Package: CrcBiomeScreen (via r-universe)

November 1, 2026

Title An R package for colorectal cancer screening and microbiome analysis

Version 1.0.0

Description A developed and benchmarked reproducible machine learning framework for microbiome-based colorectal cancer (CRC) screening. By systematically evaluating normalization strategies, taxonomic resolutions, and class imbalance handling. This R package allows users to apply the full pipeline or selectively run specific components depending on their analytical needs. It establishes a scalable foundation for developing interpretable microbiome-based screening tools to support early CRC detection. This approach could be easily implemented in a national screening programme, to improve early detection rates for this disease.

License MIT + file LICENSE

biocViews Software, Microbiome, Metagenomics, Classification, Normalization, Visualization

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 3

Depends R (>= 4.3.0)

Imports rlang, methods, dplyr, doFuture, doParallel, foreach, future, future.apply, pROC, progress, progressr, stats, tibble, tidyr, TreeSummarizedExperiment, ggplot2, GUniFrac, magrittr, parallel, withr, SummarizedExperiment, caret, ranger, utils, graphics, grDevices

Suggests rstatix, MASS, mgcv, ggplotify, ggpubr, ggrepel, ggtree, glmnet, Matrix, microbiome, phyloseq, vegan, gt, testthat (>= 3.0.0), BiocManager, devtools, knitr, rmarkdown, BiocStyle, curatedMetagenomicData, xgboost

VignetteBuilder knitr

LazyData false

URL https://github.com/omicsForestry/CrcBiomeScreen

BugReports https://github.com/omicsForestry/CrcBiomeScreen/issues

Config/pak/sysreqs cmake make libicu-dev libuv1-dev zlib1g-dev

Repository Bioconductor 3.23

Date/Publication 2026-04-28 13:07:03 UTC

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

RemoteRef RELEASE_3_23

RemoteSha 29bf03fdec23323733fef3e1dfa5a9441cdddfb3

Contents

checkClassBalance . . . 3
CrcBiomeScreen-class . . . 4
CreateCrcBiomeScreenObject . . . 4
CreateCrcBiomeScreenObjectFromTSE . . . 6
EvaluateCrcBiomeScreen . . . 7
EvaluateModel . . . 9
EvaluateRF . . . 10
EvaluateXGBoost . . . 11
FilterDataSet . . . 12
getAbsoluteAbundance . . . 14
getModelData . . . 15
getModelResult . . . 16
getNormalizedData . . . 17
getOutlierSamples . . . 18
getPredictResult . . . 18
getRelativeAbundance . . . 19
getSampleData . . . 20
getTaxaData . . . 21
getTaxaLevelData . . . 22
KeepTaxonomicLevel . . . 23
LoadTaxaTable . . . 25
ModelingRF . . . 26
ModelingRF_noweights . . . 28
ModelingXGBoost . . . 29
ModelingXGBoost_noweights . . . 30
NHSBCSP_screeningData . . . 32
NormalizeData . . . 32
PredictCrcBiomeScreen . . . 33
qcByCmdscale . . . 35
RunScreening . . . 37
setNormalizedData-setter . . . 40
setTaxaData-setter . . . 41

SplitDataSet 42
SplitTaxas 43
Thomas_2018_RelativeAbundance 45
TrainModels 45
ValidateModelOnData 47
ZellerG_2014_RelativeAbundance 50

Index **52**

`checkClassBalance` *Check the sample distribution of the dataset and give the suggestion if need the class weight or not*

Description

Check the sample distribution of the dataset and give the suggestion if need the class weight or not

Usage

```
checkClassBalance(labels, outdir = tempdir(), threshold = 0.5, plot = TRUE)
```

Arguments

`labels`	The label for distribution
`outdir`	The output directory where plots will be saved (default: tempdir()).
`threshold`	The threshold for the ratio (0.5) if it is the imbalanced dataset
`plot`	Choose to have the figures or not

Value

A A `CrcBiomeScreen` object. object with updated slots.

Examples

```
# Small toy example for runnable demonstration
train_labels <- factor(c("control", "CRC", "control", "CRC"))
checkClassBalance(train_labels)
```

CrcBiomeScreen-class *CrcBiomeScreen Class*

Description

An S4 container for CRC microbiome screening data, including abundance matrices, taxonomy, sample metadata, and model results.

Value

A A `CrcBiomeScreen` object. object.

Slots

`AbsoluteAbundance` Absolute abundance matrix.

`TaxaData` Taxonomy annotation data frame.

`SampleData` Sample metadata (must include number_reads if relative abundance is used).

`RelativeAbundance` Relative abundance matrix.

`TaxaLevelData` Optional genus-level summary.

`NormalizedData` Normalized abundance data.

`ValidationData` Optional validation dataset.

`ModelData` Processed training/testing data.

`ModelResult` Fitted model objects.

`EvaluateResult` List of evaluation metrics (RF, XGBoost, etc.).

`PredictResult` Predictions for external data.

CreateCrcBiomeScreenObject *Create a CrcBiomeScreen S4 object for microbiome-based CRC analysis*

Description

Constructor for the `CrcBiomeScreen` S4 class. This function creates a structured container for microbiome data, including absolute and relative abundance matrices, taxonomic annotations, and sample metadata. It ensures compatibility with downstream modelling and evaluation functions within the CrcBiomeScreen package.

Usage

```
CreateCrcBiomeScreenObject(
  AbsoluteAbundance = NULL,
  TaxaData = NULL,
  SampleData = NULL,
  RelativeAbundance = NULL
)
```

Arguments

AbsoluteAbundance	A numeric matrix or data frame containing absolute abundance data.
TaxaData	A data frame containing taxonomic information for each feature.
SampleData	A data frame containing sample-level metadata.
RelativeAbundance	A numeric matrix or data frame containing relative abundance data.

Details

If only relative abundance data are supplied, absolute abundance is estimated using the total number of reads in SampleData$number_reads.

Value

A A CrcBiomeScreen object. object.

- AbsoluteAbundance: Absolute abundance data.
- RelativeAbundance: Relative abundance data.
- TaxaData: Taxonomic annotations.
- SampleData: Sample metadata.
- TaxaLevelData: Optional genus-level summary data.
- NormalizedData: Normalized data.
- OrginalNormalizedData: Original normalized data.
- ValidationData: Optional validation dataset.
- OutlierSamples: Character vector of outlier sample names.
- ModelData, ModelResult, EvaluateResult, PredictResult: Optional model results and evaluation outputs

See Also

A CrcBiomeScreen object.

Examples

```
# Minimal example with tiny toy data (required for Bioconductor checks)

# Create toy abundance matrices
rel_abund <- data.frame(
  Sample1 = c(10, 20, 70),
  Sample2 = c(30, 30, 40)
)
rownames(rel_abund) <- c("TaxaA", "TaxaB", "TaxaC")

taxa_info <- data.frame(
  Taxa = rownames(rel_abund),
  stringsAsFactors = FALSE
)

sample_info <- data.frame(
  number_reads = c(10000, 12000),
  condition = c("control", "CRC"),
  row.names = c("Sample1", "Sample2"),
  stringsAsFactors = FALSE
)

# Create object
obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = rel_abund,
  TaxaData = taxa_info,
  SampleData = sample_info
)
```

CreateCrcBiomeScreenObjectFromTSE

Create a CrcBiomeScreen object from TreeSummarizedExperiment

Description

Wrapper function to directly convert a TreeSummarizedExperiment object into a A `CrcBiomeScreen` object. S4 object for downstream analysis.

Usage

```
CreateCrcBiomeScreenObjectFromTSE(tse, assay_name = NULL)
```

Arguments

`tse`	A TreeSummarizedExperiment object containing microbiome data.
`assay_name`	Which assay to use (default: "counts" or "relative_abundance").

Value

A A CrcBiomeScreen object. object.

Examples

```
# Runnable example using a minimal mock TreeSummarizedExperiment

# Load required classes (SummarizedExperiment & TreeSummarizedExperiment)
suppressMessages({
  library(SummarizedExperiment)
  library(TreeSummarizedExperiment)
})

# Create a tiny assay matrix
assay_mat <- matrix(
  c(10, 5,
    20, 7),
  nrow = 2,
  dimnames = list(c("Taxa1", "Taxa2"), c("S1", "S2"))
)

# Create row (taxa) metadata
row_meta <- DataFrame(
  Taxa = c("A;B;C", "A;D;E")
)

# Create sample metadata
col_meta <- DataFrame(
  number_reads = c(10000, 12000),
  condition = c("control", "CRC")
)

# Build a minimal TreeSummarizedExperiment
tse <- TreeSummarizedExperiment::TreeSummarizedExperiment(
  assays = list(relative_abundance = assay_mat),
  rowData = row_meta,
  colData = col_meta
)

# Convert to CrcBiomeScreen object
obj <- CreateCrcBiomeScreenObjectFromTSE(tse)

# Inspect object
obj
```

EvaluateCrcBiomeScreen

Evaluate the performance of model predictions

Description

This function calculates performance metrics (e.g., AUC) and plots the ROC curve based on prediction probabilities and true labels.

Usage

```
EvaluateCrcBiomeScreen(
  predictions,
  outdir = tempdir(),
  true_labels,
  TrueLabel = NULL,
  TaskName = "ModelEvaluation",
  PlotAUC = FALSE
)
```

Arguments

predictions	A data frame or matrix of model predictions, typically containing columns for probability scores for each class.
outdir	The output directory where plots will be saved (default: tempdir()).
true_labels	A character vector or factor of the true class labels.
TrueLabel	The positive class label (e.g., "CRC") to use for ROC/AUC calculation.
TaskName	A character string used to label the output files.
PlotAUC	A logical value indicating whether to plot the AUC curve.

Value

A A CrcBiomeScreen object. object with updated slots containing the ROC curve object and the AUC value.

Examples

```
# --- Minimal runnable example (no external dependencies) ---

# Fake prediction probabilities for 4 samples and 2 classes
pred <- data.frame(
  control = c(0.8, 0.3, 0.7, 0.2),
  CRC     = c(0.2, 0.7, 0.3, 0.8)
)

# True class labels
labels <- factor(c("control", "CRC", "control", "CRC"))

# Evaluate performance using CRC as positive class
result <- EvaluateCrcBiomeScreen(
  predictions = pred,
  true_labels = labels,
  TrueLabel = "CRC",
  PlotAUC = FALSE  # disable plotting for speed/safety
```

```
)

result$AUC
```

`EvaluateModel` *Evaluate the model to select the optimal model*

Description

Evaluate the model to select the optimal model

Usage

```
EvaluateModel(
  CrcBiomeScreenObject = NULL,
  model_type = c("RF", "XGBoost"),
  outdir = tempdir(),
  TaskName = NULL,
  TrueLabel = NULL,
  PlotAUC = NULL
)
```

Arguments

`CrcBiomeScreenObject`	A CrcBiomeScreenObject containing the model data and results
`model_type`	A character vector indicating the type of model to evaluate. Options are "RF" for Random Forest and "XGBoost" for XGBoost.
`outdir`	A character string. Path to the output directory where results (PDFs, RDS) should be saved. Defaults to tempdir().
`TaskName`	A character string used to label the output files and results.
`TrueLabel`	The true label for the classification task, which is used to evaluate the model's performance.
`PlotAUC`	A logical value indicating whether to plot the AUC curve. If TRUE, the AUC curve will be saved as a PDF file.

Value

A A `CrcBiomeScreen` object. object with with the evaluation results stored in the `EvaluateResult` slot.

Examples

```
# EvaluateModel() should be used after TrainModels() has been run.
# See the package vignette for a complete end-to-end example.
```

EvaluateRF *Evaluate the Random Forest model*

Description

Evaluate the Random Forest model

Usage

```
EvaluateRF(
  CrcBiomeScreenObject = NULL,
  outdir = tempdir(),
  TaskName = NULL,
  TrueLabel = NULL,
  PlotAUC = NULL
)
```

Arguments

CrcBiomeScreenObject	A CrcBiomeScreenObject containing the model data and results
outdir	The output directory where plots will be saved (default: tempdir()).
TaskName	A character string used to label the output files and results.
TrueLabel	The true label for the classification task, which is used to evaluate the model's performance.
PlotAUC	A logical value indicating whether to plot the AUC curve. If TRUE, the AUC curve will be saved as a PDF file.

Value

A CrcBiomeScreenObject with the evaluation results stored in the `EvaluateResult$RF` slot.

Examples

```
# Minimal runnable example demonstrating input structure for EvaluateRF

# Toy training + test matrices
train_df <- data.frame(x = c(1, 2), TrainLabel = factor(c("control", "CRC")))
test_df  <- data.frame(x = c(3, 4))

# Build minimal CrcBiomeScreen object
obj <- new("CrcBiomeScreen",
  AbsoluteAbundance = data.frame(),
  RelativeAbundance = data.frame(),
  TaxaData = data.frame(),
  SampleData = data.frame(),
  ModelData = list(
```

```
    Training = train_df,
    Test = test_df,
    TrainLabel = train_df$TrainLabel,
    TestLabel = factor(c("control", "CRC"))
  ),
  ModelResult = list(
    RF = list(best.params = list(
      num.trees = 1, mtry = 1, node_size = 1, sample_size = 1
    ))
  )
)

# NOT RUN: real evaluation uses ranger + pROC (too slow for BioC builds)
# out <- EvaluateRF(obj, TaskName = "toy", TrueLabel = "CRC")

obj
```

`EvaluateXGBoost` *Evaluate the XGBoost model*

Description

Evaluate the XGBoost model

Usage

```
EvaluateXGBoost(
  CrcBiomeScreenObject = NULL,
  outdir = tempdir(),
  TaskName = NULL,
  TrueLabel = NULL,
  PlotAUC = NULL
)
```

Arguments

`CrcBiomeScreenObject`	A CrcBiomeScreenObject containing the model data and results
`outdir`	The output directory where plots will be saved (default: tempdir()).
`TaskName`	A character string used to label the output files and results.
`TrueLabel`	The true label for the classification task, which is used to evaluate the model's performance.
`PlotAUC`	A logical value indicating whether to plot the AUC curve. If TRUE, the AUC curve will be saved as a PDF file.

Value

A CrcBiomeScreenObject with the evaluation results stored in the `EvaluateResult$XGBoost` slot.

Examples

```
# Minimal runnable example demonstrating input structure for EvaluateXGBoost

# Toy data for testing
train_df <- data.frame(x = c(1, 2), TrainLabel = factor(c("control", "CRC")))
test_df  <- data.frame(x = c(3, 4))

obj <- new("CrcBiomeScreen",
  AbsoluteAbundance = data.frame(),
  RelativeAbundance = data.frame(),
  TaxaData = data.frame(),
  SampleData = data.frame(),
  ModelData = list(
    Training = train_df,
    Test = test_df,
    TrainLabel = train_df$TrainLabel,
    TestLabel = factor(c("control", "CRC"))
  ),
  ModelResult = list(
    XGBoost = list(
      model = list(dummy_model = TRUE)  # placeholder model
    )
  )
)

# NOT RUN: actual evaluation needs xgboost + pROC
# out <- EvaluateXGBoost(obj, TaskName = "toy", TrueLabel = "CRC")

obj
```

`FilterDataSet` *Filter the CrcBiomeScreenObject dataset based on a specific label*

Description

Filter the CrcBiomeScreenObject dataset based on a specific label

Usage

```
FilterDataSet(
  CrcBiomeScreenObject = NULL,
  label = NULL,
  condition_col = "study_condition"
)
```

Arguments

`CrcBiomeScreenObject`
: A `CrcBiomeScreenObject` containing normalized microbiome data and sample metadata.

label	A character vector specifying the label(s) to filter the dataset by.
condition_col	A character string indicating the column in the SampleData that contains the condition labels (default is "study_condition").

Value

A A CrcBiomeScreen object. with filtered data based on the specified label.

Examples

```
# Create toy normalized data (5 samples, 2 taxa)
norm_data <- data.frame(
  TaxaA = c(10, 20, 15, 30, 10),
  TaxaB = c(5, 7, 6, 8, 6)
)
rownames(norm_data) <- paste0("S", 1:5)

# Create sample metadata
sample_info <- data.frame(
  study_condition = c("control", "CRC", "control", "CRC", "Adenoma"),
  country = c("US", "US", "UK", "UK", "US"),
  row.names = paste0("S", 1:5),
  stringsAsFactors = FALSE
)

# Construct a minimal CrcBiomeScreen object
toy_obj <- new(
  "CrcBiomeScreen",
  AbsoluteAbundance   = data.frame(),
  RelativeAbundance   = data.frame(),
  TaxaData            = data.frame(),
  SampleData          = sample_info,
  NormalizedData      = norm_data,
  TaxaLevelData       = NULL,
  OrginalNormalizedData = NULL,
  ValidationData      = NULL,
  ModelData           = NULL,
  ModelResult         = NULL,
  EvaluateResult      = list(),
  PredictResult       = NULL
)

# Filter to keep only CRC and control samples
filtered_obj <- FilterDataSet(
  toy_obj,
  label = c("CRC", "control"),
  condition_col = "study_condition"
)

# Inspect filtered SampleData
getSampleData(filtered_obj)
```

getAbsoluteAbundance *Accessor for AbsoluteAbundance slot of CrcBiomeScreen object*

Description

Accessor for AbsoluteAbundance slot of CrcBiomeScreen object

Usage

```
getAbsoluteAbundance(object)

## S4 method for signature 'CrcBiomeScreen'
getAbsoluteAbundance(object)
```

Arguments

object	A A CrcBiomeScreen object. object.

Value

A A CrcBiomeScreen object. object with A data.frame containing absolute abundance data.

Methods (by class)

- getAbsoluteAbundance(CrcBiomeScreen): Retrieve absolute abundance data from a CrcBiomeScreen object.

Examples

```
# Construct minimal example object
toy_obj <- CreateCrcBiomeScreenObject(
  AbsoluteAbundance = data.frame(TaxaA = c(1000, 2000)),
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

# Retrieve absolute abundance
getAbsoluteAbundance(toy_obj)
```

getModelData *Accessor for ModelData slot of CrcBiomeScreen object*

Description

Accessor for ModelData slot of CrcBiomeScreen object

Usage

```
getModelData(object)
```

Arguments

object	A A CrcBiomeScreen object. object.

Value

A A CrcBiomeScreen object. object with a data.frame containing model data.

Examples

```
rel_abund <- data.frame(
  S1 = c(10),
  S2 = c(20)
)
rownames(rel_abund) <- "TaxaA"

toy_taxa <- data.frame(
  Taxa = "TaxaA",
  stringsAsFactors = FALSE
)

toy_sample <- data.frame(
  number_reads = c(10000, 10000),
  condition = c("control", "CRC"),
  row.names = c("S1", "S2"),
  stringsAsFactors = FALSE
)

toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = rel_abund,
  TaxaData = toy_taxa,
  SampleData = toy_sample
)

getModelData(toy_obj)
```

getModelResult *Accessor for ModelResult slot of CrcBiomeScreen object*

Description

Accessor for ModelResult slot of CrcBiomeScreen object

Usage

```
getModelResult(object)

## S4 method for signature 'CrcBiomeScreen'
getModelResult(object)
```

Arguments

object	A A CrcBiomeScreen object. object.

Value

A A CrcBiomeScreen object. object with a list containing fitted model results.

Methods (by class)

- getModelResult(CrcBiomeScreen): Retrieve model results from a CrcBiomeScreen object.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

getModelData(toy_obj)
```

getNormalizedData *Accessor for NormalizedData slot of CrcBiomeScreen object*

Description

Retrieve normalized abundance data from a A `CrcBiomeScreen` object. object.

Usage

```
getNormalizedData(object)

## S4 method for signature 'CrcBiomeScreen'
getNormalizedData(object)
```

Arguments

`object`	A A `CrcBiomeScreen` object. object.

Value

A A `CrcBiomeScreen` object. object with a data.frame (or matrix) containing normalized abundance data.

Methods (by class)

- `getNormalizedData(CrcBiomeScreen)`: Retrieve normalized abundance data.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

getNormalizedData(toy_obj)
```

getOutlierSamples *Accessor for OutlierSamples*

Description

Accessor for OutlierSamples

Usage

```
getOutlierSamples(object)

## S4 method for signature 'CrcBiomeScreen'
getOutlierSamples(object)
```

Arguments

object	A A CrcBiomeScreen object. object.

Value

A character vector of detected outlier sample IDs, or NULL if no outliers have been recorded.

Methods (by class)

- getOutlierSamples(CrcBiomeScreen): results from a CrcBiomeScreen object.

Examples

```
# getOutlierSamples() is typically used after running qcByCmdscale().
# See the package vignette for a complete workflow example.
```

getPredictResult *Accessor for PredictResult slot of CrcBiomeScreen object*

Description

Accessor for PredictResult slot of CrcBiomeScreen object

Usage

```
getPredictResult(object)

## S4 method for signature 'CrcBiomeScreen'
getPredictResult(object)
```

Arguments

object	A A CrcBiomeScreen object. object.

Value

A A CrcBiomeScreen object. object with a list containing fitted Prediction results.

Methods (by class)

- getPredictResult(CrcBiomeScreen): Prediction results from a CrcBiomeScreen object.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

getPredictResult(toy_obj)
```

---

getRelativeAbundance *Accessor for RelativeAbundance slot of CrcBiomeScreen object*

Description

Accessor for RelativeAbundance slot of CrcBiomeScreen object

Usage

```
getRelativeAbundance(object)

## S4 method for signature 'CrcBiomeScreen'
getRelativeAbundance(object)
```

Arguments

object	A A CrcBiomeScreen object. object.

Value

A A CrcBiomeScreen object. object with a data.frame containing relative abundance data.

Methods (by class)

- `getRelativeAbundance(CrcBiomeScreen)`: Retrieve relative abundance data from a CrcBiomeScreen object.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

getRelativeAbundance(toy_obj)
```

`getSampleData` *Accessor for SampleData slot of CrcBiomeScreen object*

Description

Accessor for SampleData slot of CrcBiomeScreen object

Usage

```
getSampleData(object)

## S4 method for signature 'CrcBiomeScreen'
getSampleData(object)

## S4 method for signature 'CrcBiomeScreen'
getModelData(object)
```

Arguments

`object`	A A `CrcBiomeScreen` object. object.

Value

A A `CrcBiomeScreen` object. object with a data.frame containing sample metadata.

Methods (by class)

- `getSampleData(CrcBiomeScreen)`: Retrieve sample metadata from a CrcBiomeScreen object.
- `getModelData(CrcBiomeScreen)`: Retrieve sample metadata from a CrcBiomeScreen object.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

getSampleData(toy_obj)
```

`getTaxaData` *Accessor for TaxaData slot of CrcBiomeScreen object*

Description

Accessor for TaxaData slot of CrcBiomeScreen object

Usage

```
getTaxaData(object)

## S4 method for signature 'CrcBiomeScreen'
getTaxaData(object)
```

Arguments

`object`	A A `CrcBiomeScreen` object. object.

Value

A A `CrcBiomeScreen` object. object with a data.frame containing taxonomic annotations.

Methods (by class)

- `getTaxaData(CrcBiomeScreen)`: Retrieve taxonomic annotations from a CrcBiomeScreen object.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)
```

```
getTaxaData(toy_obj)
```

`getTaxaLevelData` *Accessor for TaxaLevelData slot*

Description

Retrieve the TaxaLevelData slot from a A `CrcBiomeScreen` object. object.

Usage

```
getTaxaLevelData(object)

## S4 method for signature 'CrcBiomeScreen'
getTaxaLevelData(object)
```

Arguments

`object`	A A `CrcBiomeScreen` object. object.

Value

A list containing taxonomic-level abundance data (e.g., genus-level or species-level data).

Examples

```
# Toy taxa in a simplified MetaPhlAn-like hierarchical format
toy_taxa <- data.frame(
  Taxa = c(
   "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderX|D_4__FamilyX|D_5__GenusA",
    "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderY|D_4__FamilyY|D_5__GenusB"
  ),
  stringsAsFactors = FALSE
)

# Toy abundance matrix (2 taxa, 2 samples)
toy_abs <- data.frame(
  S1 = c(10, 5),
  S2 = c(20, 15)
)
rownames(toy_abs) <- toy_taxa$Taxa

# Dummy sample metadata
toy_sample <- data.frame(
  sample_id = c("S1", "S2")
)

# Construct minimal CrcBiomeScreen object
toy_obj <- new(
```

```
  "CrcBiomeScreen",
  AbsoluteAbundance   = toy_abs,
  RelativeAbundance   = data.frame(),
  TaxaData            = toy_taxa,
  SampleData          = toy_sample,
  TaxaLevelData       = NULL,
  NormalizedData      = NULL,
  OrginalNormalizedData = NULL,
  ValidationData      = NULL,
  ModelData           = NULL,
  ModelResult         = NULL,
  EvaluateResult      = list(),
  PredictResult       = NULL
)

# Apply taxonomy splitting + keep genus level
toy_obj <- SplitTaxas(toy_obj)
genus_obj <- KeepTaxonomicLevel(toy_obj, level = "Genus")

# Inspect genus-level abundance
getTaxaLevelData(genus_obj)$GenusLevelData
```

`KeepTaxonomicLevel` *Summarize abundance data at a given taxonomic level*

Description

Aggregate absolute abundance data in a A `CrcBiomeScreen` object. object to a specified taxonomic level (e.g. "Genus" or "Family").

Usage

```
KeepTaxonomicLevel(CrcBiomeScreenObject, level = "Genus")
```

Arguments

`CrcBiomeScreenObject`	A A `CrcBiomeScreen` object. object.
`level`	Taxonomic level to summarize to. One of "Kingdom", "Phylum", "Class", "Order", "Family", "Genus", "Species".

Details

Keep a specific taxonomic level

This function aggregates abundance data to a specified taxonomic level.

Value

The CrcBiomeScreenObject with a new data frame aggregated at the specified level.

The same A `CrcBiomeScreen` object. object, updated with a slot `@TaxaLevelData` a new data frame in `@GenusLevelData` (or the corresponding level).

Examples

```
# Minimal fully runnable example for KeepTaxonomicLevel

# Toy taxa in a simplified MetaPhlAn-like hierarchical format
toy_taxa <- data.frame(
  Taxa = c(
   "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderX|D_4__FamilyX|D_5__GenusA",
    "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderY|D_4__FamilyY|D_5__GenusB"
  ),
  stringsAsFactors = FALSE
)

# Toy abundance matrix (2 taxa, 2 samples)
toy_abs <- data.frame(
  S1 = c(10, 5),
  S2 = c(20, 15)
)
rownames(toy_abs) <- toy_taxa$Taxa

# Dummy sample metadata
toy_sample <- data.frame(
  sample_id = c("S1", "S2")
)

# Construct minimal CrcBiomeScreen object
toy_obj <- new(
  "CrcBiomeScreen",
  AbsoluteAbundance   = toy_abs,
  RelativeAbundance   = data.frame(),
  TaxaData            = toy_taxa,
  SampleData          = toy_sample,
  TaxaLevelData       = NULL,
  NormalizedData      = NULL,
  OrginalNormalizedData = NULL,
  ValidationData      = NULL,
  ModelData           = NULL,
  ModelResult         = NULL,
  EvaluateResult      = list(),
  PredictResult       = NULL
)

# Apply taxonomy splitting + keep genus level
toy_obj <- SplitTaxas(toy_obj)
genus_obj <- KeepTaxonomicLevel(toy_obj, level = "Genus")

# Inspect genus-level abundance
```

```
getTaxaLevelData(genus_obj)$GenusLevelData
```

LoadTaxaTable *Load a custom taxa table for ASV/OTU data*

Description

This function allows users to load their own taxonomic assignments for ASV/OTU data. The input table should map sequence IDs to their full taxonomic lineage.

Usage

```
LoadTaxaTable(CrcBiomeScreenObject, taxa_table, id_column, taxa_column)
```

Arguments

CrcBiomeScreenObject	The CrcBiomeScreenObject to which the taxa table will be added.
taxa_table	A data frame. It must contain at least two columns: one for sequence IDs (e.g., ASV or OTU names) and another for the corresponding taxonomic lineage string.
id_column	The name of the column in taxa_table that contains the sequence IDs (ASV/OTU names).
taxa_column	The name of the column in taxa_table that contains the taxonomic lineage string.

Value

The A CrcBiomeScreen object. with the loaded taxa table.

Examples

```
## Minimal example using CreateCrcBiomeScreenObject and LoadTaxaTable

# Toy relative abundance matrix: 1 taxa (row), 2 samples (columns)
rel_abund <- data.frame(
  S1 = 10,
  S2 = 20,
  row.names = "TaxaA"
)

# Sample metadata with required 'number_reads' column
sample_info <- data.frame(
  number_reads = c(10000, 12000),
  condition    = c("control", "CRC"),
  row.names    = c("S1", "S2"),
  stringsAsFactors = FALSE
```

```
)

# Simple taxa table matching the row names of rel_abund
taxa_info <- data.frame(
  Taxa = rownames(rel_abund),
  stringsAsFactors = FALSE
)

# Construct a minimal CrcBiomeScreen object
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = rel_abund,
  TaxaData          = taxa_info,
  SampleData        = sample_info
)

# External taxonomy table to be merged in by LoadTaxaTable
my_taxa_table <- data.frame(
  ASV_ID   = "TaxaA",
  Taxonomy = "D_0__Bacteria;D_1__Firmicutes;D_2__Clostridia",
  stringsAsFactors = FALSE
)

# Load taxonomy table into the CrcBiomeScreen object
toy_obj <- LoadTaxaTable(
  CrcBiomeScreenObject = toy_obj,
  taxa_table  = my_taxa_table,
  id_column   = "ASV_ID",
  taxa_column = "Taxonomy"
)

# Inspect updated taxonomy using the accessor
head(getTaxaData(toy_obj))
```

`ModelingRF` *The packaging function for Random Forest modeling*

Description

The packaging function for Random Forest modeling

Usage

```
ModelingRF(
  CrcBiomeScreenObject = NULL,
  k.rf = n_cv,
  TaskName = NULL,
  TrueLabel = NULL,
  num_cores = NULL
)
```

Arguments

CrcBiomeScreenObject	A CrcBiomeScreenObject containing normalized microbiome data, sample metadata, etc.
k.rf	Set the number of cross validation
TaskName	A character string used to label the output
TrueLabel	This label is the future prediction target
num_cores	Set the number of the cores in parallel computing

Value

A A CrcBiomeScreen object. with the modelling results.

Examples

```
# Minimal runnable example illustrating required inputs for ModelingRF

# Create toy relative abundance matrix
rel_abund <- data.frame(S1 = 10, S2 = 20)
rownames(rel_abund) <- "TaxaA"

# Create sample metadata
sample_info <- data.frame(
  number_reads = c(10000, 12000),
  condition = c("control", "CRC"),
  row.names = c("S1", "S2")
)

# Construct minimal CrcBiomeScreen object
obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = rel_abund,
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = sample_info,
)


# NOT RUN: Actual model fitting is time-consuming
# out <- ModelingRF(
#   CrcBiomeScreenObject = obj,
#   k.rf = 2,
#   TaskName = "toy_RF",
#   TrueLabel = c("control", "CRC"),
#   num_cores = 1
# )

# The example instead demonstrates setup only
obj
```

ModelingRF_noweights *The function for modeling random forest without using class weights*

Description

The function for modeling random forest without using class weights

Usage

```
ModelingRF_noweights(
  CrcBiomeScreenObject = NULL,
  k.rf = n_cv,
  TaskName = NULL,
  TrueLabel = NULL,
  num_cores = NULL
)
```

Arguments

CrcBiomeScreenObject	A CrcBiomeScreenObject containing normalized microbiome data, sample metadata, etc.
k.rf	Set the number of cross validation
TaskName	A character string used to label the output
TrueLabel	This label is the future prediction target
num_cores	Set the number of the cores in parallel computing

Value

A A CrcBiomeScreen object. with the modelling results.

Examples

```
# Minimal runnable example for ModelingRF_noweights

rel_abund <- data.frame(S1 = 10, S2 = 20)
rownames(rel_abund) <- "TaxaA"

sample_info <- data.frame(
  number_reads = c(10000, 12000),
  condition = c("control", "CRC"),
  row.names = c("S1", "S2")
)

obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = rel_abund,
  TaxaData = data.frame(Taxa = "TaxaA"),
```

```
  SampleData = sample_info
)


# out <- ModelingRF_noweights(
#   CrcBiomeScreenObject = obj,
#   k.rf = 2,
#   TaskName = "toy_RF_nw",
#   TrueLabel = c("control", "CRC"),
#   num_cores = 1
# )

obj
```

`ModelingXGBoost` *The packaging function for XGBoost modeling*

Description

The packaging function for XGBoost modeling

Usage

```
ModelingXGBoost(
  CrcBiomeScreenObject = NULL,
  k.rf = 10,
  repeats = 5,
  TaskName = NULL,
  TrueLabel = NULL,
  num_cores = num_cores
)
```

Arguments

`CrcBiomeScreenObject`	A `CrcBiomeScreenObject` containing normalized microbiome data, sample metadata, etc.
`k.rf`	Set the number of cross validation
`repeats`	Set the number of repeats in cross validation
`TaskName`	A character string used to label the output
`TrueLabel`	This label is the future prediction target
`num_cores`	Set the number of the cores in parallel computing

Value

A A `CrcBiomeScreen` object. with the modelling results.

Examples

```
# Minimal runnable example for ModelingXGBoost

rel_abund <- data.frame(S1 = 10, S2 = 20)
rownames(rel_abund) <- "TaxaA"

sample_info <- data.frame(
  number_reads = c(10000, 12000),
  condition = c("control", "CRC"),
  row.names = c("S1", "S2")
)

obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = rel_abund,
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = sample_info
)


# out <- ModelingXGBoost(
# CrcBiomeScreenObject = obj,
#  k.rf = 2,
# TaskName = "toy_XGB",
# TrueLabel = c("control", "CRC"),
#  num_cores = 1
# )

obj
```

ModelingXGBoost_noweights

The packaging function for XGBoost modeling without using class weights

---

Description

The packaging function for XGBoost modeling without using class weights

Usage

```
ModelingXGBoost_noweights(
  CrcBiomeScreenObject = NULL,
  k.rf = 10,
  repeats = 5,
  TaskName = NULL,
  TrueLabel = NULL,
  num_cores = num_cores
)
```

Arguments

CrcBiomeScreenObject	A CrcBiomeScreenObject containing normalized microbiome data, sample metadata, etc.
k.rf	Set the number of cross validation
repeats	Set the number of repeats in cross validation
TaskName	A character string used to label the output
TrueLabel	This label is the future prediction target
num_cores	Set the number of the cores in parallel computing

Value

A A CrcBiomeScreen object. with the modelling results.

Examples

```
# Minimal runnable example for ModelingXGBoost_noweights

rel_abund <- data.frame(S1 = 10, S2 = 20)
rownames(rel_abund) <- "TaxaA"

sample_info <- data.frame(
  number_reads = c(10000, 12000),
  condition = c("control", "CRC"),
  row.names = c("S1", "S2")
)

obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = rel_abund,
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = sample_info
)


# out <- ModelingXGBoost_noweights(
# CrcBiomeScreenObject = obj,
# k.rf = 2,
# TaskName = "toy_XGB_nw",
# TrueLabel = c("control", "CRC"),
# num_cores = 1
#)

obj
```

NHSBCSP_screeningData *NHSBCSP screening dataset*

Description

A toy screening dataset derived from the NHS Bowel Cancer Screening Programme.

Usage

```
NHSBCSP_screeningData
```

Format

A data frame containing sample-level metadata and abundance data. The first two columns(index and grp) store sample identifiers and metadata, and the remaining columns contain microbial abundance features used for package examples and demonstrations.(2252 obs. of 647 variables).

Source

NHS Bowel Cancer Screening Programme

NormalizeData *Normalise the absolute data to relative data by using Total Sum Scaling and Geometric Mean of Pairwise Ratios (GMPR)*

Description

Normalise the absolute data to relative data by using Total Sum Scaling and Geometric Mean of Pairwise Ratios (GMPR)

Usage

```
NormalizeData(CrcBiomeScreenObject = NULL, method = NULL, level = NULL)
```

Arguments

CrcBiomeScreenObject	From the CreateCrcBiomeScreenObject()
method	"TSS" or "GMPR"
level	Taxonomic level for normalization, e.g., "Genus"

Value

A A CrcBiomeScreen object. with the updated NormalizedData.

Examples

```
# Minimal runnable example for NormalizeData

# Toy taxa in a simplified MetaPhlAn-like hierarchical format
toy_taxa <- data.frame(
  Taxa = c(
   "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderX|D_4__FamilyX|D_5__GenusA",
    "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderY|D_4__FamilyY|D_5__GenusB"
  ),
  stringsAsFactors = FALSE
)

# Toy abundance matrix (2 taxa, 2 samples)
toy_abs <- data.frame(
  S1 = c(10, 5),
  S2 = c(20, 15)
)
rownames(toy_abs) <- toy_taxa$Taxa

# Dummy sample metadata
toy_sample <- data.frame(
  sample_id = c("S1", "S2")
)

# Construct minimal CrcBiomeScreen object
toy_obj <- new(
  "CrcBiomeScreen",
  AbsoluteAbundance   = toy_abs,
  RelativeAbundance   = data.frame(),
  TaxaData            = toy_taxa,
  SampleData          = toy_sample,
  TaxaLevelData       = NULL,
  NormalizedData      = NULL,
  OrginalNormalizedData = NULL,
  ValidationData      = NULL,
  ModelData           = NULL,
  ModelResult         = NULL,
  EvaluateResult      = list(),
  PredictResult       = NULL
)

# Apply taxonomy splitting + keep genus level
toy_obj <- SplitTaxas(toy_obj)
toy_obj <- KeepTaxonomicLevel(toy_obj, level = "Genus")
toy_obj <- NormalizeData(toy_obj, method = "TSS", level = "Genus")
# Inspect normalized results
head(getNormalizedData(toy_obj))
```

`PredictCrcBiomeScreen` *Predict the class and probabilities for new data*

Description

Predict the class and probabilities for new data

Usage

```
PredictCrcBiomeScreen(
  CrcBiomeScreenObject,
  newdata,
  model_type = c("RF", "XGBoost")
)
```

Arguments

CrcBiomeScreenObject	The object containing the trained model.
newdata	The data frame or matrix of new features to predict on.
model_type	The type of model to use for prediction ("RF" or "XGBoost").

Value

A A CrcBiomeScreen object. with a data frame containing sample-specific predictions.

Examples

```
# --- Minimal runnable example ---

# Create a tiny toy dataset (2 samples, 2 features)
newdata <- data.frame(
  Feature1 = c(0.2, 0.8),
  Feature2 = c(0.7, 0.3)
)
rownames(newdata) <- c("S1", "S2")

# Create a minimal CrcBiomeScreen object with a fake RF model
# Instead of training, we attach a dummy model object whose predict()
# method returns fixed probabilities.

fake_rf_model <- structure(
  list(),
  class = "fakeRF"
)

# Define a simple predict method for this fake model
predict.fakeRF <- function(object, data, type = "response") {
  probs <- matrix(
    c(0.8, 0.2,   # S1: control=0.8, CRC=0.2
      0.3, 0.7),  # S2: control=0.3, CRC=0.7
    ncol = 2,
    byrow = TRUE
  )
  colnames(probs) <- c("control", "CRC")
```

```
  list(predictions = probs)
}

toy_obj <- new(
  "CrcBiomeScreen",
  AbsoluteAbundance = data.frame(),
  RelativeAbundance = data.frame(),
  TaxaData = data.frame(),
  SampleData = data.frame(),
  ModelResult = list(),
  EvaluateResult = list(
    RF = list(RF.Model = fake_rf_model)
  )
)

# Run prediction
pred_obj <- PredictCrcBiomeScreen(
  CrcBiomeScreenObject = toy_obj,
  newdata = newdata,
  model_type = "RF"
)

getPredictResult(pred_obj)$RF
```

`qcByCmdscale` *Quality control using classical MDS and outlier detection*

Description

This function performs quality control on microbiome data by applying classical multidimensional scaling (MDS) on the relative abundance matrix. It identifies outlier samples based on their Euclidean distance to the centroid in the first two MDS dimensions.

Usage

```
qcByCmdscale(
  CrcBiomeScreenObject,
  TaskName = NULL,
  outdir = tempdir(),
  normalize_method = NULL,
  threshold_sd = 1,
  plot = TRUE
)
```

Arguments

`CrcBiomeScreenObject`
: A `CrcBiomeScreenObject` containing normalized microbiome data, sample metadata, etc.

TaskName	A character string used to label the output plot and PDF filename.
outdir	The output directory where plots will be saved (default: tempdir()).description
normalize_method	A character string indicating the normalization method used (e.g., "GMPR"). Used for labeling only.
threshold_sd	Numeric value indicating how many standard deviations above the mean distance should be considered an outlier (default is 1).
plot	Logical value indicating whether to generate and save the MDS plot (default is TRUE).

Details

Outlier samples are removed from the normalized matrix and sample metadata, and the results are visualized and saved as a PDF.

The function calculates the Euclidean distance between samples in the 2D MDS space. Samples whose distance to the centroid exceeds the threshold (mean + threshold_sd * SD) are considered outliers.

A PDF plot is saved to the working directory, showing sample positions in MDS space with outliers highlighted in red.

Value

A modified CrcBiomeScreenObject where:

- NormalizedData contains filtered data with outliers removed.
- SampleData is updated to exclude outlier samples.
- OutlierSamples is a character vector of sample IDs identified as outliers.
- OrginalNormalizedData stores the unfiltered data matrix before QC.

A A CrcBiomeScreen object. with outliers.

Examples

```
# Minimal toy object for QC example
toy_sampledata <- data.frame(
  sample_id = paste0("S", 1:4),
  study_condition = c("control", "CRC", "control", "CRC")
)

toy_norm <- data.frame(
  S1 = c(1, 2, 3),
  S2 = c(2, 3, 4),
  S3 = c(1, 1, 1),
  S4 = c(3, 2, 1)
)
rownames(toy_norm) <- c("g1", "g2", "g3")

toy_obj <- new(
```

```
  "CrcBiomeScreen",
  AbsoluteAbundance = data.frame(),
  RelativeAbundance = data.frame(),
  TaxaData = data.frame(),
  SampleData = toy_sampledata,
  NormalizedData = toy_norm,
  ModelData = list()
)

# Run QC with 1 SD threshold (small example)
qc_obj <- qcByCmdscale(
  toy_obj,
  TaskName = "ToyQC",
  normalize_method = "GMPR",
  threshold = 1
)

getSampleData(qc_obj)
```

`RunScreening` *Run the screening process for the microbiome data*

Description

Run the screening process for the microbiome data

Usage

```
RunScreening(
  obj,
  model_type = NULL,
  split.requirement = NULL,
  TaskName = TaskName,
  partition = NULL,
  ClassWeights = NULL,
  n_cv = NULL,
  ValidationData = NULL,
  TrueLabel = NULL,
  num_cores = NULL
)
```

Arguments

`obj`	A `CrcBiomeScreenObject` containing normalized microbiome data, sample metadata, etc.
`model_type`	Model type to be used, default is "RF"

split.requirement
: A list containing the label and condition column for splitting the dataset, default is NULL

TaskName
: A character string used to label the output

partition
: The number of partitions for cross-validation

ClassWeights
: Whether to use class weights in the model training, default is NULL

n_cv
: The number of cross-validation folds, default is NULL

ValidationData
: A CrcBiomeScreenObject containing validation data for model evaluation, default is NULL

TrueLabel
: The true label for the classification task, which is used to evaluate the model's performance

num_cores
: Set the number of cores for parallel computing, default is NULL

Value

A A CrcBiomeScreen object. with the results of the screening process, including model training, evaluation, and validation.

Examples

```
set.seed(123)

# -------------------------
# Toy taxonomy
# -------------------------
toy_taxa_strings <- c(
  "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderA|D_4__FamilyA|D_5__GenusA",
  "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderB|D_4__FamilyB|D_5__GenusB",
  "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderC|D_4__FamilyC|D_5__GenusC",
 "D_0__Bacteria|D_1__Bacteroidetes|D_2__Bacteroidia|D_3__OrderD|D_4__FamilyD|D_5__GenusD",
 "D_0__Bacteria|D_1__Bacteroidetes|D_2__Bacteroidia|D_3__OrderE|D_4__FamilyE|D_5__GenusE",
 "D_0__Bacteria|D_1__Proteobacteria|D_2__Gammaproteobacteria|D_3__OrderF|D_4__FamilyF|D_5__GenusF"
)

toy_taxa <- data.frame(
  Taxa = toy_taxa_strings,
  stringsAsFactors = FALSE
)

# -------------------------
# Toy training data
# -------------------------
train_samples <- paste0("S", 1:12)

toy_abs <- matrix(
  c(
    rpois(6 * 6, lambda = 54.8887777),
    rpois(6 * 6, lambda = 55)
  ),
```

```
  nrow = 6,
  ncol = 12
)

rownames(toy_abs) <- toy_taxa_strings
colnames(toy_abs) <- train_samples
toy_abs <- as.data.frame(toy_abs)

toy_sample <- data.frame(
  number_reads = rep(10000, 12),
  study_condition = c(rep("control", 6), rep("CRC", 6)),
  row.names = train_samples,
  stringsAsFactors = FALSE
)

obj <- CreateCrcBiomeScreenObject(
  AbsoluteAbundance = toy_abs,
  TaxaData = toy_taxa,
  SampleData = toy_sample
)

obj <- SplitTaxas(obj)
obj <- KeepTaxonomicLevel(obj, level = "Genus")
obj <- NormalizeData(obj, method = "TSS", level = "Genus")

# -------------------------
# Toy validation data
# -------------------------
val_taxa <- data.frame(
  Taxa = toy_taxa_strings,
  stringsAsFactors = FALSE
)

val_samples <- paste0("V", 1:8)

val_abund <- matrix(
  c(
    rpois(6 * 4, lambda = 38),
    rpois(6 * 4, lambda = 48)
  ),
  nrow = 6,
  ncol = 8
)

rownames(val_abund) <- toy_taxa_strings
colnames(val_abund) <- val_samples
val_abund <- as.data.frame(val_abund)

val_sample <- data.frame(
  number_reads = rep(10000, 8),
  study_condition = c(rep("control", 4), rep("CRC", 4)),
  condition = c(rep("control", 4), rep("CRC", 4)),
  row.names = val_samples,
```

```
  stringsAsFactors = FALSE
)

val_obj <- CreateCrcBiomeScreenObject(
  AbsoluteAbundance = val_abund,
  TaxaData = val_taxa,
  SampleData = val_sample
)

val_obj <- SplitTaxas(val_obj)
val_obj <- KeepTaxonomicLevel(val_obj, level = "Genus")
val_obj <- NormalizeData(val_obj, method = "TSS", level = "Genus")

obj <- RunScreening(
  obj = obj,
  model_type = "RF",
  partition = 0.7,
  split.requirement = list(
    label = c("control", "CRC"),
    condition_col = "study_condition"
  ),
  ClassWeights = FALSE,
  n_cv = 2,
  num_cores = 1,
  TaskName = "RF_TSS_toydata",
  ValidationData = val_obj,
  TrueLabel = "CRC"
)
```

`setNormalizedData-setter` *setNormalizedData<-: Setter for NormalizedData slot of CrcBiomeScreen object*

Description

setNormalizedData<-: Setter for NormalizedData slot of CrcBiomeScreen object

Usage

```
setNormalizedData(object) <- value

## S4 replacement method for signature 'CrcBiomeScreen'
setNormalizedData(object) <- value
```

Arguments

`object`	A A `CrcBiomeScreen` object. object.
`value`	A data.frame or matrix containing normalized abundance data.

Value

A A CrcBiomeScreen object. object with a modified A CrcBiomeScreen object. object.

Functions

- setNormalizedData(CrcBiomeScreen) <- value: Replace the NormalizedData slot of a CrcBiomeScreen object.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

setNormalizedData(toy_obj) <- data.frame(n1 = 1:2)
getNormalizedData(toy_obj)
```

---

setTaxaData-setter *setTaxaData<-: Setter for TaxaData slot of CrcBiomeScreen object*

Description

setTaxaData<-: Setter for TaxaData slot of CrcBiomeScreen object

Usage

```
setTaxaData(object) <- value

## S4 replacement method for signature 'CrcBiomeScreen'
setTaxaData(object) <- value
```

Arguments

object	A A CrcBiomeScreen object. object.
value	A data.frame containing updated taxonomic annotations.

Value

A A CrcBiomeScreen object. object with a modified A CrcBiomeScreen object. object.

Functions

- setTaxaData(CrcBiomeScreen) <- value: Replace the TaxaData slot of a CrcBiomeScreen object.

Examples

```
toy_obj <- CreateCrcBiomeScreenObject(
  RelativeAbundance = data.frame(TaxaA = c(10, 20)),
  TaxaData = data.frame(Taxa = "TaxaA"),
  SampleData = data.frame(
    number_reads = 10000,
    condition = "control"
  )
)

setTaxaData(toy_obj) <- data.frame(Taxa = "NewTaxa")
getTaxaData(toy_obj)
```

`SplitDataSet` *Split the dataset into training and test sets*

Description

Split the dataset into training and test sets

Usage

```
SplitDataSet(
  CrcBiomeScreenObject = NULL,
  label = NULL,
  partition = NULL,
  condition_col = "study_condition"
)
```

Arguments

`CrcBiomeScreenObject`	From the CreateCrcBiomeScreenObject()
`label`	Divide the data set by the binary-label
`partition`	The ratio of dividing the data set
`condition_col`	The colname of label in SampleData

Value

A A `CrcBiomeScreen` object. with CrcBiomeScreenObject@ModelData

Examples

```
# Minimal toy object for dataset splitting

# Example normalized data (4 samples, 2 taxa)
toy_norm <- data.frame(
  TaxaA = c(10, 20, 15, 30),
```

```
  TaxaB = c( 5,  7,  6,  8)
)
rownames(toy_norm) <- paste0("S", 1:4)

# Sample metadata with conditions
toy_sampledata <- data.frame(
  study_condition = c("control", "CRC", "control", "CRC"),
  row.names = paste0("S", 1:4)
)

# Construct a minimal CrcBiomeScreen object
toy_obj <- new(
  "CrcBiomeScreen",
  AbsoluteAbundance   = data.frame(),
  RelativeAbundance   = data.frame(),
  TaxaData            = data.frame(),
  SampleData          = toy_sampledata,
  NormalizedData      = toy_norm,   # IMPORTANT: SplitDataSet needs this
  TaxaLevelData       = NULL,
  OrginalNormalizedData = NULL,
  ValidationData      = NULL,
  ModelData           = list(),
  ModelResult         = NULL,
  EvaluateResult      = list(),
  PredictResult       = NULL
)

# Split into training/testing sets with 70/30 ratio
toy_split <- SplitDataSet(
  toy_obj,
  label = c("control", "CRC"),
  partition = 0.7,
  condition_col = "study_condition"
)

# Inspect training labels
getModelData(toy_split)$TrainLabel
```

`SplitTaxas` *Split and clean taxonomy strings*

Description

This function automatically detects the taxonomy string format (e.g., MetaPhlAn, QIIME, SILVA, GTDB), splits the string into standard taxonomic ranks (Kingdom to Species), retains the original taxonomy string in a new column (`OriginalTaxa`), and refines labels such as "uncultured" or "unclassified" by appending the parent rank.

Usage

```
SplitTaxas(CrcBiomeScreenObject)
```

Arguments

CrcBiomeScreenObject
: A A CrcBiomeScreen object. object.

Value

A A CrcBiomeScreen object. with TaxaData.

Examples

```
# Minimal toy object for SplitTaxas demonstration

# Example taxonomic strings with up to Genus level
toy_taxa <- data.frame(
    Taxa = c(
        "D_0__Bacteria;D_1__Firmicutes;D_2__Clostridia;D_3__Lachnospirales;
        D_4__Lachnospiraceae;D_5__Roseburia","D_0__Bacteria;D_1__Firmicutes;
        D_2__Bacilli;D_3__Lactobacillales;D_4__Lactobacillaceae;
        D_5__Lactobacillus"),
    stringsAsFactors = FALSE
)

# Minimal object containing only the TaxaData slot needed for splitting
toy_obj <- new(
    "CrcBiomeScreen",
    AbsoluteAbundance   = data.frame(),
    RelativeAbundance   = data.frame(),
    TaxaData            = toy_taxa,
    SampleData          = data.frame(),
    TaxaLevelData       = NULL,
    NormalizedData      = NULL,
    OrginalNormalizedData = NULL,
    ValidationData      = NULL,
    ModelData           = NULL,
    ModelResult         = NULL,
    EvaluateResult      = list(),
    PredictResult       = NULL
)

# Run taxonomic splitting
SplitTaxas(toy_obj)
```

Thomas_2018_RelativeAbundance *Thomas 2018 relative abundance dataset*

Description

A relative abundance dataset derived from curatedMetagenomicData and included for package examples. 2021-03-31.ThomasAM_2018a.relative_abundance: Formal class 'TreeSummarizedExperiment' from package "TreeSummarizedExperiment" with 14 slots.

Usage

```
Thomas_2018_RelativeAbundance
```

Format

A named list containing relative abundance objects for demonstration.

Source

curatedMetagenomicData

TrainModels *Train the different models*

Description

Train the different models

Usage

```
TrainModels(
  CrcBiomeScreenObject = NULL,
  model_type = c("RF", "XGBoost"),
  ClassWeights = TRUE,
  n_cv = 10,
  TaskName = NULL,
  TrueLabel = NULL,
  num_cores = NULL
)
```

Arguments

CrcBiomeScreenObject	A CrcBiomeScreenObject containing normalized microbiome data, sample metadata, etc.
model_type	Select the method for modeling
ClassWeights	Choose using the class weights or not
n_cv	Set the number of cross validation
TaskName	A character string used to label the output
TrueLabel	This label is the future prediction target
num_cores	Set the number of the cores in parallel computing

Value

A A CrcBiomeScreen object. with training results.

Examples

```
set.seed(123)
toy_data <- matrix(rpois(6 * 10, 50), nrow = 6)
colnames(toy_data) <- paste0("S", 1:10)
rownames(toy_data) <- paste0("Taxa", 1:6)

toy_taxa <- data.frame(Taxa = rownames(toy_data))
toy_sample <- data.frame(
  study_condition = rep(c("control", "CRC"), each = 5),
  row.names = colnames(toy_data)
)

obj <- CreateCrcBiomeScreenObject(
  AbsoluteAbundance = as.data.frame(toy_data),
  TaxaData = toy_taxa,
  SampleData = toy_sample
)
obj <- SplitTaxas(obj)
obj <- KeepTaxonomicLevel(obj, level = "Genus")
obj <- NormalizeData(obj, method = "TSS", level = "Genus")
obj <- SplitDataSet(obj, label = c("control","CRC"), partition = 0.7)

obj <- TrainModels(
  obj,
  model_type = "RF",
  TrueLabel = "CRC",
  n_cv = 2,
  num_cores = 1
)
```

`ValidateModelOnData` *Predict the validation data by using the trained model in CrcBiomeScreenObject*

Description

Predict the validation data by using the trained model in CrcBiomeScreenObject

Usage

```
ValidateModelOnData(
  CrcBiomeScreenObject = NULL,
  model_type = NULL,
  ValidationData = NULL,
  TaskName = NULL,
  TrueLabel = NULL,
  condition_col = "study_condition",
  PlotAUC = NULL,
  outdir = tempdir()
)
```

Arguments

`CrcBiomeScreenObject`	A CrcBiomeScreenObject containing the model and data to be evaluated.
`model_type`	The type of model to be evaluated, either "RF" for Random Forest or "XGBoost".
`ValidationData`	A CrcBiomeScreenObject containing the validation data to be used for model evaluation.
`TaskName`	A character string used to label the output files and results.
`TrueLabel`	The true label for the classification task, which is used to evaluate the model's performance.
`condition_col`	The column name in the SampleData that contains the study condition labels. Default is "study_condition".
`PlotAUC`	A logical value indicating whether to plot the AUC curve. If TRUE, the AUC curve will be saved as a PDF file.
`outdir`	The output directory where plots will be saved (default: tempdir()).

Value

A CrcBiomeScreenObject with the evaluation results stored in the `PredictResult` slot for the specified model type.

Examples

```
set.seed(123)

# -------------------------
# Toy taxonomy
# -------------------------
toy_taxa_strings <- c(
  "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderA|D_4__FamilyA|D_5__GenusA",
  "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderB|D_4__FamilyB|D_5__GenusB",
  "D_0__Bacteria|D_1__Firmicutes|D_2__Clostridia|D_3__OrderC|D_4__FamilyC|D_5__GenusC",
 "D_0__Bacteria|D_1__Bacteroidetes|D_2__Bacteroidia|D_3__OrderD|D_4__FamilyD|D_5__GenusD",
 "D_0__Bacteria|D_1__Bacteroidetes|D_2__Bacteroidia|D_3__OrderE|D_4__FamilyE|D_5__GenusE",
 "D_0__Bacteria|D_1__Proteobacteria|D_2__Gammaproteobacteria|D_3__OrderF|D_4__FamilyF|D_5__GenusF"
)

toy_taxa <- data.frame(
  Taxa = toy_taxa_strings,
  stringsAsFactors = FALSE
)

# -------------------------
# Toy training data
# -------------------------
train_samples <- paste0("S", 1:12)

toy_abs <- matrix(
  c(
    rpois(6 * 6, lambda = 54.8887777),
    rpois(6 * 6, lambda = 55)
  ),
  nrow = 6,
  ncol = 12
)

rownames(toy_abs) <- toy_taxa_strings
colnames(toy_abs) <- train_samples
toy_abs <- as.data.frame(toy_abs)

toy_sample <- data.frame(
  number_reads = rep(10000, 12),
  study_condition = c(rep("control", 6), rep("CRC", 6)),
  row.names = train_samples,
  stringsAsFactors = FALSE
)

obj <- CreateCrcBiomeScreenObject(
  AbsoluteAbundance = toy_abs,
  TaxaData = toy_taxa,
  SampleData = toy_sample
)

obj <- SplitTaxas(obj)
```

```
obj <- KeepTaxonomicLevel(obj, level = "Genus")
obj <- NormalizeData(obj, method = "TSS", level = "Genus")

obj <- SplitDataSet(
  obj,
  label = c("control", "CRC"),
  partition = 0.7
)

obj <- TrainModels(
  obj,
  model_type = "RF",
  TaskName = "toy_rf",
  ClassWeights = FALSE,
  TrueLabel = "CRC",
  num_cores = 1,
  n_cv = 2
)

obj <- EvaluateModel(
  obj,
  model_type = "RF",
  TaskName = "ToyData_RF_Test",
  TrueLabel = "CRC",
  PlotAUC = FALSE
)

# -------------------------
# Toy validation data
# -------------------------
val_samples <- paste0("V", 1:8)

val_abund <- matrix(
  c(
    rpois(6 * 4, lambda = 38),
    rpois(6 * 4, lambda = 48)
  ),
  nrow = 6,
  ncol = 8
)

rownames(val_abund) <- toy_taxa_strings
colnames(val_abund) <- val_samples
val_abund <- as.data.frame(val_abund)

val_sample <- data.frame(
  number_reads = rep(10000, 8),
  study_condition = c(rep("control", 4), rep("CRC", 4)),
  condition = c(rep("control", 4), rep("CRC", 4)),
  row.names = val_samples,
  stringsAsFactors = FALSE
)
```

```
val_obj <- CreateCrcBiomeScreenObject(
  AbsoluteAbundance = val_abund,
  TaxaData = toy_taxa,
  SampleData = val_sample
)

val_obj <- SplitTaxas(val_obj)
val_obj <- KeepTaxonomicLevel(val_obj, level = "Genus")
val_obj <- NormalizeData(val_obj, method = "TSS", level = "Genus")

# -------------------------
# Align features
# -------------------------
train_norm <- getNormalizedData(obj)
val_norm <- getNormalizedData(val_obj)

common_features <- intersect(colnames(train_norm), colnames(val_norm))

setNormalizedData(obj) <- train_norm[, common_features, drop = FALSE]
setNormalizedData(val_obj) <- val_norm[, common_features, drop = FALSE]

# -------------------------
# Validate model
# -------------------------
validated_obj <- ValidateModelOnData(
  obj,
  ValidationData = val_obj,
  model_type = "RF",
  TaskName = "toy_validation",
  TrueLabel = "CRC",
  PlotAUC = FALSE
)
```

`ZellerG_2014_RelativeAbundance` *Zeller 2014 relative abundance dataset*

Description

A relative abundance dataset derived from curatedMetagenomicData and included for package examples. 2021-03-31.ZellerG_2014.relative_abundance: Formal class 'TreeSummarizedExperiment' from package "TreeSummarizedExperiment" with 14 slots.

Usage

```
ZellerG_2014_RelativeAbundance
```

Format

A named list containing relative abundance objects for demonstration.

Source

curatedMetagenomicData

Index

∗ **datasets**
NHSBCSP_screeningData, 32
Thomas_2018_RelativeAbundance, 45
ZellerG_2014_RelativeAbundance, 50

checkClassBalance, 3
CrcBiomeScreen-class, 4
CreateCrcBiomeScreenObject, 4
CreateCrcBiomeScreenObjectFromTSE, 6

EvaluateCrcBiomeScreen, 7
EvaluateModel, 9
EvaluateRF, 10
EvaluateXGBoost, 11

FilterDataSet, 12

getAbsoluteAbundance, 14
getAbsoluteAbundance,CrcBiomeScreen-method *(getAbsoluteAbundance)*, 14
getModelData, 15
getModelData,CrcBiomeScreen-method *(getSampleData)*, 20
getModelResult, 16
getModelResult,CrcBiomeScreen-method *(getModelResult)*, 16
getNormalizedData, 17
getNormalizedData,CrcBiomeScreen-method *(getNormalizedData)*, 17
getOutlierSamples, 18
getOutlierSamples,CrcBiomeScreen-method *(getOutlierSamples)*, 18
getPredictResult, 18
getPredictResult,CrcBiomeScreen-method *(getPredictResult)*, 18
getRelativeAbundance, 19
getRelativeAbundance,CrcBiomeScreen-method *(getRelativeAbundance)*, 19
getSampleData, 20
getSampleData,CrcBiomeScreen-method *(getSampleData)*, 20
getTaxaData, 21
getTaxaData,CrcBiomeScreen-method *(getTaxaData)*, 21
getTaxaLevelData, 22
getTaxaLevelData,CrcBiomeScreen-method *(getTaxaLevelData)*, 22

KeepTaxonomicLevel, 23

LoadTaxaTable, 25

ModelingRF, 26
ModelingRF_noweights, 28
ModelingXGBoost, 29
ModelingXGBoost_noweights, 30

NHSBCSP_screeningData, 32
NormalizeData, 32

PredictCrcBiomeScreen, 33

qcByCmdscale, 35

RunScreening, 37

setNormalizedData-setter, 40
setNormalizedData<- *(setNormalizedData-setter)*, 40
setNormalizedData<-,CrcBiomeScreen-method *(setNormalizedData-setter)*, 40
setTaxaData-setter, 41
setTaxaData<- *(setTaxaData-setter)*, 41
setTaxaData<-,CrcBiomeScreen-method *(setTaxaData-setter)*, 41
SplitDataSet, 42
SplitTaxas, 43

Thomas_2018_RelativeAbundance, 45
TrainModels, 45

ValidateModelOnData, 47

ZellerG_2014_RelativeAbundance, 50