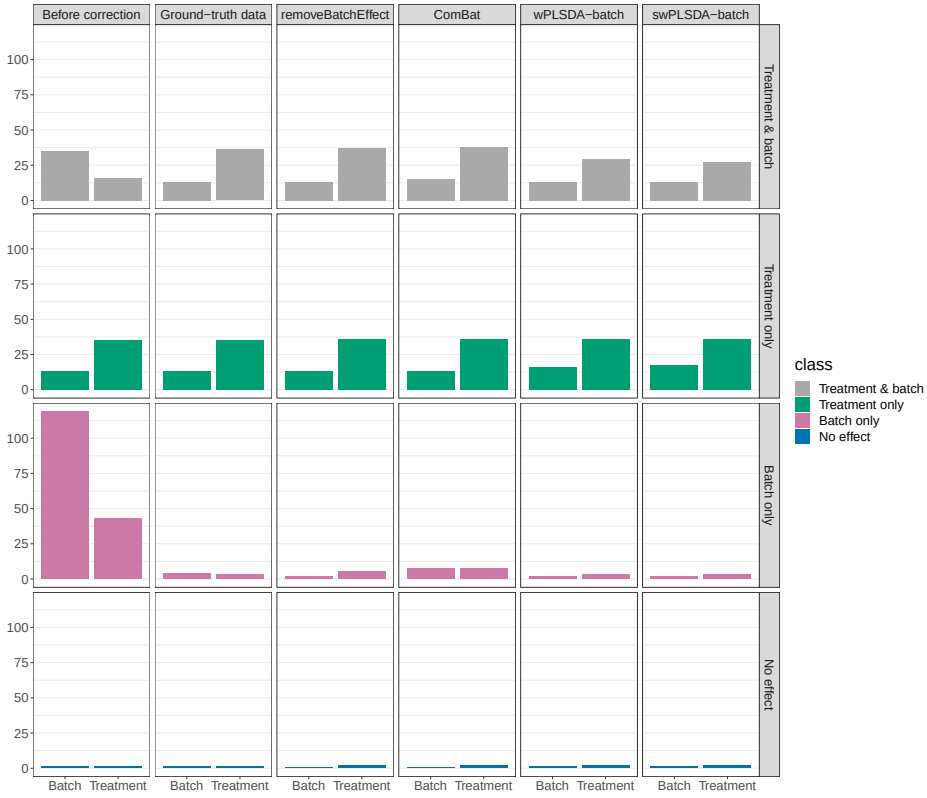

Figure 4.6: Figure 6: Simulation studies (two batch groups): the sum of R2 values for each microbial variable before and after batch effect correction for the unbalanced batch × treatment design.

Table 4.6: Table 6 Simulation studies (two batch groups): summary of accuracy measurements before and after batch effect correction for the unbalanced batch × treatment design (mean).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	wPLSDA-batch	swPLSDA-batch
Precision	0.385	0.973	0.901	0.834	0.943	0.943
Recall	0.825	0.895	0.910	0.919	0.888	0.862
F1	0.525	0.932	0.903	0.873	0.914	0.900
AUC	0.704	0.967	0.963	0.962	0.965	0.954

anced design compared to the balanced design. With wPLSDA-batch and swPLSDA-batch, variables with both treatment and batch effects explained less treatment variance after correction, compared to the ground-truth data. However, for the other variables, wPLSDA-batch and its sparse version performed as similar as the ground-truth data.

```
# precision & recall & F1 (ANOVA & sPLSDA)
## mean
acc_mean <- rbind(colMeans(precision_limma), colMeans(recall_limma),
                  colMeans(F1_limma), c(colMeans(auc_splsda), sva = NA))
rownames(acc_mean) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_mean) <- c('Before correction', 'Ground-truth data',
                        'removeBatchEffect', 'ComBat',
                        'wPLSDA-batch', 'swPLSDA-batch', 'SVA')
acc_mean <- format(acc_mean, digits = 3)
knitr::kable(acc_mean, caption = 'Table 6 Simulation studies (two batch groups): summary of accur
```

In the unbalanced design, the precision of SVA is low and very similar to the original data, indicating that the performance of SVA heavily depends on the experimental design and is likely to overfit. This may explain the somewhat inflated results of SVA in the balanced design case. wPLSDA-batch performed best with results close to those from the ground-truth data.

```
## sd
acc_sd <- rbind(apply(precision_limma, 2, sd), apply(recall_limma, 2, sd),
                apply(F1_limma, 2, sd), c(apply(auc_splsda, 2, sd), NA))
rownames(acc_sd) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_sd) <- c('Before correction', 'Ground-truth data',
                      'removeBatchEffect', 'ComBat',
                      'wPLSDA-batch', 'swPLSDA-batch', 'SVA')
acc_sd <- format(acc_sd, digits = 1)
knitr::kable(acc_sd, caption = 'Table 7 Simulation studies (two batch groups): summary of accurac
```

Table 4.7: Table 7 Simulation studies (two batch groups): summary of accuracy measurements before and after batch effect correction for the unbalanced batch × treatment design (standard deviation).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	wPLSDA-batch	swPl
Precision	0.01	0.05	0.09	0.08	0.05	0.05
Recall	0.03	0.03	0.03	0.03	0.03	0.03
F1	0.01	0.03	0.05	0.05	0.03	0.03
AUC	0.06	0.02	0.02	0.01	0.01	0.02

4.3 Simulations (three batch groups)

4.3.1 Balanced batch × treatment design

The balanced batch × treatment experimental design included 6 samples from three batches respectively in each treatment group.

Table 8: Balanced batch × treatment design in the simulation study

	Trt1	Trt2
Batch1	6	6
Batch2	6	6
Batch3	6	6

```
nitr <- 50
N = 36
p_total = 300
p_trt_relevant = 100
p_bat_relevant = 200

# global variance (RDA)
gvar.before <- gvar.clean <-
  gvar.rbe <- gvar.combat <-
  gvar.plsdab <- gvar.splsdab <- data.frame(treatment = NA, batch = NA,
                                            intersection = NA,
                                            residual = NA)

# individual variance (R2)
r2.trt.before <- r2.trt.clean <-
  r2.trt.rbe  <- r2.trt.combat <-
  r2.trt.plsdab <- r2.trt.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                       ncol = nitr))
r2.batch.before <- r2.batch.clean <-
  r2.batch.rbe  <- r2.batch.combat <-
```

```
  r2.batch.plsdab <- r2.batch.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                           ncol = nitr))

# precision & recall & F1 (ANOVA)
precision_limma <- recall_limma <- F1_limma <-
  data.frame(before = NA, clean = NA,
             rbe = NA, combat = NA,
             plsda_batch = NA, splsda_batch = NA,
             sva = NA)

# auc (splsda)
auc_splsda <-
  data.frame(before = NA, clean = NA,
             rbe = NA, combat = NA,
             plsda_batch = NA, splsda_batch = NA)


set.seed(70)
data.cor.res = corStruct(p = 300, zero_prob = 0.7)

for(i in 1: nitr){
  ### initial setup ###
  simulation <- simData_mnegbinom(batch.group = 3,
                                  mean.batch = 7,
                                  sd.batch = 8,
                                  mean.trt = 3,
                                  sd.trt = 2,
                                  mean.bg = 0,
                                  sd.bg = 0.2,
                                  N = 36,
                                  p_total = 300,
                                  p_trt_relevant = 100,
                                  p_bat_relevant = 200,
                                  percentage_overlap_samples = 0.5,
                                  percentage_overlap_variables = 0.5,
                                  data.cor = data.cor.res$data.cor,
                                  disp = 10, prob_zero = 0,
                                  seeds = i)

  set.seed(i)
  raw_count <- simulation$data
  raw_count_clean <- simulation$cleanData

  ## log transformation
  data_log <- log(raw_count + 1)
```

```
data_log_clean <- log(raw_count_clean + 1)

trt <- simulation$Y.trt
batch <- simulation$Y.bat

true.trt <- simulation$true.trt
true.batch <- simulation$true.batch

Batch_Trt.factors <- data.frame(Batch = batch, Treatment = trt)

### Original ###
X <- data_log

### Clean data ###
X.clean <- data_log_clean

#####
rownames(X) = rownames(X.clean) = names(trt) = names(batch) =
  paste0('sample', 1:N)

colnames(X) = colnames(X.clean) = paste0('otu', 1:p_total)

### Before correction ###
# global variance (RDA)
rda.before = varpart(scale(X), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.before[i,] <- rda.before$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.before <- lmFit(t(scale(X)), design = model.matrix(~ as.factor(trt)))
fit.result.before <- topTable(eBayes(fit.before), coef = 2, number = p_total)
otu.sig.before <-
  rownames(fit.result.before)[fit.result.before$adj.P.Val <= 0.05]

precision_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/
  length(otu.sig.before)
recall_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/length(true.trt)
F1_limma.before <-
  (2*precision_limma.before*recall_limma.before)/
  (precision_limma.before + recall_limma.before)

## replace NA value with 0
if(precision_limma.before == 'NaN'){
```

```
  precision_limma.before = 0
}
if(F1_limma.before == 'NaN'){
  F1_limma.before = 0
}

# individual variance (R2)
indiv.trt.before <- c()
indiv.batch.before <- c()
for(c in seq_len(ncol(X))){
  fit.res1 <- lm(scale(X)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.before <- c(indiv.trt.before, fit.summary1$r.squared)
  indiv.batch.before <- c(indiv.batch.before, fit.summary2$r.squared)
}
r2.trt.before[ ,i] <-  indiv.trt.before
r2.batch.before[ ,i] <-  indiv.batch.before


# auc (sPLSDA)
fit.before_plsda <- splsda(X = X, Y = trt, ncomp = 1)

true.response <- rep(0, p_total)
true.response[true.trt] = 1
before.predictor <- as.numeric(abs(fit.before_plsda$loadings$X))
roc.before_splsda <- roc(true.response, before.predictor, auc = TRUE)
auc.before_splsda <- roc.before_splsda$auc


################################################################################
### Ground-truth data ###
# global variance (RDA)
rda.clean = varpart(scale(X.clean), ~ Treatment, ~ Batch,
                    data = Batch_Trt.factors)
gvar.clean[i, ] <- rda.clean$part$indfract$Adj.R.squared


# precision & recall & F1 (ANOVA)
fit.clean <- lmFit(t(scale(X.clean)), design = model.matrix(~ as.factor(trt)))
fit.result.clean <- topTable(eBayes(fit.clean), coef = 2, number = p_total)
otu.sig.clean <-
  rownames(fit.result.clean)[fit.result.clean$adj.P.Val <= 0.05]
```

```
precision_limma.clean <-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/
  length(otu.sig.clean)
recall_limma.clean<-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/length(true.trt)
F1_limma.clean <-
  (2*precision_limma.clean*recall_limma.clean)/
  (precision_limma.clean + recall_limma.clean)

## replace NA value with 0
if(precision_limma.clean == 'NaN'){
  precision_limma.clean = 0
}
if(F1_limma.clean == 'NaN'){
  F1_limma.clean = 0
}

# individual variance (R2)
indiv.trt.clean <- c()
indiv.batch.clean <- c()
for(c in seq_len(ncol(X.clean))){
  fit.res1 <- lm(scale(X.clean)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.clean)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.clean <- c(indiv.trt.clean, fit.summary1$r.squared)
  indiv.batch.clean <- c(indiv.batch.clean, fit.summary2$r.squared)
}
r2.trt.clean[ ,i] <-  indiv.trt.clean
r2.batch.clean[ ,i] <-  indiv.batch.clean

# auc (sPLSDA)
fit.clean_plsda <- splsda(X = X.clean, Y = trt, ncomp = 1)

clean.predictor <- as.numeric(abs(fit.clean_plsda$loadings$X))
roc.clean_splsda <- roc(true.response, clean.predictor, auc = TRUE)
auc.clean_splsda <- roc.clean_splsda$auc

################################################################################
### removeBatchEffect corrected data ###
X.rbe <-t(removeBatchEffect(t(X), batch = batch,
                            design = model.matrix(~ as.factor(trt))))

# global variance (RDA)
rda.rbe = varpart(scale(X.rbe), ~ Treatment, ~ Batch,
```

```
                    data = Batch_Trt.factors)
gvar.rbe[i, ] <- rda.rbe$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.rbe <- lmFit(t(scale(X.rbe)),
                 design = model.matrix( ~ as.factor(trt)))
fit.result.rbe <- topTable(eBayes(fit.rbe), coef = 2, number = p_total)
otu.sig.rbe <- rownames(fit.result.rbe)[fit.result.rbe$adj.P.Val <= 0.05]

precision_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(otu.sig.rbe)
recall_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(true.trt)
F1_limma.rbe <- (2*precision_limma.rbe*recall_limma.rbe)/
  (precision_limma.rbe + recall_limma.rbe)

## replace NA value with 0
if(precision_limma.rbe == 'NaN'){
  precision_limma.rbe = 0
}
if(F1_limma.rbe == 'NaN'){
  F1_limma.rbe = 0
}

# individual variance (R2)
indiv.trt.rbe <- c()
indiv.batch.rbe <- c()
for(c in seq_len(ncol(X.rbe))){
  fit.res1 <- lm(scale(X.rbe)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.rbe)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.rbe <- c(indiv.trt.rbe, fit.summary1$r.squared)
  indiv.batch.rbe <- c(indiv.batch.rbe, fit.summary2$r.squared)
}
r2.trt.rbe[ ,i] <-  indiv.trt.rbe
r2.batch.rbe[ ,i] <-  indiv.batch.rbe


# auc (sPLSDA)
fit.rbe_plsda <- splsda(X = X.rbe, Y = trt, ncomp = 1)

rbe.predictor <- as.numeric(abs(fit.rbe_plsda$loadings$X))
roc.rbe_splsda <- roc(true.response, rbe.predictor, auc = TRUE)
auc.rbe_splsda <- roc.rbe_splsda$auc
```

```
##############################################################################
### ComBat corrected data ###
X.combat <- t(ComBat(dat = t(X), batch = batch,
                     mod = model.matrix( ~ as.factor(trt))))

# global variance (RDA)
rda.combat = varpart(scale(X.combat), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.combat[i, ] <- rda.combat$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.combat <- lmFit(t(scale(X.combat)),
                    design = model.matrix( ~ as.factor(trt)))
fit.result.combat <- topTable(eBayes(fit.combat), coef = 2, number = p_total)
otu.sig.combat <- rownames(fit.result.combat)[fit.result.combat$adj.P.Val <=
                                                0.05]

precision_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(otu.sig.combat)
recall_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(true.trt)
F1_limma.combat <- (2*precision_limma.combat*recall_limma.combat)/
  (precision_limma.combat + recall_limma.combat)

## replace NA value with 0
if(precision_limma.combat == 'NaN'){
  precision_limma.combat = 0
}
if(F1_limma.combat == 'NaN'){
  F1_limma.combat = 0
}

# individual variance (R2)
indiv.trt.combat <- c()
indiv.batch.combat <- c()
for(c in seq_len(ncol(X.combat))){
  fit.res1 <- lm(scale(X.combat)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.combat)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.combat <- c(indiv.trt.combat, fit.summary1$r.squared)
  indiv.batch.combat <- c(indiv.batch.combat, fit.summary2$r.squared)
}
```

```
r2.trt.combat[ ,i] <-  indiv.trt.combat
r2.batch.combat[ ,i] <-  indiv.batch.combat


# auc (sPLSDA)
fit.combat_plsda <- splsda(X = X.combat, Y = trt, ncomp = 1)

combat.predictor <- as.numeric(abs(fit.combat_plsda$loadings$X))
roc.combat_splsda <- roc(true.response, combat.predictor, auc = TRUE)
auc.combat_splsda <- roc.combat_splsda$auc


################################################################################
### PLSDA-batch corrected data ###
X.plsda_batch.correct <- PLSDA_batch(X = X,
                                     Y.trt = trt, Y.bat = batch,
                                     ncomp.trt = 1, ncomp.bat = 2)
X.plsda_batch <- X.plsda_batch.correct$X.nobatch

# global variance (RDA)
rda.plsda_batch = varpart(scale(X.plsda_batch), ~ Treatment, ~ Batch,
                          data = Batch_Trt.factors)
gvar.plsdab[i, ] <- rda.plsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.plsda_batch <- lmFit(t(scale(X.plsda_batch)),
                         design = model.matrix( ~ as.factor(trt)))
fit.result.plsda_batch <- topTable(eBayes(fit.plsda_batch),
                                   coef = 2, number = p_total)
otu.sig.plsda_batch <- rownames(fit.result.plsda_batch)[
  fit.result.plsda_batch$adj.P.Val <= 0.05]

precision_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(otu.sig.plsda_batch)
recall_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(true.trt)
F1_limma.plsda_batch <-
  (2*precision_limma.plsda_batch*recall_limma.plsda_batch)/
  (precision_limma.plsda_batch + recall_limma.plsda_batch)

## replace NA value with 0
if(precision_limma.plsda_batch == 'NaN'){
  precision_limma.plsda_batch = 0
```

```
}
if(F1_limma.plsda_batch == 'NaN'){
  F1_limma.plsda_batch = 0
}

# individual variance (R2)
indiv.trt.plsda_batch <- c()
indiv.batch.plsda_batch <- c()
for(c in seq_len(ncol(X.plsda_batch))){
  fit.res1 <- lm(scale(X.plsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.plsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.plsda_batch <- c(indiv.trt.plsda_batch,
                             fit.summary1$r.squared)
  indiv.batch.plsda_batch <- c(indiv.batch.plsda_batch,
                               fit.summary2$r.squared)
}
r2.trt.plsdab[ ,i] <-  indiv.trt.plsda_batch
r2.batch.plsdab[ ,i] <-  indiv.batch.plsda_batch

# auc (sPLSDA)
fit.plsda_batch_plsda <- splsda(X = X.plsda_batch, Y = trt, ncomp = 1)

plsda_batch.predictor <- as.numeric(abs(fit.plsda_batch_plsda$loadings$X))
roc.plsda_batch_splsda <- roc(true.response,
                              plsda_batch.predictor, auc = TRUE)
auc.plsda_batch_splsda <- roc.plsda_batch_splsda$auc

##############################################################################
### sPLSDA-batch corrected data ###
X.splsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt,
                                      Y.bat = batch,
                                      ncomp.trt = 1,
                                      keepX.trt = length(true.trt),
                                      ncomp.bat = 2)
X.splsda_batch <- X.splsda_batch.correct$X.nobatch

# global variance (RDA)
rda.splsda_batch = varpart(scale(X.splsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.splsdab[i, ] <- rda.splsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
```

```
fit.splsda_batch <- lmFit(t(scale(X.splsda_batch)),
                          design = model.matrix( ~ as.factor(trt)))
fit.result.splsda_batch <- topTable(eBayes(fit.splsda_batch), coef = 2,
                                    number = p_total)
otu.sig.splsda_batch <- rownames(fit.result.splsda_batch)[
  fit.result.splsda_batch$adj.P.Val <= 0.05]

precision_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(otu.sig.splsda_batch)
recall_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(true.trt)
F1_limma.splsda_batch <-
  (2*precision_limma.splsda_batch*recall_limma.splsda_batch)/
  (precision_limma.splsda_batch + recall_limma.splsda_batch)

## replace NA value with 0
if(precision_limma.splsda_batch == 'NaN'){
  precision_limma.splsda_batch = 0
}
if(F1_limma.splsda_batch == 'NaN'){
  F1_limma.splsda_batch = 0
}


# individual variance (R2)
indiv.trt.splsda_batch <- c()
indiv.batch.splsda_batch <- c()
for(c in seq_len(ncol(X.splsda_batch))){
  fit.res1 <- lm(scale(X.splsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.splsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.splsda_batch <- c(indiv.trt.splsda_batch,
                              fit.summary1$r.squared)
  indiv.batch.splsda_batch <- c(indiv.batch.splsda_batch,
                                fit.summary2$r.squared)
}
r2.trt.splsdab[ ,i] <-  indiv.trt.splsda_batch
r2.batch.splsdab[ ,i] <-  indiv.batch.splsda_batch

# auc (sPLSDA)
fit.splsda_batch_plsda <- splsda(X = X.splsda_batch, Y = trt, ncomp = 1)
```

```
splsda_batch.predictor <- as.numeric(abs(fit.splsda_batch_plsda$loadings$X))
roc.splsda_batch_splsda <- roc(true.response,
                               splsda_batch.predictor, auc = TRUE)
auc.splsda_batch_splsda <- roc.splsda_batch_splsda$auc

################################################################################
### SVA ###
X.mod <- model.matrix(~ as.factor(trt))
X.mod0 <- model.matrix(~ 1, data = as.factor(trt))
X.sva.n <- num.sv(dat = t(X), mod = X.mod, method = 'leek')
X.sva <- sva(t(X), X.mod, X.mod0, n.sv = X.sva.n)

X.mod.batch <- cbind(X.mod, X.sva$sv)
X.mod0.batch <- cbind(X.mod0, X.sva$sv)
X.sva.p <- f.pvalue(t(X), X.mod.batch, X.mod0.batch)
X.sva.p.adj <- p.adjust(X.sva.p, method = 'fdr')

otu.sig.sva <- which(X.sva.p.adj <= 0.05)

# precision & recall & F1 (ANOVA)
precision_limma.sva <-
  length(intersect(true.trt, otu.sig.sva))/length(otu.sig.sva)
recall_limma.sva <-
  length(intersect(true.trt, otu.sig.sva))/length(true.trt)
F1_limma.sva <- (2*precision_limma.sva*recall_limma.sva)/
  (precision_limma.sva + recall_limma.sva)

## replace NA value with 0
if(precision_limma.sva == 'NaN'){
  precision_limma.sva = 0
}
if(F1_limma.sva == 'NaN'){
  F1_limma.sva = 0
}


# summary
# precision & recall & F1 (ANOVA)
precision_limma[i, ] <- c(`Before correction` = precision_limma.before,
                          `Ground-truth data` = precision_limma.clean,
                          `removeBatchEffect` = precision_limma.rbe,
                          ComBat = precision_limma.combat,
                          `PLSDA-batch` = precision_limma.plsda_batch,
                          `sPLSDA-batch` = precision_limma.splsda_batch,
                          SVA = precision_limma.sva)
```

```
  recall_limma[i, ] <- c(`Before correction` = recall_limma.before,
                         `Ground-truth data` = recall_limma.clean,
                         `removeBatchEffect` = recall_limma.rbe,
                         ComBat = recall_limma.combat,
                         `PLSDA-batch` = recall_limma.plsda_batch,
                         `sPLSDA-batch` = recall_limma.splsda_batch,
                         SVA = recall_limma.sva)

  F1_limma[i, ] <- c(`Before correction` = F1_limma.before,
                     `Ground-truth data` = F1_limma.clean,
                     `removeBatchEffect` = F1_limma.rbe,
                     ComBat = F1_limma.combat,
                     `PLSDA-batch` = F1_limma.plsda_batch,
                     `sPLSDA-batch` = F1_limma.splsda_batch,
                     SVA = F1_limma.sva)

  # auc (splsda)
  auc_splsda[i, ] <- c(`Before correction` = auc.before_splsda,
                       `Ground-truth data` = auc.clean_splsda,
                       `removeBatchEffect` = auc.rbe_splsda,
                       ComBat = auc.combat_splsda,
                       `PLSDA-batch` = auc.plsda_batch_splsda,
                       `sPLSDA-batch` = auc.splsda_batch_splsda)

  # print(i)

}
```

4.3.2 Figures

```
# global variance (RDA)
prop.gvar.all <- rbind(`Before correction` = colMeans(gvar.before),
                       `Ground-truth data` = colMeans(gvar.clean),
                       removeBatchEffect = colMeans(gvar.rbe),
                       ComBat = colMeans(gvar.combat),
                       `PLSDA-batch` = colMeans(gvar.plsdab),
                       `sPLSDA-batch` = colMeans(gvar.splsdab))

prop.gvar.all[prop.gvar.all < 0] = 0
prop.gvar.all <- t(apply(prop.gvar.all, 1, function(x){x/sum(x)}))
colnames(prop.gvar.all) <- c('Treatment', 'Intersection', 'Batch', 'Residuals')

partVar_plot(prop.df = prop.gvar.all)
```

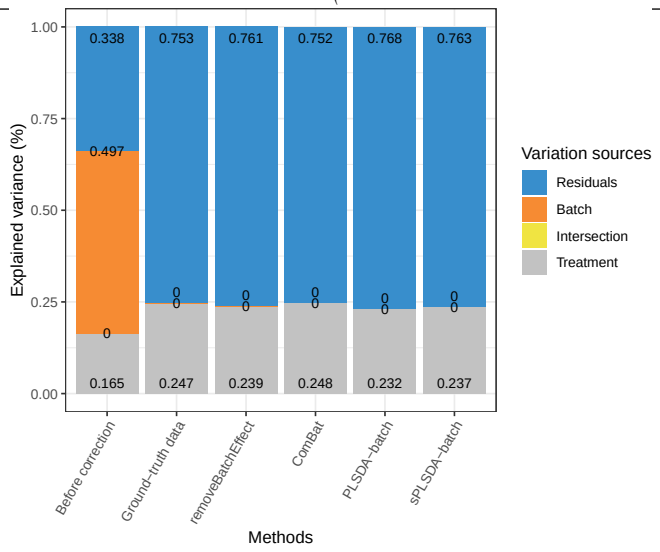

Figure 4.7: Figure 7: Simulation studies (three batch groups): comparison of explained variance before and after batch effect correction for the balanced batch × treatment design.

```
################################################################################
# individual variance (R2)
## boxplot
# class
gclass <- c(rep('Treatment only', p_trt_relevant),
            rep('Batch only', (p_total - p_trt_relevant)))
gclass[intersect(true.trt, true.batch)] = 'Treatment & batch'
gclass[setdiff(1:p_total, union(true.trt, true.batch))] = 'No effect'

gclass <- factor(gclass, levels = c('Treatment & batch',
                                    'Treatment only',
                                    'Batch only',
                                    'No effect'))

before.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.before),
                                      rowMeans(r2.batch.before)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                    each = 300)),
                               class = rep(gclass,2))
clean.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.clean),
                                     rowMeans(r2.batch.clean)),
                              type = as.factor(rep(c('Treatment','Batch'),
```

```
                                                  each = 300)),
                                class = rep(gclass,2))
rbe.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.rbe),
                                   rowMeans(r2.batch.rbe)),
                            type = as.factor(rep(c('Treatment','Batch'),
                                                 each = 300)),
                            class = rep(gclass,2))
combat.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.combat),
                                      rowMeans(r2.batch.combat)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                    each = 300)),
                               class = rep(gclass,2))
plsda_batch.r2.df.ggp <-
  data.frame(r2 = c(rowMeans(r2.trt.plsdab),
                    rowMeans(r2.batch.plsdab)),
             type = as.factor(rep(c('Treatment','Batch'),
                                  each = 300)),
             class = rep(gclass,2))
splsda_batch.r2.df.ggp <-
  data.frame(r2 = c(rowMeans(r2.trt.splsdab),
                    rowMeans(r2.batch.splsdab)),
             type = as.factor(rep(c('Treatment','Batch'),
                                  each = 300)),
             class = rep(gclass,2))

all.r2.df.ggp <- rbind(before.r2.df.ggp, clean.r2.df.ggp,
                       rbe.r2.df.ggp, combat.r2.df.ggp,
                       plsda_batch.r2.df.ggp, splsda_batch.r2.df.ggp)

all.r2.df.ggp$methods <- rep(c('Before correction',
                               'Ground-truth data',
                               'removeBatchEffect',
                               'ComBat',
                               'PLSDA-batch',
                               'sPLSDA-batch'), each = 600)

all.r2.df.ggp$methods <- factor(all.r2.df.ggp$methods,
                                levels = unique(all.r2.df.ggp$methods))

ggplot(all.r2.df.ggp, aes(x = type, y = r2, fill = class)) +
  geom_boxplot(alpha = 0.80) +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
```

```
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

Figure 4.8: Figure 8: Simulation studies (three batch groups): R2 values for each microbial variable before and after batch effect correction for the balanced batch × treatment design.

```
##############################################################################
## barplot
# class
before.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.before), gclass, sum),
                    tapply(rowMeans(r2.batch.before), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))
```

```
clean.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.clean), gclass, sum),
                    tapply(rowMeans(r2.batch.clean), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

rbe.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.rbe), gclass, sum),
                    tapply(rowMeans(r2.batch.rbe), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

combat.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.combat), gclass, sum),
                    tapply(rowMeans(r2.batch.combat), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

plsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.plsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.plsdab), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

splsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.splsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.splsdab), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))


all.r2.df.bp <- rbind(before.r2.df.bp, clean.r2.df.bp,
                      rbe.r2.df.bp, combat.r2.df.bp,
                      plsda_batch.r2.df.bp, splsda_batch.r2.df.bp)


all.r2.df.bp$methods <- rep(c('Before correction',
                              'Ground-truth data',
                              'removeBatchEffect',
                              'ComBat',
                              'PLSDA-batch',
                              'sPLSDA-batch'), each = 8)

all.r2.df.bp$methods <- factor(all.r2.df.bp$methods,
                               levels = unique(all.r2.df.bp$methods))
```

```
ggplot(all.r2.df.bp, aes(x = type, y = r2, fill = class)) +
  geom_bar(stat="identity") +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

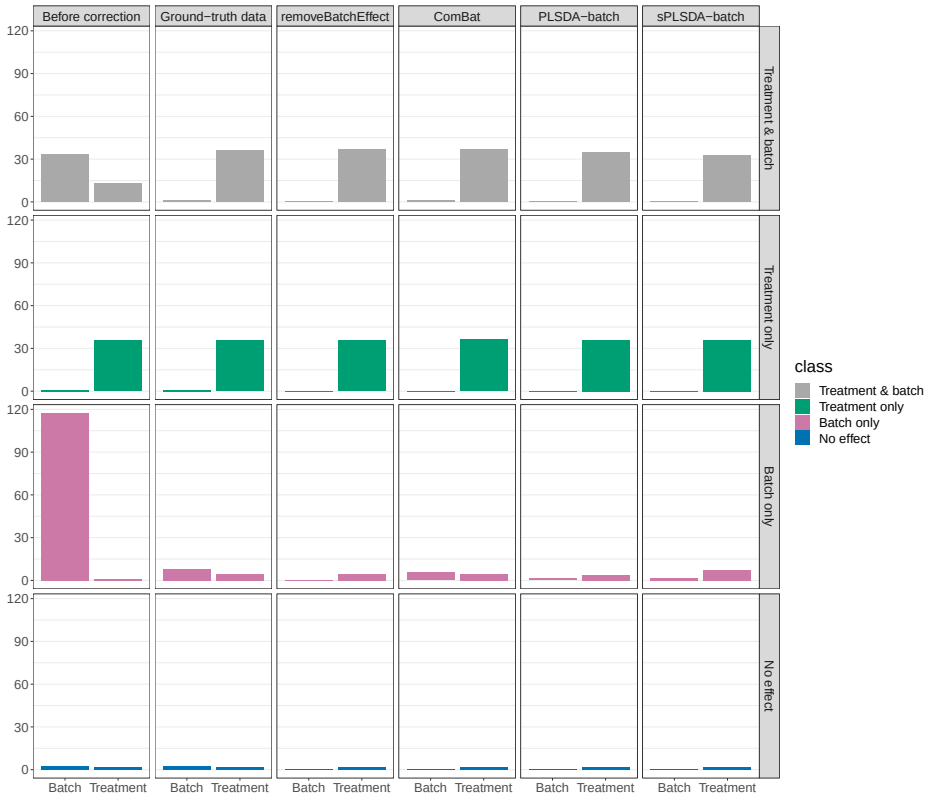

Figure 4.9: Figure 9: Simulation studies (three batch groups): the sum of R2 values for each microbial variable before and after batch effect correction for the balanced batch × treatment design.

```
# precision & recall & F1 (ANOVA & sPLSDA)
## mean
acc_mean <- rbind(colMeans(precision_limma), colMeans(recall_limma),
```

Table 4.9: Table 9: Simulation studies (three batch groups): summary of accuracy measurements before and after batch effect correction for the balanced batch × treatment design (mean).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	PLSDA-batch	sPLSDA-batch	S
Precision	0.986	0.954	0.949	0.940	0.957	0.856	0.
Recall	0.667	0.891	0.902	0.903	0.896	0.884	0.
F1	0.795	0.920	0.923	0.919	0.924	0.867	0.
AUC	0.940	0.959	0.964	0.964	0.964	0.949	N

Table 4.10: Table 10: Simulation studies (three batch groups): summary of accuracy measurements before and after batch effect correction for the balanced batch × treatment design (standard deviation).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	PLSDA-batch	sPLSDA-batch	S
Precision	0.03	0.07	0.07	0.08	0.06	0.08	0.
Recall	0.04	0.03	0.03	0.03	0.03	0.03	0.
F1	0.03	0.04	0.04	0.04	0.03	0.04	0.
AUC	0.02	0.02	0.02	0.02	0.02	0.02	N

```
                  colMeans(F1_limma), c(colMeans(auc_splsda), sva = NA))
rownames(acc_mean) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_mean) <- c('Before correction', 'Ground-truth data',
                        'removeBatchEffect', 'ComBat',
                        'PLSDA-batch', 'sPLSDA-batch', 'SVA')
acc_mean <- format(acc_mean, digits = 3)
knitr::kable(acc_mean, caption = 'Table 9: Simulation studies (three batch groups): summary of ac
```

```
## sd
acc_sd <- rbind(apply(precision_limma, 2, sd), apply(recall_limma, 2, sd),
                apply(F1_limma, 2, sd), c(apply(auc_splsda, 2, sd), NA))
rownames(acc_sd) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_sd) <- c('Before correction', 'Ground-truth data',
                      'removeBatchEffect', 'ComBat',
                      'PLSDA-batch', 'sPLSDA-batch', 'SVA')
acc_sd <- format(acc_sd, digits = 1)
knitr::kable(acc_sd, caption = 'Table 10: Simulation studies (three batch groups): summary of acc
```

4.3.3 Unbalanced batch × treatment design

The unbalanced design included 2, 10 and 2 samples from batch1, batch2 and batch3 respectively in trt1, 10, 2 and 10 samples from batch1, batch2 and batch3 in trt2.

Table 11: Unbalanced batch × treatment design in the simulation study

	Trt1	Trt2
Batch1	2	10
Batch2	10	2
Batch3	2	10

```
nitr <- 50
N = 36
p_total = 300
p_trt_relevant = 100
p_bat_relevant = 200

# global variance (RDA)
gvar.before <- gvar.clean <-
  gvar.rbe <- gvar.combat <-
  gvar.wplsdab <- gvar.swplsdab <-
  gvar.plsdab <- gvar.splsdab <- data.frame(treatment = NA, batch = NA,
                                            intersection = NA,
                                            residual = NA)

# individual variance (R2)
r2.trt.before <- r2.trt.clean <-
  r2.trt.rbe  <- r2.trt.combat <-
  r2.trt.wplsdab <- r2.trt.swplsdab <-
  r2.trt.plsdab <- r2.trt.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                       ncol = nitr))
r2.batch.before <- r2.batch.clean <-
  r2.batch.rbe  <- r2.batch.combat <-
  r2.batch.wplsdab <- r2.batch.swplsdab <-
  r2.batch.plsdab <- r2.batch.splsdab <- data.frame(matrix(NA, nrow = p_total,
                                                           ncol = nitr))

# precision & recall & F1 (ANOVA)
precision_limma <- recall_limma <- F1_limma <-
  data.frame(before = NA, clean = NA,
             rbe = NA, combat = NA,
             wplsda_batch = NA, swplsda_batch = NA,
             sva = NA)

# auc (splsda)
auc_splsda <-
  data.frame(before = NA, clean = NA,
             rbe = NA, combat = NA,
             wplsda_batch = NA, swplsda_batch = NA)
```

```
set.seed(70)
data.cor.res = corStruct(p = 300, zero_prob = 0.7)

for(i in 1: nitr){
  ### initial setup ###
  simulation <- simData_mnegbinom(batch.group = 3,
                                  mean.batch = 7,
                                  sd.batch = 8,
                                  mean.trt = 3,
                                  sd.trt = 2,
                                  mean.bg = 0,
                                  sd.bg = 0.2,
                                  N = 36,
                                  p_total = 300,
                                  p_trt_relevant = 100,
                                  p_bat_relevant = 200,
                                  percentage_overlap_samples = 1/6,
                                  percentage_overlap_variables = 0.5,
                                  data.cor = data.cor.res$data.cor,
                                  disp = 10, prob_zero = 0,
                                  seeds = i)

  set.seed(i)
  raw_count <- simulation$data
  raw_count_clean <- simulation$cleanData

  ## log transformation
  data_log <- log(raw_count + 1)
  data_log_clean <- log(raw_count_clean + 1)

  trt <- simulation$Y.trt
  batch <- simulation$Y.bat

  true.trt <- simulation$true.trt
  true.batch <- simulation$true.batch

  Batch_Trt.factors <- data.frame(Batch = batch, Treatment = trt)

  ### Original ###
  X <- data_log

  ### Clean data ###
  X.clean <- data_log_clean

  #####
```

```
rownames(X) = rownames(X.clean) = names(trt) = names(batch) =
  paste0('sample', 1:N)

colnames(X) = colnames(X.clean) = paste0('otu', 1:p_total)

### Before correction ###
# global variance (RDA)
rda.before = varpart(scale(X), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.before[i,] <- rda.before$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.before <- lmFit(t(scale(X)), design = model.matrix(~ as.factor(trt)))
fit.result.before <- topTable(eBayes(fit.before), coef = 2, number = p_total)
otu.sig.before <-
  rownames(fit.result.before)[fit.result.before$adj.P.Val <= 0.05]

precision_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/
  length(otu.sig.before)
recall_limma.before <-
  length(intersect(colnames(X)[true.trt], otu.sig.before))/length(true.trt)
F1_limma.before <-
  (2*precision_limma.before*recall_limma.before)/
  (precision_limma.before + recall_limma.before)

## replace NA value with 0
if(precision_limma.before == 'NaN'){
  precision_limma.before = 0
}
if(F1_limma.before == 'NaN'){
  F1_limma.before = 0
}

# individual variance (R2)
indiv.trt.before <- c()
indiv.batch.before <- c()
for(c in seq_len(ncol(X))){
  fit.res1 <- lm(scale(X)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.before <- c(indiv.trt.before, fit.summary1$r.squared)
  indiv.batch.before <- c(indiv.batch.before, fit.summary2$r.squared)
}
```

```
r2.trt.before[ ,i] <-  indiv.trt.before
r2.batch.before[ ,i] <-  indiv.batch.before


# auc (sPLSDA)
fit.before_plsda <- splsda(X = X, Y = trt, ncomp = 1)

true.response <- rep(0, p_total)
true.response[true.trt] = 1
before.predictor <- as.numeric(abs(fit.before_plsda$loadings$X))
roc.before_splsda <- roc(true.response, before.predictor, auc = TRUE)
auc.before_splsda <- roc.before_splsda$auc


##############################################################################
### Ground-truth data ###
# global variance (RDA)
rda.clean = varpart(scale(X.clean), ~ Treatment, ~ Batch,
                    data = Batch_Trt.factors)
gvar.clean[i, ] <- rda.clean$part$indfract$Adj.R.squared


# precision & recall & F1 (ANOVA)
fit.clean <- lmFit(t(scale(X.clean)), design = model.matrix(~ as.factor(trt)))
fit.result.clean <- topTable(eBayes(fit.clean), coef = 2, number = p_total)
otu.sig.clean <-
  rownames(fit.result.clean)[fit.result.clean$adj.P.Val <= 0.05]

precision_limma.clean <-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/
  length(otu.sig.clean)
recall_limma.clean <-
  length(intersect(colnames(X)[true.trt], otu.sig.clean))/length(true.trt)
F1_limma.clean <-
  (2*precision_limma.clean*recall_limma.clean)/
  (precision_limma.clean + recall_limma.clean)

## replace NA value with 0
if(precision_limma.clean == 'NaN'){
  precision_limma.clean = 0
}
if(F1_limma.clean == 'NaN'){
  F1_limma.clean = 0
}
```

```
# individual variance (R2)
indiv.trt.clean <- c()
indiv.batch.clean <- c()
for(c in seq_len(ncol(X.clean))){
  fit.res1 <- lm(scale(X.clean)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.clean)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.clean <- c(indiv.trt.clean, fit.summary1$r.squared)
  indiv.batch.clean <- c(indiv.batch.clean, fit.summary2$r.squared)
}
r2.trt.clean[ ,i] <-  indiv.trt.clean
r2.batch.clean[ ,i] <-  indiv.batch.clean

# auc (sPLSDA)
fit.clean_plsda <- splsda(X = X.clean, Y = trt, ncomp = 1)

clean.predictor <- as.numeric(abs(fit.clean_plsda$loadings$X))
roc.clean_splsda <- roc(true.response, clean.predictor, auc = TRUE)
auc.clean_splsda <- roc.clean_splsda$auc

##############################################################################
### removeBatchEffect corrected data ###
X.rbe <-t(removeBatchEffect(t(X), batch = batch,
                            design = model.matrix(~ as.factor(trt))))

# global variance (RDA)
rda.rbe = varpart(scale(X.rbe), ~ Treatment, ~ Batch,
                  data = Batch_Trt.factors)
gvar.rbe[i, ] <- rda.rbe$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.rbe <- lmFit(t(scale(X.rbe)),
                 design = model.matrix( ~ as.factor(trt)))
fit.result.rbe <- topTable(eBayes(fit.rbe), coef = 2, number = p_total)
otu.sig.rbe <- rownames(fit.result.rbe)[fit.result.rbe$adj.P.Val <= 0.05]

precision_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(otu.sig.rbe)
recall_limma.rbe <- length(intersect(colnames(X)[true.trt], otu.sig.rbe))/
  length(true.trt)
F1_limma.rbe <- (2*precision_limma.rbe*recall_limma.rbe)/
  (precision_limma.rbe + recall_limma.rbe)

## replace NA value with 0
```

```
if(precision_limma.rbe == 'NaN'){
  precision_limma.rbe = 0
}
if(F1_limma.rbe == 'NaN'){
  F1_limma.rbe = 0
}

# individual variance (R2)
indiv.trt.rbe <- c()
indiv.batch.rbe <- c()
for(c in seq_len(ncol(X.rbe))){
  fit.res1 <- lm(scale(X.rbe)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.rbe)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.rbe <- c(indiv.trt.rbe, fit.summary1$r.squared)
  indiv.batch.rbe <- c(indiv.batch.rbe, fit.summary2$r.squared)
}
r2.trt.rbe[ ,i] <-  indiv.trt.rbe
r2.batch.rbe[ ,i] <-  indiv.batch.rbe


# auc (sPLSDA)
fit.rbe_plsda <- splsda(X = X.rbe, Y = trt, ncomp = 1)

rbe.predictor <- as.numeric(abs(fit.rbe_plsda$loadings$X))
roc.rbe_splsda <- roc(true.response, rbe.predictor, auc = TRUE)
auc.rbe_splsda <- roc.rbe_splsda$auc

##############################################################################
### ComBat corrected data ###
X.combat <- t(ComBat(dat = t(X), batch = batch,
                     mod = model.matrix( ~ as.factor(trt))))

# global variance (RDA)
rda.combat = varpart(scale(X.combat), ~ Treatment, ~ Batch,
                     data = Batch_Trt.factors)
gvar.combat[i, ] <- rda.combat$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.combat <- lmFit(t(scale(X.combat)),
                    design = model.matrix( ~ as.factor(trt)))
fit.result.combat <- topTable(eBayes(fit.combat), coef = 2, number = p_total)
otu.sig.combat <-
  rownames(fit.result.combat)[fit.result.combat$adj.P.Val <= 0.05]
```

```
precision_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(otu.sig.combat)
recall_limma.combat <-
  length(intersect(colnames(X)[true.trt], otu.sig.combat))/
  length(true.trt)
F1_limma.combat <-
  (2*precision_limma.combat*recall_limma.combat)/
  (precision_limma.combat + recall_limma.combat)

## replace NA value with 0
if(precision_limma.combat == 'NaN'){
  precision_limma.combat = 0
}
if(F1_limma.combat == 'NaN'){
  F1_limma.combat = 0
}

# individual variance (R2)
indiv.trt.combat <- c()
indiv.batch.combat <- c()
for(c in seq_len(ncol(X.combat))){
  fit.res1 <- lm(scale(X.combat)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.combat)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.combat <- c(indiv.trt.combat, fit.summary1$r.squared)
  indiv.batch.combat <- c(indiv.batch.combat, fit.summary2$r.squared)
}
r2.trt.combat[ ,i] <-  indiv.trt.combat
r2.batch.combat[ ,i] <-  indiv.batch.combat


# auc (sPLSDA)
fit.combat_plsda <- splsda(X = X.combat, Y = trt, ncomp = 1)

combat.predictor <- as.numeric(abs(fit.combat_plsda$loadings$X))
roc.combat_splsda <- roc(true.response, combat.predictor, auc = TRUE)
auc.combat_splsda <- roc.combat_splsda$auc


################################################################################
### wPLSDA-batch corrected data ###
X.wplsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt, Y.bat = batch,
```

```
                                        ncomp.trt = 1, ncomp.bat = 2,
                                        balance = FALSE)
X.wplsda_batch <- X.wplsda_batch.correct$X.nobatch

# global variance (RDA)
rda.wplsda_batch = varpart(scale(X.wplsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.wplsdab[i, ] <- rda.wplsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.wplsda_batch <- lmFit(t(scale(X.wplsda_batch)),
                          design = model.matrix( ~ as.factor(trt)))
fit.result.wplsda_batch <- topTable(eBayes(fit.wplsda_batch),
                                    coef = 2, number = p_total)
otu.sig.wplsda_batch <- rownames(fit.result.wplsda_batch)[
  fit.result.wplsda_batch$adj.P.Val <= 0.05]

precision_limma.wplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.wplsda_batch))/
  length(otu.sig.wplsda_batch)
recall_limma.wplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.wplsda_batch))/
  length(true.trt)
F1_limma.wplsda_batch <-
  (2*precision_limma.wplsda_batch*recall_limma.wplsda_batch)/
  (precision_limma.wplsda_batch + recall_limma.wplsda_batch)

## replace NA value with 0
if(precision_limma.wplsda_batch == 'NaN'){
  precision_limma.wplsda_batch = 0
}
if(F1_limma.wplsda_batch == 'NaN'){
  F1_limma.wplsda_batch = 0
}

# individual variance (R2)
indiv.trt.wplsda_batch <- c()
indiv.batch.wplsda_batch <- c()
for(c in seq_len(ncol(X.wplsda_batch))){
  fit.res1 <- lm(scale(X.wplsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.wplsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.wplsda_batch <- c(indiv.trt.wplsda_batch,
                              fit.summary1$r.squared)
```

```
  indiv.batch.wplsda_batch <- c(indiv.batch.wplsda_batch,
                                fit.summary2$r.squared)
}
r2.trt.wplsdab[ ,i] <-  indiv.trt.wplsda_batch
r2.batch.wplsdab[ ,i] <-  indiv.batch.wplsda_batch

# auc (sPLSDA)
fit.wplsda_batch_plsda <- splsda(X = X.wplsda_batch, Y = trt, ncomp = 1)

wplsda_batch.predictor <- as.numeric(abs(fit.wplsda_batch_plsda$loadings$X))
roc.wplsda_batch_splsda <- roc(true.response,
                               wplsda_batch.predictor, auc = TRUE)
auc.wplsda_batch_splsda <- roc.wplsda_batch_splsda$auc

##############################################################################
### sPLSDA-batch corrected data ###
X.swplsda_batch.correct <- PLSDA_batch(X = X,
                                       Y.trt = trt,
                                       Y.bat = batch,
                                       ncomp.trt = 1,
                                       keepX.trt = length(true.trt),
                                       ncomp.bat = 2,
                                       balance = FALSE)
X.swplsda_batch <- X.swplsda_batch.correct$X.nobatch

# global variance (RDA)
rda.swplsda_batch = varpart(scale(X.swplsda_batch), ~ Treatment, ~ Batch,
                            data = Batch_Trt.factors)
gvar.swplsdab[i, ] <- rda.swplsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.swplsda_batch <- lmFit(t(scale(X.swplsda_batch)),
                           design = model.matrix( ~ as.factor(trt)))
fit.result.swplsda_batch <- topTable(eBayes(fit.swplsda_batch), coef = 2,
                                     number = p_total)
otu.sig.swplsda_batch <- rownames(fit.result.swplsda_batch)[
  fit.result.swplsda_batch$adj.P.Val <= 0.05]

precision_limma.swplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.swplsda_batch))/
  length(otu.sig.swplsda_batch)
recall_limma.swplsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.swplsda_batch))/
  length(true.trt)
F1_limma.swplsda_batch <-
```

```
  (2*precision_limma.swplsda_batch*recall_limma.swplsda_batch)/
  (precision_limma.swplsda_batch + recall_limma.swplsda_batch)

## replace NA value with 0
if(precision_limma.swplsda_batch == 'NaN'){
  precision_limma.swplsda_batch = 0
}
if(F1_limma.swplsda_batch == 'NaN'){
  F1_limma.swplsda_batch = 0
}


# individual variance (R2)
indiv.trt.swplsda_batch <- c()
indiv.batch.swplsda_batch <- c()
for(c in seq_len(ncol(X.swplsda_batch))){
  fit.res1 <- lm(scale(X.swplsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.swplsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.swplsda_batch <- c(indiv.trt.swplsda_batch,
                               fit.summary1$r.squared)
  indiv.batch.swplsda_batch <- c(indiv.batch.swplsda_batch,
                                 fit.summary2$r.squared)
}
r2.trt.swplsdab[ ,i] <-  indiv.trt.swplsda_batch
r2.batch.swplsdab[ ,i] <-  indiv.batch.swplsda_batch

# auc (sPLSDA)
fit.swplsda_batch_plsda <- splsda(X = X.swplsda_batch, Y = trt, ncomp = 1)

swplsda_batch.predictor <- as.numeric(abs(fit.swplsda_batch_plsda$loadings$X))
roc.swplsda_batch_splsda <- roc(true.response,
                                swplsda_batch.predictor, auc = TRUE)
auc.swplsda_batch_splsda <- roc.swplsda_batch_splsda$auc

##############################################################################
### PLSDA-batch corrected data ###
X.plsda_batch.correct <- PLSDA_batch(X = X,
                                     Y.trt = trt, Y.bat = batch,
                                     ncomp.trt = 1, ncomp.bat = 2)
X.plsda_batch <- X.plsda_batch.correct$X.nobatch

# global variance (RDA)
rda.plsda_batch = varpart(scale(X.plsda_batch), ~ Treatment, ~ Batch,
```

```
                          data = Batch_Trt.factors)
gvar.plsdab[i, ] <- rda.plsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.plsda_batch <- lmFit(t(scale(X.plsda_batch)),
                         design = model.matrix( ~ as.factor(trt)))
fit.result.plsda_batch <- topTable(eBayes(fit.plsda_batch),
                                   coef = 2, number = p_total)
otu.sig.plsda_batch <- rownames(fit.result.plsda_batch)[
  fit.result.plsda_batch$adj.P.Val <= 0.05]

precision_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(otu.sig.plsda_batch)
recall_limma.plsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.plsda_batch))/
  length(true.trt)
F1_limma.plsda_batch <-
  (2*precision_limma.plsda_batch*recall_limma.plsda_batch)/
  (precision_limma.plsda_batch + recall_limma.plsda_batch)

## replace NA value with 0
if(precision_limma.plsda_batch == 'NaN'){
  precision_limma.plsda_batch = 0
}
if(F1_limma.plsda_batch == 'NaN'){
  F1_limma.plsda_batch = 0
}

# individual variance (R2)
indiv.trt.plsda_batch <- c()
indiv.batch.plsda_batch <- c()
for(c in seq_len(ncol(X.plsda_batch))){
  fit.res1 <- lm(scale(X.plsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.plsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.plsda_batch <- c(indiv.trt.plsda_batch,
                             fit.summary1$r.squared)
  indiv.batch.plsda_batch <- c(indiv.batch.plsda_batch,
                               fit.summary2$r.squared)
}
r2.trt.plsdab[ ,i] <-  indiv.trt.plsda_batch
r2.batch.plsdab[ ,i] <-  indiv.batch.plsda_batch
```

```
# auc (sPLSDA)
fit.plsda_batch_plsda <- splsda(X = X.plsda_batch, Y = trt, ncomp = 1)

plsda_batch.predictor <- as.numeric(abs(fit.plsda_batch_plsda$loadings$X))
roc.plsda_batch_splsda <- roc(true.response,
                              plsda_batch.predictor, auc = TRUE)
auc.plsda_batch_splsda <- roc.plsda_batch_splsda$auc

##############################################################################
### sPLSDA-batch corrected data ###
X.splsda_batch.correct <- PLSDA_batch(X = X,
                                      Y.trt = trt,
                                      Y.bat = batch,
                                      ncomp.trt = 1,
                                      keepX.trt = length(true.trt),
                                      ncomp.bat = 2)
X.splsda_batch <- X.splsda_batch.correct$X.nobatch

# global variance (RDA)
rda.splsda_batch = varpart(scale(X.splsda_batch), ~ Treatment, ~ Batch,
                           data = Batch_Trt.factors)
gvar.splsdab[i, ] <- rda.splsda_batch$part$indfract$Adj.R.squared

# precision & recall & F1 (ANOVA)
fit.splsda_batch <- lmFit(t(scale(X.splsda_batch)),
                          design = model.matrix( ~ as.factor(trt)))
fit.result.splsda_batch <- topTable(eBayes(fit.splsda_batch), coef = 2,
                                    number = p_total)
otu.sig.splsda_batch <- rownames(fit.result.splsda_batch)[
  fit.result.splsda_batch$adj.P.Val <= 0.05]

precision_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(otu.sig.splsda_batch)
recall_limma.splsda_batch <-
  length(intersect(colnames(X)[true.trt], otu.sig.splsda_batch))/
  length(true.trt)
F1_limma.splsda_batch <-
  (2*precision_limma.splsda_batch*recall_limma.splsda_batch)/
  (precision_limma.splsda_batch + recall_limma.splsda_batch)

## replace NA value with 0
if(precision_limma.splsda_batch == 'NaN'){
  precision_limma.splsda_batch = 0
}
```

```
if(F1_limma.splsda_batch == 'NaN'){
  F1_limma.splsda_batch = 0
}


# individual variance (R2)
indiv.trt.splsda_batch <- c()
indiv.batch.splsda_batch <- c()
for(c in seq_len(ncol(X.splsda_batch))){
  fit.res1 <- lm(scale(X.splsda_batch)[ ,c] ~ trt)
  fit.summary1 <- summary(fit.res1)
  fit.res2 <- lm(scale(X.splsda_batch)[ ,c] ~ batch)
  fit.summary2 <- summary(fit.res2)
  indiv.trt.splsda_batch <- c(indiv.trt.splsda_batch,
                              fit.summary1$r.squared)
  indiv.batch.splsda_batch <- c(indiv.batch.splsda_batch,
                                fit.summary2$r.squared)
}
r2.trt.splsdab[ ,i] <-  indiv.trt.splsda_batch
r2.batch.splsdab[ ,i] <-  indiv.batch.splsda_batch

# auc (sPLSDA)
fit.splsda_batch_plsda <- splsda(X = X.splsda_batch, Y = trt, ncomp = 1)

splsda_batch.predictor <- as.numeric(abs(fit.splsda_batch_plsda$loadings$X))
roc.splsda_batch_splsda <- roc(true.response,
                               splsda_batch.predictor, auc = TRUE)
auc.splsda_batch_splsda <- roc.splsda_batch_splsda$auc

##############################################################################
### SVA ###
X.mod <- model.matrix(~ as.factor(trt))
X.mod0 <- model.matrix(~ 1, data = as.factor(trt))
X.sva.n <- num.sv(dat = t(X), mod = X.mod, method = 'leek')
X.sva <- sva(t(X), X.mod, X.mod0, n.sv = X.sva.n)

X.mod.batch <- cbind(X.mod, X.sva$sv)
X.mod0.batch <- cbind(X.mod0, X.sva$sv)
X.sva.p <- f.pvalue(t(X), X.mod.batch, X.mod0.batch)
X.sva.p.adj <- p.adjust(X.sva.p, method = 'fdr')

otu.sig.sva <- which(X.sva.p.adj <= 0.05)

# precision & recall & F1 (ANOVA)
precision_limma.sva <-
```

```
  length(intersect(true.trt, otu.sig.sva))/length(otu.sig.sva)
recall_limma.sva <-
  length(intersect(true.trt, otu.sig.sva))/length(true.trt)
F1_limma.sva <-
  (2*precision_limma.sva*recall_limma.sva)/
  (precision_limma.sva + recall_limma.sva)

## replace NA value with 0
if(precision_limma.sva == 'NaN'){
  precision_limma.sva = 0
}
if(F1_limma.sva == 'NaN'){
  F1_limma.sva = 0
}


# summary
# precision & recall & F1 (ANOVA)
precision_limma[i, ] <- c(`Before correction` = precision_limma.before,
                          `Ground-truth data` = precision_limma.clean,
                          `removeBatchEffect` = precision_limma.rbe,
                          ComBat = precision_limma.combat,
                          `wPLSDA-batch` = precision_limma.wplsda_batch,
                          `swPLSDA-batch` = precision_limma.swplsda_batch,
                          SVA = precision_limma.sva)

recall_limma[i, ] <- c(`Before correction` = recall_limma.before,
                       `Ground-truth data` = recall_limma.clean,
                       `removeBatchEffect` = recall_limma.rbe,
                       ComBat = recall_limma.combat,
                       `wPLSDA-batch` = recall_limma.wplsda_batch,
                       `swPLSDA-batch` = recall_limma.swplsda_batch,
                       SVA = recall_limma.sva)

F1_limma[i, ] <- c(`Before correction` = F1_limma.before,
                   `Ground-truth data` = F1_limma.clean,
                   `removeBatchEffect` = F1_limma.rbe,
                   ComBat = F1_limma.combat,
                   `wPLSDA-batch` = F1_limma.wplsda_batch,
                   `swPLSDA-batch` = F1_limma.swplsda_batch,
                   SVA = F1_limma.sva)

# auc (splsda)
auc_splsda[i, ] <- c(`Before correction` = auc.before_splsda,
                     `Ground-truth data` = auc.clean_splsda,
```

```
                        `removeBatchEffect` = auc.rbe_splsda,
                        ComBat = auc.combat_splsda,
                        `wPLSDA-batch` = auc.wplsda_batch_splsda,
                        `swPLSDA-batch` = auc.swplsda_batch_splsda)

  #  print(i)

}
```

4.3.4 Figures

```
# global variance (RDA)
prop.gvar.all <- rbind(`Before correction` = colMeans(gvar.before),
                        `Ground-truth data` = colMeans(gvar.clean),
                        removeBatchEffect = colMeans(gvar.rbe),
                        ComBat = colMeans(gvar.combat),
                        `wPLSDA-batch` = colMeans(gvar.wplsdab),
                        `swPLSDA-batch` = colMeans(gvar.swplsdab),
                        `PLSDA-batch` = colMeans(gvar.plsdab),
                        `sPLSDA-batch` = colMeans(gvar.splsdab))

prop.gvar.all[prop.gvar.all < 0] = 0
prop.gvar.all <- t(apply(prop.gvar.all, 1, function(x){x/sum(x)}))
colnames(prop.gvar.all) <- c('Treatment', 'Intersection', 'Batch', 'Residuals')

partVar_plot(prop.df = prop.gvar.all)
```

```
################################################################################
# individual variance (R2)
## boxplot
# class
gclass <- c(rep('Treatment only', p_trt_relevant),
            rep('Batch only', (p_total - p_trt_relevant)))
gclass[intersect(true.trt, true.batch)] = 'Treatment & batch'
gclass[setdiff(1:p_total, union(true.trt, true.batch))] = 'No effect'

gclass <- factor(gclass, levels = c('Treatment & batch',
                                    'Treatment only',
                                    'Batch only',
                                    'No effect'))

before.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.before),
                                      rowMeans(r2.batch.before)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                    each = 300)),
```

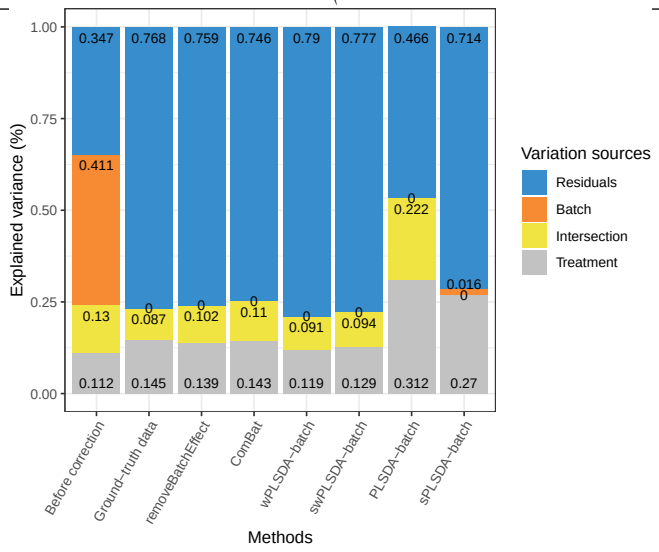

Figure 4.10: Figure 10: Simulation studies (three batch groups): comparison of explained variance before and after batch effect correction for the unbalanced batch × treatment design.

```
                                  class = rep(gclass,2))
clean.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.clean),
                                     rowMeans(r2.batch.clean)),
                              type = as.factor(rep(c('Treatment','Batch'),
                                                   each = 300)),
                              class = rep(gclass,2))
rbe.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.rbe),
                                   rowMeans(r2.batch.rbe)),
                            type = as.factor(rep(c('Treatment','Batch'),
                                                 each = 300)),
                            class = rep(gclass,2))
combat.r2.df.ggp <- data.frame(r2 = c(rowMeans(r2.trt.combat),
                                      rowMeans(r2.batch.combat)),
                               type = as.factor(rep(c('Treatment','Batch'),
                                                    each = 300)),
                               class = rep(gclass,2))
wplsda_batch.r2.df.ggp <-
  data.frame(r2 = c(rowMeans(r2.trt.wplsdab),
                    rowMeans(r2.batch.wplsdab)),
             type = as.factor(rep(c('Treatment','Batch'),
                                  each = 300)),
             class = rep(gclass,2))
```

```
swplsda_batch.r2.df.ggp <-
  data.frame(r2 = c(rowMeans(r2.trt.swplsdab),
                    rowMeans(r2.batch.swplsdab)),
             type = as.factor(rep(c('Treatment','Batch'),
                                  each = 300)),
             class = rep(gclass,2))

all.r2.df.ggp <- rbind(before.r2.df.ggp, clean.r2.df.ggp,
                       rbe.r2.df.ggp, combat.r2.df.ggp,
                       wplsda_batch.r2.df.ggp, swplsda_batch.r2.df.ggp)

all.r2.df.ggp$methods <- rep(c('Before correction',
                               'Ground-truth data',
                               'removeBatchEffect',
                               'ComBat',
                               'wPLSDA-batch',
                               'swPLSDA-batch'), each = 600)

all.r2.df.ggp$methods <- factor(all.r2.df.ggp$methods,
                                levels = unique(all.r2.df.ggp$methods))

ggplot(all.r2.df.ggp, aes(x = type, y = r2, fill = class)) +
  geom_boxplot(alpha = 0.80) +
  theme_bw() +
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

```
################################################################################
## barplot
# class
before.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.before), gclass, sum),
                    tapply(rowMeans(r2.batch.before), gclass, sum)),
             type = as.factor(rep(c('Treatment','Batch'), each = 4)),
             class = factor(rep(levels(gclass),2), levels = levels(gclass)))

clean.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.clean), gclass, sum),
                    tapply(rowMeans(r2.batch.clean), gclass, sum)),
```

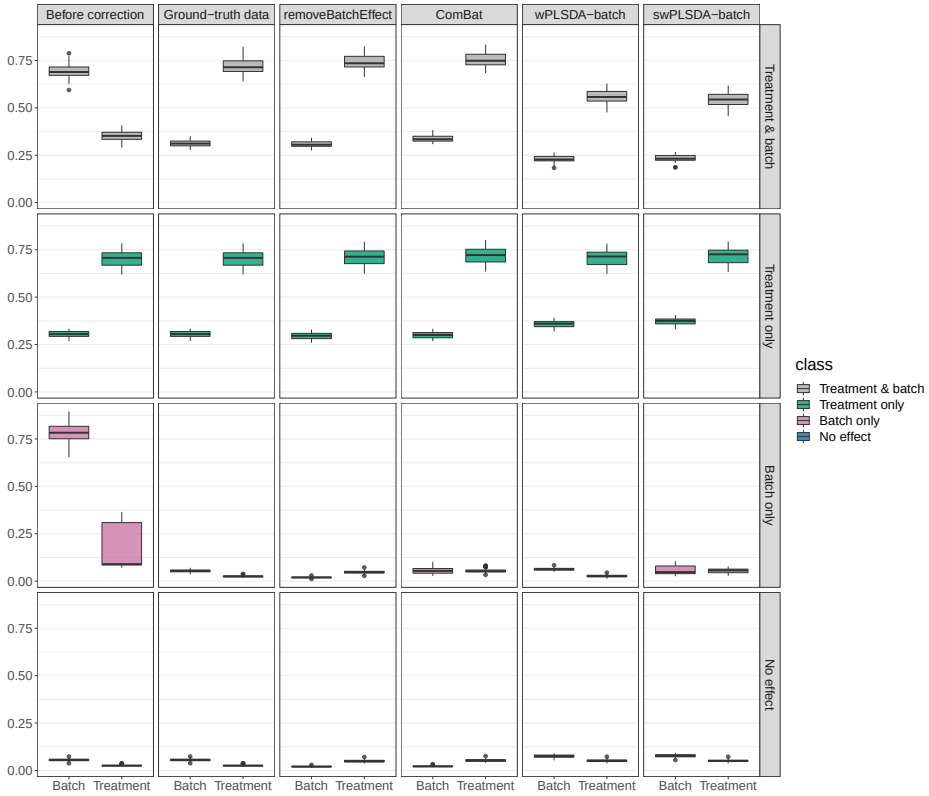

Figure 4.11: Figure 11: Simulation studies (three batch groups): R2 values for each microbial variable before and after batch effect correction for the unbalanced batch × treatment design.

```
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

rbe.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.rbe), gclass, sum),
                    tapply(rowMeans(r2.batch.rbe), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

combat.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.combat), gclass, sum),
                    tapply(rowMeans(r2.batch.combat), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

wplsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.wplsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.wplsdab), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))

swplsda_batch.r2.df.bp <-
  data.frame(r2 = c(tapply(rowMeans(r2.trt.swplsdab), gclass, sum),
                    tapply(rowMeans(r2.batch.swplsdab), gclass, sum)),
            type = as.factor(rep(c('Treatment','Batch'), each = 4)),
            class = factor(rep(levels(gclass),2), levels = levels(gclass)))


all.r2.df.bp <- rbind(before.r2.df.bp, clean.r2.df.bp,
                      rbe.r2.df.bp, combat.r2.df.bp,
                      wplsda_batch.r2.df.bp, swplsda_batch.r2.df.bp)


all.r2.df.bp$methods <- rep(c('Before correction',
                              'Ground-truth data',
                              'removeBatchEffect',
                              'ComBat',
                              'wPLSDA-batch', 'swPLSDA-batch'), each = 8)

all.r2.df.bp$methods <- factor(all.r2.df.bp$methods,
                               levels = unique(all.r2.df.bp$methods))

ggplot(all.r2.df.bp, aes(x = type, y = r2, fill = class)) +
  geom_bar(stat="identity") +
  theme_bw() +
```

```
  theme(text = element_text(size = 18),
        axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "right") + facet_grid(class ~ methods) +
  scale_fill_manual(values=c('dark gray', color.mixo(4),
                             color.mixo(5), color.mixo(9)))
```

Figure 4.12: Figure 12: Simulation studies (three batch groups): the sum of R2 values for each microbial variable before and after batch effect correction for the unbalanced batch × treatment design.

```
# precision & recall & F1 (ANOVA & sPLSDA)
## mean
acc_mean <- rbind(colMeans(precision_limma), colMeans(recall_limma),
                  colMeans(F1_limma), c(colMeans(auc_splsda), sva = NA))
rownames(acc_mean) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_mean) <- c('Before correction', 'Ground-truth data',
```

Table 4.12: Table 12: Simulation studies (three batch groups): summary of accuracy measurements before and after batch effect correction for the unbalanced batch × treatment design (mean).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	wPLSDA-batch	swPl
Precision	0.637	0.972	0.862	0.834	0.915	0.86
Recall	0.811	0.884	0.897	0.904	0.855	0.84
F1	0.713	0.925	0.874	0.863	0.882	0.85
AUC	0.826	0.963	0.954	0.955	0.948	0.92

Table 4.13: Table 13: Simulation studies (three batch groups): summary of accuracy measurements before and after batch effect correction for the unbalanced batch × treatment design (standard deviation).

	Before correction	Ground-truth data	removeBatchEffect	ComBat	wPLSDA-batch	swPl
Precision	0.03	0.04	0.12	0.12	0.08	0.10
Recall	0.03	0.03	0.03	0.03	0.04	0.03
F1	0.03	0.03	0.07	0.07	0.04	0.06
AUC	0.02	0.02	0.02	0.02	0.02	0.02

```
                        'removeBatchEffect', 'ComBat',
                        'wPLSDA-batch', 'swPLSDA-batch', 'SVA')
acc_mean <- format(acc_mean, digits = 3)
knitr::kable(acc_mean, caption = 'Table 12: Simulation studies (three batch groups): su
```

```
## sd
acc_sd <- rbind(apply(precision_limma, 2, sd), apply(recall_limma, 2, sd),
                apply(F1_limma, 2, sd), c(apply(auc_splsda, 2, sd), NA))
rownames(acc_sd) <- c('Precision', 'Recall', 'F1', 'AUC')
colnames(acc_sd) <- c('Before correction', 'Ground-truth data',
                      'removeBatchEffect', 'ComBat',
                      'wPLSDA-batch', 'swPLSDA-batch', 'SVA')
acc_sd <- format(acc_sd, digits = 1)
knitr::kable(acc_sd, caption = 'Table 13: Simulation studies (three batch groups): summ
```

4.4 References

Bibliography

Olivier Chapleur, Céline Madigou, Raphaël Civade, Yohan Rodolphe, Laurent Mazéas, and Théodore Bouchez. Increasing concentrations of phenol progressively affect anaerobic digestion of cellulose and associated microbial communities. *Biodegradation*, 27(1):15–27, 2016.

Claire Duvallet, Sean M Gibbons, Thomas Gurry, Rafael A Irizarry, and Eric J Alm. Meta-analysis of gut microbiome studies identifies disease-specific and shared responses. *Nature communications*, 8(1):1–10, 2017.

Yong Fan and Oluf Pedersen. Gut microbiota in human metabolic health and disease. *Nature Reviews Microbiology*, pages 1–17, 2020.

Glenn R Gibson, Hollie M Probert, Jan Van Loo, Robert A Rastall, and Marcel B Roberfroid. Dietary modulation of the human colonic microbiota: updating the concept of prebiotics. *Nutrition Research Reviews*, 17(2):259–275, 2004.

Carmen Haro, Oriol A Rangel-Zúñiga, Juan F Alcalá-Díaz, Francisco Gómez-Delgado, Pablo Pérez-Martínez, Javier Delgado-Lista, Gracia M Quintana-Navarro, Blanca B Landa, Juan A Navas-Cortés, Manuel Tena-Sempere, et al. Intestinal microbiota is influenced by gender and body mass index. *PloS One*, 11(5):e0154090, 2016.

Stijn Hawinkel, Federico Mattiello, Luc Bijnens, and Olivier Thas. A broken promise: microbiome differential abundance methods do not control the false discovery rate. *Briefings in bioinformatics*, 20(1):210–221, 2019.

Tina J Hieken, Jun Chen, Tanya L Hoskin, Marina Walther-Antonio, Stephen Johnson, Sheri Ramaker, Jian Xiao, Derek C Radisky, Keith L Knutson, Krishna R Kalari, et al. The microbiome of aseptically collected human breast tissue in benign and malignant disease. *Scientific reports*, 6:30751, 2016.

Dorothy Kim, Casey E Hofstaedter, Chunyu Zhao, Lisa Mattei, Ceylan Tanes, Erik Clarke, Abigail Lauder, Scott Sherrill-Mix, Christel Chehoud, Judith Kelsen, et al. Optimizing methods and dodging pitfalls in microbiome research. *Microbiome*, 5(1): 52, 2017.

Geraldine Kong, Kim-Anh Lê Cao, Louise M Judd, ShanShan Li, Thibault Renoir, and Anthony J Hannan. Microbiome profiling reveals gut dysbiosis in a transgenic mouse model of huntington's disease. *Neurobiology of disease*, 135:104268, 2020.

Jeffrey T Leek, W Evan Johnson, Hilary S Parker, Andrew E Jaffe, and John D Storey. The sva package for removing batch effects and other unwanted variation in high-throughput experiments. *Bioinformatics*, 28(6):882–883, 2012.

Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. Upset: visualization of intersecting sets. *IEEE transactions on visualization and computer graphics*, 20(12):1983–1992, 2014.

Catherine Lozupone, Jesse Stombaugh, Antonio Gonzalez, Gail Ackermann, Doug Wendel, Yoshiki Vázquez-Baeza, Janet K Jansson, Jeffrey I Gordon, and Rob Knight. Meta-analyses of studies of the human microbiota. *Genome Research*, pages gr–151803, 2013.

Daniel LÃŒdecke, Dominique Makowski, Philip Waggoner, and Indrajeet Patil. performance: Assessment of regression models performance. *CRAN*, 2020. doi: 10.5281/zenodo.3952174. URL https://easystats.github.io/performance/. R package.

Julian R Marchesi and Jacques Ravel. The vocabulary of microbiome research: a proposal, 2015.

Kevin McGregor, Aurélie Labbe, and Celia MT Greenwood. Mdine: a model to estimate differential co-occurrence networks in microbiome studies. *Bioinformatics*, 36 (6):1840–1847, 2020.

Paul J McMurdie and Susan Holmes. Waste not, want not: why rarefying microbiome data is inadmissible. *PLoS computational biology*, 10(4):e1003531, 2014.

Veronica Moskovicz, Rina Ben-El, Guy Horev, and Boaz Mizrahi. Skin microbiota dynamics following b. subtilis formulation challenge. 2020.

Shinichi Nakagawa and Holger Schielzeth. A general and simple method for obtaining r2 from generalized linear mixed-effects models. *Methods in ecology and evolution*, 4(2):133–142, 2013.

Simon Poirier, Sébastien Déjean, Cédric Midoux, Kim-Anh Lê Cao, and Olivier Chapleur. Integrating independent microbial studies to build predictive models of anaerobic digestion inhibition by ammonia and phenol. *Bioresource Technology*, 316: 123952, 2020.

Thomas P Quinn, Ionas Erb, Mark F Richardson, and Tamsyn M Crowley. Understanding sequencing data as compositions: an outlook and review. *Bioinformatics*, 34(16): 2870–2878, 2018.

Prasun Ray, Venkatachalam Lakshmanan, Jessy L Labbé, and Kelly D Craven. Microbe to microbiome: A paradigm shift in the application of microorganisms for sustainable agriculture. *Frontiers in Microbiology*, 11:3323, 2020.

Florian Rohart, Benoît Gautier, Amrit Singh, and Kim-Anh Lê Cao. mixomics: An r package for 'omics feature selection and multiple data integration. *PLoS computational biology*, 13(11):e1005752, 2017.

Oriol Sacristán-Soriano, Bernard Banaigs, Emilio O Casamayor, and Mikel A Becerro. Exploring the links between natural products and bacterial assemblages in the sponge aplysina aerophoba. *Appl. Environ. Microbiol.*, 77(3):862–870, 2011.

Amrit Singh, Casey P Shannon, Benoît Gautier, Florian Rohart, Michaël Vacher, Scott J Tebbutt, and Kim-Anh Lê Cao. Diablo: an integrative approach for identifying key molecular drivers from multi-omics assays. *Bioinformatics*, 35(17):3055–3062, 2019.

Antoni Susin, Yiwen Wang, Kim-Anh Lê Cao, and M Luz Calle. Variable selection in microbiome compositional data analysis. *NAR Genomics and Bioinformatics*, 2(2): lqaa029, 2020.

Charlotte H Wang, Linda Wu, Zengyan Wang, Magdy S Alabady, Daniel Parson, Zainab Molumo, and Sarah C Fankhauser. Characterizing changes in soil microbiome abundance and diversity due to different cover crop techniques. *PloS one*, 15 (5):e0232453, 2020.

Yiwen Wang and Kim-Anh Lê Cao. A multivariate method to correct for batch effects in microbiome data. *bioRxiv*, 2020.

Yiwen Wang and Kim-Anh LêCao. Managing batch effects in microbiome data. *Briefings in bioinformatics*, 21(6):1954–1970, 2020.

Sophie Weiss, Zhenjiang Zech Xu, Shyamal Peddada, Amnon Amir, Kyle Bittinger, Antonio Gonzalez, Catherine Lozupone, Jesse R Zaneveld, Yoshiki Vázquez-Baeza, Amanda Birmingham, et al. Normalization and microbial differential abundance strategies depend upon data characteristics. *Microbiome*, 5(1):1–18, 2017.