

Package: strainspy (via r-universe)

October 14, 2026

Type Package

Title Strain Level Microbiome Wide Association Studies

Version 0.99.3

Depends R (>= 4.1.0)

Description Fits statistical models to detect microbial strains associated with a phenotype. For small sample sizes, models can be fitted using empirical Bayes maximum a posteriori (MAP) priors to improve detection power. Supports various experimental design scenarios, including repeated measures.

License MIT + file LICENSE

Encoding UTF-8

URL <https://github.com/gtonkinhill/strainspy>,
<https://sudaraka88.r-universe.dev/strainspy>

BugReports <https://github.com/gtonkinhill/strainspy/issues>

biocViews Software, Microbiome, Metagenomics, MicrobialStrain, DifferentialExpression, Regression, ExperimentalDesign, Sequencing

Imports ape, BiocParallel, data.table, dplyr, ggforce, ggplot2, ggrepel, ggthemes, ggtree, glmmTMB, graphics, grDevices, harmonicmeanp, lmerTest, magrittr, Matrix, methods, patchwork, purrr, R.utils (>= 2.13.0), S4Vectors, stats, stringr, SummarizedExperiment, rlang, tibble, tidyr, tools, utils, viridis

Suggests betareg, BiocStyle, compositions, fastglm, future, gamlss, gamlss.dist, knitr, mgcv, rmarkdown, testthat (>= 3.0.0), vegan, emmeans

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev make libicu-dev libpng-dev libuv1-dev zlib1g-dev

Repository <https://biocstaging.r-universe.dev>

Date/Publication 2026-10-14 00:47:49 UTC

RemoteUrl <https://github.com/BiocStaging/strainspy>

RemoteRef HEAD

RemoteSha a961691294dad5448b7fd0087a47f5f0c55bd21d

Contents

abundanceFit	3
ani_distance	5
caseControlFit	6
comp_ani_diff_and_posthoc_test	7
compute_eb_priors	9
define_priors	10
estimate_beta_diversity	11
estimate_sample_richness	12
extract_fixef_priors	14
extract_strainspy	15
filter_by_presence	16
getContigNames	17
getGenomes	17
getPriorsDf	18
getResiduals	19
getSamples	19
getZICoefficients	20
glmObFit	21
glmQBFit	23
glmZiBFit	24
hadjust	26
merge_metaphlan_files	27
merge_sourmash_files	28
merge_sylph_files	29
modify_metadata	30
plot_ani_dist	31
plot_histogram	32
plot_manhattan	33
plot_mds	35
plot_pca_biplot	36
plot_prior_bootstrap	37
plot_volcano	38
prep_for_prediction	39
read_metaphlan	40
read_sourmash	41
read_sylph	43

<i>abundanceFit</i>	3
---------------------	---

read_taxonomy	44
show,straintspy_fit-method	45
show,straintspy_priors-method	45
straintspy	46
straintspy_fit-accessors	47
straintspy_fit-class	50
straintspy_priors-class	50
summary,straintspy_fit-method	51
top_hits	52

Index	53
--------------	-----------

abundanceFit	<i>abundanceFit</i>
--------------	---------------------

Description

This function fits a linear regression model using `stats::lm()` or `lmerTest::lmer()`. The latter is used for models with random effects. It takes a `SummarizedExperiment` object as input, along with a user-defined formula, and fits a linear model on the assay data.

Usage

```
abundanceFit(  
  se,  
  design,  
  nthreads = 1,  
  scale_continuous = TRUE,  
  transform = NULL,  
  BPPARAM = NULL,  
  progress = TRUE  
)
```

Arguments

se	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
design	Formula. A formula to specify the fixed and random effects, e.g., <code>~ Group + (1 Sample)</code> .
nthreads	An integer specifying the number of (CPUs or workers) to use. Defaults to 1.
scale_continuous	Logical. If TRUE, all numeric columns in <code>colData(se)</code> are z-score standardized (mean = 0, SD = 1). Defaults to TRUE.
transform	If data is already transformed, set to NULL (default). Supported options: <code>arcsin</code> transform <code>arcsin</code> or centered log ratio CLR. CLR requires <code>compositions</code> package.

BPPARAM	Optional BiocParallelParam object. If not provided, the function will configure an appropriate backend automatically.
progress	Logical. If TRUE (default), progress bars and progress messages are printed. Set to FALSE to silence them.

Details

Generally, these linear models are useful to model compositional abundance data (e.g. Taxonomic_abundance in Sylph Profile) with appropriate transforms.

Value

A `strainspy_fit` object with the following components:

<code>row_data</code>	The <code>rowData</code> of <code>se</code> for the fitted features
<code>coefficients</code>	A <code>DFrame</code> with coefficients for each feature
<code>std_errors</code>	A <code>DFrame</code> of standard errors for each feature
<code>p_values</code>	A <code>DFrame</code> of p-values for each feature
<code>residuals</code>	A <code>DFrame</code> of residual vectors for each feature
<code>convergence</code>	A named logical vector indicating convergence for each feature
<code>design</code>	Formula used in the call to <code>abundanceFit</code>
<code>call</code>	Call to <code>abundanceFit</code>

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy")

se <- read_sylph(example_path, example_meta, 'Taxonomic_abundance')
se <- filter_by_presence(se, 20)

design <- as.formula("~ Case_status + Age_at_collection")
fit <- abundanceFit(se[1:10,], design, nthreads=2, transform='CLR')
top_hits(fit, alpha = 0.5)
```

ani_distance	<i>ani_distance</i>
--------------	---------------------

Description

This function calculates the distance between samples in a SummarizedExperiment object representing strain level ANI data.

Usage

```
ani_distance(
  se,
  taxonomy = NULL,
  tax_level = "Species",
  collapse_method = "representative"
)
```

Arguments

se	A SummarizedExperiment object.
taxonomy	An optional data.frame containing the taxonomy of the genomes (rows) in se. The result of running read_taxonomy on the taxonomy file associated with the ANI data. If provided, the ANI values will be collapsed at the specified taxonomic level before calculating the distance matrix. (default=NULL)
tax_level	A character string specifying the taxonomic level at which to collapse the ANI values. (default="Species")
collapse_method	A character string specifying the method to use to collapse the ANI values. Must be one of 'representative', 'mean' or 'max'. (default='representative')

Value

A dist object of distances between samples.

Examples

```
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy")
tax_path <- system.file("extdata", "example_taxonomy.tsv.gz",
  package = "strainspy")
se <- read_sylph(example_path)
tax <- read_taxonomy(tax_path)
distance_matrix <- ani_distance(se, tax)
```

caseControlFit	<i>caseControlFit</i>
----------------	-----------------------

Description

This function fits a logistic regression model similar to that published in the original Sylph paper. It takes a SummarizedExperiment object as input, along with a user-defined formula.

Usage

```
caseControlFit(
  se,
  design,
  min_identity = 0.98,
  nthreads = 1,
  scale_continuous = TRUE,
  BPPARAM = NULL,
  progress = TRUE
)
```

Arguments

<code>se</code>	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
<code>design</code>	Formula. A formula to specify the fixed and random effects, e.g., Outcome ~ Value + Covariate + (1 Sample).
<code>min_identity</code>	A numeric value specifying the minimum identity threshold to consider (default=0.98).
<code>nthreads</code>	An integer specifying the number of (CPUs or workers) to use. Defaults to 1.
<code>scale_continuous</code>	Logical. If TRUE, all numeric columns in <code>colData(se)</code> are z-score standardized (mean = 0, SD = 1). Defaults to TRUE.
<code>BPPARAM</code>	Optional BiocParallelParam object. If not provided, the function will configure an appropriate backend automatically.
<code>progress</code>	Logical. If TRUE (default), progress bars and progress messages are printed. Set to FALSE to silence them.

Value

A `strainspy_fit` object with the following components:

<code>row_data</code>	The <code>rowData</code> of <code>se</code> for the fitted features
<code>coefficients</code>	A <code>DFrame</code> with coefficients for each feature
<code>std_errors</code>	A <code>DFrame</code> of standard errors for each feature
<code>p_values</code>	A <code>DFrame</code> of p-values for each feature

residuals	NULL; residuals are not stored by this fitter
convergence	A named logical vector indicating convergence for each feature
design	Formula used in the call to caseControlFit
call	Call to caseControlFit

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_meta$Case_status <- factor(example_meta$Case_status)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path, example_meta)
se <- filter_by_presence(se)

design <- as.formula("Case_status ~ Value + Age_at_collection")

fit <- caseControlFit(se[1:10,], design, nthreads=2)
```

comp_ani_diff_and_posthoc_test

Compute the mean ANI differences for contigs with an identity effect and perform posthoc testing to establish support for each hit.

Description

In the zero-inflated beta model, beta effects represent changes in strain identity and are modeled on the logit scale. While this scale is effective for distinguishing phenotype groups statistically, the resulting effect sizes are not directly interpretable as differences in mean ANI. The same applies to effect sizes from the ordinal beta model.

Usage

```
comp_ani_diff_and_posthoc_test(
  se,
  fit,
  th = NULL,
  simplified_formula = NULL,
  beta_min_nz = 0.1,
  beta_min_ani_diff = 0.015,
  nthreads = 1L,
  reorder_hits = FALSE,
  BPPARAM = NULL,
  progress = TRUE
)
```

Arguments

<code>se</code>	SummarizedExperiment. A SummarizedExperiment object containing assay data and metadata
<code>fit</code>	A fitted <code>strainspy_fitmodel</code>
<code>th</code>	The table returned by <code>top_hits()</code> (required).
<code>simplified_formula</code>	A formula to use instead of <code>getDesign(fit)</code> , useful when the original design was too complex and causes <code>emmeans</code> to fail during ANI estimation (default <code>NULL</code>).
<code>beta_min_nz</code>	Minimum proportion of non-zero values per phenotype group for a beta hit to be considered well supported. Hits where that proportion falls below this threshold in at least one group are flagged too few nonzero. Default 0.1. Hits supported by 3 or fewer observations are always flagged <3 nonzero.
<code>beta_min_ani_diff</code>	Minimum absolute mean ANI difference required for a beta hit to be considered meaningful. Effect sizes smaller than this are flagged: small effect. Default = 1.5e-2, i.e. 1.5 percentage points of ANI.
<code>nthreads</code>	Number of threads for parallel processing. Default 1.
<code>reorder_hits</code>	Logical. Return <code>th</code> after reordering hits to prioritise the best-supported ones. Default <code>FALSE</code> .
<code>BPPARAM</code>	Optional <code>BiocParallelParam</code> object.
<code>progress</code>	Logical. If <code>TRUE</code> (default), progress bars and progress messages are printed. Set to <code>FALSE</code> to silence them.

Details

This function converts logit-scale beta effects into response-scale mean ANI differences using marginal means and pairwise contrasts (requires `emmeans`). It therefore provides an interpretable estimate of the magnitude of strain-level differences between phenotype groups, along with post hoc quality checks for poorly supported beta signals.

Posthoc testing uses two approaches. First, it counts the number of non-zero observations supporting each beta effect. Second, it compares the estimated mean ANI difference against a preset threshold. A comment is added to each hit depending on the outcome.

By default, this function uses the formula from `getDesign(fit)`. However, if the formula contains multiple covariates, this can cause `emmeans` to behave unexpectedly. For example, if `getDesign(fit)` is `~ disease + age + sex + country + (1|study)`, this can cause this function to throw errors or warnings. Using a `simplified_formula = ~ disease` to override can be a useful approach.

This function is designed to directly work on the tibble returned from `top_hits()`.

Value

A data.frame similar to `th`, with added columns giving details on: Component (Beta, ZI or both), Mean ANI_Difference, Contrast, Non-Zero Counts of contrast components, and a general Comment on each hit. The comment will be `NA` if the hit passes all posthoc tests.

Examples

```
if (requireNamespace("emmeans", quietly = TRUE)) {
  meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
    package = "strainspy"))
  meta$Case_status <- factor(meta$Case_status)
  se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
    package = "strainspy"), meta_data = meta)
  se <- filter_by_presence(se, min_nonzero = 30)[1:10, ]
  fit <- glmZiBFit(se, ~ Case_status + Age_at_collection, nthreads = 1)
  th <- top_hits(fit, coef = 2, alpha = 1)
  posthoc <- comp_ani_diff_and_posthoc_test(se, fit, th, nthreads = 1)
  head(posthoc$ANI_Difference)
}
```

compute_eb_priors

Constructor for strainspy_priors using empirical Bayes estimation

Description

Fits fast binomial and beta models to estimate MAP prior SDs for use in zero-inflated and ordinal beta models fitted using glmmTMB.

Usage

```
compute_eb_priors(
  se,
  design,
  nthreads = 1L,
  BPPARAM = NULL,
  low_cutoff = 0.1,
  high_cutoff = 10,
  est_dispersion_prior = FALSE,
  progress = TRUE
)
```

Arguments

se	A SummarizedExperiment object with the assay matrix and colData.
design	A fixed-effects-only formula (e.g., ~ group + age + sex).
nthreads	An integer specifying the number of (CPUs or workers) to use. Defaults to 1.
BPPARAM	Optional BiocParallelParam object. If not provided, the function will configure an appropriate backend automatically.
low_cutoff	Floor for small SD estimates; values below it are raised to it. Default 0.1
high_cutoff	Ceiling for large SD estimates; values above it are capped at it. Default 10

est_dispersion_prior	Estimate a prior for dispersion (default = FALSE)
progress	Logical. If TRUE (default), progress bars and progress messages are printed. Set to FALSE to silence them.

Value

An object of class `strainspy_priors`.

Examples

```
if (requireNamespace("fastglm", quietly = TRUE) &&
    requireNamespace("betareg", quietly = TRUE)) {
  meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
    package = "strainspy"))
  meta$Case_status <- factor(meta$Case_status)
  se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
    package = "strainspy"), meta_data = meta)
  se <- filter_by_presence(se, min_nonzero = 30)
  pri <- compute_eb_priors(se[1:10, ],
    ~ Case_status + Age_at_collection, nthreads = 1)
  class(pri)
}
```

<code>define_priors</code>	<i>Constructor for strainspy_priors using preset or user defined priors</i>
----------------------------	---

Description

Constructor for `strainspy_priors` using preset or user defined priors

Usage

```
define_priors(
  se,
  design,
  method = "preset_weak",
  add_dispersion_prior = FALSE,
  priors_df = NULL
)
```

Arguments

<code>se</code>	A SummarizedExperiment object with the assay matrix and colData.
<code>design</code>	A fixed-effects-only formula (e.g., <code>~ group + age + sex</code>).
<code>method</code>	Prior type (<code>"preset_weak"</code> , <code>"preset_strong"</code> , <code>"manual"</code>). To compute empirical priors, use <code>compute_eb_priors()</code> . Default <code>preset_weak</code> .

`add_dispersion_prior` Add a prior of normal(0,1) to the dispersion parameter. Default FALSE

`priors_df` NULL if method == 'preset_weak' | 'preset_strong', else a data.frame of priors suitable for glmmTMB model fitting. See ?glmmTMB::priors for more details. Default NULL.

Value

An object of class `strainspy_priors`.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy"), meta_data = meta)
pri <- suppressWarnings(define_priors(se[1:10, ],
  ~ Case_status + Age_at_collection, method = "preset_weak"))
class(pri)
```

`estimate_beta_diversity`

Estimate beta diversity from ANI

Description

Computes between-sample distances from the ANI matrix and, unless `distance_only = TRUE`, runs NMDS, PERMANOVA and a beta-dispersion test grouped by phenotype. The `vegan` package is required for method = "jaccard" and for everything beyond the distance matrix.

Usage

```
estimate_beta_diversity(
  se,
  method = "manhattan",
  distance_only = FALSE,
  ani_threshold = 95,
  phenotype = NULL,
  show_plots = FALSE,
  return_plots = FALSE
)
```

Arguments

se	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
method	Distance method: "manhattan" (default) or "jaccard"
distance_only	Logical. If TRUE, stop after computing distance matrix
ani_threshold	Numeric. Threshold for presence/absence determination if method == jaccard (default 95)
phenotype	Any factor variable in colnames(colData(se)). Required unless distance_only = TRUE
show_plots	Logical. If TRUE, shows the plots (default = FALSE)
return_plots	Logical. If TRUE, returns ggplot2 objects (default = FALSE)

Value

If distance_only = TRUE, returns distance object. Otherwise a list with:

- distance — distance matrix
- nmds — NMDS object
- ordination — NMDS coordinates with group
- permanova — PERMANOVA results
- betadisper — betadisper object
- plots — list of ggplot2 objects (if return_plots = TRUE)

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
se <- filter_by_presence(se, min_nonzero = 30)
d <- estimate_beta_diversity(se[1:15, ], distance_only = TRUE)
```

estimate_sample_richness

Estimate species richness and coverage of reference strains using ANI

Description

This function estimates sample-level **species richness** and **strain coverage** using ANI-based presence/absence. A strain is considered present if ANI $\geq$ ani_threshold.

Usage

```
estimate_sample_richness(
  se,
  taxonomy,
  ani_threshold = 95,
  report_strain_richness = FALSE
)
```

Arguments

se SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.

taxonomy A taxonomy table read using `strainspy::read_taxonomy()` (required).

ani_threshold Numeric ANI threshold used to define presence (default: 95).

report_strain_richness Logical. Also return the strain coverage count. See description (default: FALSE)

Details

Species richness is defined as the number of species with at least one strain exceeding the ANI threshold. This requires taxonomy data, which can be read in using `read_taxonomy()`.

Experimental Strain coverage is a database-dependent, normalised score that increases with the number of reference strains detected within a species. When only one strain is present for a species, it contributes 1 towards strain coverage. As more strains are detected, the contribution increases toward 2, reaching 2 when all reference strains are detected. Although this is designed to prevent inflation for species with many reference strains, this measures the **coverage of reference strain space in the database**, not true biological strain richness. Therefore, this value is not returned by default.

The strain coverage metric is computed as:

$$1 + (k - 1)/N$$

where k is the number of reference strains detected for a species and N is the total number of reference strains for that species in the database.

This approach does **not** resolve true strain multiplicity and may overestimate coverage when a single biological strain matches many highly similar reference genomes. Alternative approaches are being investigated.

Value

A named list with:

species_richness Integer vector giving the number of species detected per sample.

strain_coverage NA or Numeric vector giving the database-normalised strain coverage per sample.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy"))
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy"), meta_data = meta)
tax <- read_taxonomy(system.file("extdata", "example_taxonomy.tsv.gz",
  package = "strainspy"))
rich <- estimate_sample_richness(se[1:20, ], tax)
```

extract_fixef_priors *Access fixef priors needed for ordinal beta regression*

Description

Access fixef priors needed for ordinal beta regression

Usage

```
extract_fixef_priors(object)

## S4 method for signature 'strainspy_priors'
extract_fixef_priors(object)
```

Arguments

object A strainspy_priors object.

Value

A data frame of prior parameters for fixed effects.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy"), meta_data = meta)
pri <- suppressWarnings(define_priors(se[1:10, ],
  ~ Case_status + Age_at_collection, method = "preset_weak"))
head(extract_fixef_priors(pri))
```

extract_strainspy	<i>Extract data from a StrainSpy SummarizedExperiment object in long format</i>
-------------------	---

Description

Extract data from a StrainSpy SummarizedExperiment object in long format

Usage

```
extract_strainspy(
  se,
  variables = NULL,
  contigs = NULL,
  taxonomy = NULL,
  drop_zeros = FALSE
)
```

Arguments

se	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
variables	One or multiple variables in colnames(colData(se)), except Sample_file.
contigs	A character vector of contigs (rownames of se) to extract.
taxonomy	An optional taxonomy file read using strainspy::read_taxonomy (default=NULL).
drop_zeros	Bool. If TRUE, rows with an ANI or abundance of 0 are dropped. Default FALSE.

Value

A long-format data.frame with sample metadata, contig identity columns, ANI values, row-level annotations, and optional taxonomy columns.

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy")
example_meta <- read.csv(example_meta_path)
# Change variable from character to factor
example_meta$Case_status = factor(example_meta$Case_status)

example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy")
se <- read_sylph(example_path, example_meta)
```

```
tax_path <- system.file("extdata", "example_taxonomy.tsv.gz",
package = "strainspy")
tax <- read_taxonomy(tax_path)

extract_strainspy(se, variables = c("Case_status", "Age_at_collection"),
contigs = rownames(se)[1:5], taxonomy=tax)
```

filter_by_presence	<i>filter_by_presence</i>
--------------------	---------------------------

Description

This function filters a SummarizedExperiment object to keep only rows (features/strains) that are present in a specified number of samples. The same rows are filtered from all assays in the SummarizedExperiment.

Usage

```
filter_by_presence(se, min_nonzero = 10)
```

Arguments

se	A SummarizedExperiment object.
min_nonzero	Integer. The minimum number of non-zero entries required to retain a row. Default is 10.

Value

A filtered SummarizedExperiment object with only the rows that meet the criteria.

Examples

```
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path)

# Filter to keep only rows with at least 20 non-zero entries
filtered_se <- filter_by_presence(se, min_nonzero = 20)
```

getContigNames	<i>Get Contig names in a fit object</i>
----------------	---

Description

Gets the ordered list of fitted contig names in a strainspy_fit

Usage

```
getContigNames(object)

## S4 method for signature 'strainspy_fit'
getContigNames(object)
```

Arguments

object An object of class strainspy_fit.

Value

A vector of contig names

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
fit <- glmZiBFit(filter_by_presence(se, min_nonzero = 30)[1:10, ],
~ Case_status + Age_at_collection, nthreads = 1, return_vcov = FALSE)
head(getContigNames(fit))
```

getGenomes	<i>Get Genomes in a fit object</i>
------------	------------------------------------

Description

Gets the ordered list of fitted genomes in a strainspy_fit

Usage

```
getGenomes(object)

## S4 method for signature 'strainspy_fit'
getGenomes(object)
```

Arguments

object An object of class `strainspy_fit`.

Value

A vector of fitted genomes

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
fit <- glmZiBfit(filter_by_presence(se, min_nonzero = 30)[1:10, ],
~ Case_status + Age_at_collection, nthreads = 1, return_vcov = FALSE)
head(getGenomes(fit))
```

getPriorsDf

Access the prior specification table

Description

Returns the full prior data.frame stored in a `strainspy_priors` object, including both fixed-effect and zero-inflation rows. See `extract_fixef_priors()` for the fixed-effect rows only.

Usage

```
getPriorsDf(object)

## S4 method for signature 'strainspy_priors'
getPriorsDf(object)
```

Arguments

object An object of class `strainspy_priors`.

Value

A data.frame with columns `prior`, `class` and `coef`.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
pr <- define_priors(se, ~ Case_status + Age_at_collection, method = "preset_weak")
getPriorsDf(pr)
```

getResiduals	<i>Access Residuals</i>
--------------	-------------------------

Description

Returns the residuals from a `strainspy_fit` object.

Usage

```
getResiduals(object)

## S4 method for signature 'strainspy_fit'
getResiduals(object)
```

Arguments

`object` An object of class `strainspy_fit`.

Value

A `DataFrame` containing residuals.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
fit <- glmZiBFit(filter_by_presence(se, min_nonzero = 30)[1:10, ],
~ Case_status + Age_at_collection, nthreads = 1, return_vcov = FALSE)
class(getResiduals(fit))
```

getSamples	<i>Get Sample names in a fit object</i>
------------	---

Description

Gets the ordered list of samples used in `strainspy_fit`

Usage

```
getSamples(object)

## S4 method for signature 'strainspy_fit'
getSamples(object)
```

Arguments

object An object of class `strainspy_fit`.

Value

A vector of fitted sample names

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
fit <- glmZiBFit(filter_by_presence(se, min_nonzero = 30)[1:10, ],
~ Case_status + Age_at_collection, nthreads = 1, return_vcov = FALSE)
head(getSamples(fit))
```

getZICoefficients	<i>Access Zero-Inflation Coefficients</i>
-------------------	---

Description

Returns the zero-inflation coefficients from a `strainspy_fit` object.

Usage

```
getZICoefficients(object)

## S4 method for signature 'strainspy_fit'
getZICoefficients(object)
```

Arguments

object An object of class `strainspy_fit`.

Value

A `DataFrame` containing zero-inflation coefficients.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
fit <- glmZiBFit(filter_by_presence(se, min_nonzero = 30)[1:10, ],
~ Case_status + Age_at_collection, nthreads = 1, return_vcov = FALSE)
class(getZICoefficients(fit))
```

glmObFit

*glmObFit***Description**

This function fits an ordinal beta regression model using the `glmmTMB` package. It takes a `SummarizedExperiment` object as input, along with a user-defined formula, and fits an ordinal beta regression model on the assay data.

Usage

```
glmObFit(
  se,
  design,
  nthreads = 1L,
  scale_continuous = TRUE,
  MAP_prior = "preset_weak",
  family = glmmTMB::ordbeta(),
  BPPARAM = NULL,
  progress = TRUE
)
```

Arguments

<code>se</code>	<code>SummarizedExperiment</code> . A <code>SummarizedExperiment</code> object containing the assay data and metadata.
<code>design</code>	Formula. A formula to specify the fixed and random effects, e.g., <code>~ Group + (1 Sample)</code> .
<code>nthreads</code>	An integer specifying the number of (CPUs or workers) to use. Defaults to 1.
<code>scale_continuous</code>	Logical. If <code>TRUE</code> , all numeric columns in <code>colData(se)</code> are z-score standardized (mean = 0, SD = 1). Defaults to <code>TRUE</code> .
<code>MAP_prior</code>	One of <code>strainspy_priors</code> object, character string (<code>"preset_weak"</code> , <code>"preset_strong"</code>), a <code>data.frame</code> of priors, or <code>NULL</code> . Default <code>"preset_weak"</code> . See Details.
<code>family</code>	A <code>glmmTMB</code> family object. Defaults to <code>glmmTMB::ordbeta()</code> . For general use, leave this at the default. Advanced users can experiment with other families in <code>?glmmTMB::family_glmmTMB</code>
<code>BPPARAM</code>	Optional <code>BiocParallelParam</code> object. If not provided, the function will configure an appropriate backend automatically.
<code>progress</code>	Logical. If <code>TRUE</code> (default), progress bars and progress messages are printed. Set to <code>FALSE</code> to silence them.

Details

For robust model fitting, it is recommended to use maximum a posteriori (MAP) estimation. Empirical Bayes MAP priors can be computed using `strainspy::compute_eb_priors()`. They can also be manually defined using `strainspy::define_priors()`. The resulting `strainspy_priors` object can be passed to the `MAP_prior` argument.

Alternatively, a character string ("preset_weak" or "preset_strong") or a valid prior data.frame (see `?glmmTMB::priors`) can be provided instead. This will result in a call to `strainspy::define_priors()` to generate priors. To disable MAP estimation and fit by maximum likelihood without priors, set `MAP_prior = NULL`.

Variance-covariance matrices are not returned by this fitter: the `vcov` slot of the result is `NULL`, so the fit cannot be used with `comp_ani_diff_and_posthoc_test()`.

Value

A `strainspy_fit` object with the following components:

<code>row_data</code>	The <code>rowData</code> of <code>se</code> for the fitted features
<code>coefficients</code>	A <code>DFrame</code> with coefficients for each feature
<code>std_errors</code>	A <code>DFrame</code> of standard errors for each feature
<code>p_values</code>	A <code>DFrame</code> of p-values for each feature
<code>residuals</code>	A <code>DFrame</code> of residual vectors for each feature
<code>convergence</code>	A named logical vector indicating convergence for each feature
<code>design</code>	Formula used in the call to <code>glmObFit</code>
<code>call</code>	Call to <code>glmObFit</code>

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path, example_meta)
se <- filter_by_presence(se)

design <- as.formula("~ Case_status + Age_at_collection")

fit <- glmObFit(se[1:10,], design, family=glmmTMB::ordbeta())
```

glmQBFit

glmQBFit

Description

This function fits a quasibinomial regression model using `stats::glm()`.

Usage

```
glmQBFit(
  se,
  design,
  nthreads = 1L,
  scale_continuous = TRUE,
  BPPARAM = NULL,
  progress = TRUE
)
```

Arguments

<code>se</code>	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
<code>design</code>	Formula. A formula to specify fixed effects, e.g., <code>~ Age + BMI</code> .
<code>nthreads</code>	An integer specifying the number of (CPUs or workers) to use. Defaults to 1.
<code>scale_continuous</code>	Logical. If TRUE, all numeric columns in <code>colData(se)</code> are z-score standardized (mean = 0, SD = 1). Default TRUE.
<code>BPPARAM</code>	Optional BiocParallelParam object. If not provided, the function will configure an appropriate backend automatically.
<code>progress</code>	Logical. If TRUE (default), progress bars and progress messages are printed. Set to FALSE to silence them.

Details

It takes a SummarizedExperiment object as input, along with a user-defined formula containing only fixed effect parameters, and fits a quasibinomial regression model on the assay data.

Variance-covariance matrices are not returned by this fitter: the `vcov` slot of the result is NULL, so the fit cannot be used with `comp_ani_diff_and_posthoc_test()`.

Value

A `strainspy_fit` object with the following components:

<code>row_data</code>	The <code>rowData</code> of <code>se</code> for the fitted features
<code>coefficients</code>	A <code>DFrame</code> with coefficients for each feature

<code>std_errors</code>	A DFrame of standard errors for each feature
<code>p_values</code>	A DFrame of p-values for each feature
<code>residuals</code>	A DFrame of residual vectors for each feature
<code>convergence</code>	A named logical vector indicating convergence for each feature
<code>design</code>	Formula used in the call to <code>glmQBFit</code>
<code>call</code>	Call to <code>glmQBFit</code>

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path, example_meta)
se <- filter_by_presence(se)

design <- as.formula(" ~ Case_status")

fit <- glmQBFit(se[1:10,], design, nthreads=2)
```

glmZiBFit

glmZiBFit

Description

This function fits a zero-inflated beta regression model using either `glmmTMB` or `gamlss` packages. It takes a `SummarizedExperiment` object as input, along with a user-defined formula, and fits a zero-inflated beta regression model on the assay data.

Usage

```
glmZiBFit(
  se,
  design,
  nthreads = 1,
  scale_continuous = TRUE,
  MAP_prior = "preset_weak",
  BPPARAM = NULL,
  method = "glmmTMB",
  return_vcov = TRUE,
  progress = TRUE
)
```

Arguments

<code>se</code>	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
<code>design</code>	Formula. A formula to specify the fixed and random effects, e.g., $\sim \text{Group} + (1 \text{Sample})$.
<code>nthreads</code>	An integer specifying the number of (CPUs or workers) to use. Defaults to 1.
<code>scale_continuous</code>	Logical. If TRUE, all numeric columns in <code>colData(se)</code> are z-score standardized (mean = 0, SD = 1). Defaults to TRUE.
<code>MAP_prior</code>	One of <code>strainspy_priors</code> object, character string ("preset_weak", "preset_strong"), a data.frame of priors, or NULL. Default "preset_weak". See <code>?define_priors</code> .
<code>BPPARAM</code>	Optional BiocParallelParam object. If not provided, the function will configure an appropriate backend automatically.
<code>method</code>	Character. The method to use for fitting the model. Either 'glmmTMB' (default) or 'gamlss'.
<code>return_vcov</code>	Logical. Return variance-covariance matrices for each model, required for <code>comp_ani_diff_and_posthoc</code> . Disabling will result in a much smaller <code>strainspy_fit</code> object that is useful for most analyses. Defaults to TRUE.
<code>progress</code>	Logical. If TRUE (default), progress bars and progress messages are printed. Set to FALSE to silence them.

Value

A `strainspy_fit` object with the following components:

<code>row_data</code>	The <code>rowData</code> of <code>se</code> for the fitted features
<code>coefficients</code>	A DFrame with coefficients for each feature
<code>std_errors</code>	A DFrame of standard errors for each feature
<code>p_values</code>	A DFrame of p-values for each feature
<code>zi_coefficients</code>	A DFrame of zero inflated coefficients for each feature
<code>zi_std_errors</code>	A DFrame of zero inflated standard errors for each feature
<code>zi_p_values</code>	A DFrame of zero inflated p-values for each feature
<code>residuals</code>	A DFrame of residual vectors for each feature
<code>convergence</code>	A named logical vector indicating convergence for each feature
<code>design</code>	Formula used in the call to <code>glmZiBFit</code>
<code>call</code>	Call to <code>glmZiBFit</code>

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
```

```

example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path, example_meta)
se <- filter_by_presence(se)

design <- as.formula("~ Case_status")

fit <- glmZiBFit(se[1:10,], design, nthreads=2)

```

hadjust

Performs hierarchical taxonomy aware adjustment of p-values.

Description

This function performs multiple testing adjustment on the results of a `strainspy_fit` object at each taxonomic level, using either the Harmonic Mean P-value (HMP) or Bonferroni correction.

Usage

```
hadjust(object, coef = 2, method = "HMP", taxonomy, index_range = FALSE)
```

Arguments

<code>object</code>	A <code>strainspy_fit</code> object.
<code>coef</code>	The coefficient to use for p-value adjustment. Defaults to 2.
<code>method</code>	The method for p-value adjustment: "HMP" (default) or "bonferroni".
<code>taxonomy</code>	Required. A taxonomy table from <code>read_taxonomy()</code> , or the path to the taxonomy file.
<code>index_range</code>	Logical. Also return <code>index_min</code> and <code>index_max</code> columns, used to draw Manhattan plot segments for higher taxonomic levels. Default FALSE.

Value

A tibble with one row per taxon and level, with columns `Level`, `Model`, `Name`, the number of strains (`strain_count` for HMP, `sequence_count` for Bonferroni), `mean_coefficient` and the adjusted p-value `p_adjust`.

Examples

```

meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
se <- filter_by_presence(se, min_nonzero = 30)
tax <- read_taxonomy(system.file("extdata", "example_taxonomy.tsv.gz",

```

```

package = "strainspy"))
fit <- glmZiBFit(se[1:10, ], ~ Case_status + Age_at_collection,
nthreads = 1, return_vcov = FALSE)
out <- hadjust(fit, coef = 2, taxonomy = tax)
head(out)

```

merge_metaphlan_files *merge_metaphlan_files*

Description

Merge MetaPhlAn ≥ 4.0 profiles into a single file to use with read_metaphlan() and generate the taxonomy file required for downstream analysis.

Usage

```

merge_metaphlan_files(
  metaphlan_files,
  output_folder,
  output_filename = "metaphlan",
  sgb_database = TRUE
)

```

Arguments

metaphlan_files	A character vector of file paths to merge. Optionally, if the input is named, names(metaphlan_files) will be used as sample identifiers in place of the filename.
output_folder	Folder path to save outputs.
output_filename	File name for the two outputs. By default, the two files will be saved as metaphlan_merged.tsv.gz and metaphlan_taxonomy.tsv.gz
sgb_database	bool. Specify whether the default sgb database was used for profiling. Set to FALSE if sgb_to_gtdb_profile.py was run on the profiles as a post processing step. Default TRUE

Value

Invisibly returns NULL. The function writes two files to output_folder: a merged profile table (*_merged.tsv.gz) and a taxonomy table (*_taxonomy.tsv.gz).

Examples

```

files <- character(0)
is.character(files)

```

merge_sourmash_files *merge_sourmash_files*

Description

Merge the outputs of `sourmash search` or `sourmash gather` into a single file. The output of this function is suitable to use in `read_sourmash()`

Usage

```
merge_sourmash_files(
  sourmash_files,
  manifest_file = NULL,
  output,
  strip_unusable = TRUE
)
```

Arguments

sourmash_files A character vector of file paths to merge.

manifest_file If one or several indexed databases were used for searching or gathering, the required genome column can be missing. The path to the output of `sourmash sig` manifest can be provided to try and add back the data automatically (Default = NULL). If provided, columns `md5`, `name` and `filename` are required. This function uses `name` and `md5` to match the manifest data with `sourmash` outputs. In the final output, `filename` will be used to identify each hit. If the `filename` is a path, the `basename` without extensions will be used instead.

output Save path for the output file. Internally, `data.table::fwrite` is used, provide the extension `gz` to save a compressed file.

strip_unusable Remove columns containing data not useful for further analysis. Default TRUE.

Details

Refer to [sourmash documentation](#) for more information.

Value

Invisibly returns NULL. The merged `sourmash` table is written to `output`.

Examples

```
# Split the bundled sourmash output into two per-query files, then merge them.
hits <- data.table::fread(
  system.file("extdata", "example_sourmash.csv.gz", package = "strainspy")
)
queries <- head(unique(hits$query_name), 2)
parts <- vapply(queries, function(q) {
```

```

    f <- tempfile(fileext = ".csv")
    data.table::fwrite(hits[hits$query_name == q, ], f)
    f
  }, character(1))

merged <- tempfile(fileext = ".csv.gz")
merge_sourmash_files(parts, output = merged)

sm <- read_sourmash(merged)
dim(sm)

```

merge_sylph_files	<i>merge_sylph_files</i>
-------------------	--------------------------

Description

Merge the outputs of sylph query or sylph profile into a single file. The output of this function is suitable to use in read_sylph()

Usage

```
merge_sylph_files(sylph_files, output)
```

Arguments

sylph_files	A character vector of sylph output file paths to merge.
output	Save path for the output file. Internally, data.table::fwrite is used, provide the extension gz to save a compressed file.

Value

Invisibly returns output. The merged sylph table is written to that path.

Examples

```

# Split the bundled profile into two per-sample files, then merge them back.
profile <- data.table::fread(
  system.file("extdata", "example_sylph_profile.tsv.gz", package = "strainspy")
)
samples <- head(unique(profile$Sample_file), 2)
parts <- vapply(samples, function(s) {
  f <- tempfile(fileext = ".tsv")
  data.table::fwrite(profile[profile$Sample_file == s, ], f, sep = "\t")
  f
}, character(1))

merged <- tempfile(fileext = ".tsv.gz")
merge_sylph_files(parts, merged)

```

```
# The merged file carries one header and both samples
se <- read_sylph(merged)
dim(se)
```

modify_metadata	<i>Add or modify metadata in a SummarizedExperiment object</i>
-----------------	--

Description

This function adds or modifies metadata in a SummarizedExperiment object.

Usage

```
modify_metadata(se, meta_data, replace = FALSE, convert = TRUE)
```

Arguments

se	A SummarizedExperiment object.
meta_data	A data.frame with sample metadata. The first column must contain all samples returned by colnames(se).
replace	Logical. If TRUE, replaces existing colData. Default FALSE.
convert	Logical. If TRUE, attempts to auto-convert types as suited for model fitting (int to numeric, char to factor). Disable if meta_data is fully configured for model fitting. Default TRUE. See Details.

Details

If convert = TRUE, character columns with 5 or fewer unique (non-NA) values will be automatically converted to factors, with levels ordered alphabetically. For meaningful association testing, however, factor levels should reflect the intended reference level. It is recommended to manually convert such variables to factors with explicitly set levels *before* calling modify_metadata().

Value

Updated SummarizedExperiment object.

Examples

```
# Add or modify metadata subsequently
example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path)
se <- modify_metadata(se, example_meta)
```

plot_ani_dist	<i>Generate a violin or box plot for a subset of contigs for a categorical phenotype.</i>
---------------	---

Description

Generate a violin or box plot for a subset of contigs for a categorical phenotype.

Usage

```
plot_ani_dist(
  se,
  phenotype,
  contigs,
  facet_phenotype = NULL,
  contig_names = NULL,
  show_zero_plot = TRUE,
  show_points = TRUE,
  plot = TRUE,
  plot_type = "box"
)
```

Arguments

se	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
phenotype	Any factor variable in <code>colnames(colData(se))</code> .
contigs	A vector of contigs to generate distribution plots
facet_phenotype	Any factor variable in <code>colnames(colData(se))</code> , except phenotype. If provided, this will be used to facet the plot. See <code>?ggplot2::facet_wrap()</code> . Default NULL.
contig_names	Optional character vector to replace long contig names in the plot. Must match the length and order of contigs. If NULL (default), shortening will be attempted automatically.
show_zero_plot	Bool. If TRUE, distribution of the proportion of zeros will also be plotted. Default TRUE
show_points	Bool. If TRUE, points will be overlaid with jitter using <code>ggforce::geom_sina()</code> . Default TRUE
plot	If set to false, the function will return a tibble with data used to generate the plot.
plot_type	Specify whether box or violin plot is required. Default box.

Value

If `plot = TRUE`, returns a `ggplot` object (or a combined patchwork plot when `show_zero_plot = TRUE`). If `plot = FALSE`, returns a long-format table used to generate the ANI distribution plot.

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz", package = "strainspy")
example_meta <- read.csv(example_meta_path)
# Change variable from character to factor
example_meta$Case_status = factor(example_meta$Case_status)

example_path <- system.file("extdata", "example_sylph_profile.tsv.gz", package = "strainspy")
se <- read_sylph(example_path, example_meta)

plot_ani_dist(se, phenotype = "Case_status", contigs = rownames(se)[1:5])
```

plot_histogram	<i>Generate a histogram for a single contig for a categorical phenotype.</i>
----------------	--

Description

Generate a histogram for a single contig for a categorical phenotype.

Usage

```
plot_histogram(
  se,
  phenotype,
  contig,
  bins = 250,
  drop_zeros = FALSE,
  show_quantiles = TRUE,
  fit_spline = FALSE,
  fit_control = 100,
  separate_facets = TRUE
)
```

Arguments

se	SummarizedExperiment. A SummarizedExperiment object containing the assay data and metadata.
phenotype	One variable in colnames(colData(se)), except Sample_file. The selected variable must be categorical
contig	Contig name to visualise
bins	Numeric. Number of bins to plot the histogram. This parameter should be adjusted to properly visualise the ANI differences. Default 250
drop_zeros	Bool. If TRUE, ANI or Abundance 0 values will not be included in the histogram. Default FALSE

show_quantiles Bool. If TRUE, the minimum, first quartile, median, mean, third quartile and maximum are shown as vertical lines. Default TRUE

fit_spline Bool. If TRUE, a spline will be fitted to each histogram. Default FALSE

fit_control Numeric. Smaller values increase the smoothness of the fitted spline. Default 100

separate_facets Bool. If TRUE, each category of the phenotype will be plotted in a separate facet. Default TRUE

Value

A ggplot histogram object. A summary table of per-group statistics is attached as attr(plot, "summary_table").

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
# Change variable from character to factor
example_meta$Case_status = factor(example_meta$Case_status)

example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path, example_meta)

plot_histogram(se, phenotype = "Case_status", contig = rownames(se)[1])
```

plot_manhattan

Generate a strain Manhattan plot from a strainspy_fit object

Description

This function can generate multiple Manhattan plots from a strainspy_fit object.

Usage

```
plot_manhattan(
  object,
  coef = 2,
  taxonomy = NULL,
  aggregate_by_taxa = NULL,
  method = NULL,
  alpha = 0.05,
```

```

tax_levels = c("Phylum", "Genus", "Species", "Strain"),
plot = TRUE
)

```

Arguments

object	A strainspy_fit object.
coef	The number of the coefficient from which to generate the plot (default=2).
taxonomy	An optional taxonomy file read using strainspy::read_taxonomy (default=NULL).
aggregate_by_taxa	Logical. If TRUE, aggregate p-values across tax_levels and visualise as a stacked segment plot. Requires taxonomy. If NULL (default), the function automatically sets it to TRUE when taxonomy is provided.
method	Character. Multiple testing correction method. With taxonomy, it is passed to hadjust() and must be "HMP" or "bonferroni"; without, it is passed to top_hits() and must be one of "holm", "bonferroni", "BH" or "BY". Default NULL, meaning "HMP" with a taxonomy and "holm" without.
alpha	Numeric. Significance threshold for adjusted p-values. Defaults to 0.05.
tax_levels	Character vector. Subset of taxonomic levels: c("Domain", "Phylum", "Class", "Order", "Family", "Genus", "Species", "Strain") to include in the plot. Defaults to c("Phylum", "Genus", "Species", "Strain"). Provided levels will be reordered by the standard hierarchy and the highest level is used for colouring. If it has >30 categories, the legend is replaced by a named vector. Only applicable with aggregate_by_taxa = TRUE.
plot	If set to false, the function will return a tibble with data used to generate the plot. NOTE: Will be disregarded if taxonomy is provided with the option aggregate_by_taxa = FALSE.

Details

The generated plot will be more informative if taxonomic information is provided. Two types of plots can be generated with taxonomic information.

1. Set aggregate_by_taxa = TRUE to aggregate p-values across taxonomic levels specified by tax_levels. This generates a stacked segment Manhattan plot. In this plot, -log10 p-values adjusted using method will be plotted in taxonomic ordering. Significance threshold will be alpha.
2. Set aggregate_by_taxa = FALSE to generate a traditional Manhattan plot for all strains coloured by phylum. Unadjusted p-values will be plotted and Bonferroni significance thresholds (0.05, 0.01) will be shown. Strains will be in taxonomic ordering and a tree showing the taxonomic relationship between strains will be plotted below the Manhattan plot.

If taxonomic information is not provided (taxonomy = NULL), a plot with -log10 p-values adjusted using method will be generated. Strains will be plotted in alphabetical order. Significance threshold will be alpha.

If Zero-Inflated Beta regression was used to model object, separate facets will be used for zero-inflated and beta components of the model.

Value

A ggplot object showing the Manhattan plot if `plot = TRUE`, else a tibble with the plot data.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy"), meta_data = meta)
se <- filter_by_presence(se, min_nonzero = 30)
tax <- read_taxonomy(system.file("extdata", "example_taxonomy.tsv.gz",
  package = "strainspy"))
fit <- glmZiBFit(se[1:10, ], ~ Case_status + Age_at_collection,
  nthreads = 1, return_vcov = FALSE)
mh_data <- plot_manhattan(fit, taxonomy = tax, plot = FALSE)
head(mh_data)
```

plot_mds

*Plot MDS of ANI data***Description**

Performs classical multidimensional scaling (MDS) on a `SummarizedExperiment` object and plots the result, optionally colored by a phenotype.

Usage

```
plot_mds(
  se,
  coef = 2,
  taxonomy = NULL,
  palette = c("#DAA520", "#3CB371", "#483D8B"),
  plot = TRUE
)
```

Arguments

<code>se</code>	A <code>SummarizedExperiment</code> object containing ANI data.
<code>coef</code>	Integer. Index of the column in <code>colData(se)</code> to use as the grouping or color variable (default is 2).
<code>taxonomy</code>	Optional taxonomy table to use in distance calculation.
<code>palette</code>	A vector of color hex codes for numeric phenotype gradients. Defaults to <code>viridis</code> palette.
<code>plot</code>	Logical. If <code>TRUE</code> , returns a ggplot object. If <code>FALSE</code> , returns the underlying MDS coordinates as a tibble.

Value

A ggplot object showing the MDS plot, or a tibble of MDS coordinates if plot = FALSE.

Examples

```
example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path, example_meta)
plot_mds(se, coef = 2)
```

plot_pca_biplot	<i>plot_pca_biplot</i>
-----------------	------------------------

Description

Perform PCA and Generate a Biplot

Usage

```
plot_pca_biplot(se, phenotype, plot = TRUE)
```

Arguments

se	A SummarizedExperiment object containing ANI data.
phenotype	A character string specifying the name of a phenotype or an integer specifying the column number in colData(se)
plot	A logical value indicating whether to return the PCA plot (TRUE) or PCA data (FALSE). Default is TRUE.

Details

This function performs Principal Component Analysis (PCA) on a SummarizedExperiment object and generates a PCA biplot colored by a specified phenotype.

Value

If plot = TRUE, returns a ggplot2 PCA plot. If plot = FALSE, returns a tibble with PCA results.

Examples

```
example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy")
se <- read_sylph(example_path, example_meta)
plot_pca_biplot(se, phenotype = 3)
```

plot_prior_bootstrap *Plot the bootstrap distributions behind a strainspy_priors object*

Description

Plot the bootstrap distributions behind a strainspy_priors object

Usage

```
plot_prior_bootstrap(object, prior)

## S4 method for signature 'strainspy_priors'
plot_prior_bootstrap(object, prior)
```

Arguments

object	An object of class strainspy_priors that has bootstrap information
prior	Name of the coefficient whose bootstrap SD to plot, or "dispersion".

Value

A patchwork of ggplot2 histograms.

Examples

```
methods::hasMethod("plot_prior_bootstrap", "strainspy_priors")
```

plot_volcano	<i>Generate a Volcano Plot from a strainspy_fit Object</i>
--------------	--

Description

This function creates a volcano plot from the results of a strainspy model fit, highlighting the relationship between effect size (coefficients) and statistical significance ($-\log_{10}$ p-values).

Usage

```
plot_volcano(
  object,
  coef = 2,
  alpha = 0.05,
  method = "holm",
  palette = rev(viridis::viridis(3)),
  label = FALSE,
  contig_names = NULL,
  plot = TRUE
)
```

Arguments

object	A strainspy_fit object containing model results.
coef	Integer, specifying the coefficient index of the fit to extract p values and effect size for plotting. Default is 2.
alpha	Float indicating significance level. Default is 0.05
method	The method for p-value adjustment. For available options, see ?top_hits(). Default holm.
palette	A vector of user defined palette to colour plot. Default is the viridis palette.
label	Logical, whether to label most significant points of the plot with Species name.
contig_names	Optional character vector of labels for the significant hits. Must match the length and order of top_hits(object, coef = coef, alpha = alpha)\$Contig_name. If NULL (default), contig names are shortened automatically.
plot	Logical, whether to return a ggplot object (default: TRUE). If FALSE, returns the processed data.

Value

A ggplot2 object if plot = TRUE, otherwise a dataframe containing the relevant statistics.

Examples

```
example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
se <- read_sylph(example_path, example_meta)
se <- filter_by_presence(se)
design <- as.formula(" ~ Case_status + Sex")
fit <- glmZiBFit(se[1:10,], design, nthreads=2)
plot_volcano(fit, coef = 2, label = TRUE)
```

prep_for_prediction	<i>Prepare a caret-compatible prediction data frame from a Summarized-Experiment</i>
---------------------	--

Description

Formats strain-level data and optional covariates into a `data.frame` object ready for modeling with **caret**. The function extracts selected contigs from the `sy` assay matrix, combines them with the specified outcome and optional covariates from the metadata, and performs basic validation and cleaning. Metadata is pulled from `SummarizedExperiment::colData(sy)`, and generally assumes `strainspy::modify_metadata()` was used upstream.

Usage

```
prep_for_prediction(
  sy,
  outcome,
  contigs,
  covariates = NULL,
  use_genome_names = TRUE
)
```

Arguments

<code>sy</code>	A <code>SummarizedExperiment</code> object containing the assay data and metadata.
<code>outcome</code>	Character string giving the name of the outcome variable present in <code>colData(sy)</code> . This is usually the left-hand side of the prediction model formula.
<code>contigs</code>	Character vector of contigs (features) to include as predictors. Must be present in <code>rownames(sy)</code> .
<code>covariates</code>	Optional character vector of additional metadata variables to include as covariates. Default <code>NULL</code> for no covariates.
<code>use_genome_names</code>	Bool. Use <code>Genome_file</code> instead of <code>Contig_name</code> in <code>colnames(output)</code> . <code>Genome_file</code> is extracted from <code>SummarizedExperiment::elementMetadata(sy)</code> . If not available, <code>Contig_name</code> will be used. Default <code>TRUE</code> .

Details

Character variables are automatically converted to factors. Numeric variables are returned unchanged. For pre processing, refer to `caret::train()`.

Value

A data.frame containing the specified outcome variable, optional covariates, and selected contig features. The object is ready for direct use in **caret** functions such as `caret::train()`.

Examples

```
library(strainspy)

example_meta_path <- system.file("extdata", "example_metadata.csv.gz",
package = "strainspy")
example_meta <- read.csv(example_meta_path)
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
sy <- read_sylph(example_path, example_meta)
contigs <- rownames(sy)[1:10]
df <- prep_for_prediction(
  sy = sy,
  outcome = "Case_status",
  contigs = contigs,
  covariates = c("Sex", "BMI")
)
```

read_metaphlan

read_metaphlan

Description

This function reads a merged MetaPhlAn profile output and creates a SummarizedExperiment object. An optional metadata table can be provided and added directly to colData via an internal call to `modify_metadata()`.

Usage

```
read_metaphlan(
  file_path,
  meta_data = NULL,
  variable = "relative_abundance",
  clean_names = TRUE
)
```

Arguments

<code>file_path</code>	Character. Path to the merged metaphlan output file.
<code>meta_data</code>	data.frame. An optional tibble or data frame containing sample metadata. See <code>?modify_metadata</code> for the requirements.
<code>variable</code>	Character. Name of the input variable to import. Defaults to <code>relative_abundance</code> .
<code>clean_names</code>	Logical. If TRUE, file paths will be stripped of their directory path and file extension, leaving only the base file name. Defaults to TRUE.

Value

A SummarizedExperiment object with the following components:

<code>assays</code>	A matrix containing numeric features such as <code>relative_abundance</code>
<code>rowData</code>	DataFrame containing metadata for each sequence, such as <code>Contig_name</code> and <code>Genome_file</code> .
<code>colData</code>	DataFrame containing information for each sample (derived from the <code>Sample_file</code> field).

Note

Since MetaPhlAn profiles are usually generated per sample, first use `merge_metaphlan_files()` to merge and generate the input for this function.

If metadata is provided, it must meet the input requirements described in `modify_metadata()`. If appending metadata fails due to pre processing requirements, this function will issue a detailed warning and still return the SummarizedExperiment object without metadata. In that case, apply necessary fixes and subsequently call `modify_metadata()`.

Examples

```
# Read a merged metaphlan profile file into a SummarizedExperiment object
example_path <- system.file("extdata", "metaphlan_merged.tsv.gz",
  package = "strainspy")
mp <- read_metaphlan(example_path)
```

read_sourmash

read_sourmash

Description

This function reads a `sourmash gather` or `sourmash search` -like output CSV file and creates a SummarizedExperiment object. An optional metadata table can be provided and added directly to `colData` via an internal call to `modify_metadata()`.

Usage

```
read_sourmash(
  file_path,
  meta_data = NULL,
  variable = "match_containment_ani",
  clean_names = TRUE
)
```

Arguments

<code>file_path</code>	Character. Path to the sourmash gather file (CSV format).
<code>meta_data</code>	data.frame. A tibble or data frame containing sample metadata. Metadata requirements are identical to <code>read_sylph()</code> .
<code>variable</code>	Character. Name of the input variable to import. Defaults to <code>match_containment_ani</code> .
<code>clean_names</code>	Logical. If TRUE, file paths will be stripped of their directory path and file extension, leaving only the base file name. Defaults to TRUE.

Value

A SummarizedExperiment object with the following components:

<code>assays</code>	A matrix of the selected variable, by default <code>match_containment_ani</code> .
<code>rowData</code>	DataFrame containing metadata for each sequence, such as <code>Contig_name</code> and <code>Genome_file</code> .
<code>colData</code>	DataFrame containing information for each sample (derived from the <code>Sample_file</code> field).

Note

Some sourmash subcommands, like `gather`, are designed to take exactly one query file. To merge and save a CSV file suitable to use with this function, use `merge_sourmash_files()`.

If metadata is provided, it must meet the input requirements described in `modify_metadata()`. If appending metadata fails due to pre processing requirements, this function will issue a detailed warning and still return the SummarizedExperiment object without metadata. In that case, apply necessary fixes and subsequently call `modify_metadata()`.

Examples

```
# Read a merged sourmash file into a SummarizedExperiment object
example_path <- system.file("extdata", "example_sourmash.csv.gz",
  package = "strainspy")
sm <- read_sourmash(example_path)
```

read_sylph	<i>Read Sylph query or profile output file and create a SummarizedExperiment object</i>
------------	---

Description

This function reads a Sylph output file (either query or profile), parses the data, and returns a SummarizedExperiment object. An optional metadata table can be provided and added directly to colData via an internal call to modify_metadata().

Usage

```
read_sylph(  
  file_path,  
  meta_data = NULL,  
  variable = "Adjusted_ANI",  
  min_identity = 95,  
  clean_names = TRUE  
)
```

Arguments

file_path	Character. Path to the Sylph query or profile output file(s) in tsv format. A single merged sylph output file processed with merge_sylph_files(), or a character vector of paths to individual sylph output files can be provided. If multiple files are provided, they will be merged internally and saved to a temporary file. If the internal merging fails, use merge_sylph_files() to merge the files externally and provide the merged file path.
meta_data	data.frame. An optional tibble or data frame containing sample metadata. See Details
variable	Character. Name of the input variable to import. Defaults to Adjusted_ANI, other options can include Naive_ANI, Taxonomic_abundance and Sequence_abundance.
min_identity	Numeric. Minimum identity threshold for filtering ANI values. Defaults to 95.
clean_names	Logical. If TRUE, file paths will be stripped of their directory path and file extension, leaving only the base file name. Defaults to TRUE.

Details

If metadata is provided, it must meet the input requirements described in modify_metadata(). If appending metadata fails due to pre processing requirements, this function will issue a detailed warning and still return the SummarizedExperiment object without metadata. In that case, apply necessary fixes and subsequently call modify_metadata().

Value

A SummarizedExperiment object with the following components:

assays	A matrix of the selected variable, by default Adjusted_ANI.
rowData	DataFrame containing metadata for each sequence, such as Contig_name and Genome_file.
colData	DataFrame containing information for each sample (derived from the Sample_file field).

Examples

```
# Read a Sylph file
example_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy")
sy <- read_sylph(example_path)
```

read_taxonomy	<i>Read and process a taxonomy file</i>
---------------	---

Description

This function reads a headerless, two-column, tab-separated taxonomy file: genome accession and GTDB-style lineage string (d__...;p__...;...;s__...), as distributed with the sylph GTDB databases.

Usage

```
read_taxonomy(file_path)
```

Arguments

file_path	Path to the taxonomy file.
-----------	----------------------------

Value

A data.table with columns Genome, Taxonomy, Domain, Phylum, Class, Order, Family, Genus, Species and index (the row order after sorting by lineage).

Examples

```
tax <- read_taxonomy(system.file("extdata", "example_taxonomy.tsv.gz",
package = "strainspy"))
head(tax)
```

`show, strainspy_fit-method`*Show Method for strainspy_fit*

Description

Displays a summary of a `strainspy_fit` object, including model design, convergence, and coefficient matrix dimensions.

Usage

```
## S4 method for signature 'strainspy_fit'
show(object)
```

Arguments

`object` An object of class `strainspy_fit`.

Value

Invisibly returns `NULL` after printing a compact summary.

`show, strainspy_priors-method`*Show method for strainspy_priors objects*

Description

Show method for `strainspy_priors` objects

Usage

```
## S4 method for signature 'strainspy_priors'
show(object)
```

Arguments

`object` An object of class `strainspy_priors`

Value

Invisibly returns the object

strainspy

Strainspy Wrapper

Description

A wrapper to run a full strainspy workflow with explicit input validation.

Usage

```
strainspy(
  meta_path,
  sylph_path,
  design_formula,
  coef,
  min_nonzero = NULL,
  taxonomy_path = NULL,
  output_path = NULL,
  alpha = 0.05,
  p_adjust_method = "holm",
  model_method = "glmmTMB",
  scale_continuous = TRUE,
  nthreads = 2,
  return_vcov = FALSE,
  attach_taxonomy = TRUE,
  overwrite = TRUE,
  progress = TRUE
)
```

Arguments

meta_path	Path to the metadata file (CSV format).
sylph_path	Path to the sylph output file (TSV format).
design_formula	One-sided formula (or string coercible to formula) for the design matrix, e.g. ~ Condition + Age + (1 Subject).
coef	Coefficient index to report in top_hits(). This determines which design term is tested.
min_nonzero	Minimum number of samples for strain presence filtering. If NULL (default), this is set to ceiling(0.10 * n_samples).
taxonomy_path	Optional path to a taxonomy file (TSV format). If provided, taxonomy columns are appended to top hits.
output_path	Optional path to save top hits as CSV. If NULL, no file is written.
alpha	Significance threshold used by top_hits().
p_adjust_method	Multiple-testing correction method for top_hits().

<code>model_method</code>	Backend used by <code>glmZiBfit()</code> : "glmmTMB" or "gamlss".
<code>scale_continuous</code>	If TRUE, continuous metadata covariates are z-scored by <code>glmZiBfit()</code> .
<code>nthreads</code>	Number of threads for parallel processing (default 2).
<code>return_vcov</code>	If TRUE, include per-feature variance-covariance matrices in fit.
<code>attach_taxonomy</code>	If TRUE and <code>taxonomy_path</code> is provided, append taxonomy columns to top hits.
<code>overwrite</code>	If FALSE, abort when <code>output_path</code> already exists.
<code>progress</code>	Logical. If TRUE (default), progress bars and progress messages are printed. Set to FALSE to silence them.

Value

A data frame of top hits. If `taxonomy_path` is supplied and `attach_taxonomy` = TRUE, taxonomy columns are appended.

Examples

```
meta_path <- system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy")
sylph_path <- system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy")
taxonomy_path <- system.file("extdata", "example_taxonomy.tsv.gz",
  package = "strainspy")

# Decision 1: choose predictors for the design formula
# Decision 2: choose which coefficient index to report in top_hits()
# For ~ Case_status + Age_at_collection, coef = 2 corresponds to Case_status

out <- strainspy(
  meta_path = meta_path,
  sylph_path = sylph_path,
  design_formula = "~ Case_status + Age_at_collection",
  coef = 2,
  taxonomy_path = taxonomy_path,
  output_path = tempfile(fileext = ".csv"),
  nthreads = 2,
  alpha = 1
)
head(out)
```

strainspy_fit-accessors

Accessors for strainspy_fit objects

Description

Getters for the per-feature statistics and fitting metadata stored in a `strainspy_fit` object. Use these rather than reaching into the object with `@`.

Usage

```
getRowData(object)

## S4 method for signature 'strainspy_fit'
getRowData(object)

getCoefficients(object)

## S4 method for signature 'strainspy_fit'
getCoefficients(object)

getStdErrors(object)

## S4 method for signature 'strainspy_fit'
getStdErrors(object)

getPValues(object)

## S4 method for signature 'strainspy_fit'
getPValues(object)

getZStdErrors(object)

## S4 method for signature 'strainspy_fit'
getZStdErrors(object)

getZIPValues(object)

## S4 method for signature 'strainspy_fit'
getZIPValues(object)

getVcov(object)

## S4 method for signature 'strainspy_fit'
getVcov(object)

getZIVcov(object)

## S4 method for signature 'strainspy_fit'
getZIVcov(object)

getDesign(object)

## S4 method for signature 'strainspy_fit'
getDesign(object)

getFamily(object)
```

```
## S4 method for signature 'strainspy_fit'
getFamily(object)

getFitCall(object)

## S4 method for signature 'strainspy_fit'
getFitCall(object)

isScaleContinuous(object)

## S4 method for signature 'strainspy_fit'
isScaleContinuous(object)

getConvergence(object)

## S4 method for signature 'strainspy_fit'
getConvergence(object)
```

Arguments

object An object of class `strainspy_fit`.

Value

- `getRowData()`: a `DataFrame` of feature annotations, one row per feature.
- `getCoefficients()`, `getStdErrors()`, `getPValues()`: `DataFrames` of per-feature estimates, one column per model term.
- `getZIStdErrors()`, `getZIPValues()`: the zero-inflation counterparts, or `NULL` when the model has no zero-inflation component.
- `getVcov()`, `getZIVcov()`: a `List` of per-feature variance-covariance matrices, or `NULL` when they were not stored.
- `getDesign()`: the model formula.
- `getFamily()`: the model family object.
- `getFitCall()`: the call that produced the fit.
- `isScaleContinuous()`: `TRUE` if continuous covariates were z-scored before fitting.
- `getConvergence()`: a logical vector, one entry per feature, `TRUE` where the model converged.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
package = "strainspy"), meta_data = meta)
fit <- glmZiBFit(filter_by_presence(se, min_nonzero = 30)[1:10, ],
~ Case_status + Age_at_collection, nthreads = 1, return_vcov = FALSE)
head(getCoefficients(fit))
```

```
getDesign(fit)
isScaleContinuous(fit)
```

strainspy_fit-class *strainspy_fit: A Class for storing fit data from all strainspy models*

Description

The strainspy_fit class stores the results of a strainspy model fit.

Slots

row_data A DataFrame containing row data.

priors A strainspy_priors object, or NULL for an unpenalised maximum-likelihood fit.

scale_continuous A logical indicating whether all numeric columns were z-score standardized

coefficients A DataFrame containing model coefficients.

std_errors A DataFrame containing standard errors of the coefficients.

p_values A DataFrame containing p-values for the coefficients.

vcov A List of per-feature variance-covariance matrices, or NULL.

zi_coefficients A DataFrame containing zero-inflation coefficients.

zi_std_errors A DataFrame containing zero-inflation standard errors.

zi_p_values A DataFrame containing zero-inflation p-values.

zi_vcov A List of per-feature zero-inflation variance-covariance matrices, or NULL.

residuals A DataFrame containing residuals from the model.

convergence A logical indicating whether model fitting converged.

design The model formula.

call The matched call of the model.

family The family of the fit.

strainspy_priors-class *strainspy_priors: A Class for storing model fitting priors*

Description

The strainspy_priors class is designed to store MAP priors used for ordinal and zero-inflated beta model fitting using glmmTMB.

Slots

method Character. Prior estimation method used ("preset_weak", "preset_strong", "empirical", "manual").

priors_df Data frame. Data frame containing prior parameters.

boot_fixef Numeric matrix. Bootstrap SD values for fixed effect priors

boot_fixef_ZI Numeric matrix. Bootstrap SD values for zero inflated fixed effect priors

boot_disp Numeric matrix. Bootstrap mean and SD values for the dispersion term of beta component.

design The formula with the effect terms used

call The matched call of the model.

summary, strainspy_fit-method

Summary Method for strainspy_fit

Description

Provides a high-level summary of a strainspy_fit object: its call, the first rows of the coefficients and zero-inflation coefficients, and whether residuals are available.

Usage

```
## S4 method for signature 'strainspy_fit'
summary(object)
```

Arguments

object An object of class strainspy_fit.

Value

A list summarizing the strainspy_fit object.

top_hits	<i>Provides the top results of a strainspy_fit object, sorted by adjusted p-value.</i>
----------	--

Description

Adjusts the p-values of one coefficient of a strainspy_fit for multiple testing with p.adjust() and returns the features that pass alpha, sorted by adjusted p-value. For taxonomy-aware adjustment see hadjust().

Usage

```
top_hits(fit, coef = 2, method = "holm", alpha = 0.05)
```

Arguments

fit	A strainspy_fit object.
coef	The coefficient to report and use for p-value adjustment. Defaults to 2.
method	The method for p-value adjustment, passed to p.adjust(). One of "holm" (default), "bonferroni", "BH" (Benjamini-Hochberg) or "BY".
alpha	Numeric. Significance threshold for adjusted p-values. Defaults to 0.05.

Value

A tibble with the top hits, sorted by adjusted p-value.

Examples

```
meta <- read.csv(system.file("extdata", "example_metadata.csv.gz",
  package = "strainspy"))
meta$Case_status <- factor(meta$Case_status)
se <- read_sylph(system.file("extdata", "example_sylph_profile.tsv.gz",
  package = "strainspy"), meta_data = meta)
fit <- glmZiBFit(filter_by_presence(se, min_nonzero = 30)[1:10, ],
  ~ Case_status + Age_at_collection, nthreads = 1, return_vcov = FALSE)
top_hits(fit, coef = 2, alpha = 1)
```

Index

abundanceFit, [3](#)
ani_distance, [5](#)

caseControlFit, [6](#)
comp_ani_diff_and_posthoc_test, [7](#)
compute_eb_priors, [9](#)

define_priors, [10](#)

estimate_beta_diversity, [11](#)
estimate_sample_richness, [12](#)
extract_fixef_priors, [14](#)
extract_fixef_priors, strainspy_priors-method
 (extract_fixef_priors), [14](#)
extract_strainspy, [15](#)

filter_by_presence, [16](#)

getCoefficients
 (strainspy_fit-accessors), [47](#)
getCoefficients, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getContigNames, [17](#)
getContigNames, strainspy_fit-method
 (getContigNames), [17](#)
getConvergence
 (strainspy_fit-accessors), [47](#)
getConvergence, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getDesign (strainspy_fit-accessors), [47](#)
getDesign, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getFamily (strainspy_fit-accessors), [47](#)
getFamily, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getFitCall (strainspy_fit-accessors), [47](#)
getFitCall, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getGenomes, [17](#)
getGenomes, strainspy_fit-method
 (getGenomes), [17](#)

getPriorsDf, [18](#)
getPriorsDf, strainspy_priors-method
 (getPriorsDf), [18](#)
getPValues (strainspy_fit-accessors), [47](#)
getPValues, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getResiduals, [19](#)
getResiduals, strainspy_fit-method
 (getResiduals), [19](#)
getRowData (strainspy_fit-accessors), [47](#)
getRowData, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getSamples, [19](#)
getSamples, strainspy_fit-method
 (getSamples), [19](#)
getStdErrors (strainspy_fit-accessors),
 [47](#)
getStdErrors, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getVcov (strainspy_fit-accessors), [47](#)
getVcov, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getZICoefficients, [20](#)
getZICoefficients, strainspy_fit-method
 (getZICoefficients), [20](#)
getZIPValues (strainspy_fit-accessors),
 [47](#)
getZIPValues, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getZIStdErrors
 (strainspy_fit-accessors), [47](#)
getZIStdErrors, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)
getZIVcov (strainspy_fit-accessors), [47](#)
getZIVcov, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)

glmObFit, [21](#)
glmQBFit, [23](#)
glmZiBFit, [24](#)

hadjust, [26](#)

isScaleContinuous
 (strainspy_fit-accessors), [47](#)

isScaleContinuous, strainspy_fit-method
 (strainspy_fit-accessors), [47](#)

merge_metaphlan_files, [27](#)

merge_sourmash_files, [28](#)

merge_sylph_files, [29](#)

modify_metadata, [30](#)

plot_ani_dist, [31](#)

plot_histogram, [32](#)

plot_manhattan, [33](#)

plot_mds, [35](#)

plot_pca_biplot, [36](#)

plot_prior_bootstrap, [37](#)

plot_prior_bootstrap, strainspy_priors-method
 (plot_prior_bootstrap), [37](#)

plot_volcano, [38](#)

prep_for_prediction, [39](#)

read_metaphlan, [40](#)

read_sourmash, [41](#)

read_sylph, [43](#)

read_taxonomy, [44](#)

show, strainspy_fit-method, [45](#)

show, strainspy_priors-method, [45](#)

strainspy, [46](#)

strainspy_fit-accessors, [47](#)

strainspy_fit-class, [50](#)

strainspy_priors-class, [50](#)

summary, strainspy_fit-method, [51](#)

top_hits, [52](#)