Package: SIAMCAT (via r-universe)

September 28, 2024

Type Package

Title Statistical Inference of Associations between Microbial Communities And host phenoTypes

Version 2.9.0

Description Pipeline for Statistical Inference of Associations between Microbial Communities And host phenoTypes (SIAMCAT). A primary goal of analyzing microbiome data is to determine changes in community composition that are associated with environmental factors. In particular, linking human microbiome composition to host phenotypes such as diseases has become an area of intense research. For this, robust statistical modeling and biomarker extraction toolkits are crucially needed. SIAMCAT provides a full pipeline supporting data preprocessing, statistical association testing, statistical modeling (LASSO logistic regression) including tools for evaluation and interpretation of these models (such as cross validation, parameter selection, ROC analysis and diagnostic model plots).

Depends R (>= 4.2.0), mlr3, phyloseq

Imports beanplot, glmnet, graphics, grDevices, grid, gridBase, gridExtra, LiblineaR, matrixStats, methods, pROC, PRROC, RColorBrewer, scales, stats, stringr, utils, infotheo, progress, corrplot, lmerTest, mlr3learners, mlr3tuning, paradox, lgr

License GPL-3

LazyData true

Encoding UTF-8

RoxygenNote 7.3.1

biocViews ImmunoOncology, Metagenomics, Classification, Microbiome, Sequencing, Preprocessing, Clustering, FeatureExtraction, GeneticVariability, MultipleComparison,Regression

Suggests BiocStyle, testthat, knitr, rmarkdown, tidyverse, ggpubr

VignetteBuilder knitr

Repository https://bioc.r-universe.dev

RemoteUrl https://github.com/bioc/SIAMCAT

RemoteRef HEAD

RemoteSha 665aeef7301d5bf0f25ffd4e7e537bfae7aac9b8

Contents

`SIAMCAT-package` *SIAMCAT: Statistical Inference of Associations between Microbial Communities And host phenoTypes*

Description

Pipeline for Statistical Inference of Associations between Microbial Communities And host phenoTypes (SIAMCAT). A primary goal of analyzing microbiome data is to determine changes in community composition that are associated with environmental factors. In particular, linking human microbiome composition to host phenotypes such as diseases has become an area of intense research. For this, robust statistical modeling and biomarker extraction toolkits are crucially needed. SIAMCAT provides a full pipeline supporting data preprocessing, statistical association testing, statistical modeling (LASSO logistic regression) including tools for evaluation and interpretation of these models (such as cross validation, parameter selection, ROC analysis and diagnostic model plots).

Details

SIAMCAT is a pipeline for Statistical Inference of Associations between Microbial Communities And host phenoTypes. A primary goal of analyzing microbiome data is to determine changes in community composition that are associated with environmental factors. In particular, linking human microbiome composition to host phenotypes such as diseases has become an area of intense research. For this, robust statistical modeling and biomarker extraction toolkits are crucially needed!

Author(s)

Maintainer: Jakob Wirbel <jakob.wirbel@embl.de> (ORCID)

Authors:

- Konrad Zych <konrad.zych@embl.de> (ORCID)
- Georg Zeller <zeller@embl.de> (ORCID)

Other contributors:

- Morgan Essex <morgan.essex@embl.de> [contributor]
- Nicolai Karcher [contributor]
- Kersten Breuer [contributor]

`add.meta.pred` *Add metadata as predictors*

Description

This function adds metadata to the feature matrix to be later used as predictors

Usage

```
add.meta.pred(siamcat, pred.names, std.meta = TRUE,
feature.type='normalized', verbose = 1)
```

Arguments

`siamcat`	object of class siamcat-class
`pred.names`	vector of names of the variables within the metadata to be added to the feature matrix as predictors
`std.meta`	boolean, should added metadata features be standardized?, defaults to `TRUE`
`feature.type`	string, on which type of features should the function work? Can be either `"original"`, `"filtered"`, or `"normalized"`. Please only change this paramter if you know what you are doing!
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This functions adds one or several metadata variables to the set of features, so that they can be included for model training.

Usually, this function should be called before train.model.

Numerical meta-variables are added as z-scores to the feature matrix unless specified otherwise.

Please be aware, that non-numerical metadata variables will be converted to numerical values by using `as.numeric()` and could therefore lead to errors. Thus, it makes sense to encode non-numerical metadata variables to numerically before you start the SIAMCAT workflow.

Value

an object of class siamcat-class with metadata added to the features

Examples

```
data(siamcat_example)

# Add the Age of the patients as potential predictor
siamcat_age_added <- add.meta.pred(siamcat_example, pred.names=c('Age'))
```

```
# Add Age and BMI as potential predictors
# Additionally, prevent standardization of the added features
siamcat_meta_added <- add.meta.pred(siamcat_example,
    pred.names=c('Age', 'BMI'), std.meta=FALSE)
```

`association.plot` *Visualize associations between features and classes*

Description

This function visualizes different measures of association between features and the label, computed previously with the check.associations function

Usage

```
association.plot(siamcat, fn.plot=NULL, color.scheme = "RdYlBu",
sort.by = "fc", max.show = 50, plot.type = "quantile.box",
panels = c("fc", "auroc"), prompt=TRUE, verbose = 1)
```

Arguments

`siamcat`	object of class siamcat-class
`fn.plot`	string, filename for the pdf-plot. If `fn.plot` is `NULL`, the plot will be produced in the active graphics device.
`color.scheme`	valid R color scheme or vector of valid R colors (must be of the same length as the number of classes), defaults to `'RdYlBu'`
`sort.by`	string, sort features by p-value (`"p.val"`), by fold change (`"fc"`) or by prevalence shift (`"pr.shift"`), defaults to `"fc"`
`max.show`	integer, how many associated features should be shown, defaults to `50`
`plot.type`	string, specify how the abundance should be plotted, must be one of these: `c("bean", "box", "quantile.box", "quantile.rect")`, defaults to `"quantile.box"`
`panels`	vector, name of the panels to be plotted next to the abundances, possible entries are `c("fc", "auroc", "prevalence")`, defaults to `c("fc", "auroc")`
`prompt`	boolean, turn on/off prompting user input when not plotting into a pdf-file, defaults to TRUE
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This function visualizes the results of the computations carried out in the check.associations function. It produces a plot of the top `max.show` associated features at a user-specified significance level `alpha`.

For binary classification problems, the plot will show the distribution of the log10-transformed abundances for both classes, a P-value from the significance test, and user-selected panels for the effect size (AU-ROC, prevalence shift, or generalized fold change). For regression problems, the plot will show the Spearman correlation, the significance, and the linear model effect size.

Value

Does not return anything, but instead produces association plot

Examples

```
# Example data
data(siamcat_example)

# Simple example
association.plot(siamcat_example, fn.plot = "./assoc_plot.pdf")

# Plot associations as box plot
association.plot(siamcat_example,
    fn.plot = "./assoc_plot_box.pdf",
    plot.type = "box")

# Additionally, sort by p-value instead of by fold change
association.plot(siamcat_example,
    fn.plot = "./assoc_plot_fc.pdf",
    plot.type = "box", sort.by = "p.val")

# Custom colors
association.plot(siamcat_example,
    fn.plot = "./assoc_plot_blue_yellow.pdf",
    plot.type = "box", color.scheme = c("cornflowerblue", "#ffc125"))
```

`associations` *Retrieve the results of association testing from a SIAMCAT object*

Description

Function to retrieve the results of association testing

Usage

```
associations(siamcat, verbose=1)

## S4 method for signature 'siamcat'
associations(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class containing the results of association testing
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function returns the results of the association testing procedure as dataframe. See check.associations for more details.

Value

A `data.frame` of association testing results or `NULL`

Examples

```
data(siamcat_example)
temp <- associations(siamcat_example)
head(temp)
```

---

assoc_param — *Retrieve the list of parameters for association testing from a SIAMCAT object*

Description

Function to retrieve the list of parameters for association testing

Usage

```
assoc_param(siamcat, verbose=1)

## S4 method for signature 'siamcat'
assoc_param(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class containing the results from association testing
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function returns the list of parameters used in association testing. See check.associations for more details.

Value

A list of parameters for association testing or NULL

Examples

```
data(siamcat_example)
temp <- assoc_param(siamcat_example)
names(temp)
```

---

check.associations *Calculate associations between features and labels*

Description

This function computes different measures of association between features and the label and stores the results in the association slot of the SIAMCAT object

Usage

```
check.associations(siamcat, formula="feat~label", test='wilcoxon',
alpha=0.05, mult.corr="fdr", log.n0=1e-06, pr.cutoff=1e-06,
probs.fc=seq(.1, .9, .05), paired=NULL, feature.type='filtered',
verbose = 1)
```

Arguments

siamcat	object of class siamcat-class
formula	string, formula used for testing, see Details for more information, defaults to "feat~label"
test	string, statistical test used for the association testing, can be either 'wilcoxon' or 'lm', see Details for more information, defaults to 'wilcoxon'
alpha	float, significance level, defaults to 0.05
mult.corr	string, multiple hypothesis correction method, see p.adjust, defaults to "fdr"
log.n0	float, pseudo-count to be added before log-transformation of the data, defaults to 1e-06. Will be ignored if feature.type is "normalized".
pr.cutoff	float, cutoff for the prevalence computation, defaults to 1e-06
probs.fc	numeric vector, quantiles used to calculate the generalized fold change between groups, see Details for more information, defaults to seq(.1, .9, .05)
paired	character, column name of the meta-variable containing information for a paired test, defaults to NULL
feature.type	string, on which type of features should the function work? Can be either c("original", "filtered", or "normalized"). Please only change this parameter if you know what you are doing! If feature.type is "normalized", the normalized abundances will not be log10-transformed.

`verbose` integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Value

object of class siamcat-class with the slot `associations` filled

Statistical testing

The function uses the Wilcoxon test as default statistical test for binary classification problems. Alternatively, a simple linear model (as implemented in lm) can be used as well. For regression problems, the function defaults to the linear model.

Effect sizes

The function calculates several measures for the effect size of the assocations between microbial features and the label. For binary classification problems, these associations are:

- AUROC (area under the Receiver Operating Characteristics curve) as a non-parametric measure of enrichment,
- the generalized fold change (gFC), a pseudo-fold change which is calculated as geometric mean of the differences between quantiles across both groups,
- prevalence shift (difference in prevalence between the two groups).

For regression problems, the effect sizes are:

- Spearman correlation between the feature and the label.

Confounder-corrected testing

To correct for possible confounders while testing for association, the function uses linear mixed effect models as implemented in the lmerTest package. To do so, the test formula needs to be adjusted to include the confounder. For example, when correcting for the metadata information `Sex`, the formula would be: `'feat~label+(1|Sex)'` (see also the example below).

Please note that modifying the formula parameter in this function might lead to unexpected results!

Paired testing

For paired testing, e.g. when the same patient has been sampled before and after an intervention, the ‘paired‘ parameter can be supplied to the function. This indicated a column in the metadata table that holds the information about pairing.

Examples

```
# Example data
data(siamcat_example)

# Simple example
siamcat_example <- check.associations(siamcat_example)
```

```
# Confounder-corrected testing (corrected for Sex)
#
# this is not run during checks
# siamcat_example <- check.associations(siamcat_example,
#     formula='feat~label+(1|Sex)', test='lm')

# Paired testing
#
# this is not run during checks
# siamcat_paired <- check.associations(siamcat_paired,
#     paired='Individual_ID')
```

`check.confounders` *Check for potential confounders in the metadata*

Description

Checks potential confounders in the metadata and visualize the results

Usage

```
check.confounders(siamcat, fn.plot, meta.in = NULL,
feature.type='filtered', verbose = 1)
```

Arguments

`siamcat`	an object of class siamcat-class
`fn.plot`	string, filename for the pdf-plot
`meta.in`	vector, specific metadata variable names to analyze, defaults to NULL (all metadata variables will be analyzed)
`feature.type`	string, on which type of features should the function work? Can be either `c()"original", "filtered", or "normalized")`. Please only change this paramter if you know what you are doing!
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This function checks for associations between class labels and potential confounders (e.g. Age, Sex, or BMI) that are present in the metadata. Statistical testing is performed with Fisher's exact test or Wilcoxon test, while associations are visualized either as barplot or Q-Q plot, depending on the type of metadata.

Additionally, it evaluates associations among metadata variables using conditional entropy and associations with the label using generalized linear models, producing a correlation heatmap and appropriate quantitative barplots, respectively.

Please note that the confounder check is currently only available for binary classification problems!

Value

Does not return anything, but outputs plots to specified pdf file

Examples

```
# Example data
data(siamcat_example)

# Simple working example
check.confounders(siamcat_example, './conf_plot.pdf')
```

---

create.data.split *Split a dataset into training and a test sets.*

Description

This function prepares the cross-validation by splitting the data into `num.folds` training and test folds for `num.resample` times.

Usage

```
create.data.split(siamcat, num.folds = 2, num.resample = 1,
stratify = TRUE, inseparable = NULL, verbose = 1)
```

Arguments

`siamcat`	object of class siamcat-class
`num.folds`	integer number of cross-validation folds (needs to be >=2), defaults to 2
`num.resample`	integer, resampling rounds (values <= 1 deactivate resampling), defaults to 1
`stratify`	boolean, should the splits be stratified so that an equal proportion of classes are present in each fold?, will be ignored for regression tasks, defaults to `TRUE`
`inseparable`	string, name of metadata variable to be inseparable, defaults to `NULL`, see Details below
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This function splits the labels within a siamcat-class object and prepares the internal cross-validation for the model training (see train.model).

The function saves the training and test instances for the different cross-validation folds within a list in the `data_split`-slot of the siamcat-class object, which is a list with four entries:

- `num.folds` - the number of cross-validation folds
- `num.resample` - the number of repetitions for the cross-validation

- `training.folds` - a list containing the indices for the training instances
- `test.folds` - a list containing the indices for the test instances

If provided, the data split will take into account a metadata variable for the data split (by providing the `inseparable` argument). For example, if the data contains several samples for the same individual, it makes sense to keep data from the same individual within the same fold.

If `inseparable` is given, the `stratify` argument will be ignored.

Value

object of class siamcat-class with the `data_split`-slot filled

Examples

```
data(siamcat_example)

# simple working example
siamcat_split <- create.data.split(siamcat_example, num.folds=10,
num.resample=5, stratify=TRUE)
```

`create.label` *Create a label list*

Description

This function creates a label object from metadata or an atomic vector

Usage

```
create.label(label, case, meta=NULL, control=NULL,
p.lab = NULL, n.lab = NULL, remove.meta.column=FALSE, verbose=1)
```

Arguments

`label`	named vector to create the label or the name of the metadata column that will be used to create the label
`case`	name of the group that will be used as a positive label. If the variable is binary, the other label will be used as a negative one. If the variable has multiple values, all the other values will be used a negative label (testing one vs rest).
`meta`	metadata dataframe object or an object of class sample_data-class
`control`	name of a label or vector with names that will be used as a negative label. All values that are nor equal to case and control will be dropped. Default to NULL in which case: If the variable is binary, the value not equal to case will be used as negative. If the variable has multiple values, all the values not equal to cases will be used a negative label (testing one vs rest).
`p.lab`	name of the positive group (useful mostly for visualizations). Default to NULL in which case the value of the positive group will be used.

`n.lab` name of the negative group (useful mostly for visualizations). Default to NULL in which case the value of the negative group will be used for binary variables and "rest" will be used for variables with multiple values.

`remove.meta.column` boolean indicating if the label column in the metadata should be retained. Please note that if this is set to `TRUE`, the function will return a list as result. Defaults to `FALSE`

`verbose` integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

The function creates a list to be used as label in a SIAMCAT object. Mainly for interal use, but it can be used to customize your label (`p.lab` and `n.lab` will be used as labels during plotting, for example).

The input for the function can be either a named vector encoding the label or the name of a column in the metadata (needs to be provided as well) which contains the label information.

Value

return either

- a list to be used in a SIMCAT object **OR**
- a list with entries `meta` and `label`, if `remove.meta.column` is set to `TRUE`

Examples

```
data('meta_crc_zeller')

label <- create.label(label='Group', case='CRC', meta=meta.crc.zeller)
```

`data_split` *Retrieve the data split from a SIAMCAT object*

Description

Function to retrieve the data split stored in the `data_split` slot within a SIAMCAT object

Usage

```
data_split(siamcat, verbose=1)

## S4 method for signature 'siamcat'
data_split(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class containing a data split
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function returns a list containing information about the data split. See create.data.split for more details.

Value

A list containing the data split information or `NULL`

Examples

```
data(siamcat_example)
temp <- data_split(siamcat_example)
names(temp)
```

`evaluate.predictions` *Evaluate prediction results*

Description

This function compares the predictions (from [make.predictions]) and true labels for all samples and evaluates the results.

Usage

```
evaluate.predictions(siamcat, verbose = 1)
```

Arguments

`siamcat`	object of class siamcat-class
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Value

object of class siamcat-class with the slot `eval_data` filled

Binary classification problems

This function calculates several metrics for the predictions in the `pred_matrix`-slot of the siamcat-class-object. The Area Under the Receiver Operating Characteristic (ROC) Curve (AU-ROC) and the Precision-Recall Curve will be evaluated and the results will be saved in the `eval_data`-slot of the supplied siamcat-class- object. The `eval_data`-slot contains a list with several entries:

- `$roc` - average ROC-curve across repeats or a single ROC-curve on complete dataset (see roc);
- `$auroc` - AUC value for the average ROC-curve;
- `$prc` - list containing the positive predictive value (precision) and true positive rate (recall) values used to plot the mean PR curve;
- `$auprc` - AUC value for the mean PR curve;
- `$ev` - list containing for different decision thresholds the number of false positives, false negatives, true negatives, and true positives.

For the case of repeated cross-validation, the function will additionally return

- `$roc.all` - list of roc objects (see roc) for every repeat;
- `$auroc.all` - vector of AUC values for the ROC curves for every repeat;
- `$prc.all` - list of PR curves for every repeat;
- `$auprc.all` - vector of AUC values for the PR curves for every repeat;
- `$ev.all` - list of `ev` lists (see above) for every repeat.

Regression problems

This function calculates several metrics for the evaluation of predictions and will store the results in the `eval_data`-slot of the supplied siamcat-class objects. The `eval_data`-slot will contain:

- `r2` - the mean R squared value across repeats or a single R-squared value on the complete dataset;
- `mae` - them mean absolute error of the predictions;
- `mse` - the mean squared error of the predictions.

For the case of repeated cross-validation, the function will additionally compute all three of these measures for the individual cross-validation repeats and will store the results in the `eval_data` slot as `r2.all`, `mae.all`, and `mse.all`.

Examples

```
data(siamcat_example)

siamcat_evaluated <- evaluate.predictions(siamcat_example)
```

eval_data — *Retrieve the evaluation metrics from a SIAMCAT object*

Description

Function to retrieve the evaluation metrics from a SIAMCAT object

Usage

```
eval_data(siamcat, verbose=1)

## S4 method for signature 'siamcat'
eval_data(siamcat, verbose = 1)
```

Arguments

siamcat	(Required). A siamcat-class object that contains evaluation data
verbose	integer, if the slot is empty, should a message be printed? values can be either 0 (no output) or 1 (print message)

Details

The functions returns a list containing the evaluation metrics from a SIAMCAT object. See evaluate.predictions for more information on evaluation data.

Value

The list of evaluation data or NULL

Examples

```
data(siamcat_example)
temp <- eval_data(siamcat_example)
names(temp)
temp$auroc
```

feat.crc.zeller — *Example feature matrix*

Description

Feature matrix (as data.frame) of the CRC dataset from Zeller et al. MSB 2014 (see http://msb.embopress.org/content/10/11/766), containing 141 samples and 1754 bacterial species (features).

Source

http://msb.embopress.org/content/10/11/766

feature_type — *Retrieve the feature type used for model training from a SIAMCAT object*

Description

Function to retrieve information on which type of features the models were trained

Usage

```
feature_type(siamcat, verbose=1)

## S4 method for signature 'siamcat'
feature_type(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class that contains trained models
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function extracts the information on which type of features the models were trained.

Value

The string describing type of feature used for the model training or `NULL`

Examples

```
data(siamcat_example)
feature_type(siamcat_example)
```

feature_weights — *Retrieve the matrix of feature weights from a SIAMCAT object*

Description

Function to extract the feature weights from a SIAMCAT object

Usage

```
feature_weights(siamcat, verbose=1)

## S4 method for signature 'siamcat'
feature_weights(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). A siamcat-class object that contains trained models
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function extracts the weight matrix from all trained models (see weight_matrix) and computes several metrics on the feature weights:

- `mean.weight` - mean weight across trained models
- `median.weight` - median weight across trained models
- `sd.weight` - standard deviation of the weight across trained models
- `mean.rel.weight` - mean **relative** weight across trained models (each model is normalized by the absolute of all weights)
- `median.rel.weight` - median **relative** weight across trained models
- `sd.rel.weight` - standard deviation of the **relative** weight across trained models
- `percentage` - percentage of models in which this feature was selected (i.e. non-zero)

Value

A dataframe containing mean/median feature weight and additional info or `NULL`

Examples

```
data(siamcat_example)
temp <- feature_weights(siamcat_example)
head(temp)
```

`filter.features` *Perform unsupervised feature filtering.*

Description

This function performs unsupervised feature filtering.

Usage

```
filter.features(siamcat, filter.method = "abundance",
cutoff = 0.001, rm.unmapped = TRUE, feature.type='original', verbose = 1)
```

Arguments

`siamcat`	an object of class siamcat-class
`filter.method`	string, method used for filtering the features, can be one of these: `c('abundance', 'cum.abundance', 'prevalence','variance', 'pass')`, defaults to `'abundance'`
`cutoff`	float, abundace, prevalence, or variance cutoff, defaults to `0.001` (see Details below)
`rm.unmapped`	boolean, should unmapped reads be discarded?, defaults to `TRUE`
`feature.type`	string, on which type of features should the function work? Can be either `"original"`, `"filtered"`, or `"normalized"`. Please only change this paramter if you know what you are doing!
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This function filters the features in a siamcat-class object in a unsupervised manner.

The different filter methods work in the following way:

- `'abundace'` - remove features whose maximum abundance is never above the threshold value in any of the samples
- `'cum.abundance'` - remove features with very low abundance in all samples, i.e. those that are never among the most abundant entities that collectively make up (1-cutoff) of the reads in any sample
- `'prevalence'` - remove features with low prevalence across samples, i.e. those that are undetected (relative abundance of 0) in more than `1 - cutoff` percent of samples.
- `'variance'` - remove features with low variance across samples, i.e. those that have a variance lower than `cutoff`
- `'pass'` - pass-through filtering will not change the features

Features can also be filtered repeatedly with different methods, e.g. first using the maximum abundance filtering and then using prevalence filtering. However, if a filtering method has already been applied to the dataset, SIAMCAT will default back on the original features for filtering.

Value

siamcat an object of class siamcat-class

Examples

```
# Example dataset
data(siamcat_example)

# Simple examples
siamcat_filtered <- filter.features(siamcat_example,
    filter.method='abundance',
    cutoff=1e-03)
```

```
# 5% prevalence filtering
siamcat_filtered <- filter.features(siamcat_example,
    filter.method='prevalence',
    cutoff=0.05)

# filter first for abundance and then for prevalence
siamcat_filt <- filter.features(siamcat_example,
    filter.method='abundance', cutoff=1e-03)
siamcat_filt <- filter.features(siamcat_filt, filter.method='prevalence',
    cutoff=0.05, feature.type='filtered')
```

`filter.label` *Filter the label of a SIMACAT object*

Description

This functions filters the label in a SIAMCAT object

Usage

```
filter.label(siamcat, ids, verbose = 1)
```

Arguments

`siamcat`	an object of class siamcat-class
`ids`	vector, can contain either names or indices of samples to be retained
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This function filters the label contained in a SIAMCAT object, based on the provided `ids`. The IDs can be either sample names or indices to be retained.

Predominantly for internal use...

Please note: It makes sense to run validate.data after filtering the label.

Value

siamcat an object of class siamcat-class

Examples

```
data(siamcat_example)

# simple working example
siamcat_filtered <- filter.label(siamcat_example, ids=c(1:20))
```

filt_params — *Retrieve the list of parameters for feature filtering from a SIAMCAT object*

Description

Function to retrieve the list of parameters for feature filtering

Usage

```
filt_params(siamcat, verbose=1)

## S4 method for signature 'siamcat'
filt_params(siamcat, verbose = 1)
```

Arguments

siamcat	(Required). An instance of siamcat-class containing filtered features
verbose	integer, if the slot is empty, should a message be printed? values can be either 0 (no output) or 1 (print message)

Details

The function returns the list of feature filtering parameters. See filter.features for more details.

Value

A list of feature filtering parameters or NULL

Examples

```
data(siamcat_example)
temp <- filt_params(siamcat_example)
names(temp)
```

get.filt_feat.matrix — *Retrieve the filtered features from a SIAMCAT object*

Description

Function to retrieve the filtered features from a SIAMCAT object

Usage

```
get.filt_feat.matrix(siamcat)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class containing filtered features

Details

The function returns the filtered features as matrix. See filter.features for more details.

Value

A matrix containing the filtered features

Examples

```
data(siamcat_example)
feat.filt <- get.filt_feat.matrix(siamcat_example)
feat.filt[1:3, 1:3]
```

`get.norm_feat.matrix` *Retrieve the normalized features from a SIAMCAT object*

Description

Function to retrieve the normalized features from a SIAMCAT object

Usage

```
get.norm_feat.matrix(siamcat)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class containing normalized features

Details

The function returns the normalized features as matrix. See normalize.features for more details.

Value

A matrix containing the normalized features

Examples

```
data(siamcat_example)
feat.norm <- get.norm_feat.matrix(siamcat_example)
feat.norm[1:3, 1:3]
```

get.orig_feat.matrix *Retrieve the original features from a SIAMCAT object*

Description

Function to retrieve the original features from a SIAMCAT object

Usage

```
get.orig_feat.matrix(siamcat)
```

Arguments

siamcat	(Required). An instance of siamcat-class

Details

The function returns the original features as matrix.

Value

A matrix containing the original features

Examples

```
data(siamcat_example)
feat.original <- get.orig_feat.matrix(siamcat_example)
feat.original[1:3, 1:3]
```

label *Retrieve the label from a SIAMCAT object*

Description

Retrieve the label from a SIAMCAT object

Usage

```
label(siamcat, verbose=1)

## S4 method for signature 'siamcat'
label(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). A siamcat-class object
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

This function will retrieve the label information from a SIAMCAT object. The label will contain three entries:

- `label`: The label as named vector, in which the classes are encoded numerically
- `info`: Information about the different classes
- `type`: What kind of label is it?

Value

The label or NULL.

Examples

```
data(siamcat_example)
temp <- label(siamcat_example)
head(temp$label)
temp$info
temp$type
```

`make.predictions` *Make predictions on a test set*

Description

This function takes a siamcat-class-object containing a model trained by train.model and performs predictions on a given test-set.

Usage

```
make.predictions(siamcat, siamcat.holdout = NULL,
normalize.holdout = TRUE, verbose = 1)
```

Arguments

`siamcat`	object of class siamcat-class
`siamcat.holdout`	optional, object of class siamcat-class on which to make predictions, defaults to `NULL`

`normalize.holdout`
: boolean, should the holdout features be normalized with a frozen normalization (see normalize.features) using the normalization parameters in `siamcat`?, defaults to `TRUE`

`verbose`
: integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This functions uses the model in the `model_list`-slot of the `siamcat` object to make predictions on a given test set. The test set can either consist of the test instances in the cross-validation, saved in the `data_split`-slot of the same `siamcat` object, or a completely external feature set, given in the form of another `siamcat` object (`siamcat.holdout`).

Value

object of class siamcat-class with the slot `pred_matrix` filled

Examples

```
data(siamcat_example)

# Simple example
siamcat_example <- train.model(siamcat_example, method='lasso')
siamcat.pred <- make.predictions(siamcat_example)

# Predictions on a holdout-set (not run)
# pred.mat <- make.predictions(siamcat.trained, siamcat.holdout,
#     normalize.holdout=TRUE)
```

`meta` *Retrieve the metadata from a SIAMCAT object*

Description

Retrieve the metadata from a SIAMCAT object

Usage

```
meta(siamcat)

## S4 method for signature 'siamcat'
meta(siamcat)

## S4 method for signature 'sample_data'
meta(siamcat)
```

Arguments

`siamcat` (Required). A siamcat-class object

Details

This function will retrieve the metadata from a SIAMCAT object. The metadata is a object of the sample_data-class.

Value

The metadata as sample_data-class object

Examples

```
data(siamcat_example)
temp <- meta(siamcat_example)
head(temp)
```

---

meta.crc.zeller *Example metadata matrix*

Description

Metadata (as data.frame) of the CRC dataset from Zeller et al. MSB 2014 (see http://msb.embopress.org/content/10/11/766), containing 6 metadata variables variables (e.g. Age or BMI) for 141 samples.

Source

http://msb.embopress.org/content/10/11/766

---

model.evaluation.plot *Model Evaluation Plot*

Description

Produces plots for model evaluation.

Usage

```
model.evaluation.plot(..., fn.plot = NULL,
colours=NULL, show.all=FALSE, verbose = 1)
```

Arguments

`...`	one or more object of class siamcat-class, can be named
`fn.plot`	string, filename for the pdf-plot
`colours`	colour specification for the different siamcat-class- objects, defaults to `NULL` which will cause the colours to be picked from the `'Set1'` palette
`show.all`	boolean, Should the results from repeated cross-validation models be plotted? Defaults to `FALSE`, leading to a single line for the mean across cross-valdiation repeats
`verbose`	control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Value

Does not return anything, but produces the model evaluation plot.

Binary classification problems

The first plot shows the Receiver Operating Characteristic (ROC)-curve, the other plot the Precision-recall (PR)-curve for the model. If `show.all == FALSE` (which is the default), a single line representing the mean across cross-validation repeats will be plotted, otherwise the individual cross-validation repeats will be included as lightly shaded lines.

Regression problems

For regression problems, this function will produce a scatter plot between the real and predicted values. If several siamcat-class-objects are supplied, a single plot for each object will be produced.

Examples

```
data(siamcat_example)

# simple working example
model.evaluation.plot(siamcat_example, fn.plot='./eval.pdf')

# plot several named SIAMCAT object
# although we use only one example object here
model.evaluation.plot('Example_1'=siamcat_example,
    'Example_2'=siamcat_example, colours=c('red', 'blue'),
    fn.plot='./eval.pdf')

# show indiviudal cross-validation repeats
model.evaluation.plot(siamcat_example, fn.plot='./eval.pdf', show.all=TRUE)
```

model.interpretation.plot

Model Interpretation Plot

Description

This function produces a plot for model interpretation

Usage

```
model.interpretation.plot(siamcat, fn.plot = NULL,
color.scheme = "BrBG", consens.thres = 0.5, heatmap.type = "zscore",
limits = c(-3, 3), log.n0 = 1e-06, max.show = 50, prompt=TRUE,
verbose = 1)
```

Arguments

`siamcat`	object of class siamcat-class
`fn.plot`	string, filename for the pdf-plot
`color.scheme`	color scheme for the heatmap, defaults to `'BrBG'`
`consens.thres`	float, minimal ratio of models incorporating a feature in order to include it into the heatmap, defaults to `0.5` **Note that for** `'randomForest'` **models, this cutoff specifies the minimum median Gini coefficient for a feature to be included and should therefore be much lower, e.g.** `0.01`
`heatmap.type`	string, type of the heatmap, can be either `'fc'` or `'zscore'`, defaults to `'zscore'`
`limits`	vector, cutoff for extreme values in the heatmap, defaults to `c(-3, 3)`
`log.n0`	float, pseudocount to be added before log-transformation of features, defaults to `1e-06`
`max.show`	integer, maximum number of features to be shown in the model interpretation plot, defaults to 50
`prompt`	boolean, turn on/off prompting user input when not plotting into a pdf-file, defaults to TRUE
`verbose`	control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

Produces a plot consisting of

- a barplot showing the feature weights and their robustness (i.e. in what proportion of models have they been incorporated)
- a heatmap showing the z-scores of the metagenomic features across samples
- another heatmap displaying the metadata categories (if applicable)
- a boxplot displaying the poportion of weight per model that is actually shown for the features that are incorporated into more than `consens.thres` percent of the models.

Value

Does not return anything, but produces the model interpretation plot.

Examples

```
data(siamcat_example)

# simple working example
siamcat_example <- train.model(siamcat_example, method='lasso')
model.interpretation.plot(siamcat_example, fn.plot='./interpretion.pdf',
    heatmap.type='zscore')
```

---

models — *Retrieve list of trained models from a SIAMCAT object*

Description

Function to retrieve the list of trained models

Usage

```
models(siamcat, verbose=1)

## S4 method for signature 'siamcat'
models(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class that contains trained models
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function extracts the list of trained models.

Value

The list of models or `NULL`

Examples

```
data(siamcat_example)
temp <- models(siamcat_example)
temp[[1]]
```

model_type — *Retrieve the machine learning method from a SIAMCAT object*

Description

Function to retrieve information on which type of machine learning method was used for model training

Usage

```
model_type(siamcat, verbose=1)

## S4 method for signature 'siamcat'
model_type(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class that contains trained models
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function extracts the information on which type of machine learning method was used for model training.

Value

The string describing the machine learning method or `NULL`

Examples

```
data(siamcat_example)
model_type(siamcat_example)
```

normalize.features — *Perform feature normalization*

Description

This function performs feature normalization according to user-specified parameters.

Usage

```
normalize.features(siamcat, norm.method = c("rank.unit", "rank.std",
"log.std", "log.unit", "log.clr", "std", "pass"),
norm.param = list(log.n0 = 1e-06, sd.min.q = 0.1, n.p = 2, norm.margin = 1),
feature.type='filtered', verbose = 1)
```

Arguments

`siamcat`	an object of class siamcat-class
`norm.method`	string, normalization method, can be one of these: `c('rank.unit', 'rank.std', 'log.std', 'log.unit', 'log.clr','std','pass')`
`norm.param`	list, specifying the parameters of the different normalization methods, see Details for more information
`feature.type`	string, on which type of features should the function work? Can be either `"original"`, `"filtered"`, or `"normalized"`. Please only change this paramter if you know what you are doing!
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Value

an object of class siamcat-class with normalized features

Implemented methods

There are seven different normalization methods available, which might need additional parameters, which are passed via the `norm.param` list:

- `'rank.unit'` - converts features to ranks and normalizes each column (=sample) by the square root of the sum of ranks This method does not require additional parameters.
- `'rank.std'` - converts features to ranks and applies z-score standardization. This method requires `sd.min.q` (minimum quantile of the standard deviation to be added to all features in order to avoid underestimation of standard deviation) as additional parameter.
- `'log.clr'` - centered log-ratio transformation. This methods requires a pseudocount (`log.n0`) before log-transformation.
- `'log.std'` - log-transforms features and applies z-score standardization. This method requires both a pseudocount (`log.n0`) and `sd.min.q`
- `'log.unit'` - log-transforms features and normalizes by features or samples with different norms. This method requires a pseudocount (`log.n0`) and then additionally the parameters `norm.maring` (margin over which to normalize, similarly to the `apply`-syntax: Allowed values are `1` for normalization over features, `2` over samples, and `3` for normalization by the global maximum) and the parameter `n.p` (vector norm to be used, can be either `1` for `x/sum(x)` or `2` for `x/sqrt(sum(x^2))`).
- `'std'` - z-score standardization without any other transformation This method only requires the `sd.min.q` parameter
- `'pass'` - pass-through normalization will not change the features

Frozen normalization

The function additionally allows to perform a frozen normalization on a different dataset. After normalizing the first dataset, the `norm_feat` slot in the SIAMCAT object contains all parameters of the normalization, which you can access via the norm_params accessor.

In order to perform a frozen normalization of a new dataset, you can run the function supplying the normalization parameters as argument to `norm.param`: `norm.param=norm_params(siamcat_reference)`. See also the example below.

Examples

```
# Example data
data(siamcat_example)

# Simple example
siamcat_norm <- normalize.features(siamcat_example,
    norm.method='rank.unit')

# log.unit example
siamcat_norm <- normalize.features(siamcat_example,
    norm.method='log.unit',
    norm.param=list(log.n0=1e-05, n.p=1, norm.margin=1))

# log.std example
siamcat_norm <- normalize.features(siamcat_example,
    norm.method='log.std',
    norm.param=list(log.n0=1e-05, sd.min.q=.1))

# Frozen normalization
# normalize the object siamcat with the same parameters as used in
# siamcat_reference
#
# this is not run
# siamcat_norm <- normalize.features(siamcat,
#   norm.param=norm_params(siamcat_reference))
```

---

norm_params — *Retrieve the list of parameters for feature normalization from a SIAMCAT object*

Description

Function to retrieve the list of parameters for feature normalization

Usage

```
norm_params(siamcat, verbose=1)

## S4 method for signature 'siamcat'
norm_params(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class containing normalized features
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function returns the list of normalization parameters used in the feature normalization procedure. See normalize.features for more details.

Value

A list of normalization parameters or `NULL`

Examples

```
data(siamcat_example)
temp <- norm_params(siamcat_example)
names(temp)
```

---

pred_matrix *Retrieve the prediction matrix from a SIAMCAT object*

Description

Function to retrieve the prediction matrix from a SIAMCAT object

Usage

```
pred_matrix(siamcat, verbose=1)

## S4 method for signature 'siamcat'
pred_matrix(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). A siamcat-class object that contains a prediction matrix
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The functions returns a matrix containing the predictions for all samples across the different cross-validation repeats. See make.predictions for more information.

Value

A matrix containing predictions or `NULL`

Examples

```
data(siamcat_example)
temp <- pred_matrix(siamcat_example)
head(temp)
```

`read.label` *Read label file*

Description

Read label information from a file

Usage

```
read.label(fn.in.label)
```

Arguments

`fn.in.label`	name of the tsv file containing labels

Details

This function reads in a tsv file with labels and converts it into a label.

First row is expected to be

`#BINARY:1=[label for cases]; -1=[label for controls]`.

Second row should contain the sample identifiers as tab-separated list (consistent with feature and metadata).

Third row is expected to contain the actual class labels (tab-separated): `1` for each case and `-1` for each control.

Note: Labels can take other numeric values (but not characters or strings); importantly, the label for cases has to be greater than the one for controls

Value

label object containing several entries:

- `$label` named vector containing the numerical labels from the file;
- `$info` information about the classes in the label;
- `$type` information about the label type (e.g. `BINARY`);

Examples

```
# run with example data
fn.label <- system.file('extdata',
    'label_crc_zeller_msb_mocat_specI.tsv',
    package = 'SIAMCAT')

crc.zeller.label <- read.label(fn.label)
```

select.samples — *Select samples based on metadata*

Description

This function select samples based on information given in the metadata

Usage

```
select.samples(siamcat, filter, allowed.set = NULL,
allowed.range = NULL, verbose = 1)
```

Arguments

`siamcat`	an object of class siamcat-class
`filter`	string, name of the meta variable on which the selection should be done
`allowed.set`	a vector of allowed values
`allowed.range`	a range of allowed values
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

This functions selects labels and metadata based on a specific column in the metadata. Provided with a column-name in the metadata and a range or a set of allowed values, the function will filter the siamcat-class object accordingly.

Value

an object of class siamcat-class with labels and metadata filtered in order to contain only allowed values

Examples

```
data(siamcat_example)

# Select all samples that fall into an Age-range between 25 and 80 years
siamcat_selected <- select.samples(siamcat_example,
    filter='Age',
    allowed.range=c(25, 80))

# Select only female samples
siamcat_female <- select.samples(siamcat_example,
    filter='Gender',
    allowed.set=c('F'))
```

`siamcat` *SIAMCAT constructor function*

Description

Function to construct an object of class siamcat-class

Usage

```
siamcat(..., feat=NULL, label=NULL, meta=NULL,
    phyloseq=NULL, validate=TRUE, verbose=3)
```

Arguments

`...`	additional arguments
`feat`	feature information for SIAMCAT (see details)
`label`	label information for SIAMCAT (see details)
`meta`	(optional) metadata information for SIAMCAT (see details)
`phyloseq`	(optional) a phyloseq object for the creation of an SIAMCAT object (see details)
`validate`	boolean, should the newly constructed SIAMCAT object be validated? defaults to TRUE (**we strongly recommend against setting this parameter to FALSE**)
`verbose`	control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Details

Build siamcat-class objects from their components.

This functions creates a SIAMCAT object (see siamcat-class). In order to do so, the function needs

- `feat` the feature information for SIAMCAT, should be either a matrix, a data.frame, or a otu_table-class. The columns should correspond to the different samples (e.g. patients) and the rows the different features (e.g. taxa). Columns and rows should be named.
- `meta` metadata information for the different samples in the feature matrix. Metadata is optional for the SIAMCAT workflow. Should be either a data.frame (with the rownames corresponding to the sample names of the feature matrix) or an object of class sample_data-class
- `phyloseq` Alternatively to supplying both feat and meta, SIAMCAT can also work with a phyloseq object containing an otu_table and other optional slots (like sample_data for meta-variables).

Notice: do supply **either** the feature information as matrix/data.frame/otu_table (and optionally metadata) **or** a phyloseq object, but not both.

The label information for SIAMCAT can take several forms:

- metadata column: if there is metadata (either via meta or as sample_data in the phyloseq object), the label object can be created by taking the information in a specific metadata column. In order to do so, `label` should be the name of the column, and `case` should indicate which group(s) should be the positive group(s). A typical example could look like that:

  `siamcat <- siamcat(feat=feat.matrix, meta=metadata, label='DiseaseState', case='CRC')`

  for the construction of a label to predict CRC status (which is encoded in the column `"DiseaseState"` of the metadata). For more control (e.g. specific labels for plotting or specific control state), the label can also be created outside of the `siamcat` function using the create.label function.
- named vector: the label can also be supplied as named vector which encodes the label either as characters (e.g. "Healthy" and "Diseased"), as factor, or numerically (e.g. -1 and 1). The vector must be named with the names of samples (corresponding to the samples in features). Also here, the information about the positive group(s) is needed via the `case` parameter. Internally, the vector is given to the create.label function.
- label object: A label object can be created with the create.label function or by reading a dedicated label file with read.label.

Value

A new siamcat-class object

Examples

```
# example with package data
data("feat_crc_zeller", package="SIAMCAT")
data("meta_crc_zeller", package="SIAMCAT")

siamcat <- siamcat(feat=feat.crc.zeller,
    meta=meta.crc.zeller,
    label='Group',
    case='CRC')
```

`siamcat-class` *The S4 SIAMCAT class*

Description

The SIAMCAT class

Details

The S4 SIAMCAT class stores the results from the SIAMCAT workflow in different slots. The different slots will be filled by different functions (referenced in the description below).

In order to contruct a SIAMCAT class object, please refer to the documentation of the construction function siamcat.

The SIAMCAT class is based on the phyloseq-class. Therefore, you can easily import a `phyloseq` object into SIAMCAT.

Slots

`phyloseq` object of class phyloseq-class

`label` list containing the label information for the samples and some metadata about the label, created by create.label or when creating the siamcat-class object by calling siamcat

`filt_feat` list containing the filtered features as matrix and the list of filtering parameters, created by calling the filter.features function

`associations` list containing the parameters for association testing and the results of association testing with these parameters in a dataframe, created by calling the check.associations function

`norm_feat` list containing the normalized features as matrix and the list of normalziation parameters (for frozen normalization), created by calling the normalize.features function

`data_split` list containing cross-validation instances, created by calling the create.data.split function

`model_list` list containing the trained models, the type of model that was trained, and on which kind of features it was trained, created by calling the train.model function

`pred_matrix` matrix of predictions, created by calling the make.predictions function

`eval_data` list containing different evaluation metrics, created by calling the evaluate.predictions function

---

`siamcat_example` *SIAMCAT example*

Description

Reduced version of the CRC dataset from Zeller et al. MSB 2014 (see http://msb.embopress.org/content/10/11/766), containing 100 features (15 associated features at 5% FDR in the original dataset and 85 random other features) and 141 samples, saved after the complete SIAMCAT pipelinehas been run.

Thus, the example dataset contains entries in every slot of the SIAMCAT object (see siamcat-class), e.g, `eval_data` or `data_split`.

Mainly used for running the examples in the function documentation.

Source

http://msb.embopress.org/content/10/11/766

train.model *Model training*

Description

This function trains the a machine learning model on the training data

Usage

```
train.model(siamcat, method = "lasso", measure = "classif.acc",
param.set = NULL, grid.size=11, min.nonzero=5, perform.fs = FALSE,
param.fs = list(no_features = 100, method = "AUC", direction="absolute"),
feature.type='normalized', verbose = 1)
```

Arguments

`siamcat`	object of class siamcat-class
`method`	string, specifies the type of model to be trained, may be one of these: `c('lasso', 'enet', 'ridge', 'lasso_ll', 'ridge_ll','randomForest')`
`measure`	character, specifies the model selection criterion during internal cross-validation, see mlr_measures for more details, defaults to `'classif.acc'`
`param.set`	list, set of extra parameters for mlr, see below for details, defaults to `NULL`
`grid.size`	integer, grid size for internal tuning (needed for some machine learning methods, for example `lasso_ll`), defaults to `11`
`min.nonzero`	integer number of minimum nonzero coefficients that should be present in the model (only for `'lasso'`, `'ridge'`, and `'enet'`), defaults to `5`
`perform.fs`	boolean, should feature selection be performed? Defaults to `FALSE`
`param.fs`	list, parameters for the feature selection, see Details, defaults to `list(thres.fs=100, method.fs="AUC", direction='absolute')`
`feature.type`	string, on which type of features should the function work? Can be either `"original"`, `"filtered"`, or `"normalized"`. Please only change this paramter if you know what you are doing!
`verbose`	integer, control output: `0` for no output at all, `1` for only information about progress and success, `2` for normal level of information and `3` for full debug information, defaults to `1`

Value

object of class siamcat-class with added `model_list`

Machine learning methods

This functions performs the training of the machine learning model and functions as an interface to the `mlr3`-package.

The function expects a siamcat-class-object with a prepared cross-validation (see create.data.split) in the `data_split`-slot of the object. It then trains a model for each fold of the data split.

The different machine learning methods are implemented as Learners from the mlr3learners package:

- `'lasso'`, `'enet'`, and `'ridge'` use the `'classif.cv_glmnet'` or `'regr.cv_glmnet'` Learners, which interface to the glmnet package,
- `'lasso_ll'` and `'ridge_ll'` use a custom Learner, which is only available for classification tasks. The underlying package is the LiblineaR packge.
- `'randomForest'` is implemented via the `'classif.ranger'` or `regr.ranger` Learners available trough the ranger package.

Hyperparameter tuning

There is additional control over the machine learning procedure by supplying information through the `param.set` parameter within the function. We encourage you to check out the excellent mlr documentation for more in-depth information.

Here is a short overview which parameters you can supply in which form:

- enet The **alpha** parameter describes the mixture between lasso and ridge penalty and is -per default- determined using internal cross-validation (the default would be equivalent to `param.set=list('alpha'=c(0,1))`). You can supply either the limits of the hyperparameter exploration (e.g. with limits 0.2 and 0.8: `param.set=list('alpha'=c(0.2,0.8))`) or you can supply a fixed alpha value as well (`param.set=list('alpha'=0.5)`).
- lasso_ll/ridge_ll You can supply both **class.weights** and the **cost** parameter (cost of the constraints violation, see LiblineaR for more info). The default values would be equal to `param.set=list('class.weight 1),'cost'=c(-2, 3))`.
- randomForest You can supply the two parameters **num.trees** (Number of trees to grow) and **mtry** (Number of variables randomly sampled as candidates at each split). See also ranger for more info. The default values correspond to `param.set=list('num.trees'=c(100, 1000), 'mtry'= c(round(sqrt.mdim / 2), round(sqrt.mdim), round(sqrt.mdim * 2)))` with `sqrt.mdim=sqrt(nrow(da`

Feature selection

If feature selection should be performed (for example for functional data with a large number of features), the `param.fs` list should contain:

- `no_features` - Number of features to be retained after feature selection,
- `method` - method for the feature selection, may be `AUC`, `gFC`, or `Wilcoxon` for binary classification problems or `spearman`, `pearson`, or `MI` (mutual information) for regression problems
- `direction` - indicates if the feature selection should be performed in a single direction only. Can be either
  - `absolute` - select the top associated features (independent of the sign of enrichment),

- positivethe top positively associated featured (enriched in the case group for binary classification or enriched in higher values for regression),
- negative the top negatively associated features (inverse of positive)

Direction will be ignored for Wilcoxon and MI.

Examples

```
data(siamcat_example)

# simple working example
siamcat_example <- train.model(siamcat_example, method='lasso')
```

---

validate.data *Validate samples in labels, features, and metadata*

Description

This function checks if labels are available for all samples in features. Additionally validates metadata, if available.

Usage

```
validate.data(siamcat, verbose = 1)
```

Arguments

siamcat	an object of class siamcat-class
verbose	integer, control output: 0 for no output at all, 1 for only information about progress and success, 2 for normal level of information and 3 for full debug information, defaults to 1

Details

This function validates the data by checking that labels are available for all samples in the feature matrix. Furthermore, the number of samples per class is checked to ensure a minimum number. If metadata is available, the overlap between labels and metadata is checked as well.

This function is run when a siamcat-class object is created.

Value

an object of class siamcat-class

Examples

```
data(siamcat_example)

# validate.data should be run before completing the pipeline
# since the complete pipeline had been run on siamcat_example, we
# construct a new siamcat object for the example
feat <- orig_feat(siamcat_example)
label <- label(siamcat_example)
siamcat <- siamcat(feat=feat, label=label, validate=FALSE)
siamcat <- validate.data(siamcat, verbose=2)
```

---

volcano.plot *Visualize associations between features and classes as volcano plot*

Description

This function creates a volcano plot to vizualize the association between features and the label

Usage

```
volcano.plot(siamcat, fn.plot=NULL, color.scheme="RdYlBu",
annotate=5)
```

Arguments

`siamcat`	object of class siamcat-class
`fn.plot`	string, filename for the pdf-plot. If `fn.plot` is `NULL`, the plot will be produced in the active graphics device.
`color.scheme`	valid R color scheme or vector of valid R colors (must be of the same length as the number of classes), defaults to `'RdYlBu'`
`annotate`	integer, number of features to annotate with the name

Details

bla bla bla

Value

Does not return anything, but produces a volcano plot based on association measures

Examples

```
# Example data
data(siamcat_example)

# Simple example
volcano.plot(siamcat_example, fn.plot='./volcano.pdf')
```

`weight_matrix` *Retrieve the weight matrix from a SIAMCAT object*

Description

Function to retrieve the feature weights from a SIAMCAT object

Usage

```
weight_matrix(siamcat, verbose=1)

## S4 method for signature 'siamcat'
weight_matrix(siamcat, verbose = 1)
```

Arguments

`siamcat`	(Required). An instance of siamcat-class that contains trained models
`verbose`	integer, if the slot is empty, should a message be printed? values can be either `0` (no output) or `1` (print message)

Details

The function extracts the feature weights from all trained models acorss all cross-validation folds and repeats.

Value

A matrix containing the feature weights or `NULL`

Examples

```
data(siamcat_example)
temp <- weight_matrix(siamcat_example)
temp[1:3, 1:3]
```

Index