

# Package ‘SIAMCAT’

October 14, 2021

**Type** Package

**Title** Statistical Inference of Associations between Microbial Communities And host phenotypes

**Version** 1.12.0

**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 (>= 3.6.0), mlr, phyloseq

**Imports** beanplot, glmnet, graphics, grDevices, grid, gridBase, gridExtra, LiblineaR, matrixStats, methods, ParamHelpers, pROC, PRROC, RColorBrewer, scales, stats, stringr, utils, infotheo, progress, corplot

**License** GPL-3

**LazyData** true

**RoxygenNote** 7.1.1

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

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

**VignetteBuilder** knitr

**git\_url** <https://git.bioconductor.org/packages/SIAMCAT>

**git\_branch** RELEASE\_3\_13

**git\_last\_commit** fb65d54

**git\_last\_commit\_date** 2021-05-19

**Date/Publication** 2021-10-14

**Author** Konrad Zych [aut] (<<https://orcid.org/0000-0001-7426-0516>>),  
 Jakob Wirbel [aut, cre] (<<https://orcid.org/0000-0002-4073-3562>>),  
 Georg Zeller [aut] (<<https://orcid.org/0000-0003-1429-7485>>),  
 Morgan Essex [ctb],  
 Nicolai Karcher [ctb],  
 Kersten Breuer [ctb]

**Maintainer** Jakob Wirbel <jakob.wirbel@embl.de>

## R topics documented:

SIAMCAT-package . . . . .	3
accessSlot . . . . .	4
add.meta.pred . . . . .	5
associations . . . . .	6
associations<- . . . . .	7
assoc_param . . . . .	7
check.associations . . . . .	8
check.confounders . . . . .	10
create.data.split . . . . .	11
create.label . . . . .	12
data_split . . . . .	14
data_split<- . . . . .	14
evaluate.predictions . . . . .	15
eval_data . . . . .	16
eval_data<- . . . . .	17
feat.crc.zeller . . . . .	18
feature_type . . . . .	18
feature_weights . . . . .	19
filter.features . . . . .	20
filter.label . . . . .	21
filt_feat . . . . .	22
filt_feat<- . . . . .	23
filt_params . . . . .	23
get.filt_feat.matrix . . . . .	24
get.norm_feat.matrix . . . . .	25
get.orig_feat.matrix . . . . .	25
label . . . . .	26
label<- . . . . .	27
make.predictions . . . . .	27
meta . . . . .	28
meta.crc.zeller . . . . .	29
meta<- . . . . .	30
model.evaluation.plot . . . . .	30

model.interpretation.plot . . . . .	31
models . . . . .	33
model_list . . . . .	34
model_list<- . . . . .	35
model_type . . . . .	35
normalize.features . . . . .	36
norm_feat . . . . .	38
norm_feat<- . . . . .	39
norm_params . . . . .	39
orig_feat . . . . .	40
orig_feat<- . . . . .	41
physeq . . . . .	42
physeq<- . . . . .	42
pred_matrix . . . . .	43
pred_matrix<- . . . . .	44
read.label . . . . .	44
read.lefse . . . . .	45
select.samples . . . . .	46
siamcat . . . . .	47
siamcat-class . . . . .	49
siamcat.to.lefse . . . . .	50
siamcat_example . . . . .	50
train.model . . . . .	51
validate.data . . . . .	53
weight_matrix . . . . .	54
<b>Index</b>	<b>55</b>

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]

---

accessSlot

*Universal slot accessor function for siamcat-class.*

---

## Description

This function is used internally by many accessors.

## Usage

```
accessSlot(siamcat, slot, verbose=1)
```

## Arguments

siamcat	an object of <a href="#">siamcat-class</a> .
slot	A character string indicating the slot (not data class) of the component data type that is desired.
verbose	If the slot is empty, should a message be printed? values can be either 0 (no output) or 1 (print message)

## Value

Returns the component object specified by the argument slot. Returns NULL if slot does not exist.

**Examples**

```
#
data(siamcat_example)
accessSlot(siamcat_example, "label")
accessSlot(siamcat_example, "model_list")
```

---

add.meta.pred	<i>Add metadata as predictors</i>
---------------	-----------------------------------

---

**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 <a href="#">siamcat-class</a>
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 parameter 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)
```

---

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 <a href="#">siamcat-class</a> 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)
```

---

associations<-	<i>Assign a new associations object to x</i>
----------------	--

---

**Description**

Assign a new associations object to x

**Usage**

```
associations(x) <- value

## S4 replacement method for signature 'siamcat,list'
associations(x) <- value
```

**Arguments**

x	an object of class <a href="#">siamcat-class</a>
value	an associations object

**Value**

none

**Examples**

```
data(siamcat_example)
associations(siamcat_example) <- list(
  'assoc.results'=associations(siamcat_example),
  'assoc.param'=assoc_param(siamcat_example))
```

---

assoc_param	<i>Retrieve the list of parameters for association testing from a SIAMCAT object</i>
-------------	--

---

**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 <a href="#">siamcat-class</a> 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      *Check and visualize associations between features and classes*

---

**Description**

This function computes different measures of association between features and the label and visualizes the results

**Usage**

```
check.associations(siamcat, fn.plot=NULL, color.scheme = "RdYlBu",
  alpha = 0.05, mult.corr = "fdr", sort.by = "fc",
  detect.lim = 1e-06, pr.cutoff = 1e-6, max.show = 50,
  plot.type = "quantile.box",
  panels = c("fc", "auroc"), prompt = TRUE,
  feature.type = 'filtered', verbose = 1)
```

**Arguments**

siamcat	object of class <a href="#">siamcat-class</a>
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'
alpha	float, significance level, defaults to 0.05



<code>mult.corr</code>	string, multiple hypothesis correction method, see <a href="#">p.adjust</a> , defaults to "fdr"
<code>sort.by</code>	string, sort features by p-value ("p.val"), by fold change ("fc") or by prevalence shift ("pr.shift"), defaults to "fc"
<code>detect.lim</code>	float, pseudocount to be added before log-transformation of the data, defaults to $1e-06$ . Will be ignored if <code>feature.type</code> is "normalized".
<code>pr.cutoff</code>	float, cutoff for the prevalence computation, defaults to $1e-06$
<code>max.show</code>	integer, how many associated features should be shown, defaults to 50
<code>plot.type</code>	string, specify how the abundance should be plotted, must be one of these: <code>c("bean", "box", "quantile.box", "quantile.rect")</code> , defaults to "quantile.box"
<code>panels</code>	vector, name of the panels to be plotted next to the abundances, possible entries are <code>c("fc", "auroc", "prevalence")</code> , defaults to <code>c("fc", "auroc")</code>
<code>prompt</code>	boolean, turn on/off prompting user input when not plotting into a pdf-file, defaults to TRUE
<code>feature.type</code>	string, on which type of features should the function work? Can be either <code>c("original", "filtered", or "normalized")</code> . Please only change this parameter if you know what you are doing!  If <code>feature.type</code> is "normalized", the normalized abundances will not be log10-transformed.
<code>verbose</code>	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

For each feature, this function calculates different measures of association between the feature and the label. In detail, these associations are:

- Significance as computed by a Wilcoxon test followed by multiple hypothesis testing correction.
- AUROC (Area Under the Receiver Operating Characteristics Curve) as a non-parametric measure of enrichment (corresponds to the effect size of the Wilcoxon test).
- The generalized Fold Change (gFC) is a pseudo fold change which is calculated as geometric mean of the differences between the quantiles for the different classes found in the label.
- The prevalence shift between the two different classes found in the label.

Finally, the function produces a plot of the top `max.show` associated features at a user-specified significance level `alpha`, showing the distribution of the log10-transformed abundances for both classes, and user-selected panels for the effect (AU-ROC, Prevalence Shift, and Fold Change).

## Value

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

## Examples

```
# Example data
data(siamcat_example)

# Simple example
siamcat_example <- check.associations(siamcat_example,
  fn.plot='./assoc_plot.pdf')

# Plot associations as box plot
siamcat_example <- check.associations(siamcat_example,
  fn.plot='./assoc_plot_box.pdf', plot.type='box')

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

# Custom colors
siamcat_example <- check.associations(siamcat_example,
  fn.plot='./assoc_plot_blue_yellow.pdf', plot.type='box',
  color.scheme=c('cornflowerblue', '#ffc125'))
```

---

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

---

## Description

Checks potential confounders in the metadata and produces some visualizations

## Usage

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

## Arguments

siamcat	an object of class <a href="#">siamcat-class</a>
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 <code>c()</code> "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.

**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 <a href="#">siamcat-class</a>
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?, 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 would make 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 <a href="#">sample_data-class</a>
control	name of a label or vector with names that will be used as a negative label. All values that are not 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 internal 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	<i>Retrieve the data split from a SIAMCAT object</i>
------------	--

---

### 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

<code>siamcat</code>	(Required). An instance of <a href="#">siamcat-class</a> containing a data split
<code>verbose</code>	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)
```

---

data_split<-	<i>Assign a new list containing a cross-validation split to a SIAMCAT object</i>
--------------	--

---

### Description

Assign a new list containing a cross-validation split to a SIAMCAT object

**Usage**

```
data_split(x) <- value

## S4 replacement method for signature 'siamcat,list'
data_split(x) <- value
```

**Arguments**

x	an object of class <a href="#">siamcat-class</a>
value	list containing a cross-validation split

**Value**

none

**Examples**

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

---

evaluate.predictions *Evaluate prediction results*

---

**Description**

This function takes the correct labels and predictions for all samples and evaluates the results using the

- Area Under the Receiver Operating Characteristic (ROC) Curve (AU-ROC)
- and the Precision-Recall Curve (PR)

as metric. Predictions can be supplied either for a single case or as matrix after resampling of the dataset.

Prediction results are usually produced with the function [make.predictions](#).

**Usage**

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

**Arguments**

siamcat	object of class <a href="#">siamcat-class</a>
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 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.

## Value

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

## 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
```

---

eval\_data<- *Assign a new list with evaluation data to a SIAMCAT object*

---

**Description**

Assign a new list with evaluation data to a SIAMCAT object

**Usage**

```
eval_data(x) <- value

## S4 replacement method for signature 'siamcat,list'
eval_data(x) <- value
```

**Arguments**

x an object of class [siamcat-class](#)  
 value a list of evaluation data

**Value**

none

**Examples**

```
data(siamcat_example)
eval_data(siamcat_example) <- eval_data(siamcat_example)
```

---

feat.crc.zeller	<i>Example feature matrix</i>
-----------------	-------------------------------

---

**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	<i>Retrieve the feature type used for model training from a SIAMCAT object</i>
--------------	--

---

**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 <a href="#">siamcat-class</a> 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	<i>Retrieve the matrix of feature weights from a SIAMCAT object</i>
-----------------	---

---

### 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 <a href="#">siamcat-class</a> 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	<i>Perform unsupervised feature filtering.</i>
-----------------	--

---

### Description

This function performs unsupervised feature filtering. Features can be filtered based on abundance, prevalence, or on variance. Additionally, unmapped reads may be removed.

### Usage

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

### Arguments

siamcat	an object of class <a href="#">siamcat-class</a>
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, abundance, 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 parameter 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 an unsupervised manner.

The different filter methods work in the following way:

- 'abundance' - 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.label

*Filter the label of a SIAMCAT object*

---

### Description

This functions filters the label in a SIAMCAT object

### Usage

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

### Arguments

siamcat	an object of class <a href="#">siamcat-class</a>
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_feat	<i>Retrieve the information stored in the filt_feat slot within a SIAMCAT object</i>
-----------	--

---

**Description**

Function to retrieve the information stored in the filt\_feat slot within a SIAMCAT object

**Usage**

```
filt_feat(siamcat, verbose=1)

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

**Arguments**

siamcat	(Required). An instance of <a href="#">siamcat-class</a> that contains 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 will return a list containing the information stored in the filt\_feat slot of a SIAMCAT object. This list contains:

- `filt.feats` - filtered features as matrix, see [get.filt\\_feat.matrix](#)
- `filt.param` - parameters used for feature filtering, see [get.filt\\_feat.matrix](#)

**Value**

The list stored in the filt\_feat slot of the SIAMCAT object or NULL

**Examples**

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

---

filt_feat<-	<i>Assign a new filt_feat object to x</i>
-------------	---

---

**Description**

Assign a new filt\_feat object to x

**Usage**

```

filt_feat(x) <- value

## S4 replacement method for signature 'siamcat,list'
filt_feat(x) <- value

```

**Arguments**

x	an object of class <a href="#">siamcat-class</a>
value	an filt_feat object

**Value**

none

**Examples**

```

data(siamcat_example)
filt_feat(siamcat_example) <- list(
  filt_feat=filt_feat(siamcat_example),
  filt_param=filt_params(siamcat_example))

```

---

filt_params	<i>Retrieve the list of parameters for feature filtering from a SIAMCAT object</i>
-------------	--

---

**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	<i>Retrieve the label from a SIAMCAT object</i>
-------	---

---

**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 <a href="#">siamcat-class</a> 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
```



**Arguments**

<code>siamcat</code>	object of class <a href="#">siamcat-class</a>
<code>siamcat.holdout</code>	optional, object of class <a href="#">siamcat-class</a> on which to make predictions, defaults to NULL
<code>normalize.holdout</code>	boolean, should the holdout features be normalized with a frozen normalization (see <a href="#">normalize.features</a> ) using the normalization parameters in <code>siamcat?</code> , defaults to TRUE
<code>verbose</code>	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.pred <- make.predictions(siamcat_example)

# Predictions on a holdout-set
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	<i>Example metadata matrix</i>
-----------------	--------------------------------

---

**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>

```
meta<-          Assign a new sam_data object to x
```

---

### Description

Assign a new sam\_data object to x

### Usage

```
meta(x) <- value

## S4 replacement method for signature 'siamcat,sample_data'
meta(x) <- value
```

### Arguments

x                    an object of class [siamcat-class](#)  
value                an object of class [sample\\_data-class](#)

### Value

none

### Examples

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

---

```
model.evaluation.plot Model Evaluation Plot
```

---

### Description

Produces two plots for model evaluation. The first plot shows the Receiver Operating Characteristic (ROC)-curves, the other the Precision-recall (PR)-curves for the different cross-validation repetitions.

### Usage

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

**Arguments**

...	one or more object of class <code>siamcat-class</code> , can be named
<code>fn.plot</code>	string, filename for the pdf-plot
<code>colours</code>	colour specification for the different <code>siamcat-class</code> - objects, defaults to NULL which will cause the colours to be picked from the 'Set1' palette
<code>show.all</code>	boolean, Should all repeated cross-validation models be plotted?
<code>verbose</code>	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.

**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')
```

---

model.interpretation.plot

*Model Interpretation Plot*

---

**Description**

This function produces a plot for model interpretation, displaying

- the feature weights,
- the robustness of feature weights
- the features scores across samples,
- the distribution of metadata across samples, and
- the proportion of model weights shown.

**Usage**

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

**Arguments**

siamcat	object of class <a href="#">siamcat-class</a>
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)
detect.lim	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 patients
- 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
model.interpretation.plot(siamcat_example, fn.plot='./interpretation.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 <a href="#">siamcat-class</a> 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_list	<i>Retrieve the information stored in the model_list slot within a SIAMCAT object</i>
------------	---

---

### Description

Function to retrieve the information stored in the model\_list slot within a SIAMCAT object

### Usage

```
model_list(siamcat, verbose=1)

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

### Arguments

siamcat	(Required). An instance of <a href="#">siamcat-class</a> 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 will return a list containing the information stored in the model\_list slot of a SIAMCAT object. This list contains:

- models - list of trained models
- model\_type - machine learning method used for training
- feature\_type - string describing on which type of features the models were trained

### Value

The list stored in the model\_list slot of the SIAMCAT object or NULL

### Examples

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

---

model_list<-	<i>Assign a new list containing trained models to a SIAMCAT object</i>
--------------	--

---

**Description**

Assign a new list containing trained models to a SIAMCAT object

**Usage**

```
model_list(x) <- value

## S4 replacement method for signature 'siamcat,list'
model_list(x) <- value
```

**Arguments**

x	an object of class <a href="#">siamcat-class</a>
value	list containing trained models, type of models and of features

**Value**

none

**Examples**

```
data(siamcat_example)
model_list(siamcat_example) <- model_list(siamcat_example)
```

---

model_type	<i>Retrieve the machine learning method from a SIAMCAT object</i>
------------	---

---

**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 <a href="#">siamcat-class</a> 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 <a href="#">siamcat-class</a>
norm.method	string, normalization method, can be one of these: 'c('rank.unit', 'rank.std', 'log.std', 'log.uni
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

## Details

There are seven different normalization methods available:

- 'rank.unit' - converts features to ranks and normalizes each column (=sample) by the square root of the sum of ranks
- 'rank.std' - converts features to ranks and applies z-score standardization
- 'log.clr' - centered log-ratio transformation (with the addition of pseudocounts)
- 'log.std' - log-transforms features (after addition of pseudocounts) and applies z-score standardization
- 'log.unit' - log-transforms features (after addition of pseudocounts) and normalizes by features or samples with different norms
- 'std' - z-score standardization without any other transformation
- 'pass' - pass-through normalization will not change the features

The list entries in 'norm.param' specify the normalization parameters, which are dependant on the normalization method of choice:

- 'rank.unit' or 'pass' does not require any other parameters
- 'rank.std' and 'std' requires `sd.min.q`, quantile of the distribution of standard deviations of all features that will be added to the denominator during standardization in order to avoid underestimation of the standard deviation, defaults to 0.1
- 'log.clr' requires `log.n0`, which is the pseudocount to be added before log-transformation, defaults to NULL leading to the estimation of `log.n0` from the data
- 'log.std' requires both `log.n0` and `sd.min.q`, using the same default values
- 'log.unit' requires next to `log.n0` also the parameters `n.p` and `norm.margin`. `n.p` specifies the vector norm to be used, can be either 1 for  $x/\text{sum}(x)$  or 2 for  $x/\sqrt{\text{sum}(x^2)}$ . The parameter `norm.margin` specifies the 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.

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.

## Value

an object of class [siamcat-class](#) with normalized features

## 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
siamcat_norm <- normalize.features(siamcat,
  norm.param=norm_params(siamcat_reference))

```

---

norm_feat	<i>Retrieve the information stored in the norm_feat slot within a SIAM-CAT object</i>
-----------	---

---

## Description

Function to retrieve the information stored in the norm\_feat slot within a SIAMCAT object

## Usage

```

norm_feat(siamcat, verbose=1)

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

```

## Arguments

siamcat	(Required). An instance of <a href="#">siamcat-class</a> that contains 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 will return a list containing the information stored in the norm\_feat slot of a SIAM-CAT object. This list contains:

- norm.feats - normalized features as matrix, see [get.norm\\_feats.matrix](#)
- norm.param - parameters used for normalization, see [normalize.features](#)

## Value

The list stored in the norm\_feat slot of the SIAMCAT object or NULL

**Examples**

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

---

norm_feat<-	<i>Assign a new list containing normalization parameters and normalized features to a SIAMCAT object</i>
-------------	--

---

**Description**

Assign a new list containing normalization parameters and normalized features to a SIAMCAT object

**Usage**

```
norm_feat(x) <- value

## S4 replacement method for signature 'siamcat,list'
norm_feat(x) <- value
```

**Arguments**

x	an object of class <a href="#">siamcat-class</a>
value	a list containing normalization parameters and features

**Value**

none

**Examples**

```
data(siamcat_example)
norm_feat(siamcat_example) <- norm_feat(siamcat_example)
```

---

norm_params	<i>Retrieve the list of parameters for feature normalization from a SIAMCAT object</i>
-------------	--

---

**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)
```

---

orig_feat	<i>Retrieve a <a href="#">otu_table-class</a> object from otu_table slot in the phyloseq slot in a siamcat object</i>
-----------	---

---

**Description**

Retrieve a [otu\\_table-class](#) object from otu\_table slot in the phyloseq slot in a siamcat object

**Usage**

```
orig_feat(siamcat)

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

## S4 method for signature 'otu_table'
orig_feat(siamcat)
```



`orig_feat<-`

41

### Arguments

`siamcat` (Required). An instance of [siamcat-class](#) that contains a label or instance of [otu\\_table-class](#).

### Value

The [otu\\_table-class](#) object or NULL.

### Examples

```
data(siamcat_example)
temp <- orig_feat(siamcat_example)
```

---

`orig_feat<-` *Assign a new otu\_table object to x orig\_feat slot*

---

### Description

Assign a new `otu_table` object to `x orig_feat` slot

### Usage

```
orig_feat(x) <- value

## S4 replacement method for signature 'siamcat,otu_table'
orig_feat(x) <- value
```

### Arguments

`x` an object of class [siamcat-class](#)  
`value` an object of class [otu\\_table-class](#)

### Value

none

### Examples

```
data(siamcat_example)
orig_feat(siamcat_example) <- orig_feat(siamcat_example)
```

---

physeq	<i>Retrieve a <a href="#">phyloseq-class</a> object from object.</i>
--------	--

---

**Description**

Retrieve a [phyloseq-class](#) object from object.

**Usage**

```
physeq(siamcat, verbose=1)

## S4 method for signature 'ANY'
physeq(siamcat, verbose = 1)

## S4 method for signature 'phyloseq'
physeq(siamcat)
```

**Arguments**

siamcat	(Required). An instance of <a href="#">siamcat-class</a> that contains a label or instance of <a href="#">phyloseq-class</a> .
verbose	If the slot is empty, should a message be printed? values can be either 0 (no output) or 1 (print message)

**Value**

The [phyloseq-class](#) object or NULL.

**Examples**

```
data(siamcat_example)
physeq(siamcat_example)
```

---

physeq<-	<i>Assign a new <a href="#">phyloseq</a> object to x</i>
----------	--

---

**Description**

Assign a new [phyloseq](#) object to x

**Usage**

```
physeq(x) <- value

## S4 replacement method for signature 'siamcat,phyloseq'
physeq(x) <- value
```

**Arguments**

x                    an object of class [siamcat-class](#)  
value                an object of class [phyloseq-class](#)

**Value**

none

**Examples**

```
data(siamcat_example)
physeq(siamcat_example) <- physeq(siamcat_example)
```

---

pred_matrix	<i>Retrieve the prediction matrix from a SIAMCAT object</i>
-------------	---

---

**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)
```

---

```
pred_matrix<-          Assign a new matrix with predictions to a SIAMCAT object
```

---

**Description**

Assign a new matrix with predictions to a SIAMCAT object

**Usage**

```
pred_matrix(x) <- value

## S4 replacement method for signature 'siamcat,matrix'
pred_matrix(x) <- value
```

**Arguments**

x                    an object of class [siamcat-class](#)  
value                a matrix containing predictions

**Value**

none

**Examples**

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

---

```
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)
```

---

read.lefse	<i>read an input file in a LEfSe input format</i>
------------	---

---

**Description**

This reads an input file in a LEfSe input format

**Usage**

```
read.lefse(filename = "data.txt", rows.meta = 1, row.samples = 2)
```

**Arguments**

filename	name of the input file in a LEfSe input format
rows.meta	specifies in which rows metadata variables are stored
row.samples	specifies in which row sample names are stored

**Value**

a list with two elements:

- feat a features matrix
- meta a metadata matrix

**Examples**

```
fn.in.lefse<- system.file("extdata",
"LEfSe_crc_zeller_msb_mocat_specI.tsv",package = "SIAMCAT")
meta.and.features <- read.lefse(fn.in.lefse, rows.meta = 1:6,
row.samples = 7)
meta <- meta.and.features$meta
feat <- meta.and.features$feat
label <- create.label(meta=meta, label="label", case = "cancer")
siamcat <- siamcat(feat=feat, label=label, meta=meta)
```

---

select.samples	<i>Select samples based on metadata</i>
----------------	---

---

**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 <a href="#">siamcat-class</a>
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**

<code>...</code>	additional arguments
<code>feat</code>	feature information for SIAMCAT (see details)
<code>label</code>	label information for SIAMCAT (see details)
<code>meta</code>	(optional) metadata information for SIAMCAT (see details)
<code>phyloseq</code>	(optional) a phyloseq object for the creation of an SIAMCAT object (see details)
<code>validate</code>	boolean, should the newly constructed SIAMCAT object be validated? defaults to TRUE ( <b>we strongly recommend against setting this parameter to FALSE</b> )
<code>verbose</code>	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	<i>The S4 SIAMCAT class</i>
---------------	-----------------------------

---

## 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 construct 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 normalization 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.to.lefse	<i>create a lefse input file from siamcat object</i>
------------------	--

---

### Description

This function creates a lefse input file from siamcat object

### Usage

```
siamcat.to.lefse(siamcat, filename = "siamcat_output.txt")
```

### Arguments

siamcat	object of class <a href="#">siamcat-class</a>
filename	name of the input file to which data will be save

### Value

nothing but data is written to a file

### Examples

```
data(siamcat_example)  
siamcat.to.lefse(siamcat_example)
```

---

siamcat_example	<i>SIAMCAT example</i>
-----------------	------------------------

---

### 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 pipeline has 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	<i>Model training</i>
-------------	-----------------------

---

## Description

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

## Usage

```
train.model(siamcat, method = c("lasso", "enet", "ridge", "lasso_ll",
"ridge_ll", "randomForest"), stratify = TRUE, modsel.crit = list("auc"),
min.nonzero.coeff = 1, param.set = NULL, perform.fs = FALSE, param.fs =
list(thres.fs = 100, method.fs = "AUC", direction='absolute'),
feature.type='normalized', verbose = 1)
```

## Arguments

siamcat	object of class <a href="#">siamcat-class</a>
method	string, specifies the type of model to be trained, may be one of these: c('lasso', 'enet', 'ridge', 'lasso_ll', 'ridge_ll', 'randomForest')
stratify	boolean, should the folds in the internal cross-validation be stratified?, defaults to TRUE
modsel.crit	list, specifies the model selection criterion during internal cross-validation, may contain these: c('auc', 'f1', 'acc', 'pr'), defaults to list('auc')
min.nonzero.coeff	integer number of minimum nonzero coefficients that should be present in the model (only for 'lasso', 'ridge', and 'enet'), defaults to 1
param.set	list, set of extra parameters for mlr run, may contain: <ul style="list-style-type: none"> <li>• cost and class.weights - for lasso_ll and ridge_ll</li> <li>• alpha - for enet</li> <li>• ntree and mtry - for RandomForrest.</li> </ul> See below for details. Defaults to NULL
perform.fs	boolean, should feature selection be performed? Defaults to FALSE
param.fs	list, parameters for the feature selection, must contain: <ul style="list-style-type: none"> <li>• thres.fs - threshold for the feature selection,</li> <li>• method.fs - method for the feature selection, may be AUC, gFC, or Wilcoxon</li> <li>• direction - for AUC and gFC, select either the top associated features (independent of the sign of enrichment), the top positively associated featured, or the top negatively associated features, may be absolute, positive, or negative. Will be ignored for Wilcoxon.</li> </ul> See Details for more information. 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

## Details

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

For the machine learning methods that require additional hyperparameters (e.g. `lasso_ll`), the optimal hyperparameters are tuned with the function `tuneParams` within the `mlr`-package.

The different machine learning methods are implemented as `mlr`-tasks:

- 'lasso', 'enet', and 'ridge' use the 'classif.cvglmnet' Learner,
- 'lasso\_ll' and 'ridge\_ll' use the 'classif.LiblineaRL1LogReg' and the 'classif.LiblineaRL2LogReg' Learners respectively
- 'randomForest' is implemented via the 'classif.randomForest' Learner.

**Hyperparameters** You also have 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.weights'^ seq(-2,3,length = 6 + 5 + 10))`.
- `randomForest` You can supply the two parameters **ntree** (Number of trees to grow) and **mtry** (Number of variables randomly sampled as candidates at each split). See also [randomForest](#) for more info. The default values correspond to `param.set=list('ntree'=c(100,1000), 'mtry'=c(round(sqrt.mdim / 2), round(sqrt.mdim), round(sqrt.mdim * 2)))` with `sqrt.mdim=sqrt(nrow(data))`.

**Feature selection** The function can also perform feature selection on each individual fold. At the moment, three methods for feature selection are implemented:

- 'AUC' - computes the Area Under the Receiver Operating Characteristics Curve for each single feature and selects the top `param.fs$thres.fs`, e.g. 100 features
- 'gFC' - computes the generalized Fold Change (see [check.associations](#)) for each feature and likewise selects the top `param.fs$thres.fs`, e.g. 100 features
- Wilcoxon - computes the p-Value for each single feature with the Wilcoxon test and selects features with a p-value smaller than `param.fs$thres.fs`

For AUC and gFC, feature selection can also be directed, that means that the features will be selected either based on the overall association (absolute - gFC will be converted to absolute values and AUC values below 0.5 will be converted by 1 - AUC), or on associations in a certain direction (positive - positive enrichment as measured by positive values of the gFC or AUC values higher than 0.5 - and reversely for negative).

### Value

object of class [siamcat-class](#) with added `model_list`

### Examples

```
data(siamcat_example)

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

---

validate.data	<i>Validate samples in labels, features, and metadata</i>
---------------	---

---

### 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

<code>siamcat</code>	an object of class <a href="#">siamcat-class</a>
<code>verbose</code>	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)
```

---

weight_matrix	<i>Retrieve the weight matrix from a SIAMCAT object</i>
---------------	---

---

## 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 <a href="#">siamcat-class</a> 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 across 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

- \* **SIAMCAT**
    - add.meta.pred, 5
    - check.associations, 8
    - check.confounders, 10
    - create.data.split, 11
    - evaluate.predictions, 15
    - filter.features, 20
    - make.predictions, 27
    - model.evaluation.plot, 30
    - model.interpretation.plot, 31
    - normalize.features, 36
    - select.samples, 46
    - train.model, 51
    - validate.data, 53
  - \* **add.meta.pred**
    - add.meta.pred, 5
  - \* **check.associations**
    - check.associations, 8
  - \* **check.confounders**
    - check.confounders, 10
  - \* **create.data.split**
    - create.data.split, 11
  - \* **create.label**
    - create.label, 12
  - \* **data**
    - feat.crc.zeller, 18
    - meta.crc.zeller, 29
    - siamcat\_example, 50
  - \* **evaluate.predictions**
    - evaluate.predictions, 15
  - \* **filter.features**
    - filter.features, 20
  - \* **filter.label**
    - filter.label, 21
  - \* **make.predictions**
    - make.predictions, 27
  - \* **model.evaluation.plot**
    - model.evaluation.plot, 30
  - \* **model.interpretation.plot**
    - model.interpretation.plot, 31
  - \* **normalize.features**
    - normalize.features, 36
  - \* **plm.trainer**
    - train.model, 51
  - \* **read.lefse**
    - read.lefse, 45
  - \* **select.samples**
    - select.samples, 46
  - \* **siamcat.to.lefse**
    - siamcat.to.lefse, 50
  - \* **validate.data**
    - validate.data, 53
- 
- accessSlot, 4
  - add.meta.pred, 5
  - assign-associations (associations<-), 7
  - assign-data\_split (data\_split<-), 14
  - assign-eval\_data (eval\_data<-), 17
  - assign-filt\_feat (filt\_feat<-), 23
  - assign-label (label<-), 27
  - assign-meta (meta<-), 30
  - assign-model\_list (model\_list<-), 35
  - assign-norm\_feat (norm\_feat<-), 39
  - assign-orig\_feat (orig\_feat<-), 41
  - assign-physeq (physeq<-), 42
  - assign-pred\_matrix (pred\_matrix<-), 44
  - assoc\_param, 7
  - assoc\_param, siamcat-method (assoc\_param), 7
  - assoc\_param\_param (assoc\_param), 7
  - associations, 6
  - associations, siamcat-method (associations), 6
  - associations<-, 7
  - associations<-, siamcat, list-method (associations<-), 7
  - check.associations, 6, 8, 8, 49, 52
  - check.confounders, 10

- create.data.split, [11](#), [14](#), [49](#), [52](#)
- create.label, [12](#), [48](#), [49](#)
- data\_split, [14](#)
- data\_split, siamcat-method (data\_split), [14](#)
- data\_split<-, [14](#)
- data\_split<-, siamcat, list-method (data\_split<-), [14](#)
- eval\_data, [16](#)
- eval\_data, siamcat-method (eval\_data), [16](#)
- eval\_data<-, [17](#)
- eval\_data<-, siamcat, list-method (eval\_data<-), [17](#)
- evaluate.predictions, [15](#), [17](#), [49](#)
- feat.crc.zeller, [18](#)
- feature\_type, [18](#)
- feature\_type, siamcat-method (feature\_type), [18](#)
- feature\_weights, [19](#)
- feature\_weights, siamcat-method (feature\_weights), [19](#)
- filt\_feat, [22](#)
- filt\_feat, siamcat-method (filt\_feat), [22](#)
- filt\_feat<-, [23](#)
- filt\_feat<-, siamcat, list-method (filt\_feat<-), [23](#)
- filt\_params, [23](#)
- filt\_params, siamcat-method (filt\_params), [23](#)
- filter.features, [20](#), [24](#), [49](#)
- filter.label, [21](#)
- get.filt\_feat.matrix, [22](#), [24](#)
- get.norm\_feat.matrix, [25](#), [38](#)
- get.orig\_feat.matrix, [25](#)
- label, [26](#)
- label, siamcat-method (label), [26](#)
- label<-, [27](#)
- label<-, siamcat, list-method (label<-), [27](#)
- LiblineaR, [52](#)
- make.predictions, [15](#), [27](#), [43](#), [49](#)
- meta, [28](#)
- meta, sample\_data-method (meta), [28](#)
- meta, siamcat-method (meta), [28](#)
- meta.crc.zeller, [29](#)
- meta<-, [30](#)
- meta<-, siamcat, sample\_data-method (meta<-), [30](#)
- model.evaluation.plot, [30](#)
- model.interpretation.plot, [31](#)
- model\_list, [34](#)
- model\_list, siamcat-method (model\_list), [34](#)
- model\_list<-, [35](#)
- model\_list<-, siamcat, list-method (model\_list<-), [35](#)
- model\_type, [35](#)
- model\_type, siamcat-method (model\_type), [35](#)
- models, [33](#)
- models, siamcat-method (models), [33](#)
- norm\_feat, [38](#)
- norm\_feat, siamcat-method (norm\_feat), [38](#)
- norm\_feat<-, [39](#)
- norm\_feat<-, siamcat, list-method (norm\_feat<-), [39](#)
- norm\_params, [37](#), [39](#)
- norm\_params, siamcat-method (norm\_params), [39](#)
- normalize.features, [25](#), [28](#), [36](#), [38](#), [40](#), [49](#)
- orig\_feat, [40](#)
- orig\_feat, otu\_table-method (orig\_feat), [40](#)
- orig\_feat, siamcat-method (orig\_feat), [40](#)
- orig\_feat<-, [41](#)
- orig\_feat<-, siamcat, otu\_table-method (orig\_feat<-), [41](#)
- otu\_table-class, [40](#), [41](#), [48](#)
- p.adjust, [9](#)
- phyloseq-class, [42](#), [43](#), [49](#)
- physeq, [42](#)
- physeq, ANY-method (physeq), [42](#)
- physeq, phyloseq-method (physeq), [42](#)
- physeq<-, [42](#)
- physeq<-, siamcat, phyloseq-method (physeq<-), [42](#)
- pred\_matrix, [43](#)
- pred\_matrix, siamcat-method (pred\_matrix), [43](#)
- pred\_matrix<-, [44](#)



`pred_matrix<-`, `siamcat`, `matrix-method`  
    (`pred_matrix<-`), 44

`randomForest`, 52

`read.label`, 44, 48

`read.lefse`, 45

`roc`, 16

`sample_data-class`, 13, 29, 30, 48

`select.samples`, 46

SIAMCAT (SIAMCAT-package), 3

`siamcat`, 47, 49

`siamcat-class`, 4–12, 14–44, 46–49, 49,  
    50–54

SIAMCAT-package, 3

`siamcat.to.lefse`, 50

`siamcat_example`, 50

`train.model`, 5, 12, 27, 49, 51

`tuneParams`, 52

`validate.data`, 21, 53

`weight_matrix`, 19, 54

`weight_matrix`, `siamcat-method`  
    (`weight_matrix`), 54