Package 'MBECS'

April 10, 2023

Title Evaluation and correction of batch effects in microbiome data-sets

Version 1.2.0

Description The Microbiome Batch Effect Correction Suite (MBECS) provides a set of functions to evaluate and mitigate unwated noise due to processing in batches. To that end it incorporates a host of batch correcting algorithms (BECA) from various packages. In addition it offers a correction and reporting pipeline that provides a preliminary look at the characteristics of a data-set before and after correcting for batch effects.

biocViews BatchEffect, Microbiome, ReportWriting, Visualization, Normalization, QualityControl

URL https://github.com/rmolbrich/MBECS

BugReports https://github.com/rmolbrich/MBECS/issues/new

License Artistic-2.0

Encoding UTF-8

LazyData false

RoxygenNote 7.1.2

Imports methods, magrittr, phyloseq, limma, lme4, lmerTest, pheatmap, rmarkdown, cluster, dplyr, ggplot2, gridExtra, ruv, sva, tibble, tidyr, vegan, stats, utils, Matrix

Suggests knitr, markdown, BiocStyle, testthat (>= 3.0.0)

Depends R (>= 4.1)

Collate 'MBECS-package.R' 'data.R' 'mbecs_classes.R' 'mbecs_analyses.R' 'mbecs_corrections.R' 'mbecs_helper.R' 'mbecs_plots.R' 'mbecs_reports.R'

VignetteBuilder knitr

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/MBECS

git_branch RELEASE_3_16

git_last_commit 58656c6

git_last_commit_date 2022-11-01

Date/Publication 2023-04-10

Author Michael Olbrich [aut, cre] (<https://orcid.org/0000-0003-2789-3382>)

Maintainer Michael Olbrich <M.Olbrich@protonmail.com>

R topics documented:

`.mbecGetData` *Mbec-Data Getter*

Description

This function extracts abundance matrix and meta-data in the chosen orientation from the input.

Usage

```
.mbecGetData(
  input.obj,
  orientation = "fxs",
  required.col = NULL,
  type = c("otu", "ass", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

input.obj	MbecData object
orientation,	Select either 'fxs' or 'sxf' to retrieve features in rows or columns respectively.
required.col	Vector of column names that are required from the covariate-table.
type	Specify which type of data to add, by using one of 'ass' (Assessement), 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
label	For types 'ass' and 'cor' this specifies the name within the lists.

Details

The parameter 'orientation' determines if the output has features as columns (sxf) or if the columns contain samples (fxs). This is mainly used to retrieve correctly oriented matrices for the different analysis and correction functions.

The parameter 'required.col' is a vector of column names (technically positions would work) in the metadata, that are required for the analysis at hand. The function actually only checks if they are present in the data, but it will return the whole meta-frame.

The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor" and assessment vectors "ass". The later two additionally require the use of the argument 'label' that specifies the name within the respective lists of corrections and assessments.

Value

A list that contains count-matrix (in chosen orientation) and meta-data table.

Examples

```
# This will return the un-transformed (OTU) abundance matrix with features as
# columns and it will test if the columns "group" and "batch" are present in
# the meta-data table.
data(dummy.mbec)
list.obj <- mbecGetData(input.obj=dummy.mbec, orientation="sxf",
    required.col=c("group","batch"), type="otu")

# This will return the clr-transformed abundance matrix with features as
# rows and it will test if the columns "group" and "batch" are present in
# the meta-data table.
list.obj <- mbecGetData(input.obj=dummy.mbec, orientation="fxs",
    required.col=c("group","batch"), type="clr")
```

`.mbecGetPhyloseq` *Return Phyloseq after correction*

Description

This function extracts the abundance table of choice and returns a phyloseq object for downstream analyses.

Usage

```
.mbecGetPhyloseq(
  input.obj,
  type = c("otu", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

`input.obj`	MbecData object
`type`	Specify which type of data to add, by using one of 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
`label`	For type 'cor' this specifies the name within the list.

Details

The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor". The later additionally requires the use of the argument 'label' that specifies the name within the list of corrected matrices.

Value

A phyloseq object that contains the chosen abundance table as otu_table.

Examples

```
# This will return a phyloseq object that contains the clr-transformed
# abundances as otu_table
data(dummy.mbec)
ps.clr.obj <- mbecGetPhyloseq(input.obj=dummy.mbec, type="clr")
```

`.mbecSetData` *Mbec-Data Setter*

Description

Sets and/or replaces selected feature abundance matrix and handles correct orientation. The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor" and assessment vectors "ass". The later two additionally require the use of the argument 'label' that specifies the name within the respective lists of corrections and assessments.

Usage

```
.mbecSetData(
  input.obj,
  new.cnts = NULL,
  type = c("otu", "ass", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

`input.obj`	MbecData object to work on.
`new.cnts`	A matrix-like object with same dimension as 'otu_table' in input.obj.
`type`	Specify which type of data to add, by using one of 'ass' (Assessement), 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
`label`	For types 'ass' and 'cor' this sets the name within the lists.

Value

Input object with updated attributes.

Examples

```
# This will fill the 'tss' slot with the supplied matrix.
data(dummy.mbec, dummy.list)
MBEC.obj <- mbecSetData(input.obj=dummy.mbec, new.cnts=dummy.list$cnts,
    type='tss')

# This will put the given matrix into the list of corrected counts under the
# name "nameOfMethod".
MBEC.obj <- mbecSetData(input.obj=dummy.mbec, new.cnts=dummy.list$cnts,
    type='cor', label="nameOfMethod")
```

colinScore *Variable Correlation Linear (Mixed) Models*

Description

Takes a fitted model and computes maximum correlation between covariates as return value. Return value contains actual correlation-matrix as 'vcor' attribute.

Usage

```
colinScore(model.fit)
```

Arguments

model.fit	lm() or lmm() output

Details

ToDo: maybe some additional validation steps and more informative output.

Value

Maximum amount of correlation for given model variables.

Examples

```
# This will return the maximum colinearity score in the given model
data(dummy.list)
limimo <- lme4::lmer(dummy.list$cnts[,1] ~ group + (1|batch),
data=dummy.list$meta)
num.max_corr <- colinScore(model.fit=limimo)
```

dummy.list *Mock-up microbiome abundance table and meta-data.*

Description

An artificial data-set containing pre-processed abundance table of microbial communities and a matrix of covariate information. The data was created using the mbecDummy function for the sole purpose of running examples and showing the package workflow.

Usage

```
dummy.list
```

Format

A list object containing counts and meta-data:

cnts Compositional Abundance Data

meta Covariate Information

Examples

```
data(dummy.list)
```

`dummy.mbec` *Mock-up microbiome abundance table and meta-data.*

Description

An artificial data-set containing pre-processed abundance table of microbial communities and a matrix of covariate information. The data was created using the `mbecDummy` function for the sole purpose of running examples and showing the package workflow. This particular object was also processed with `mbecTransform` function in order to generate "clr" and "tss" transformed abundance matrices.

Usage

```
dummy.mbec
```

Format

An mbecData object including tss and clr transformed counts:

otu Compositional Abundance Data

tss Compositional Abundance Data Sum-Scaled

clr Compositional Abundance Data Log-Ratio Transformed

meta Covariate Information

Examples

```
data(dummy.mbec)
```

dummy.ps *Mock-up microbiome abundance table and meta-data.*

Description

An artificial data-set containing pre-processed abundance table of microbial communities and a matrix of covariate information. The data was created using the `mbecDummy` function for the sole purpose of running examples and showing the package workflow. This particular object was then converted using `phyloseq`.

Usage

```
dummy.ps
```

Format

A phyloseq object containing counts and meta-data:

otu_table Compositional Abundance Data

sam_data Covariate Information

Examples

```
data(dummy.ps)
```

mbecBat *Combat Batch Effects (ComBat)*

Description

This method uses an non-/parametric empirical Bayes framework to correct for BEs. Described by Johnson et al. 2007 this method was initially conceived to work with gene expression data and is part of the sva-package in R.

Usage

```
mbecBat(input.obj, model.vars, type = c("clr", "otu", "tss"))
```

Arguments

`input.obj`	phyloseq object or numeric matrix (correct orientation is handeled internally)
`model.vars`	Vector of covariate names. First element relates to batch.
`type`	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a matrix of corrected abundances.

Value

A matrix of batch-effect corrected counts

---

mbecBMC *Batch Mean Centering (BMC)*

Description

For known BEs, this method takes the batches, i.e., subgroup of samples within a particular batch, and centers them to their mean.

Usage

```
mbecBMC(input.obj, model.vars, type = c("clr", "otu", "tss"))
```

Arguments

input.obj	phyloseq object or numeric matrix (correct orientation is handeled internally)
model.vars	Vector of covariate names. First element relates to batch.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a matrix of corrected abundances.

Value

A matrix of batch-effect corrected counts

mbecBox *Feature Differential Abundance Box-Plot*

Description

Displays the abundance of a selected feature, grouped/colored by a covariate, i.e., batch, in a box-plot. Includes the density-plot, i.e., the distribution of counts for each sub-group. Selection methods for features are "TOP" and "ALL" which select the top-n or all features respectively. The default value for the argument 'n' is 10. If 'n' is supplied with a vector of feature names, e.g., c("OTU1","OTU5", "OTU10"), of arbitrary length, the argument 'method' will be ignored and only the given features selected for plotting.

Usage

```
mbecBox(
  input.obj,
  method = c("ALL", "TOP"),
  n = 10,
  model.var = "batch",
  type = "clr",
  label = character(),
  return.data = FALSE
)
```

Arguments

input.obj	MbecData object
method	One of 'ALL' or 'TOP' for 'n' most variable features, DEFAULT is 'ALL'.
n	Number of OTUs to display for 'TOP' method, or vector of specific feature names to select.
model.var	Covariate to group by, default is "batch".
type	Which abundance matrix to use for the calculation.
label	Which corrected abundance matrix to use for analysis.
return.data	logical if TRUE returns the data.frame required for plotting. Default (FALSE) will return plot object.

Details

The function returns either a plot-frame or the finished ggplot object. Input is an MbecData-object. If cumulative log-ratio (clr) and total sum-scaled (tss) abundance matrices are part of the input, i.e., 'mbecTransform()' was used, they can be selected as input by using the 'type' argument with either "otu", "clr" or "tss". If batch effect corrected matrices are available, they can be used by specifying the 'type' argument as "cor" and using the 'label' argument to select the appropriate matrix by its denominator, e.g., for batch correction method ComBat this would be "bat", for RemoveBatchEffects from the limma package this is "rbe". Default correction method-labels are "ruv3", "bmc","bat","rbe","pn","svd".

The combination of 'type' and 'label' argument basically accesses the attribute 'cor', a list that stores all matrices of corrected counts. This list can also be accessed via getter and setter methods. Hence, the user can supply their own matrices with own names.

Value

either a ggplot2 object or a formatted data-frame to plot from

Examples

```
# This will return the plot-frame of all features in the data-set.
data(dummy.mbec)
data.Box <- mbecBox(input.obj=dummy.mbec, method='ALL', model.var='batch',
type='clr', return.data=TRUE)

# This will return the ggplot2 object of the top 5 most variable features.
plot.Box <- mbecBox(input.obj=dummy.mbec, method='TOP', n=5,
model.var='batch', type='otu', return.data=FALSE)
```

---

mbecBoxPlot *Variability boxes plotting function*

Description

Takes data.frame from mbecBox and produces a ggplot2 object.

Usage

```
mbecBoxPlot(tmp, otu.idx, model.var, label = NULL)
```

Arguments

tmp	Count of selected features.
otu.idx	Index of selected Otus in the data.
model.var	Which covariate to group Otus by.
label	Name of the plot displayed as legend title.

Value

ggplot2 object

Examples

```
# This will return a list of the five most variable features grouped by the
# covariate 'batch'.
data(dummy.mbec)
box.df <- mbecBox(input.obj=dummy.mbec, method='TOP', n=5,
model.var='batch', type="otu", return.data=TRUE)
plot.box <- mbecBoxPlot(box.df[[1]], box.df[[2]], 'batch')
```

mbecCLR *Centered Log-Ratio Transformation*

Description

Internal function that performs CLR-transformation on input-matrix. Formula is: clr(mtx) = ln(mtx / geometric_mean(mtx_samples))

Usage

```
mbecCLR(input.mtx, offset = 0)
```

Arguments

input.mtx	A matrix of counts (samples x features).
offset	An (OPTIONAL) offset in case of sparse matrix. Function will add an offset of 1/#features if matrix is sparse and offset not provided.

Value

A matrix of transformed counts of same size and orientation as the input.

mbecCorrection *Batch Effect Correction Wrapper*

Description

Either corrects or accounts for (known) batch effects with one of several algorithms.

Usage

```
mbecCorrection(
  input.obj,
  model.vars = c("batch", "group"),
  method = c("lm", "lmm", "sva", "ruv2", "ruv4", "ruv3", "bmc", "bat", "rbe", "pn",
    "svd"),
  type = c("clr", "otu", "tss"),
  nc.features = NULL
)
```

Arguments

input.obj	An MbecData object with 'tss' and 'clr' matrices.
model.vars	Vector of covariate names. First element relates to batch.
method	Denotes the algorithms to use. One of 'lm, lmm, sva, ruv2, ruv4' for assessment methods or one of 'ruv3, bmc, bat, rbe, pn, svd' for correction algorithms.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr' but percentile normalization is supposed to work on tss-abundances.
nc.features	(OPTIONAL) A vector of features names to be used as negative controls in RUV-2/3/4. If not supplied, the algorithm will use a linear model to find pseudo-negative controls

Details

ASSESSMENT METHODS The assessment methods 'lm, lmm, sva, ruv-2 and ruv-4" estimate the significance of the batch effect and update the attribute 'assessments' with vectors of p-values.

Linear (Mixed) Models: A simple linear mixed model with covariates 'treatment' and 'batch', or respective variables in your particular data-set, will be fitted to each feature and the significance for the treatment variable extracted.

Surrogate variable Analysis (SVA): Surrogate Variable Analysis (SVA): Two step approach that (1.) identify the number of latent factors to be estimated by fitting a full-model with effect of interest and a null-model with no effects. The function num.sv then calculates the number of latent factors. In the next (2.) step, the sva function will estimate the surrogate variables. And adjust for them in full/null-model . Subsequent F-test gives significance values for each feature - these P-values and Q-values are accounting for surrogate variables (estimated BEs).

Remove unwanted Variation 2 (RUV-2): Estimates unknown BEs by using negative control variables that, in principle, are unaffected by treatment/biological effect, i.e., aka the effect of interest in an experiment. These variables are generally determined prior to the experiment. An approach to RUV-2 without the presence of negative control variables is the estimation of pseudo-negative controls. To that end, an lm or lmm (depending on whether or not the study design is balanced) with treatment is fitted to each feature and the significance calculated. The features that are not significantly affected by treatment are considered as pseudo-negative control variables. Subsequently, the actual RUV-2 function is applied to the data and returns the p-values for treatment, considering unwanted BEs (whatever that means).

Remove Unwanted Variation 4 (RUV-4): The updated version of RUV-2 also incorporates the residual matrix (w/o treatment effect) to estimate the unknown BEs. To that end it follows the same procedure in case there are no negative control variables and computes pseudo-controls from the data via l(m)m. As RUV-2, this algorithm also uses the parameter 'k' for the number of latent factors. RUV-4 brings the function 'getK()' that estimates this factor from the data itself. The calculated values are however not always reliable. A value of k=0 fo example can occur and should be set to 1 instead. The output is the same as with RUV-2.

CORRECTION METHODS The correction methods 'ruv3, bmc, bat, rbe, pn, svd' attempt to mitigate the batch effect and update the attribute 'corrections' with the resulting abundance matrices of corrected counts.

Remove Unwanted Variation 3 (RUV-3): This algorithm requires negative control-features, i.e., OTUs that are known to be unaffected by the batch effect, as well as technical replicates. The

algorithm will check for the existence of a replicate column in the covariate data. If the column is not present, the execution stops and a warning message will be displayed.

Batch Mean Centering (BMC): For known BEs, this method takes the batches, i.e., subgroup of samples within a particular batch, and centers them to their mean.

Combat Batch Effects (ComBat): This method uses an non-/parametric empirical Bayes framework to correct for BEs. Described by Johnson et al. 2007 this method was initially conceived to work with gene expression data and is part of the sva-package in R.

Remove Batch Effects (RBE): As part of the limma-package this method was designed to remove BEs from Microarray Data. The algorithm fits the full- model to the data, i.e., all relevant covariates whose effect should not be removed, and a model that only contains the known BEs. The difference between these models produces a residual matrix that (should) contain only the full- model-effect, e.g., treatment. As of now the mbecs-correction only uses the first input for batch-effect grouping. ToDo: think about implementing a version for more complex models.

Percentile Normalization (PN): This method was actually developed specifically to facilitate the integration of microbiome data from different studies/experimental set-ups. This problem is similar to the mitigation of BEs, i.e., when collectively analyzing two or more data-sets, every study is effectively a batch on its own (not withstanding the probable BEs within studies). The algorithm iterates over the unique batches and converts the relative abundance of control samples into their percentiles. The relative abundance of case-samples within the respective batches is then transformed into percentiles of the associated control-distribution. Basically, the procedure assumes that the control-group is unaffected by any effect of interest, e.g., treatment or sickness, but both groups within a batch are affected by that BE. The switch to percentiles (kinda) flattens the effective difference in count values due to batch - as compared to the other batches. This also introduces the two limiting aspects in percentile normalization. It can only be applied to case/control designs because it requires a reference group. In addition, the transformation into percentiles removes information from the data.

Singular Value Decomposition (SVD): Basically perform matrix factorization and compute singular eigenvectors (SEV). Assume that the first SEV captures the batch-effect and remove this effect from the data. The interesting thing is that this works pretty well (with the test-data anyway) But since the SEVs are latent factors that are (most likely) confounded with other effects it is not obvious to me that this is the optimal approach to solve this issue.

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be as input, but assessments or corrections-lists will contain the result of the respective chosen method.

Value

An updated object of class MbecData.

Examples

```
# This call will use 'ComBat' for batch effect correction on CLR-transformed
# abundances and store the new counts in the 'corrections' attribute.
data(dummy.mbec)
study.obj <- mbecCorrection(input.obj=dummy.mbec,
model.vars=c("batch","group"), method="bat", type="clr")

# This call will use 'Percentile Normalization' for batch effect correction
```

```
# on TSS-transformed counts and store the new counts in the 'corrections'
# attribute.
study.obj <- mbecCorrection(dummy.mbec, model.vars=c("batch","group"),
method="pn", type="tss")
```

MbecData *Define MbecData-class*

Description

An extension of phyloseq-class that contains the additional attributes 'tss', 'clr', 'corrections' and 'assessments' to enable the MBECS functionality.

Constructor for the package class MbecData. Minimum input is an abundance matrix for the argument 'cnt_table' and any type of frame that contains columns of covariate information. The argument 'cnt_table' requires col/row- names that correspond to features and samples. The correct orientation will be handled internally. The argument 'meta_data' requires row-names that correspond to samples. Although it is an exported function, the user should utilize the function 'mbecProcessInput()' for safe initialization of an MbecData-object from phyloseq or list(counts, metadata) inputs.

Usage

```
MbecData(
  cnt_table = NULL,
  meta_data = NULL,
  tax_table = NULL,
  phy_tree = NULL,
  refseq = NULL,
  assessments = list(),
  corrections = list(),
  tss = NULL,
  clr = NULL
)

MbecData(
  cnt_table = NULL,
  meta_data = NULL,
  tax_table = NULL,
  phy_tree = NULL,
  refseq = NULL,
  assessments = list(),
  corrections = list(),
  tss = NULL,
  clr = NULL
)
```

Arguments

cnt_table	either class phyloseq or a matrix of counts
meta_data	A table with covariate information, whose row-names correspond to sample-IDs.
tax_table	Taxonomic table from phyloseq as optional input.
phy_tree	Phylogenetic tree as optional input.
refseq	Reference sequences as optional input.
assessments	A list for the results of BEAAs.
corrections	A list for the results of BECAs.
tss	Total-sum-squared features matrix.
clr	Cumulative log-ratio transformed feature matrix.

Details

Additional (OPTIONAL) arguments are 'tax_table', 'phy_tree' and 'ref_seq' from phyloseq-objects.

The (OPTIONAL) arguments 'tss' and 'clr' are feature abundance matrices that should contain total-sum-scaled or cumulative log-ratio transformed counts respectively. They should however be calculated by the package-function 'mbecTransform()'.

The lists for Assessments and Corrections will be initialized empty and should only be accessed via the available Get/Set-functions.

Value

produces an R-object of type MbecData

Slots

`otu_table` Class phyloseq::otu_table, (usually sparse) matrix of abundance values.

`sample_data` Dataframe of covariate variables.

`tax_table` Taxonomic table from phyloseq as optional input

`phy_tree` Phylogenetic tree as optional input

`refseq` Reference sequences as optional input

`assessments` A list for the results of batch effect assessment algorithms (BEAA) that produce p-values for all features.

`corrections` A list for the results of batch effect correction algorithms (BECA) that produce adjusted abundance matrices.

`tss` Total-sum-squared feature abundance matrix.

`clr` Cumulative log-ratio transformed feature abundance matrix.

Examples

```
# use constructor with default parameters to create object from count-matrix
# and meta-data table.
data(dummy.list)
mbec.obj <- MbecData(cnt_table=dummy.list$cnts, meta_data = dummy.list$meta)
# use constructor with default parameters to create object from count-matrix
# and meta-data table.
data(dummy.list)
mbec.obj <- MbecData(cnt_table=dummy.list$cnts, meta_data = dummy.list$meta)
```

`mbecDummy` *Creates a dummy data-set with abundance matrix and meta-data.*

Description

For given number of otus and samples this will create mockup microbiome data that contains systematic and non-systematic batch effects. Comes with meta data that denotes study groups and batches. The replicate column is fake and only used to test RUV-implementations.

Usage

```
mbecDummy(n.otus = 500, n.samples = 40)
```

Arguments

`n.otus`	Integer to determine number of features to "simulate".
`n.samples`	Even integer to set number of samples to "simulate".

Details

'Group' and 'batch' variables are actually taken into account in data creation, but only to the degree that the random draws for values are performed with different parameters respectively.

THIS HAS ONLY A CONCEPTUAL SIMILARITY TO MICROBIOME DATA AT BEST AND IS IN NO WAY USEFUL OTHER THAN TESTING PACKAGE FUNCTIONS AND VISUALIZING WORKFLOWS!

This function is also the basis for the included mockup data-sets.

Value

A list object that contains the made up abundance and the accompanying meta-data.

Examples

```
dummy.list <- mbecDummy(n.otus=100, n.samples=30)
```

mbecGetData *Mbec-Data Getter*

Description

This function extracts abundance matrix and meta-data in the chosen orientation from the input.

Usage

```
mbecGetData(
  input.obj,
  orientation = "fxs",
  required.col = NULL,
  type = c("otu", "ass", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

input.obj	MbecData object
orientation,	Select either 'fxs' or 'sxf' to retrieve features in rows or columns respectively.
required.col	Vector of column names that are required from the covariate-table.
type	Specify which type of data to add, by using one of 'ass' (Assessement), 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
label	For types 'ass' and 'cor' this specifies the name within the lists.

Details

The parameter 'orientation' determines if the output has features as columns (sxf) or if the columns contain samples (fxs). This is mainly used to retrieve correctly oriented matrices for the different analysis and correction functions.

The parameter 'required.col' is a vector of column names (technically positions would work) in the metadata, that are required for the analysis at hand. The function actually only checks if they are present in the data, but it will return the whole meta-frame.

The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor" and assessment vectors "ass". The later two additionally require the use of the argument 'label' that specifies the name within the respective lists of corrections and assessments.

Value

A list that contains count-matrix (in chosen orientation) and meta-data table.

Examples

```
# This will return the un-transformed (OTU) abundance matrix with features as
# columns and it will test if the columns "group" and "batch" are present in
# the meta-data table.
data(dummy.mbec)
list.obj <- mbecGetData(input.obj=dummy.mbec, orientation="sxf",
    required.col=c("group","batch"), type="otu")

# This will return the clr-transformed abundance matrix with features as
# rows and it will test if the columns "group" and "batch" are present in
# the meta-data table.
list.obj <- mbecGetData(input.obj=dummy.mbec, orientation="fxs",
    required.col=c("group","batch"), type="clr")
```

---

mbecGetData,MbecData-method

Mbec-Data Getter

Description

This function extracts abundance matrix and meta-data in the chosen orientation from the input.

Usage

```
## S4 method for signature 'MbecData'
mbecGetData(
  input.obj,
  orientation = "fxs",
  required.col = NULL,
  type = c("otu", "ass", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

input.obj	MbecData object
orientation,	Select either 'fxs' or 'sxf' to retrieve features in rows or columns respectively.
required.col	Vector of column names that are required from the covariate-table.
type	Specify which type of data to add, by using one of 'ass' (Assessement), 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
label	For types 'ass' and 'cor' this specifies the name within the lists.

Details

The parameter 'orientation' determines if the output has features as columns (sxf) or if the columns contain samples (fxs). This is mainly used to retrieve correctly oriented matrices for the different analysis and correction functions.

The parameter 'required.col' is a vector of column names (technically positions would work) in the metadata, that are required for the analysis at hand. The function actually only checks if they are present in the data, but it will return the whole meta-frame.

The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor" and assessment vectors "ass". The later two additionally require the use of the argument 'label' that specifies the name within the respective lists of corrections and assessments.

Value

A list that contains count-matrix (in chosen orientation) and meta-data table.

Examples

```
# This will return the un-transformed (OTU) abundance matrix with features as
# columns and it will test if the columns "group" and "batch" are present in
# the meta-data table.
data(dummy.mbec)
list.obj <- mbecGetData(input.obj=dummy.mbec, orientation="sxf",
    required.col=c("group","batch"), type="otu")

# This will return the clr-transformed abundance matrix with features as
# rows and it will test if the columns "group" and "batch" are present in
# the meta-data table.
list.obj <- mbecGetData(input.obj=dummy.mbec, orientation="fxs",
    required.col=c("group","batch"), type="clr")
```

---

mbecGetPhyloseq *Return Phyloseq after correction*

Description

This function extracts the abundance table of choice and returns a phyloseq object for downstream analyses.

Usage

```
mbecGetPhyloseq(
  input.obj,
  type = c("otu", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

input.obj	MbecData object
type	Specify which type of data to add, by using one of 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
label	For type 'cor' this specifies the name within the list.

Details

The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor". The later additionally requires the use of the argument 'label' that specifies the name within the list of corrected matrices.

Value

A phyloseq object that contains the chosen abundance table as otu_table.

Examples

```
# This will return a phyloseq object that contains the clr-transformed
# abundances as otu_table
data(dummy.mbec)
ps.clr.obj <- mbecGetPhyloseq(input.obj=dummy.mbec, type="clr")
```

---

mbecGetPhyloseq,MbecData-method

Return Phyloseq after correction

Description

This function extracts the abundance table of choice and returns a phyloseq object for downstream analyses.

Usage

```
## S4 method for signature 'MbecData'
mbecGetPhyloseq(
  input.obj,
  type = c("otu", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

input.obj	MbecData object
type	Specify which type of data to add, by using one of 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
label	For type 'cor' this specifies the name within the list.

Details

The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor". The later additionally requires the use of the argument 'label' that specifies the name within the list of corrected matrices.

Value

A phyloseq object that contains the chosen abundance table as otu_table.

Examples

```
# This will return a phyloseq object that contains the clr-transformed
# abundances as otu_table
data(dummy.mbec)
ps.clr.obj <- mbecGetPhyloseq(input.obj=dummy.mbec, type="clr")
```

---

mbecHeat *Feature Differential Abundance Heatmap*

Description

Shows the abundance value of selected features in a heatmap. By default, the function expects two covariates group and batch to depict clustering in these groups. More covariates can be included. Selection methods for features are "TOP" and "ALL" which select the top-n or all features respectively. The default value for the argument 'n' is 10. If 'n' is supplied with a vector of feature names, e.g., c("OTU1","OTU5", "OTU10"), of arbitrary length, the argument method' will be ignored and only the given features selected for plotting.

Usage

```
mbecHeat(
  input.obj,
  model.vars = c("batch", "group"),
  center = TRUE,
  scale = TRUE,
  method = "TOP",
  n = 10,
  type = "clr",
  label = character(),
  return.data = FALSE
)
```

Arguments

input.obj	MbecData object
model.vars	Covariates of interest to show in heatmap.
center	Flag to activate centering, DEFAULT is TRUE.
scale	Flag to activate scaling, DEFAULT is TRUE.
method	One of 'ALL' or 'TOP' or a vector of feature names.
n	Number of features to select in method TOP.
type	Which abundance matrix to use for the calculation.
label	Which corrected abundance matrix to use for analysis.
return.data	Logical if TRUE returns the data.frame required for plotting. Default (FALSE) will return plot object.

Details

The function returns either a plot-frame or the finished ggplot object. Input is an MbecData-object. If cumulative log-ratio (clr) and total sum-scaled (tss) abundance matrices are part of the input, i.e., 'mbecTransform()' was used, they can be selected as input by using the 'type' argument with either "otu", "clr" or "tss". If batch effect corrected matrices are available, they can be used by specifying the 'type' argument as "cor" and using the 'label' argument to select the appropriate matrix by its denominator, e.g., for batch correction method ComBat this would be "bat", for RemoveBatchEffects from the limma package this is "rbe". Default correction method-labels are "ruv3", "bmc","bat","rbe","pn","svd".

The combination of 'type' and 'label' argument basically accesses the attribute 'cor', a list that stores all matrices of corrected counts. This list can also be accessed via getter and setter methods. Hence, the user can supply their own matrices with own names.

Value

either a ggplot2 object or a formatted data-frame to plot from

Examples

```
# This will return the plot-frame of all features in the data-set.
data(dummy.mbec)
data.Heat <- mbecHeat(input.obj=dummy.mbec, model.vars=c('group','batch'),
center=TRUE, scale=TRUE, method='ALL', return.data=TRUE)

# This will return the ggplot2 object of the top 5 most variable features.
plot.Heat <- mbecHeat(input.obj=dummy.mbec, model.vars=c('group','batch'),
center=TRUE, scale=TRUE, method='TOP', n=5, return.data=FALSE)
```

mbecHeatPlot *Heatmap plotting function*

Description

Takes data.frame from 'mbecHeat()' and produces a ggplot2 object.

Usage

```
mbecHeatPlot(tmp.cnts, tmp.meta, model.vars, label = NULL)
```

Arguments

tmp.cnts	Count values of selected features.
tmp.meta	Covariate information for potting.
model.vars	Two covariates of interest to select by first variable selects panels and second one determines coloring.
label	Name of the plot displayed as legend title.

Value

ggplot2 object

Examples

```
# This will return a paneled plot that shows results for the variance
# assessment.
data(dummy.mbec)
heat.df <- mbecHeat(input.obj=dummy.mbec, model.vars=c('group','batch'),
center=TRUE, scale=TRUE, method='TOP', n=5, return.data=TRUE)
plot.heat <- mbecHeatPlot(tmp.cnts=heat.df[[1]],
tmp.meta=heat.df[[2]], model.vars=c('group','batch'))
```

mbecLM *Linear (Mixed) Model Feature to Batch Fit*

Description

Helper function that fits lm/lmm with covariates 'treatment' and 'batch' to every feature in the dataset. Returns the fdr corrected significance value for the "treatment" variable. The method 'lm' will fit the linear model `y ~ model.vars[1] + model.vars[2]` and the linear mixed model will consider the second term as random effect, i.e., `y ~ model.vars[1] + (1|model.vars[2])`.

Usage

```
mbecLM(
  input.obj,
  method = c("lm", "lmm"),
  model.vars = c("batch", "group"),
  type = c("clr", "otu", "tss", "cor"),
  label = character()
)
```

Arguments

input.obj	MbecData object
method	Either 'lm' or 'lmm' for linear models and linear mixed models.
model.vars	Covariates of interest, first relates to batch and second to treatment.
type	Which abundance matrix to use, one of 'otu, tss, clr, cor'. DEFAULT is clr' and the use of 'cor' requires the parameter label to be set as well.
label	Which corrected abundance matrix to use for analysis in case 'cor' was selected as type.

Details

The function returns either a plot-frame or the finished ggplot object. Input for th data-set can be an MbecData-object, a phyloseq-object or a list that contains counts and covariate data. The covariate table requires an 'sID' column that contains sample IDs equal to the sample naming in the counts table. Correct orientation of counts will be handled internally.

Value

A vector of fdr corrected p-values that show significance of treatment for every feature

Examples

```
# This will return p-value for the linear model fit of every feature.
data(dummy.mbec)
val.score <- mbecLM(input.obj=dummy.mbec, model.vars=c("batch","group"),
method="lm")
```

---

mbecMixedVariance *Mixed Model Variance-Component Extraction*

Description

A helper function that extracts the variance components of linear mixed models, i.e., residuals, random-effects, fixed-effects, scales them to sample-size and returns a list of components.

Usage

```
mbecMixedVariance(model.fit)
```

Arguments

model.fit	A linear mixed model object of class 'lmerMod'.

Details

Uses 'lme4::VarCorr' to extract Residuals and random-effects components. Standard Deviation of Residuals is stored as 'sc' attribute in the output of 'VarCorr'.

Uses 'lme4::fixef' to extract fixed-effects components, i.e., parameter estimates. The attribute 'pp' of the model contains the dense model matrix for fixed-effects parameters (X). The fixed effects variance, sigma2f, is the variance of the matrix-multiplication beta times X (parameter vector by model matrix)

Value

A named list, containing proportional variance for model terms that describe mixed effects.

Examples

```
# This will return the variance of random/mixed components.
data(dummy.list)
limimo <- lme4::lmer(dummy.list$cnts[,1] ~ group + (1|batch),
data=dummy.list$meta)
list.variance <- mbecMixedVariance(model.fit=limimo)
```

---

mbecModelVariance *Estimate Explained Variance*

Description

The function offers a selection of methods/algorithms to estimate the proportion of variance that can be attributed to covariates of interest. This shows, how much variation is explained by the treatment effect, which proportion is introduced by processing in batches and the leftover variance, i.e., residuals that are not currently explained. Covariates of interest (CoI) are selected by the user and the function will incorporate them into the model building for the respective algorithm. The user can select from five different approaches to adapt to the characteristics of the data-set, e.g., LMMs are a better choice than LMs for a very unbalanced study design. Available approaches are: Linear Model (lm), Linear Mixed Model (lmm), Redundancy Analysis (rda), Principal Variance Component Analysis (pvca) or Silhouette Coefficient (s.coef).

Usage

```
mbecModelVariance(
  input.obj,
  model.vars = character(),
  method = c("lm", "lmm", "rda", "pvca", "s.coef"),
  model.form = NULL,
  type = c("otu", "clr", "tss", "ass", "cor"),
  label = character(),
  no.warning = TRUE,
  na.action = NULL
)
```

Arguments

input.obj	MbecData object
model.vars	Vector of covariates to include in model-construction, in case parameter 'model.form' is not supplied.
method	Select method of modeling: Linear Model (lm), Linear Mixed Model (lmm), Redundancy Analysis (rda), Principal Variance Component Analysis (pvca) or Silhouette Coefficient (s.coef).
model.form	string that describes a model formula, i.e., 'y ~ covariate1 + (1|covariate2)'.
type	Which abundance matrix to use for the calculation.
label	Which corrected abundance matrix to use for analysis.
no.warning	(OPTIONAL) True/False-flag that should turn of singularity warnings, but it doesn't quite work
na.action	(OPTIONAL) set NA handling, will take global option if not supplied

Details

Linear Model (lm): An additive model of all covariates is fitted to each feature respectively and the proportion of variance is extracted for each covariate (OTU_x ~ covariate_1 + covariate_2 + ...).

Linear Mixed Model (lmm): All but the first covariate are considered mixed effects. A model is fitted to each OTU respectively and the proportion of variance extracted for each covariate. (OTU_x ~ covariate_1 + (1|covariate_2) + (1|...)).

partial Redundancy Analysis (rda): Iterates over given covariates, builds a model of all covariates that includes one variable as condition/constraint and then fits it to the feature abundance matrix. The difference in explained variance between the full- and the constrained-model is then attributed to the constraint. (cnts ~ group + Condition(batch) vs. cnts ~ group + batch)

Principal Variance Component Analysis (pvca): Algorithm - calculate the correlation of the fxs count-matrix - from there extract the eigenvectors and eigenvalues and calculate the proportion of explained variance per eigenvector (i.e. principal component) by dividing the eigenvalues by the sum of eigenvalues. Now select as many PCs as required to fill a chosen quota for the total proportion of explained variance. Iterate over all PCs and fit a linear mixed model that contains all covariates as random effect and all unique interactions between two covariates. Compute variance covariance components form the resulting model –> From there we get the Variance that each

covariate(variable) contributes to this particular PC. Then just standardize variance by dividing it through the sum of variance for that model. Scale each PCs results by the proportion this PC accounted for in the first place. And then do it again by dividing it through the total amount of explained variance, i.e. the cutoff to select the number of PCs to take (obviously not the cutoff but rather the actual values for the selected PCs). Finally take the average over each random variable and interaction term and display in a nice plot.

Silhouette Coefficient (s.coef): Calculate principal components and get sample-wise distances on the resulting (sxPC) matrix. Then iterate over all the covariates and calculate the cluster silhouette (which is basically either zero, if the cluster contains only a single element, or it is the distance to the closest different cluster minus the distance of the sample within its own cluster divided (scaled) by the maximum distance). Average over each element in a cluster for all clusters and there is the representation of how good the clustering is. This shows how good a particular covariate characterizes the data, i.e., a treatment variable for instance may differentiate the samples into treated and untreated groups which implies two clusters. In an ideal scenario, the treatment variable, i.e., indicator for some biological effect would produce a perfect clustering. In reality, the confounding variables, e.g., batch, sex or age, will also influence the ordination of samples. Hence, the clustering coefficient is somewhat similar to the amount of explained variance metric that the previous methods used. If used to compare an uncorrected data-set to a batch-corrected set, the expected result would be an increase of clustering coefficient for the biological effect (and all other covariates - because a certain amount of uncertainty was removed from the data) and a decrease for the batch effect.

The function returns a data-frame for further analysis - the report functions (mbecReport and mbecReportPrelim) will automatically produce plots. Input for the data-set can be an MbecData-object, a phyloseq-object or a list that contains counts and covariate data. The covariate table requires an 'sID' column that contains sample IDs equal to the sample naming in the counts table. Correct orientation of counts will be handled internally.

Value

Data.frame that contains proportions of variance for given covariates in every feature.

Examples

```
# This will return a data-frame that contains the variance attributable to
# group and batch according to linear additive model.
data(dummy.mbec)
df.var.lm <- mbecModelVariance(input.obj=dummy.mbec,
model.vars=c("batch", "group"), method='lm', type='clr')
# This will return a data-frame that contains the variance attributable to
# group and batch according to principal variance component analysis.
df.var.pvca <- mbecModelVariance(input.obj=dummy.mbec,
model.vars=c("batch", "group"), method='pvca')
```

`mbecModelVarianceLM` *Estimate Explained Variance with Linear Models*

Description

The function uses a linear modeling approach to estimate the proportion of variance that can be attributed to covariates of interest. This shows, how much variation is explained by the treatment effect, which proportion is introduced by processing in batches and the leftover variance, i.e., residuals that are not currently explained. Covariates of interest (CoI) are selected by the user and the function will incorporate them into the model.

Usage

```
mbecModelVarianceLM(model.form, model.vars, tmp.cnts, tmp.meta, type)
```

Arguments

`model.form`	Formula for linear model, function will create simple additive linear model if this argument is not supplied.
`model.vars`	Covariates to use for model building if argument 'model.form' is not given.
`tmp.cnts`	Abundance matrix in 'sample x feature' orientation.
`tmp.meta`	Covariate table that contains at least the used variables.
`type`	String the denotes data source, i.e., one of "otu","clr" or "tss" for the transformed counts or the label of the batch corrected count-matrix.

Details

Linear Model (lm): An additive model of all covariates is fitted to each feature respectively and the proportion of variance is extracted for each covariate (OTU_x ~ covariate_1 + covariate_2 + ...).

Value

Data.frame that contains proportions of variance for given covariates in a linear modelling approach.

`mbecModelVarianceLMM` *Estimate Explained Variance with Linear Mixed Models*

Description

The function uses a linear mixed modeling approach to estimate the proportion of variance that can be attributed to covariates of interest. This shows, how much variation is explained by the treatment effect, which proportion is introduced by processing in batches and the leftover variance, i.e., residuals that are not currently explained. Covariates of interest (CoI) are selected by the user and the function will incorporate them into the model.

Usage

```
mbecModelVarianceLMM(model.form, model.vars, tmp.cnts, tmp.meta, type)
```

Arguments

model.form	Formula for linear mixed model, function will create simple additive linear mixed model if this argument is not supplied.
model.vars	Covariates to use for model building if argument 'model.form' is not given.
tmp.cnts	Abundance matrix in 'sample x feature' orientation.
tmp.meta	Covariate table that contains at least the used variables.
type	String the denotes data source, i.e., one of "otu","clr" or "tss" for the transformed counts or the label of the batch corrected count-matrix.

Details

Linear Mixed Model (lmm): Only the first covariate is considered a mixed effect. A model is fitted to each OTU respectively and the proportion of variance extracted for each covariate. (OTU_x ~ covariate_2.. + covariate_n + (1|covariate_1)

Value

Data.frame that contains proportions of variance for given covariates in a linear mixed modelling approach.

---

mbecModelVariancePVCA *Estimate Explained Variance with Principal Variance Component Analysis*

Description

The function offers a selection of methods/algorithms to estimate the proportion of variance that can be attributed to covariates of interest. This shows, how much variation is explained by the treatment effect, which proportion is introduced by processing in batches and the leftover variance, i.e., residuals that are not currently explained. Covariates of interest (CoI) are selected by the user and the function will incorporate them into the model.

Usage

```
mbecModelVariancePVCA(
  model.vars,
  tmp.cnts,
  tmp.meta,
  type,
  pct_threshold,
  na.action
)
```

Arguments

model.vars	Covariates to use for model building.
tmp.cnts	Abundance matrix in 'sample x feature' orientation.
tmp.meta	Covariate table that contains at least the used variables.
type	String the denotes data source, i.e., one of "otu","clr" or "tss" for the transformed counts or the label of the batch corrected count-matrix.
pct_threshold	Cutoff value for accumulated variance in principal components.
na.action	Set NA handling, will take global option if not supplied.

Details

Principal Variance Component Analysis (pvca): Algorithm - calculate the correlation of the fxs count-matrix - from there extract the eigenvectors and eigenvalues and calculate the proportion of explained variance per eigenvector (i.e. principal component) by dividing the eigenvalues by the sum of eigenvalues. Now select as many PCs as required to fill a chosen quota for the total proportion of explained variance. Iterate over all PCs and fit a linear mixed model that contains all covariates as random effect and all unique interactions between two covariates. Compute variance covariance components form the resulting model –> From there we get the Variance that each covariate(variable) contributes to this particular PC. Then just standardize variance by dividing it through the sum of variance for that model. Scale each PCs results by the proportion this PC accounted for in the first place. And then do it again by dividing it through the total amount of explained variance, i.e. the cutoff to select the number of PCs to take (obviously not the cutoff but rather the actual values for the selected PCs). Finally take the average over each random variable and interaction term and display in a nice plot.

Value

Data.frame that contains proportions of variance for given covariates in a principal variance component analysis approach.

---

mbecModelVarianceRDA *Estimate Explained Variance with Redundancy Analysis*

Description

The function offers a selection of methods/algorithms to estimate the proportion of variance that can be attributed to covariates of interest. This shows, how much variation is explained by the treatment effect, which proportion is introduced by processing in batches and the leftover variance, i.e., residuals that are not currently explained. Covariates of interest (CoI) are selected by the user and the function will incorporate them into the model.

Usage

```
mbecModelVarianceRDA(model.vars, tmp.cnts, tmp.meta, type)
```

Arguments

model.vars	Covariates to use for model building.
tmp.cnts	Abundance matrix in 'sample x feature' orientation.
tmp.meta	Covariate table that contains at least the used variables.
type	String the denotes data source, i.e., one of "otu","clr" or "tss" for the transformed counts or the label of the batch corrected count-matrix.

Details

partial Redundancy Analysis (rda): Iterates over given covariates, builds a model of all covariates that includes one variable as condition/constraint and then fits it to the feature abundance matrix. The difference in explained variance between the full- and the constrained-model is then attributed to the constraint. (cnts ~ group + Condition(batch) vs. cnts ~ group + batch)

Value

Data.frame that contains proportions of variance for given covariates in a partial redundancy analysis approach.

---

mbecModelVarianceSCOEF

Estimate Explained Variance with Silhouette Coefficient

Description

The function offers a selection of methods/algorithms to estimate the proportion of variance that can be attributed to covariates of interest. This shows, how much variation is explained by the treatment effect, which proportion is introduced by processing in batches and the leftover variance, i.e., residuals that are not currently explained. Covariates of interest (CoI) are selected by the user and the function will incorporate them into the model.

Usage

```
mbecModelVarianceSCOEF(model.vars, tmp.cnts, tmp.meta, type)
```

Arguments

model.vars	Covariates to use for model building.
tmp.cnts	Abundance matrix in 'sample x feature' orientation.
tmp.meta	Covariate table that contains at least the used variables.
type	String the denotes data source, i.e., one of "otu","clr" or "tss" for the transformed counts or the label of the batch corrected count-matrix.

Details

Silhouette Coefficient (s.coef): Calculate principal components and get sample-wise distances on the resulting (sxPC) matrix. Then iterate over all the covariates and calculate the cluster silhouette (which is basically either zero, if the cluster contains only a single element, or it is the distance to the closest different cluster minus the distance of the sample within its own cluster divided (scaled) by the maximum distance). Average over each element in a cluster for all clusters and there is the representation of how good the clustering is. This shows how good a particular covariate characterizes the data, i.e., a treatment variable for instance may differentiate the samples into treated and untreated groups which implies two clusters. In an ideal scenario, the treatment variable, i.e., indicator for some biological effect would produce a perfect clustering. In reality, the confounding variables, e.g., batch, sex or age, will also influence the ordination of samples. Hence, the clustering coefficient is somewhat similar to the amount of explained variance metric that the previous methods used. If used to compare an uncorrected data-set to a batch-corrected set, the expected result would be an increase of clustering coefficient for the biological effect (and all other covariates - because a certain amount of uncertainty was removed from the data) and a decrease for the batch effect.

Value

Data.frame that contains proportions of variance for given covariates in a silhouette coefficient analysis approach.

`mbecMosaic` *Mosaic Sample Group Allocation*

Description

Depicts the dispersion of samples over two (preferentially categorical) covariates of interest. Effectively showing, the un-/evenness within and between covariates to inform the choice of methods for the subsequent steps in an analysis.

Usage

```
mbecMosaic(input.obj, model.vars = c("batch", "group"), return.data = FALSE)
```

Arguments

`input.obj`	MbecData object
`model.vars`	Two covariates of interest to the sample allocation.
`return.data`	Logical if TRUE returns the data.frame required for plotting. Default (FALSE) will return plot object.

Details

The function returns either a plot-frame or the finished ggplot object. Input for the data-set can be an MbecData-object.

Value

either a ggplot2 object or a formatted data-frame to plot from

Examples

```
# This will return the plot-df of the samples grouped by group and batch.
data(dummy.mbec)
data.Mosaic <- mbecMosaic(input.obj=dummy.mbec,
model.vars=c('group','batch'), return.data=TRUE)

# Return the ggplot2 object of the samples grouped by group and batch
plot.Mosaic <- mbecMosaic(input.obj=dummy.mbec,
model.vars=c('group','batch'), return.data=FALSE)
```

---

mbecMosaicPlot *Mosaic plotting function*

Description

Takes data.frame from mbecMosaic and produces a ggplot2 object.

Usage

```
mbecMosaicPlot(study.summary, model.vars)
```

Arguments

study.summary	'mbecMosaic' output object.
model.vars	two covariates of interest to select by first variable selects panels and second one determines coloring.

Value

ggplot2 object

Examples

```
# This will return a paneled plot that shows results for the variance
# assessment.
data(dummy.mbec)
mosaic.df <- mbecMosaic(input.obj=dummy.mbec, model.vars=c('group','batch'),
return.data=TRUE)
plot.mosaic <- mbecMosaicPlot(study.summary=mosaic.df,
model.vars=c('group','batch'))
```

mbecPCA *Principal Component Analysis Plot*

Description

Takes two covariates, i.e., group and batch, and computes the ordination-plot for user-selected principal components. Covariates determine sample-shape and color and can be switched to shift the emphasis on either group. In addition to the ordination-plot, the function will show the distribution of eigenvalues (colored by the second covariate) on their respective principal components.

Usage

```
mbecPCA(
  input.obj,
  model.vars = c("batch", "group"),
  pca.axes = c(1, 2),
  type = "clr",
  label = character(),
  return.data = FALSE
)
```

Arguments

input.obj	list(cnts, meta), phyloseq, MbecData object (correct orientation is handled internally)
model.vars	two covariates of interest to select by first variable selects color (batch) and second one determines shape (group)
pca.axes	numeric vector which axes to plot, first is X and second is Y
type	Which abundance matrix to use for the calculation.
label	Which corrected abundance matrix to use for analysis.
return.data	logical if TRUE returns the data.frame required for plotting. Default (FALSE) will return plot object.

Details

The function returns either a plot-frame or the finished ggplot object. Input is an MbecData-object. If cumulative log-ratio (clr) and total sum-scaled (tss) abundance matrices are part of the input, i.e., 'mbecTransform()' was used, they can be selected as input by using the 'type' argument with either "otu", "clr" or "tss". If batch effect corrected matrices are available, they can be used by specifying the 'type' argument as "cor" and using the 'label' argument to select the appropriate matrix by its denominator, e.g., for batch correction method ComBat this would be "bat", for RemoveBatchEffects from the limma package this is "rbe". Default correction method-labels are "ruv3", "bmc","bat","rbe","pn","svd".

The combination of 'type' and 'label' argument basically accesses the attribute 'cor', a list that stores all matrices of corrected counts. This list can also be accessed via getter and setter methods. Hence, the user can supply their own matrices with own names.

Value

either a ggplot2 object or a formatted data-frame to plot from

Examples

```
# This will return the data.frame for plotting.
data(dummy.mbec)
data.PCA <- mbecPCA(input.obj=dummy.mbec,
model.vars=c('group','batch'), pca.axes=c(1,2), return.data=TRUE)

# This will return the ggplot2 object for display, saving and modification.
# Selected PCs are PC3 on x-axis and PC2 on y-axis.
plot.PCA <- mbecPCA(input.obj=dummy.mbec,
model.vars=c('group','batch'), pca.axes=c(3,2), return.data=FALSE)
```

---

mbecPCA,MbecData-method

Principal Component Analysis Plot for MbecData

Description

Takes two covariates, i.e., group and batch, and computes the ordination-plot for user-selected principal components. Covariates determine sample-shape and color and can be switched to shift the emphasis on either group. In addition to the ordination-plot, the function will show the distribution of eigenvalues (colored by the second covariate) on their respective principal components.

Usage

```
## S4 method for signature 'MbecData'
mbecPCA(
  input.obj,
  model.vars = c("batch", "group"),
  pca.axes = c(1, 2),
  type = "clr",
  label = character(),
  return.data = FALSE
)
```

Arguments

input.obj	MbecData object
model.vars	two covariates of interest to select by first variable selects color (batch) and second one determines shape (group).
pca.axes	numeric vector which axes to plot, first is X and second is Y
type	Which abundance matrix to use for the calculation.
label	Which corrected abundance matrix to use for analysis.
return.data	logical if TRUE returns the data.frame required for plotting. Default (FALSE) will return plot object.

Details

The function returns either a plot-frame or the finished ggplot object. Input is an MbecData-object. If cumulative log-ratio (clr) and total sum-scaled (tss) abundance matrices are part of the input, i.e., 'mbecTransform()' was used, they can be selected as input by using the 'type' argument with either "otu", "clr" or "tss". If batch effect corrected matrices are available, they can be used by specifying the 'type' argument as "cor" and using the 'label' argument to select the appropriate matrix by its denominator, e.g., for batch correction method ComBat this would be "bat", for RemoveBatchEffects from the limma package this is "rbe". Default correction method-labels are "ruv3", "bmc","bat","rbe","pn","svd".

The combination of 'type' and 'label' argument basically accesses the attribute 'cor', a list that stores all matrices of corrected counts. This list can also be accessed via getter and setter methods. Hence, the user can supply their own matrices with own names.

Value

either a ggplot2 object or a formatted data-frame to plot from

Examples

```
# This will return the data.frame for plotting.
data(dummy.mbec)
data.PCA <- mbecPCA(input.obj=dummy.mbec,
model.vars=c('group','batch'), pca.axes=c(1,2), return.data=TRUE)

# This will return the ggplot2 object for display, saving and modification.
# Selected PCs are PC3 on x-axis and PC2 on y-axis.
plot.PCA <- mbecPCA(input.obj=dummy.mbec,
model.vars=c('group','batch'), pca.axes=c(3,2), return.data=FALSE)
```

---

mbecPCAPlot *PCA plotting function*

Description

Takes data.frame from mbecPCA and produces a ggplot2 object.

Usage

```
mbecPCAPlot(plot.df, metric.df, model.vars, pca.axes, label = NULL)
```

Arguments

plot.df	Data.frame containing principal component data.
metric.df	Data.frame containing covariate data.
model.vars	two covariates of interest to select by first variable selects panels and second one determines coloring.
pca.axes	NMumerical two-piece vector that selects PCs to plot.
label	Name of the plot displayed as legend title.

Value

ggplot2 object

Examples

```
# This will return a paneled plot that shows results for the variance
# assessment.
data(dummy.mbec)
pca.df <- mbecPCA(input.obj=dummy.mbec,
model.vars=c('group','batch'), pca.axes=c(1,2), return.data=TRUE)
plot.pca <- mbecPCAPlot(plot.df=pca.df[[1]], metric.df=pca.df[[2]],
model.vars=c('group','batch'), pca.axes=c(1,2))
```

---

mbecPN *Percentile Normalization (PN)*

Description

This method was actually developed specifically to facilitate the integration of microbiome data from different studies/experimental set-ups. This problem is similar to the mitigation of BEs, i.e., when collectively analyzing two or more data-sets, every study is effectively a batch on its own (not withstanding the probable BEs within studies). The algorithm iterates over the unique batches and converts the relative abundance of control samples into their percentiles. The relative abundance of case-samples within the respective batches is then transformed into percentiles of the associated control-distribution. Basically, the procedure assumes that the control-group is unaffected by any effect of interest, e.g., treatment or sickness, but both groups within a batch are affected by that BE. The switch to percentiles (kinda) flattens the effective difference in count values due to batch - as compared to the other batches. This also introduces the two limiting aspects in percentile normalization. It can only be applied to case/control designs because it requires a reference group. In addition, the transformation into percentiles removes information from the data.

Usage

```
mbecPN(input.obj, model.vars, type = c("clr", "otu", "tss"))
```

Arguments

input.obj	phyloseq object or numeric matrix (correct orientation is handeled internally)
model.vars	Vector of covariate names. First element relates to batch.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'tss'.

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a matrix of corrected abundances.

Value

A matrix of batch-effect corrected counts

`mbecProcessInput` *Mbec-Data Constructor Wrapper*

Description

This function is a wrapper for the constructor of MbecData-objects from phyloseq objects and lists of counts and sample data.

Usage

```
mbecProcessInput(input.obj, required.col = NULL)
```

Arguments

`input.obj`	One of MbecData, phyloseq or list(counts, meta-data).
`required.col`	Vector of column names that need to be present in the meta-data table.

Details

The (OPTIONAL) argument 'required.col' is a vector of column-names that will enable a sanity test for the presence in the meta-data table. Which is also the second use-case for objects that are already of class MbecData.

Value

An object of type MbecData that has been validated.

Examples

```
# This will check for the presence of variables 'group' and 'batch' in the
# meta-data and return an object of class 'MbecData'.
data(dummy.mbec)
MbecData.obj <- mbecProcessInput(input.obj=dummy.mbec,
    required.col=c("group","batch"))
```

mbecProcessInput,list-method

Mbec-Data Constructor Wrapper

Description

This function is a wrapper for the constructor of MbecData-objects from phyloseq objects and lists of counts and sample data.

Usage

```
## S4 method for signature 'list'
mbecProcessInput(input.obj, required.col = NULL)
```

Arguments

input.obj	One of MbecData, phyloseq or list(counts, meta-data).
required.col	Vector of column names that need to be present in the meta-data table.

Details

The (OPTIONAL) argument 'required.col' is a vector of column-names that will enable a sanity test for the presence in the meta-data table. Which is also the second use-case for objects that are already of class MbecData.

Value

An object of type MbecData that has been validated.

Examples

```
# This will check for the presence of variables 'group' and 'batch' in the
# meta-data and return an object of class 'MbecData'.
data(dummy.mbec)
MbecData.obj <- mbecProcessInput(input.obj=dummy.mbec,
    required.col=c("group","batch"))
```

mbecProcessInput,MbecData-method

Mbec-Data Constructor Wrapper

Description

This function is a wrapper for the constructor of MbecData-objects from phyloseq objects and lists of counts and sample data.

Usage

```
## S4 method for signature 'MbecData'
mbecProcessInput(input.obj, required.col = NULL)
```

Arguments

input.obj	One of MbecData, phyloseq or list(counts, meta-data).
required.col	Vector of column names that need to be present in the meta-data table.

Details

The (OPTIONAL) argument 'required.col' is a vector of column-names that will enable a sanity test for the presence in the meta-data table. Which is also the second use-case for objects that are already of class MbecData.

Value

An object of type MbecData that has been validated.

Examples

```
# This will check for the presence of variables 'group' and 'batch' in the
# meta-data and return an object of class 'MbecData'.
data(dummy.mbec)
MbecData.obj <- mbecProcessInput(input.obj=dummy.mbec,
    required.col=c("group","batch"))
```

---

mbecProcessInput,phyloseq-method

Mbec-Data Constructor Wrapper

Description

This function is a wrapper for the constructor of MbecData-objects from phyloseq objects and lists of counts and sample data.

Usage

```
## S4 method for signature 'phyloseq'
mbecProcessInput(input.obj, required.col = NULL)
```

Arguments

input.obj	One of MbecData, phyloseq or list(counts, meta-data).
required.col	Vector of column names that need to be present in the meta-data table.

Details

The (OPTIONAL) argument 'required.col' is a vector of column-names that will enable a sanity test for the presence in the meta-data table. Which is also the second use-case for objects that are already of class MbecData.

Value

An object of type MbecData that has been validated.

Examples

```
# This will check for the presence of variables 'group' and 'batch' in the
# meta-data and return an object of class 'MbecData'.
data(dummy.ps)
MbecData.obj <- mbecProcessInput(input.obj=dummy.ps,
    required.col=c("group","batch"))
```

---

mbecPVCAStatsPlot *Plot Proportion of Variance for PVCA*

Description

Covariate-Variances as modeled by PVCA will be displayed as box-plots. It works with the output of 'mbecVarianceStats()' for the method 'pvca'. Format of this output is a data.frame that contains a column for every model variable and as many rows as there are features (OTUs, Genes, ..). Multiple frames may be used as input by putting them into a list - IF the data.frames contain a column named 'type', this function will use 'facet_grid()' to display side-by-side panels to enable easy comparison.

Usage

```
mbecPVCAStatsPlot(pvca.obj)
```

Arguments

pvca.obj,	output of 'mbecVarianceStats' with method pvca

Value

A ggplot2 box-plot object.

Examples

```
# This will return a paneled plot that shows results for the variance
# assessment.
data(dummy.mbec)
df.var.pvca <- mbecModelVariance(input.obj=dummy.mbec,
model.vars=c('batch','group'), method='pvca', type='clr')
plot.pvca <- mbecPVCAStatsPlot(pvca.obj=df.var.pvca)
```

mbecRBE *Remove Batch Effects (RBE)*

Description

As part of the limma-package this method was designed to remove BEs from Microarray Data. The algorithm fits the full-model to the data, i.e., all relevant covariates whose effect should not be removed, and a model that only contains the known BEs. The difference between these models produces a residual matrix that (should) contain only the full-model-effect, e.g., treatment. As of now the mbecs-correction only uses the first input for batch-effect grouping. ToDo: think about implementing a version for more complex models.

Usage

```
mbecRBE(input.obj, model.vars, type = c("clr", "otu", "tss"))
```

Arguments

input.obj	phyloseq object or numeric matrix (correct orientation is handeled internally)
model.vars	Vector of covariate names. First element relates to batch.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a matrix of corrected abundances.

Value

A matrix of batch-effect corrected counts

mbecRDAStatsPlot *Plot Proportion of Variance for pRDA*

Description

Covariate-Variances as modeled by pRDA will be displayed as box-plots. It works with the output of 'mbecVarianceStats()' for the method 'rda'. Format of this output is a data.frame that contains a column for every model variable and as many rows as there are features (OTUs, Genes, ..). Multiple frames may be used as input by putting them into a list - IF the data.frames contain a column named 'type', this function will use 'facet_grid()' to display side-by-side panels to enable easy comparison.

Usage

```
mbecRDAStatsPlot(rda.obj)
```

Arguments

rda.obj,	list or single output of 'mbecVarianceStats' with method rda

Value

A ggplot2 box-plot object.

Examples

```
# This will return a paneled plot that shows results for three variance
# assessments.
data(dummy.mbec)
df.var.rda <- mbecModelVariance(input.obj=dummy.mbec,
model.vars=c('group','batch'), method='rda', type='clr')
plot.rda <- mbecRDAStatsPlot(rda.obj=df.var.rda)
```

---

mbecReportPost *Constructs a comparative report of batch corrected data.*

Description

Constructs a comparative report of batch corrected data.

Usage

```
mbecReportPost(
  input.obj,
  model.vars = c("batch", "group"),
  type = "clr",
  file.name = NULL,
  file.dir = getwd(),
  return.data = FALSE
)
```

Arguments

input.obj,	list of phyloseq objects to compare, first element is considered uncorrected data
model.vars,	required covariates to build models
type	One of 'otu', 'tss' or 'clr' to determine the abundance matrix to use for evaluation.
file.name	Optional file name, parameter defaults to NULL and template name will be used
file.dir	Optional output directory, parameter defaults to current working directory.
return.data,	TRUE will return a list of all produced plots, FALSE will start rendering the report

Value

either a ggplot2 object or a formatted data-frame to plot from

Examples

```
data(dummy.list)
dummy.test <- mbecTransform(list(dummy.list$cnts[,seq(20)],
dummy.list$meta), method="clr")
dummy.corrected <- mbecCorrection(input.obj=dummy.test,
model.vars=c("batch","group"), method="bat", type="clr" )

report.data <- mbecReportPost(input.obj=dummy.corrected,
model.vars=c("batch","group"), type="clr", file.name=NULL, file.dir=NULL,
return.data = TRUE)
```

---

mbecReportPrelim *Constructs an initial report of a single data-set.*

Description

Input can be of class MbecData, phyloseq or list(counts, meta-data). The function will check if required covariates are present and apply normalization with default parameters according to chosen type, i.e., 'clr' (cumulative log-ratio) or 'tss' (total sum scaled).

Usage

```
mbecReportPrelim(
  input.obj,
  model.vars = c("batch", "group"),
  type = c("clr", "otu", "tss"),
  file.name = NULL,
  file.dir = getwd(),
  return.data = FALSE
)
```

Arguments

input.obj	list of phyloseq objects to compare, first element is considered uncorrected data
model.vars	required covariates to build models
type	One of 'otu', 'tss' or 'clr' to determine the abundance matrix to use for evaluation.
file.name	Optional file name, parameter defaults to NULL and template name will be used
file.dir	Optional output directory, parameter defaults to current working directory.
return.data	TRUE will return a list of all produced plots, FALSE will start rendering the report

Value

either a ggplot2 object or a formatted data-frame to plot from

Examples

```
data(dummy.list)
report.data <- mbecReportPrelim(input.obj=list(dummy.list$cnts[,seq(20)],
dummy.list$meta), model.vars=c("batch","group"),
type="clr", file.name=NULL, file.dir=NULL, return.data=TRUE)
```

---

mbecRLE *Relative Log Expression Plot*

Description

Takes two covariates, i.e., group and batch, and computes the RLE-plot over the grouping of the first covariate, colored by the second covariate. Effectively illustrating the relative expression between samples from different batches within the respective study groups. Other covariates can be chosen as input and the function will check for factors and convert if necessary. Categorical factors, e.g., group membership, sex and batch, produce the best result.

Usage

```
mbecRLE(
  input.obj,
  model.vars = c("batch", "group"),
  type = "clr",
  label = character(),
  return.data = FALSE
)
```

Arguments

input.obj	MbecData-object
model.vars	two covariates of interest to select by. First relates to 'batch' and the second to relevant grouping.
type	Which abundance matrix to use for the calculation.
label	Which corrected abundance matrix to use for analysis.
return.data	logical if TRUE returns the data.frame required for plotting. Default (FALSE) will return plot object.

Details

The function returns either a plot-frame or the finished ggplot object. Input is an MbecData-object. If cumulative log-ratio (clr) and total sum-scaled (tss) abundance matrices are part of the input, i.e., 'mbecTransform()' was used, they can be selected as input by using the 'type' argument with either "otu", "clr" or "tss". If batch effect corrected matrices are available, they can be used by specifying the 'type' argument as "cor" and using the 'label' argument to select the appropriate matrix by its denominator, e.g., for batch correction method ComBat this would be "bat", for RemoveBatchEffects from the limma package this is "rbe". Default correction method-labels are "ruv3", "bmc","bat","rbe","pn","svd".

The combination of 'type' and 'label' argument basically accesses the attribute 'cor', a list that stores all matrices of corrected counts. This list can also be accessed via getter and setter methods. Hence, the user can supply their own matrices with own names.

Value

Either a ggplot2 object or a formatted data-frame to plot from.

Examples

```
# This will return the data.frame for plotting.
data(dummy.mbec)
data.RLE <- mbecRLE(input.obj=dummy.mbec, type="clr",
model.vars=c('group','batch'), return.data=TRUE)

# This will return the ggplot2 object for display, saving and modification.
plot.RLE <- mbecRLE(input.obj=dummy.mbec, model.vars=c('group','batch'),
type="clr", return.data=FALSE)
```

`mbecRLEPlot` *RLE plotting function*

Description

Takes data.frame from mbecRLE and produces a ggplot2 object.

Usage

```
mbecRLEPlot(rle.df, model.vars, label = NULL)
```

Arguments

`rle.df`	'mbecRLE' data output
`model.vars`	two covariates of interest to select by first variable selects panels and second one determines coloring
`label`	Name of the plot displayed as legend title.

Value

ggplot2 object

Examples

```
# This will return a paneled plot that shows results for the variance
# assessment.
data(dummy.mbec)
rle.df <- mbecRLE(input.obj=dummy.mbec, model.vars=c('group','batch'),
type="clr", return.data=TRUE)
plot.rle <- mbecRLEPlot(rle.df, c('group','batch'))
```

---

mbecRunCorrections *Run Correction Pipeline*

Description

Run all correction algorithms selected by method and add corrected counts as matrices to the dataset.

Usage

```
mbecRunCorrections(
  input.obj,
  model.vars = c("batch", "group"),
  type = "clr",
  method = c("ruv3", "bmc", "bat", "rbe", "pn", "svd"),
  nc.features = NULL
)
```

Arguments

input.obj	Phyloseq object or a list that contains numeric matrix and meta-data table. Requires sample names as row/col-names to handle correct orientation.
model.vars	Two covariates of interest to select by first variable selects panels and second one determines coloring.
type	One of 'otu', 'tss' or 'clr' to determine the abundance matrix to use for evaluation.
method	algorithms to use
nc.features	(OPTIONAL) A vector of features names to be used as negative controls in RUV-3. If not supplied, the algorithm will use an 'lm' to find pseudo-negative controls

Value

an object of class MbecDataSet

Examples

```
# This call will use 'ComBat' for batch effect correction and store the new
# counts in a list-obj in the output.
data(dummy.mbec)
study.obj <- mbecRunCorrections(input.obj=dummy.mbec,
model.vars=c("batch","group"), method=c("bat","bmc"))

# This call will use 'Percentile Normalization' for batch effect correction
# and replace the old count matrix.
study.obj <- mbecRunCorrections(dummy.mbec, model.vars=c("batch","group"),
method=c("pn"))
```

---

mbecRUV2 *Remove unwanted Variation 2 (RUV-2)*

Description

Estimates unknown BEs by using negative control variables that, in principle, are unaffected by treatment/study/biological effect (aka the effect of interest in an experiment). These variables are generally determined prior to the experiment. An approach to RUV-2 without the presence of negative control variables is the estimation of pseudo-negative controls. To that end an lm or lmm (depending on whether or not the study design is balanced) with treatment is fitted to each feature and the significance calculated. The features that are not significantly affected by treatment are considered as pseudo-negative control variables. Subsequently, the actual RUV-2 function is applied to the data and returns the p-values for treatment, considering unwanted BEs (whatever that means).

Usage

```
mbecRUV2(
  input.obj,
  model.vars,
  type = c("clr", "otu", "tss"),
  nc.features = NULL
)
```

Arguments

input.obj	phyloseq object or numeric matrix (correct orientation is handeled internally)
model.vars	Vector of covariate names. First element relates to batch.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.
nc.features	(OPTIONAL) A vector of features names to be used as negative controls in RUV-3. If not supplied, the algorithm will use an 'lm' to find pseudo-negative controls

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a vector of p-values.

Value

A vector of p-values that indicate significance of the batch-effect for the features.

`mbecRUV3` *Remove Unwanted Variation 3 (RUV-3)*

Description

This algorithm requires negative control-features, i.e., OTUs that are known to be unaffected by the batch effect, as well as technical replicates. The algorithm will check for the existence of a replicate column in the covariate data. If the column is not present, the execution stops and a warning message will be displayed.

Usage

```
mbecRUV3(
  input.obj,
  model.vars,
  type = c("clr", "otu", "tss"),
  nc.features = NULL
)
```

Arguments

`input.obj`	phyloseq object or numeric matrix (correct orientation is handeled internally)
`model.vars`	Vector of covariate names. First element relates to batch.
`type`	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.
`nc.features`	(OPTIONAL) A vector of features names to be used as negative controls in RUV-3. If not supplied, the algorithm will use an 'lm' to find pseudo-negative controls

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a matrix of corrected abundances.

Value

A matrix of batch-effect corrected counts

mbecRUV4 *Remove Unwanted Variation 4 (RUV-4)*

Description

The updated version of RUV-2 also incorporates the residual matrix (w/o treatment effect) to estimate the unknown BEs. To that end it follows the same procedure in case there are no negative control variables and computes pseudo-controls from the data via l(m)m. As RUV-2, this algorithm also uses the parameter 'k' for the number of latent factors. RUV-4 brings the function 'getK()' that estimates this factor from the data itself. The calculated values are however not always reliable. A value of k=0 fo example can occur and should be set to 1 instead.

Usage

```
mbecRUV4(
  input.obj,
  model.vars,
  type = c("clr", "otu", "tss"),
  nc.features = NULL
)
```

Arguments

input.obj	phyloseq object or numeric matrix (correct orientation is handeled internally)
model.vars	Vector of covariate names. First element relates to batch.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.
nc.features	(OPTIONAL) A vector of features names to be used as negative controls in RUV-3. If not supplied, the algorithm will use an 'lm' to find pseudo-negative controls

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a vector of p-values.

Value

A vector of p-values that indicate significance of the batch-effect for the features.

MBECS *MBECS: Evaluation and correction of batch effects in microbiome data-sets.*

Description

The Microbiome Batch-Effect Correction Suite aims to provide a toolkit for stringent assessment and correction of batch-effects in microbiome data sets. To that end, the package offers wrapper-functions to summarize study-design and data, e.g., PCA, Heatmap and Mosaic-plots, and to estimate the proportion of variance that can be attributed to the batch effect. The function `mbecCorrection` acts as a wrapper for various batch effects correction algorithms (BECA) and in conjunction with the aforementioned tools, it can be used to compare the effectiveness of correction methods on particular sets of data. All functions of this package are accessible on their own or within the preliminary and comparative report pipelines respectively.

Pipeline

- `mbecProcessInput`
- `mbecTransform`
- `mbecReportPrelim`
- `mbecCorrection`
- `mbecRunCorrections`
- `mbecReportPost`

Exploratory functions

- `mbecRLE`
- `mbecPCA`
- `mbecBox`
- `mbecHeat`
- `mbecMosaic`

Variance functions

- `mbecModelVariance`
- `mbecVarianceStatsPlot`
- `mbecRDAStatsPlot`
- `mbecPVCAStatsPlot`
- `mbecSCOEFStatsPlot`

mbecSCOEFStatsPlot *Plot Silhouette Coefficient*

Description

The goodness of clustering assessed by the silhouette coefficient. It works with the output of 'mbecVarianceStats()' for the method 's.coef'. Format of this output is a data.frame that contains a column for every model variable and as many rows as there are features (OTUs, Genes, ..). Multiple frames may be used as input by putting them into a list - IF the data.frames contain a column named 'type', this function will use 'facet_grid()' to display side-by-side panels to enable easy comparison.

Usage

```
mbecSCOEFStatsPlot(scoef.obj)
```

Arguments

scoef.obj,	output of 'mbecVarianceStats' with method s.coef

Value

A ggplot2 dot-plot object.

Examples

```
# This will return a paneled plot that shows results for the variance
# assessment.
data(dummy.mbec)
df.var.scoef <- mbecModelVariance(input.obj=dummy.mbec,
model.vars=c('batch','group'), method='s.coef', type='clr')
plot.scoef <- mbecSCOEFStatsPlot(scoef.obj=df.var.scoef)
```

mbecSetData *Mbec-Data Setter*

Description

Sets and/or replaces selected feature abundance matrix and handles correct orientation. The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor" and assessment vectors "ass". The later two additionally require the use of the argument 'label' that specifies the name within the respective lists of corrections and assessments.

Usage

```
mbecSetData(
  input.obj,
  new.cnts = NULL,
  type = c("otu", "ass", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

input.obj	MbecData object to work on.
new.cnts	A matrix-like object with same dimension as 'otu_table' in input.obj.
type	Specify which type of data to add, by using one of 'ass' (Assessement), 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
label	For types 'ass' and 'cor' this sets the name within the lists.

Value

Input object with updated attributes.

Examples

```
# This will fill the 'tss' slot with the supplied matrix.
data(dummy.mbec, dummy.list)
MBEC.obj <- mbecSetData(input.obj=dummy.mbec, new.cnts=dummy.list$cnts,
    type='tss')

# This will put the given matrix into the list of corrected counts under the
# name "nameOfMethod".
MBEC.obj <- mbecSetData(input.obj=dummy.mbec, new.cnts=dummy.list$cnts,
    type='cor', label="nameOfMethod")
```

---

mbecSetData,MbecData-method

Mbec-Data Setter

Description

Sets and/or replaces selected feature abundance matrix and handles correct orientation. The argument type determines which slot to access, i.e. the base matrices for un-transformed counts "otu", total sum-scaled counts "tss", cumulative log-ratio transformed counts "clr" and batch effect corrected counts "cor" and assessment vectors "ass". The later two additionally require the use of the argument 'label' that specifies the name within the respective lists of corrections and assessments.

Usage

```
## S4 method for signature 'MbecData'
mbecSetData(
  input.obj,
  new.cnts = NULL,
  type = c("otu", "ass", "cor", "clr", "tss"),
  label = character()
)
```

Arguments

input.obj	MbecData object to work on.
new.cnts	A matrix-like object with same dimension as 'otu_table' in input.obj.
type	Specify which type of data to add, by using one of 'ass' (Assessement), 'cor' (Correction), 'clr' (Cumulative Log-Ratio) or 'tss' (Total Scaled-Sum).
label	For types 'ass' and 'cor' this sets the name within the lists.

Value

Input object with updated attributes.

Examples

```
# This will fill the 'tss' slot with the supplied matrix.
data(dummy.mbec, dummy.list)
MBEC.obj <- mbecSetData(input.obj=dummy.mbec, new.cnts=dummy.list$cnts,
    type='tss')

# This will put the given matrix into the list of corrected counts under the
# name "nameOfMethod".
MBEC.obj <- mbecSetData(input.obj=dummy.mbec, new.cnts=dummy.list$cnts,
    type='cor', label="nameOfMethod")
```

---

mbecSVA *Surrogate variable Analysis (SVA)*

Description

Two step approach that (1.) identify the number of latent factors to be estimated by fitting a full-model with effect of interest and a null-model with no effects. The function 'num.sv()' then calculates the number of latent factors. In the next (2.) step, the sva function will estimate the surrogate variables. And adjust for them in full/null-model . Subsequent F-test gives significance values for each feature - these P-values and Q-values are accounting for surrogate variables (estimated BEs).

Usage

```
mbecSVA(input.obj, model.vars, type = c("clr", "otu", "tss"))
```

Arguments

input.obj	MbecData object
model.vars	Vector of covariate names. First element relates to variable of interest.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.

Details

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a vector of p-values.

Value

A vector of p-values that indicate significance of the batch-effect for the features.

---

mbecSVD *Singular Value Decomposition (SVD)*

Description

Basically perform matrix factorization and compute singular eigenvectors (SEV). Assume that the first SEV captures the batch-effect and remove this effect from the data. The interesting thing is that this works pretty well. But since the SEVs are latent factors that are (most likely) confounded with other effects it is not obvious to me that this is the optimal approach to solve this issue.

Usage

```
mbecSVD(input.obj, model.vars, type = c("clr", "otu", "tss"))
```

Arguments

input.obj	phyloseq object or numeric matrix (correct orientation is handeled internally)
model.vars	Vector of covariate names. First element relates to batch.
type	Which abundance matrix to use, one of 'otu, tss, clr'. DEFAULT is 'clr'.

Details

ToDo: IF I find the time to works on "my-own-approach" then this is the point to start from!!!

The input for this function is supposed to be an MbecData object that contains total sum-scaled and cumulative log-ratio transformed abundance matrices. Output will be a matrix of corrected abundances.

Value

A matrix of batch-effect corrected counts

mbecTestModel *Check If Model Is Estimable*

Description

Applies Limma's 'nonEstimable()' to a given model and returns NULL if everything works out, or a warning and a vector of problematic covariates in case there is a problem.

Usage

```
mbecTestModel(input.obj, model.vars = NULL, model.form = NULL)
```

Arguments

input.obj	MbecData, phyloseq or list (counts, meta-data).
model.vars	Names of covariates to construct formula from.
model.form	Formula for a linear model to test.

Details

The usefull part is that you can just put in all the covariates of interest as model.vars and the function will build a simple linear model and its model.matrix for testing. You can also provide more complex linear models and the function will do the rest.

Value

Either NULL if everything is fine or a vector of strings that denote covariates and their respective problematic levels.

Examples

```
# This will return NULL because it is estimable.
data(dummy.mbec)
eval.obj <- mbecTestModel(input.obj=dummy.mbec,
model.vars=c("group","batch"))
```

mbecTransform *Normalizing Transformations*

Description

Wrapper to help perform cumulative log-ratio and total sum-scaling transformations ,adapted from packages 'mixOmics' and robCompositions' to work on matrices and Phyloseq objects alike.

Usage

```
mbecTransform(
  input.obj,
  method = c("clr", "tss"),
  offset = 0,
  required.col = NULL
)
```

Arguments

input.obj	MbecData, phyloseq, list(counts, meta-data)
method	one of 'CLR' or 'TSS'
offset	(OPTIONAL) Offset in case of sparse matrix, for DEFAULT (0) an offset will be calculated if required.
required.col	(OPTIONAL) A vector of column names in the meta-data that need to be present. Sanity check for subsequent steps.

Details

The function returns an MbecData object with transformed counts and covariate information. Input for the data-set can be of type MbecData, phyloseq or a list that contains counts and covariate data. Correct orientation of counts will be handled internally, as long as both abundance table contain sample names.

Value

MbecData with transformed counts in 'clr' and 'tss' attributes respectively.

Examples

```
# This will return the cumulative log-ratio transformed counts in an
# MbecData object.
data(dummy.mbec)
mbec.CLR <- mbecTransform(input.obj=dummy.mbec, method="clr", offset=0,
required.col=c("batch","group"))

# This will return total sum-scaled counts in an MbecData object.
mbec.CLR <- mbecTransform(input.obj=dummy.mbec, method="tss", offset=0,
required.col=c("batch","group"))
```

---

mbecUpperCase *Capitalize Word Beginning*

Description

Change the first letter of the input to uppercase. Used in plotting functions to make covariates, i.e., axis-labels look nicer.

Usage

```
mbecUpperCase(input = character())
```

Arguments

input	Any string whose first letter should be capitalized.

Value

Input with first letter capitalized

---

mbecValidateModel *Validate Linear (Mixed) Models*

Description

A helper function that calculates the collinearity between model variables and stops execution, if the maximum value is bigger than the allowed threshold.

Usage

```
mbecValidateModel(model.fit, colinearityThreshold = 0.999)
```

Arguments

model.fit	lm() or lmm() output.
colinearityThreshold	Cut-off for model rejection, I=[0,1].

Details

ToDo: maybe some additional validation steps and more informative output.

Value

No return values. Stops execution if validation fails.

Examples

```
# This will just go through if colinearity threshold is met.
data(dummy.list)
limimo <- lme4::lmer(dummy.list$cnts[,1] ~ group + (1|batch),
data=dummy.list$meta)
mbecValidateModel(model.fit=limimo, colinearityThreshold=0.999)
```

mbecVarianceStats *Wrapper for Model Variable Variance Extraction*

Description

For a Linear (Mixed) Model, this function extracts the proportion of variance that can be explained by terms and interactions and returns a named row-vector.

Usage

```
mbecVarianceStats(model.fit)
```

Arguments

model.fit	A linear (mixed) model object of class 'lm' or 'lmerMod'.

Details

Linear Model: Perform an analysis of variance (ANOVA) on the model.fit and return the Sum of squares for each term, scaled by the total sum of squares.

Linear Mixed Model: employ helper function 'mbecMixedVariance' to extract residuals, random effects and fixed effects components from the model. The components are then transformed to reflect explained proportions of variance for the model coefficients. The function implements transformation for varying coefficients as well, but NO ADJUSTMENT for single or multiple coefficients at this point.

Value

A named row-vector, containing proportional variance for model terms.

Examples

```
# This will return the data.frame for plotting.
data(dummy.list)
limo <- stats::lm(dummy.list$cnts[,1] ~ group + batch, data=dummy.list$meta)
vec.variance <- mbecVarianceStats(model.fit=limo)
```

mbecVarianceStatsLM *Model Variable Variance Extraction from LM*

Description

For a Linear Model, this function extracts the proportion of variance that can be explained by terms and interactions and returns a named row-vector.

Usage

```
mbecVarianceStatsLM(model.fit)
```

Arguments

model.fit	A linear model object of class 'lm'.

Details

Linear Model: Perform an analysis of variance (ANOVA) on the model.fit and return the Sum of squares for each term, scaled by the total sum of squares.

Value

A named row-vector, containing proportional variance for model terms.

---

mbecVarianceStatsLMM *Model Variable Variance Extraction from LMM*

Description

For a Linear Mixed Model, this function extracts the proportion of variance that can be explained by terms and interactions and returns a named row-vector.

Usage

```
mbecVarianceStatsLMM(model.fit)
```

Arguments

model.fit	A linear mixed model object of class 'lmerMod'.

Details

Linear Mixed Model: employ helper function 'mbecMixedVariance' to extract residuals, random effects and fixed effects components from the model. The components are then transformed to reflect explained proportions of variance for the model coefficients. The function implements transformation for varying coefficients as well, but NO ADJUSTMENT for single or multiple coefficients at this point.

Value

A named row-vector, containing proportional variance for model terms.

mbecVarianceStatsPlot *Plot Proportion of Variance for L(M)M*

Description

Covariate-Variances as modeled by linear (mixed) models will be displayed as box-plots. It works with the output of 'mbecVarianceStats()' for methods 'lm' and 'lmm'. Format of this output is a data.frame that contains a column for every model variable and as many rows as there are features (OTUs, Genes, ..). Multiple frames may be used as input by putting them into a list - IF the data.frames contain a column named 'type', this function will use 'facet_grid()' to display side-by-side panels to enable easy comparison.

Usage

```
mbecVarianceStatsPlot(variance.obj)
```

Arguments

variance.obj,	output of 'mbecVarianceStats' with method lm

Value

A ggplot2 box-plot object.

Examples

```
# This will return a paneled plot that shows results for the variance
# assessments.
data(dummy.mbec)
df.var.lm <- mbecModelVariance(input.obj=dummy.mbec,
model.vars=c('group','batch'), method='lm', type='clr')
plot.lm <- mbecVarianceStatsPlot(variance.obj=df.var.lm)
```

percentileNorm *Percentile Normalization*

Description

Wrapper to help perform percentile normalization on a matrix of counts. Takes counts and a dataframe of grouping variables and returns a matrix of transformed counts. This is designed (by the Developers of the procedure) to work with case/control experiments by taking the untreated group as reference and adjusting the other groupings of TRT x Batch to it.

Usage

```
percentileNorm(cnts, meta)
```

Arguments

cnts	A numeric matrix of abundances (samples x features).
meta	Data-frame of covariate columns, first column contains batches, second column contains grouping.

Details

The function returns a matrix of normalized abundances.

Value

Numeric matrix of corrected/normalized counts.

Examples

```
# This will return a matrix of normalized counts, according to the covariate
# information in meta
data(dummy.list)
mtx.pn_counts <- percentileNorm(cnts=dummy.list$cnts,
meta=dummy.list$meta[,c("batch","group")])
```

---

poscore *Percentile of Score*

Description

Helper function that calculates percentiles of scores for batch-correction method 'pn' (percentile normalization). R-implementation of Claire Duvallet's 'percentileofscore()' for python.

Usage

```
poscore(cnt.vec, cnt, type = c("rank", "weak", "strict", "mean"))
```

Arguments

cnt.vec	A vector of counts that acts as reference for score calculation.
cnt	A numeric value to calculate percentile-score for.
type	One of 'rank', 'weak', 'strict' or 'mean' to determine how the score is calculated.

Details

Calculates the number of values that bigger than reference (left) and the number of values that are smaller than the reference (right). Percentiles of scores are given in the interval $I : [0, 100]$. Depending on type of calculation, the score will be computed as follows:

`rank = (right + left + ifelse(right > left, 1, 0)) * 50/n`

`weak = right / n*100`

`strict = left / n*100`

`mean = (right + left) * 50/n)`

Value

A score for given count in relation to reference counts.

Examples

```
# This will return a score for the supplied vector with default evaluation
# (strict).
val.score <- poscore(cnt.vec=runif(100, min=0, max=100), cnt=42,
type="strict")
```

Index