Package ‘BANDITS’

September 20, 2024

Type Package

Title BANDITS: Bayesian ANalysis of DIfferenTial Splicing

Version 1.21.0

Author Simone Tiberi [aut, cre].

Maintainer Simone Tiberi <simone.tiberi@uzh.ch>

Description BANDITS is a Bayesian hierarchical model for detecting differential splicing of genes and transcripts,
via differential transcript usage (DTU), between two or more conditions.
The method uses a Bayesian hierarchical framework, which allows for sample specific proportions in a Dirichlet-Multinomial model, and samples the allocation of fragments to the transcripts.
Parameters are inferred via Markov chain Monte Carlo (MCMC) techniques and a DTU test is performed
via a multivariate Wald test on the posterior densities for the average relative abundance of transcripts.

biocViews DifferentialSplicing, AlternativeSplicing, Bayesian, Genetics, RNASeq, Sequencing, DifferentialExpression, GeneExpression, MultipleComparison, Software, Transcription, StatisticalMethod, Visualization

License GPL (>= 3)

Depends R (>= 4.3.0)

Imports Rcpp, doRNG, MASS, data.table, R.utils, doParallel, parallel, foreach, methods, stats, graphics, ggplot2, DRIMSeq, BiocParallel

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, testthat, tximport, BiocStyle, GenomicFeatures, Biostrings

SystemRequirements C++17

VignetteBuilder knitr

RoxygenNote 7.2.3

ByteCompile true

URL https://github.com/SimoneTiberi/BANDITS

BugReports https://github.com/SimoneTiberi/BANDITS/issues

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

git_branch devel

git_last_commit f75f35f

git_last_commit_date 2024-04-30

Repository Bioconductor 3.20

Date/Publication 2024-09-20

Contents

`BANDITS-package` *BANDITS: Bayesian ANalysis of DIfferenTial Splicing*

Description

BANDITS is a Bayesian hierarchical model for detecting differential splicing of genes and transcripts, via differential transcript usage (DTU), between two or more conditions. The method uses a Bayesian hierarchical framework, which allows for sample specific proportions in a Dirichlet-Multinomial model, and samples the allocation of fragments to the transcripts. Parameters are inferred via Markov chain Monte Carlo (MCMC) techniques and a DTU test is performed via a multivariate Wald test on the posterior densities for the average relative abundance of transcripts.

Questions relative to BANDITS should be either written to the Bioconductor support site, tagging the question with 'BANDITS', or reported as a new issue at BugReports.

To access the vignettes, type: `browseVignettes("BANDITS")`.

Author(s)

Simone Tiberi [aut, cre].

Maintainer: Simone Tiberi <simone.tiberi@uzh.ch>

BANDITS_data-class *BANDITS_data class*

Description

BANDITS_data contains all the information required to perform differential transcript usage (DTU). BANDITS_data associates each gene (genes), to its transcript ids (transcripts), effective transcript lengths (effLen), equivalence classes (classes) and respective counts (counts). The same structure is also used for groups of genes with reads/fragments compatible with >1 gene (with uniqueId == FALSE); in this case the 'genes' field contains all the genes ids in the group. Created via create_data.

Usage

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

Arguments

object	a 'BANDITS_data' object.

Value

- show(object): returns the number of genes and transcripts in the BANDITS_data object.

Slots

genes list of gene names: each element is a vector of 1 or more gene names indicating the genes to be analyzed together.

transcripts list of transcript names: each element is a vector of 1 or more transcript names indicating the transcripts matching the gene names in the corresponding element of @genes object.

effLen list of transcript effective lengths: each element is a vector of 1 or more numbers, indicating the effective length of the transcripts in the corresponding element of @transcripts object.

classes list of matrices: the (i,j) element of each matrix is 1 if the i-th transcript is present in the j-th equivalence class, 0 otherwise.

counts list of matrices: the (i,j) element indicates the reads/fragments compatible with the i-th equivalence class in sample j.

uniqueId logical, it indicates if the element contains one gene to be analyzed alone (TRUE), or more genes to be analyzed jointly (FALSE).

all_genes vector, it lists all the genes to be analyzed (with at least 2 transcripts).

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

create_data, filter_transcripts, eff_len_compute

Examples

```
# load the pre-computed data:
data("input_data", package = "BANDITS")
show(input_data)
```

BANDITS_test-class *BANDITS_test class*

Description

BANDITS_test contains the results of the differential transcript usage (DTU) test. BANDITS_test is organized in three data.frames containing: gene-level results, transcript-level results and convergence diagnostics of the Markov chain Monte Carlo (MCMC) posterior chains. Created via test_DTU. To test for convergence, we use Heidelberger and Welch's convergence diagnostic, implemented in coda::heidel.diag, to test for the stationarity of the chain for the full log-posterior density; we use a 0.01 threshold on the p.value to reject the null hypotehsis of stationarity.

Usage

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

## S4 method for signature 'BANDITS_test'
convergence(x)

## S4 method for signature 'BANDITS_test'
top_genes(x, n = Inf, sort_by_g = "p.value")

## S4 method for signature 'BANDITS_test'
top_transcripts(x, n = Inf, sort_by_tr = "gene")

## S4 method for signature 'BANDITS_test'
gene(x, gene_id)

## S4 method for signature 'BANDITS_test'
transcript(x, transcript_id)

## S4 method for signature 'BANDITS_test'
plot_proportions(x, gene_id, CI = TRUE, CI_level = 0.95)
```

Arguments

object, x	a 'BANDITS_test' object.
n	the number of genes or transcripts to report. By default n = Inf and all results will be reported.
sort_by_g	"p.value" for sorting results according to gene-level significance (i.e., p.value); "DTU_measure" for sorting results according to the 'DTU_measure' (check the vignette for details).

`sort_by_tr`	"gene" for sorting results according to gene-level significance (i.e., p.value); "transcript" for sorting results according to transcript-level significance (i.e., p.value).
`gene_id`	a character string or vector indicating the gene or genes whose results should be retrieved.
`transcript_id`	a character string or vector indicating the transcript or transcripts whose results should be retrieved.
`CI`	a logical element indicating whether to also plot confidence boundaries (TRUE) or not (FALSE).
`CI_level`	a number between 0 and 1, indicating the level of the confidence interval to plot.

Value

- `show(object)`: prints the number of gene and transcript level results in the `BANDITS_test` object.
- `top_genes(x, n = Inf, sort_by_g = "p.value")`: returns the gene-level results of the DTU test for the top 'n' significant genes. By default n = Inf and all results will be reported. sort_by_g = "gene" for sorting results according to gene-level significance; sort_by_g = "DTU_measure" for sorting results according to the 'DTU_measure'.
- `top_transcripts(x, n = Inf, sort_by_tr = "gene")`: returns the transcript-level results of the DTU test for the top 'n' significant genes. By default n = Inf and all results will be reported. sort_by_tr = "gene" for sorting results according to gene-level significance; sort_by_tr = "transcript" for sorting results according to transcript-level significance.
- `convergence(x)`: returns the convergence diagnostic of the posterior MCMC chains for every gene.
- `gene(x, gene_id)`: returns a list with all results for the gene(s) specified in 'gene_id': gene results, corresponding transcript results and convergence diagnostic.
- `transcript(x, transcript_id)`: returns a list with all results for the trancript specified in 'transcript_id': transcript results, corresponding gene results and convergence diagnostic.
- `plot_proportions(x, gene_id, CI = TRUE, CI_level = 0.95)`: plots the posterior means of the average transcripts relative expression (i.e., the proportions) of each condition, for the gene specified in 'gene_id'. If 'CI' is TRUE, a profile Wald type confidence interval will also be plotted for each transcript estimated proportion; the level of the confidence interval is specified by 'CI_level'.

Slots

`Gene_results` a `data.frame` containing the gene-level results of the DTU test, structured in the following columns:

- Gene_id contains the gene names;
- p.values is the gene-level p.values of the DTU test;
- adj.p.values is the Benjamini-Hochberg adjusted p.values (via `p.adjust`);
- p.values_inverted (only available for 2-group comparisons) is a conservative p.value, accounting for the inversion of the dominant transcript between conditions: p.values_inverted = p.values, if the dominant transcript varies between conditions, and p.values_inverted = sqrt(p.values) if both conditions have the same dominant transcript;
- adj.p.values_inverted (only available for 2-group comparisons) is the Benjamini-Hochberg adjusted p.values_inverted, via `p.adjust`;

- DTU_measure (only available for 2-group comparisons) represents a measure of the intensity of changes between conditions. This measure ranges between 0, when proportions are identical between groups, and 2, when an isoform is always expressed in group A and a different transcript is always chosen in group B;
- Mean log-prec "group_name" indicates the posterior mean of the logarithm of the Dirichlet precision parameter in group "group_name". The precision parameter models the degree of over-dispersion between samples: the higher the precision parameter (or its logarithm), the lower the sample-to-sample variability.
- SD log-prec "group_name" indicates the standard deviation of the logarithm of the Dirichlet precision parameter in group "group_name".

`Transcript_results` a `data.frame` containing the transcript-level results of the DTU test, structured in the following columns:

- Gene_id contains the gene names;
- Transcript_id contains the transcript names;
- p.values is the transcript-level p.values of the DTU test;
- adj.p.values is the Benjamini-Hochberg adjusted p.values (via `p.adjust`);
- Max_Gene_Tr.p.val is a conservative p.value and represents the maximum between the transcript p.value and corresponding gene p.value;
- Max_Gene_Tr.Adj.p.val is the Benjamini-Hochberg adjusted Max_Gene_Tr.p.val (via `p.adjust`);
- Mean "group_name" indicates the posterior mean of the average relative abundance of the transcript in group "group_name" (an `NaN` value indicates that no data was available for a group to estimate parameters);
- SD "group_name" indicates the standard deviation of the average relative abundance of the transcript in group "group_name" (an `NaN` value indicates that no data was available for a group to estimate parameters); this column indicates the variability in the mean estimate and is used to plot a Wald type confidence interval for the mean relative abundance via `plot_proportions`.

`Convergence` a `data.frame` containing the convercence diagnostics of the DTU test, structured in the following columns:

- Gene_id contains the gene names;
- converged is 1 if convergence was reached, 0 otherwise;
- burn_in indicates what fraction of the chain was removed to ensure convergence (excluding the `burn_in` parameter specified in `test_DTU`.

`samples_design` a `data.frame` containing the design of the experiment, with one row for each sample and two columns with names 'sample_id' and 'group', specifying the id and group of each sample, respectively. It is provided by the user to `test_DTU`.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

`test_DTU`, `create_data`, `BANDITS_data`

Examples

```
# load the pre-computed results:
data("results", package = "BANDITS")
```

```
show(results)

# Visualize the most significant Genes, sorted by gene level significance.
head(top_genes(results))

# Alternatively, gene-level results can also be sorted according to DTU_measure,
# which is a measure of the strength of the change between the
# average relative abundances of the two groups.
head(top_genes(results, sort_by = "DTU_measure"))

# Visualize the most significant transcripts, sorted by transcript level significance.
head(top_transcripts(results, sort_by = "transcript"))

# Visualize the convergence output for the most significant genes,
# sorted by gene level significance.
head(convergence(results))

# We can further use the 'gene' function to gather all output for a specific gene:
# gene level, transcript level and convergence results.
top_gene = top_genes(results, n = 1)
gene(results, top_gene$Gene_id)

# Similarly we can use the 'transcript function to gather all output
# for a specific transcript.
top_transcript = top_transcripts(results, n = 1)
transcript(results, top_transcript$Transcript_id)

#Finally, we can plot the estimated average transcript relative expression
# in the two groups for a specific gene via 'plot_proportions'.
plot_proportions(results, top_gene$Gene_id)
```

`create_data` *Create a 'BANDITS_data' object*

Description

`create_data` imports the equivalence classes and create a 'BANDITS_data' object.

Usage

```
create_data(
  salmon_or_kallisto,
  gene_to_transcript,
  salmon_path_to_eq_classes = NULL,
  kallisto_equiv_classes = NULL,
  kallisto_equiv_counts = NULL,
  kallisto_counts = NULL,
  eff_len,
  n_cores = NULL,
  transcripts_to_keep = NULL,
  max_genes_per_group = 50
)
```

Arguments

`salmon_or_kallisto`
: a character string indicating the input data: 'salmon' or 'kallisto'.

`gene_to_transcript`
: a matrix or data.frame with a list of gene-to-transcript correspondances. The first column represents the gene id, while the second one contains the transcript id.

`salmon_path_to_eq_classes`
: (for salmon input only) a vector of length equals to the number of samples: each element indicates the path to the equivalence classes of the respective sample (computed by salmon).

`kallisto_equiv_classes`
: (for kallisto input only) a vector of length equals to the number of samples: each element indicates the path to the equivalence classes ('.ec' files) of the respective sample (computed by kallisto).

`kallisto_equiv_counts`
: (for kallisto input only) a vector of length equals to the number of samples: each element indicates the path to the counts of the equivalence classes ('.tsv' files) of the respective sample (computed by kallisto).

`kallisto_counts`
: (for kallisto input only) a matrix or data.frame, with 1 column per sample and 1 row per transcript, containing the estimated abundances for each transcript in each sample, computed by kallisto. The matrix must be unfiltered and the order or rows must be unchanged.

`eff_len`
: a vector containing the effective length of transcripts; the vector names indicate the transcript ids. Ideally, created via `eff_len_compute`.

`n_cores`
: the number of cores to parallelize the tasks on. It is highly suggested to use at least one core per sample (default if not specificied by the user).

`transcripts_to_keep`
: a vector containing the list of transcripts to keep. Ideally, created via `filter_transcripts`.

`max_genes_per_group`
: an integer number specifying the maximum number of genes that each group can contain. When equivalence classes contain transcripts from distinct genes, these genes are analyzed together. For computational reasons, 'max_genes_per_group' sets a limit to the number of genes that each group can contain.

Value

A `BANDITS_data` object.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

`eff_len_compute`, `filter_transcripts`, `filter_genes`, `BANDITS_data`

Examples

```
# specify the directory of the internal data:
data_dir = system.file("extdata", package = "BANDITS")

# load gene_to_transcript matching:
data("gene_tr_id", package = "BANDITS")

# Specify the directory of the transcript level estimated counts.
sample_names = paste0("sample", seq_len(4))
quant_files = file.path(data_dir, "STAR-salmon", sample_names, "quant.sf")

# Load the transcript level estimated counts via tximport:
library(tximport)
txi = tximport(files = quant_files, type = "salmon", txOut = TRUE)
counts = txi$counts

# Optional (recommended): transcript pre-filtering
transcripts_to_keep = filter_transcripts(gene_to_transcript = gene_tr_id,
                                         transcript_counts = counts,
                                         min_transcript_proportion = 0.01,
                                         min_transcript_counts = 10,
                                         min_gene_counts = 20)

# compute the Median estimated effective length for each transcript:
eff_len = eff_len_compute(x_eff_len = txi$length)

# specify the path to the equivalence classes:
equiv_classes_files = file.path(data_dir, "STAR-salmon", sample_names, "aux_info", "eq_classes.txt")

# create data from 'salmon' and filter internally lowly abundant transcripts:
input_data = create_data(salmon_or_kallisto = "salmon",
                         gene_to_transcript = gene_tr_id,
                         salmon_path_to_eq_classes = equiv_classes_files,
                         eff_len = eff_len,
                         n_cores = 2,
                         transcripts_to_keep = transcripts_to_keep)
input_data

# create data from 'kallisto' and filter internally lowly abundant transcripts:
kallisto_equiv_classes = file.path(data_dir, "kallisto", sample_names, "pseudoalignments.ec")
kallisto_equiv_counts  = file.path(data_dir, "kallisto", sample_names, "pseudoalignments.tsv")

input_data_2 = create_data(salmon_or_kallisto = "kallisto",
                           gene_to_transcript = gene_tr_id,
                           kallisto_equiv_classes = kallisto_equiv_classes,
                           kallisto_equiv_counts = kallisto_equiv_counts,
                           kallisto_counts = counts,
                           eff_len = eff_len, n_cores = 2,
                           transcripts_to_keep = transcripts_to_keep)
input_data_2
```

`eff_len_compute` *Compute the median effective length of transcripts.*

Description

`eff_len_compute` inputs the estimated effective length of transcripts from every sample, and computes the median effective length of each transcript across samples.

Usage

```
eff_len_compute(x_eff_len)
```

Arguments

`x_eff_len`	is a list: each element of the list refers to a specific sample and is a matrix or data.frame with the estimated effective length under the column 'EffectiveLength' and the transcript name under the column 'Name'.

Value

A vector containing the effective length of transcripts; the vector names indicate the transcript ids.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

`filter_transcripts`, `create_data`

Examples

```
# specify the directory of the internal data:
data_dir = system.file("extdata", package = "BANDITS")

# Specify the directory of the transcript level estimated counts.
quant_files = file.path(data_dir, "STAR-salmon", paste0("sample", seq_len(4)), "quant.sf")

# Load the transcript level estimated counts via tximport:
library(tximport)
txi = tximport(files = quant_files, type = "salmon", txOut = TRUE)

# compute the Median estimated effective length for each transcript:
eff_len = eff_len_compute(x_eff_len = txi$length)
head(eff_len)
```

`filter_genes` *Filter lowly abundant genes.*

Description

`filter_genes` filters genes, according to the overall number of counts (across all samples) compatible with the gene. The filtering also applies to groups of genes with reads/fragments compatible with >1 gene; in this case, the number of counts considered is across all genes in the group.

Usage

```
filter_genes(BANDITS_data, min_counts_per_gene = 10)
```

Arguments

BANDITS_data a 'BANDITS_data' object, created with the create_data function.

min_counts_per_gene the minimum number of counts compatible with a gene (across all samples).

Details

The function inputs a 'BANDITS_data' object, and returns again a 'BANDITS_data' object after filtering genes and groups of genes.

Value

A BANDITS_data object.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

filter_transcripts, create_data, BANDITS_data

Examples

```
# load the pre-computed data:
data("input_data", package = "BANDITS")
input_data

# Filter lowly abundant genes:
input_data = filter_genes(input_data, min_counts_per_gene = 20)
```

---

filter_transcripts *Filter lowly abundant transcripts.*

Description

filter_transcripts filters transcripts, before loading the data, according to estimated transcript level counts. The function outputs a vector containing the list of transcripts which respect the filtering criteria across all samples (i.e., min_transcript_proportion, min_transcript_counts and min_gene_counts).

Usage

```
filter_transcripts(
  gene_to_transcript,
  transcript_counts,
  min_transcript_proportion = 0.01,
  min_transcript_counts = 1,
  min_gene_counts = 10
)
```

Arguments

`gene_to_transcript`
: a matrix or data.frame with a list of gene-to-transcript correspondances. The first column represents the gene id, while the second one contains the transcript id.

`transcript_counts`
: a matrix or data.frame, with 1 column per sample and 1 row per transcript, containing the estimated abundances for each transcript in each sample.

`min_transcript_proportion`
: the minimum relative abundance (i.e., proportion) of a transcript in a gene.

`min_transcript_counts`
: the minimum overall abundance of a transcript (adding counts from all samples).

`min_gene_counts`
: the minimum overall abundance of a gene (adding counts from all samples).

Details

Transcript pre-filtering is highly suggested: it both improves the performance of the method and decreases its computational cost.

Value

A vector containing the list of transcripts which respect the filtering criteria.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

filter_genes, create_data, BANDITS_data

Examples

```
# specify the directory of the internal data:
data_dir = system.file("extdata", package = "BANDITS")

# load gene_to_transcript matching:
data("gene_tr_id", package = "BANDITS")

# Load the transcript level estimated counts via tximport:
library(tximport)
quant_files = file.path(data_dir, "STAR-salmon", paste0("sample", seq_len(4)), "quant.sf")
txi = tximport(files = quant_files, type = "salmon", txOut = TRUE)
counts = txi$counts

# transcript pre-filtering:
transcripts_to_keep = filter_transcripts(gene_to_transcript = gene_tr_id,
                                         transcript_counts = counts,
                                         min_transcript_proportion = 0.01,
                                         min_transcript_counts = 10,
                                         min_gene_counts = 20)
head(transcripts_to_keep)
```

gene_tr_id — *Gene-transcript matching*

Description

Gene-transcript matching

Arguments

gene_tr_id	a data.frame containing the matching between gene (1st column) and transcript identifiers (2nd column). The gtf file used was downloaded from the ARMOR github repository here.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

Examples

```
# Compute the 'gene_tr_id' object from the gtf file as shown in the vignettes,
# see: browseVignettes("BANDITS").
# suppressMessages(library(GenomicFeatures))
# tx = makeTxDbFromGFF("Homo_sapiens.GRCh38.93.1.1.10M.gtf")
# ss = unlist(transcriptsBy(tx, by="gene"))
# gene_tr_id_gtf = data.frame(gene_id = names(ss), transcript_id = ss$tx_name )
# gene_tr_id_gtf = gene_tr_id_gtf[ rowSums( is.na(gene_tr_id_gtf)) == 0, ] # remove eventual NA's
# gene_tr_id_gtf = unique(gene_tr_id_gtf) # remove eventual duplicated rows

# load the Gene-Transcript data.frame and visualize its top
data(gene_tr_id, package = "BANDITS")
head(gene_tr_id)
```

input_data — *A* BANDITS_data *object, generated with* create_data

Description

A BANDITS_data object, generated with create_data

Arguments

input_data	a BANDITS_data object, generated via create_data.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

create_data, BANDITS_data

Examples

```
# Object 'input_data' is generated via 'create_data' as shown in the vignettes:
# see browseVignettes("BANDITS").

# load the pre-computed data:
data("input_data", package = "BANDITS")
input_data
```

`plot_precision` *Plot the log-precision estimates*

Description

`plot_precision` plots a histogram of the estimates for the log-precision parameter of the Dirichlet-Multinomial distribution, obtained via `prior_precision`. The solid line represents the normal prior for the log-precision parameter.

Usage

```
plot_precision(prior)
```

Arguments

`prior`	the prior of the log-precision parameter, computed via `prior_precision`.

Value

A plot.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

`test_DTU`, `prior_precision`

Examples

```
# load the pre-computed precision estimates:
data(precision, package = "BANDITS")

# Plot the histogram of the genewise log-precision estimates.
# The black solid line represents the normally distributed prior distribution
# for the log-precision parameter.
plot_precision(precision)
```

precision — *Estimates for the log-precision parameter, generated with* prior_precision

Description

Estimates for the log-precision parameter, generated with prior_precision

Arguments

precision — a list, generated via prior_precision, with two elements:

- 'prior', a numeric vector containing the mean and standard deviation prior estimates for the log-precision parameter of the Dirichlet-multinomial distribution;
- 'genewise_log_precision', a numeric vector containing the gene-wise precision estimates of the log-precision parameter of the Dirichlet-multinomial distribution.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

prior_precision, plot_precision

Examples

```
# Object 'precision' is generated via 'prior_precision' as shown in the vignettes:
# see browseVignettes("BANDITS").

# load the pre-computed precision estimates:
data(precision, package = "BANDITS")

precision$prior
head(precision$genewise_log_precision)
```

prior_precision — *Infer an informative prior for the precision*

Description

prior_precision uses DRIMSeq's pipeline to infer an informative prior for the precision parameter of the Dirichlet-Multinomial distribution. The function computes the genewise estimates for the precision via DRIMSeq::dmPrecision, and calculates the mean and standard deviation of the log-precision estimates.

Usage

```
prior_precision(
  gene_to_transcript,
  transcript_counts,
  n_cores = 1,
  transcripts_to_keep = NULL,
  max_n_genes_used = 100
)
```

Arguments

`gene_to_transcript`
: a matrix or data.frame with a list of gene-to-transcript correspondances. The first column represents the gene id, while the second one contains the transcript id.

`transcript_counts`
: a matrix or data.frame, with 1 column per sample and 1 row per transcript, containing the estimated abundances for each transcript in each sample.

`n_cores`
: the number of cores to parallelize the tasks on.

`transcripts_to_keep`
: a vector containing the list of transcripts to keep. Ideally, created via `filter_transcripts`.

`max_n_genes_used`
: the maximum number of genes to compute the prior on. First, genes with at least 2 transcripts are selected. Then, if more than 'max_n_genes_used' such genes are available, 'max_n_genes_used' of these genes are sampled at random and used to calculate the prior of the precision parameter. A smaller 'max_n_genes_used' (minimum 100) will lead to faster but more approximate prior estimates.

Value

A list with 2 objects containing:

- prior: a vector containing the mean and standard deviation of the log-precision, used to formulate an informative prior in `test_DTU`;
- genewise_log_precision: a numeric vector with the individual genewise estimates for the log-precision.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

`test_DTU`, `plot_precision`

Examples

```
# specify the directory of the internal data:
data_dir = system.file("extdata", package = "BANDITS")

# load gene_to_transcript matching:
data("gene_tr_id", package = "BANDITS")
```

```
# Load the transcript level estimated counts via tximport:
library(tximport)
quant_files = file.path(data_dir, "STAR-salmon", paste0("sample", seq_len(4)), "quant.sf")
txi = tximport(files = quant_files, type = "salmon", txOut = TRUE)
counts = txi$counts

# Optional (recommended): transcript pre-filtering
transcripts_to_keep = filter_transcripts(gene_to_transcript = gene_tr_id,
                                         transcript_counts = counts,
                                         min_transcript_proportion = 0.01,
                                         min_transcript_counts = 10,
                                         min_gene_counts = 20)

# Infer an informative prior for the precision parameter
# Use the same filtering criteria as in 'create_data', by choosing the same argument for 'transcripts_to_keep'.
# If transcript pre-filtering is not performed, leave 'transcripts_to_keep' unspecified.
set.seed(61217)
precision = prior_precision(gene_to_transcript = gene_tr_id, transcript_counts = counts,
                            n_cores = 2, transcripts_to_keep = transcripts_to_keep)
precision$prior
head(precision$genewise_log_precision)
```

---

results *Results of the DTU test, generated with* test_DTU

Description

Results of the DTU test, generated with test_DTU

Arguments

results	a BANDITS_test object containing the results of the DTU test, generated with test_DTU.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

test_DTU, BANDITS_test

Examples

```
# Object 'results' is generated via 'test_DTU' as shown in the vignettes:
# see browseVignettes("BANDITS").

# load the pre-computed results:
data("results", package = "BANDITS")
results
```

test_DTU *Perform differential splicing*

Description

test_DTU performs differential splicing, via differential transcript usage (DTU), between 2 or more groups. Parameters are inferred via Markov chain Monte Carlo (MCMC) techniques and a DTU test is performed via a multivariate Wald test on the posterior densities for the average relative abundance of transcripts. Warning: the samples in samples_design must have the same order as those in the 'path_to_eq_classes' parameter of the create_data function.

Usage

```
test_DTU(
  BANDITS_data,
  precision = NULL,
  R = 10^4,
  burn_in = 2 * 10^3,
  samples_design,
  group_col_name = "group",
  n_cores = 1,
  gene_to_transcript,
  theshold_pval = 0.1
)
```

Arguments

BANDITS_data	a 'BANDITS_data' object.
precision	a vector with the mean and standard deviation of the log-precision parameter.
R	the number of iterations for the MCMC algorithm (after the burn-in). Min 10^4. Albeit no difference was observed in simulation studies when increasing 'R' above 10^4, we encourage users to possibly use higher values of R (e.g., 2*10^4), if the computational time allows it, particularly for comparisons between 3 or more groups.
burn_in	the length of the burn-in to be discarded (before convergence is reached). Min 2*10^3. Albeit no difference was observed in simulation studies when increasing 'burn_in' above 2*10^3, we encourage users to possibly use higher values of R (e.g., double) if the computational time allows it.
samples_design	a data.frame indicating the design of the experiment with one row for each sample: samples_design must contain a column with the sample id and one with the group id. Warning: the samples in samples_design must have the same order as those in the 'path_to_eq_classes' parameter of the create_data function.
group_col_name	the name of the column of 'samples_design' containing the group id. By default group_col_name = "group".
n_cores	the number of cores to parallelize the tasks on.
gene_to_transcript	a matrix or data.frame with a list of gene-to-transcript correspondances. The first column represents the gene id, while the second one contains the transcript id.

theshold_pval is a threshold between 0 and 1; when running test_DTU, if the p.value of a gene is < theshold_pval, a second (independent) MCMC chain is run and the p.value is re-computed on the aggregation of the two chains. By defauls theshold_pval = 0.1, while theshold_pval = 1 corresponds to running all chains twice, and theshold_pval = 0 means all chains will only run once.

Value

A BANDITS_test object.

Author(s)

Simone Tiberi <simone.tiberi@uzh.ch>

See Also

create_data, BANDITS_data, BANDITS_test

Examples

```
# load gene_to_transcript matching:
data("gene_tr_id", package = "BANDITS")

# We define the design of the study
samples_design = data.frame(sample_id = paste0("sample", seq_len(4)),
                            group = c("A", "A", "B", "B"))

# load the pre-computed data:
data("input_data", package = "BANDITS")
input_data

# Filter lowly abundant genes:
input_data = filter_genes(input_data, min_counts_per_gene = 20)

# load the pre-computed precision estimates:
data(precision, package = "BANDITS")

## Test for DTU
set.seed(61217)
results = test_DTU(BANDITS_data = input_data,
                   precision = precision$prior,
                   samples_design = samples_design,
                   R = 10^4, burn_in = 2*10^3, n_cores = 2,
                   gene_to_transcript = gene_tr_id)
results
```

Index