Package 'dasper'

September 7, 2021

Title Detecting abberant splicing events from RNA-sequencing data

Version 1.2.0

Date 2020-10-26

Description The aim of dasper is to detect aberrant splicing events from RNA-seq data. dasper will use as input both junction and coverage data from RNA-seq to calculate the deviation of each splicing event in a patient from a set of user-defined controls. dasper uses an unsupervised outlier detection algorithm to score each splicing event in the patient with an outlier score representing the degree to which that splicing event looks abnormal.

License Artistic-2.0

URL https://github.com/dzhang32/dasper

BugReports https://support.bioconductor.org/t/dasper

biocViews Software, RNASeq, Transcriptomics, AlternativeSplicing, Coverage, Sequencing

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Suggests BiocStyle, covr, testthat, GenomicState, ggplot2, ggpubr, ggrepel, grid, knitcitations, knitr, recount, rmarkdown, sessioninfo, rtracklayer, tibble

VignetteBuilder knitr

Imports basilisk, BiocFileCache, BiocParallel, data.table, dplyr, GenomeInfoDb, GenomicFeatures, GenomicRanges, IRanges, magrittr, megadepth, methods, plyranges, readr, reticulate, S4Vectors, stringr, SummarizedExperiment, tidyr

Depends R (>= 4.0)

StagedInstall no

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

git_branch RELEASE_3_13

git_last_commit 9bf018b

git_last_commit_date 2021-05-19

Date/Publication 2021-09-07

Author David Zhang [aut, cre] (<https://orcid.org/0000-0003-2382-8460>),
Leonardo Collado-Torres [ctb] (<https://orcid.org/0000-0003-2140-308X>)

Maintainer David Zhang <david.zhang.12@ucl.ac.uk>

R topics documented:

`coverage_norm` *Processing coverage*

Description

The set of functions prefixed with "coverage_" are used to process coverage data. They are designed to be run after you have processed your junctions in the order `coverage_norm`, `coverage_score`. Or, alternatively the wrapper function `coverage_process` can be used to run the 2 functions stated above in one go. For more details of the individual functions, see "Details".

Usage

```
coverage_norm(
  junctions,
  ref,
  unannot_width = 20,
  coverage_paths_case,
  coverage_paths_control,
  coverage_chr_control = NULL,
  load_func = .coverage_load,
  bp_param = BiocParallel::SerialParam(),
  norm_const = 1
)

coverage_process(
  junctions,
```

```
  ref,
  unannot_width = 20,
  coverage_paths_case,
  coverage_paths_control,
  coverage_chr_control = NULL,
  load_func = .coverage_load,
  bp_param = BiocParallel::SerialParam(),
  norm_const = 1,
  score_func = .zscore,
  ...
)

coverage_score(junctions, coverage, score_func = .zscore, ...)
```

Arguments

`junctions`	junction data as a RangedSummarizedExperiment-class object.
`ref`	either path to gtf/gff3 or object of class TxDb-class.
`unannot_width`	integer scalar determining the width of the region to obtain coverage from when the end of of a junction does not overlap an existing exon.
`coverage_paths_case`	paths to the BigWig files containing the coverage of your case samples. Must be the same length and order to the samples in `junctions`.
`coverage_paths_control`	paths to the BigWig files
`coverage_chr_control`	either "chr" or "no_chr", indicating the chromosome format of control coverage data. Only required if the chromosome format of the control BigWig files is different to that of your junctions.
`load_func`	a function to use to load coverage. Currently only for internal use to increase testing speed.
`bp_param`	a BiocParallelParam-class instance denoting whether to parallelise the loading of coverage across BigWig files.
`norm_const`	numeric scaler to add to the normalisation coverage to avoid dividing by 0s and resulting NaN or Inf values.
`score_func`	function to score junctions by their abnormality. By default, will use a z-score but can be switched to a user-defined function. This function must take as input an `x` and `y` argument, containing case and control counts respectively. This must return a numeric vector equal to the length of `x` with elements corresponding to a abnormality of each junction.
`...`	additional arguments passed to `score_func`.
`coverage`	list containing normalised coverage data that is outputted from coverage_norm.

Details

`coverage_process` wraps all "coverage_" prefixed functions in dasper. This is designed to simplify processing of the coverage data for those familiar or uninterested with the intermediates.

coverage_norm obtains regions of interest for each junction where coverage disruptions would be expected. These consist of the intron itself the overlapping exon definitions (if ends of junctions are annotated), picking the shortest exon when multiple overlap one end. If ends are unannotated, coverage_norm will use a user-defined width set by unannot_width. Then, coverage will be loaded using megadepth and normalised to a set region per junction. By default, the boundaries of each gene associated to a junction are used as the region to normalise to.

coverage_score will score disruptions in the coverage across the intronic/exonic regions associated with each junction. This abnormality score generated by score_func operates by calculating the deviation of the coverage in patients to a coverage across the same regions in controls. Then, for each junction it obtains the score of the region with the greatest disruption.

Value

junctions as SummarizedExperiment object with additional assays named "coverage_region" and "coverage_score". "coverage_region" labels the region of greatest disruption (1 = exon_start, 2 = exon_end, 3 = intron) and "coverage_score" contains the abnormality scores of the region with the greatest disruption.

Functions

- coverage_norm: Load and normalise coverage from RNA-sequencing data
- coverage_score: Score coverage by their abnormality

Examples

```
##### Set up txdb #####

# use GenomicState to load txdb (GENCODE v31)
ref <- GenomicState::GenomicStateHub(
    version = "31",
    genome = "hg38",
    filetype = "TxDb"
)[[1]]

##### Set up BigWig #####

# obtain path to example bw on recount2
bw_path <- recount::download_study(
    project = "SRP012682",
    type = "samples",
    download = FALSE
)[[1]]


##### junction_process #####

junctions_processed <- junction_process(
    junctions_example,
    ref,
    types = c("ambig_gene", "unannotated"),
)
```

```
##### coverage_norm #####

coverage_normed <- coverage_norm(
    junctions_processed,
    ref,
    unannot_width = 20,
    coverage_paths_case = rep(bw_path, 2),
    coverage_paths_control = rep(bw_path, 2)
)

##### coverage_score #####

junctions <- coverage_score(junctions_processed, coverage_normed)

##### coverage_process #####

# this wrapper will obtain coverage scores identical to those
# obtained through running the individual wrapped functions shown below
junctions_w_coverage <- coverage_process(
    junctions_processed,
    ref,
    coverage_paths_case = rep(bw_path, 2),
    coverage_paths_control = rep(bw_path, 3)
)

# the two objects are equivalent
all.equal(junctions_w_coverage, junctions, check.attributes = FALSE)
```

`dasper` *dasper: detecting abberant splicing events from RNA-seq data*

Description

Placeholder for package description - to be updated

`junctions_example` *Set of example junctions*

Description

A dataset containing the example junction data for 2 case and 3 control samples outputted from `junction_load`. The junctions have been filtered for only those lying on chromosome 21 or 22.

Usage

```
junctions_example
```

Format

RangedSummarizedExperiment-class object from SummarizedExperiment detailing the counts, co-ordinates of junctions lying on chromosome 21/22 for 2 example samples and 3 controls:

assays matrix with counts for junctions (rows) and 5 samples (cols)

colData example sample metadata

rowRanges GRanges-class object describing the co-ordinates and strand of each junction

Source

generated using data-raw/junctions_example.R

`junction_annot` *Processing junctions*

Description

The set of functions prefixed with "junction_" are used to process junction data. They are designed to be run in a sequential manner in the order `junction_annot`, `junction_filter`, `junction_norm`, `junction_score`. Or, alternatively the wrapper function `junction_process` can be used to run all 4 of the functions stated above in one go. For more details of the individual functions, see "Details".

Usage

```
junction_annot(junctions, ref)

junction_filter(
  junctions,
  count_thresh = c(raw = 5),
  n_samp = c(raw = 1),
  width_range = NULL,
  types = NULL,
  regions = NULL
)

junction_norm(junctions)

junction_process(
  junctions,
  ref,
  count_thresh = c(raw = 5),
  n_samp = c(raw = 1),
  width_range = NULL,
  types = NULL,
  regions = NULL,
  score_func = .zscore,
```

```
  ...
)

junction_score(junctions, score_func = .zscore, ...)
```

Arguments

junctions	junction data as a RangedSummarizedExperiment-class object.
ref	either path to gtf/gff3 or object of class TxDb-class.
count_thresh	named vector with names matching the names of the assays in junctions. Values denote the number of counts below which a junction will be filtered out.
n_samp	named vector with names matching the names of the assays in junctions. Values denotes number of samples that have to express the junction above the count_thresh in order for that junction to not be filtered.
width_range	numeric vector of length 2. The first element denoting the lower limit of junction width and the second the upper limit. Junctions with widths outside this range will be filtered out.
types	any junctions matching these types, derived form junction_annot will be filtered out.
regions	any junctions overlapping this set of regions (in a GRanges-class format) will be filtered out.
score_func	function to score junctions by their abnormality. By default, will use a z-score but can be switched to a user-defined function. This function must take as input an x and y argument, containing case and control counts respectively. This must return a numeric vector equal to the length of x with elements corresponding to a abnormality of each junction.
...	additional arguments passed to score_func.

Details

junction_process wraps all "junction_" prefixed functions in dasper except junction_load. This is designed to simplify processing of the junction data for those familiar or uninterested with the intermediates.

junction_annot annotates junctions by 1. whether their start and/or end position precisely overlaps with an annotated exon boundary and 2. whether that junction matches an intron definition from existing annotation. Using this information along with the strand, junctions are categorised into "annotated", "novel_acceptor", "novel_donor", "novel_combo", "novel_exon_skip", "ambig_gene" and "unannotated".

junction_filter filters out "noisy" junctions based on counts, the width of junctions, annotation category of the junction returned from junction_annot and whether the junction overlaps with a set of (blacklist) regions.

junction_norm normalises the raw junction counts by 1. building junction clusters by finding junctions that share an acceptor or donor position and 2. calculating a proportion-spliced-in (PSI) for each junction by dividing the raw junction count by the total number of counts in it's associated cluster.

`junction_score` will use the counts contained within the "norm" assay to calculate a deviation of each patient junction from the expected distribution of control junction counts. The function used to calculate this abnormality score can be user-inputted or left as the default z-score. Junctions will also be labelled based on whether they are up-regulated (+1) or down-regulated (-1) with respect to controls junction and this information is stored in the assay "direction" for use in outlier_aggregate.

Value

RangedSummarizedExperiment-class object containing filtered, annotated, normalised junction data with abnormality scores.

Functions

- `junction_annot`: Annotate junctions using reference annotation
- `junction_filter`: Filter junctions by count, width, annotation or region
- `junction_norm`: Normalise junction counts by cluster
- `junction_score`: Score patient junctions by their abnormality

See Also

ENCODE blacklist regions are recommended to be included as `regions` for `junction_filter` and can be downloaded from https://github.com/Boyle-Lab/Blacklist/blob/master/lists/hg38-blacklist.v2.bed.gz. Further information can be found via the publication https://www.nature.com/articles/s41598-019-45839-z.

Examples

```
##### Set up txdb #####

# use GenomicState to load txdb (GENCODE v31)
ref <- GenomicState::GenomicStateHub(
    version = "31",
    genome = "hg38",
    filetype = "TxDb"
)[[1]]

##### junction_annot #####

junctions <- junction_annot(junctions_example, ref)

##### junction_filter #####

junctions <- junction_filter(
    junctions,
    types = c("ambig_gene", "unannotated")
)

##### junction_norm #####

junctions <- junction_norm(junctions)
```

```
##### junction_score #####

junctions <- junction_score(junctions)

##### junction_process #####

junctions_processed <- junction_process(
    junctions_example,
    ref,
    types = c("ambig_gene", "unannotated")
)

# the two objects are equivalent
all.equal(junctions_processed, junctions, check.attributes = FALSE)
```

`junction_load` *Load junctions from RNA-sequencing data*

Description

`junction_load` loads in raw patient and control junction data and formats it into a RangedSummarizedExperiment-class object. Control samples can be the user's in-house samples or selected from GTEx v6 data publicly released through the recount2 and downloaded through snaptron. By default, `junction_load` expects the junction data to be in STAR aligned format (SJ.out) but this can be modified via the argument `load_func`.

Usage

```
junction_load(
  junction_paths,
  metadata = dplyr::tibble(samp_id = stringr::str_c("samp_",
    seq_along(junction_paths))),
  controls = rep(FALSE, length(junction_paths)),
  load_func = .STAR_load,
  chrs = NULL,
  coord_system = 1
)
```

Arguments

`junction_paths`	path(s) to junction data.
`metadata`	data.frame containing sample metadata with rows in the same order as `junction_paths`.
`controls`	either a logical vector of the same length as `junction_paths` with TRUE representing controls. Or, one of "fibroblasts", "lymphocytes", "skeletal_muscle", "whole_blood" representing the samples of which GTEx tissue to use as controls. By default, will assume all samples are patients.

load_func	function to load in junctions. By default, requires STAR formatted junctions (SJ.out). But this can be switched dependent on the format of the user's junction data. Function must take as input a junction path then return a data.frame with the columns "chr", "start", "end", "strand" and "count".
chrs	chromosomes to keep. By default, no filter is applied.
coord_system	1 (1-based) or 0 (0-based) denoting the co-ordinate system corresponding to the user junctions from junction_paths. Only used when controls is set to "fibroblasts" to ensure GTEx data is harmonised to match the co-ordinate system of the user's junctions.

Value

RangedSummarizedExperiment-class object containing junction data.

Examples

```
junctions_example_1_path <-
    system.file("extdata",
        "junctions_example_1.txt",
        package = "dasper",
        mustWork = TRUE
    )
junctions_example_2_path <-
    system.file("extdata",
        "junctions_example_2.txt",
        package = "dasper",
        mustWork = TRUE
    )

junctions <-
    junction_load(
        junction_paths = c(junctions_example_1_path, junctions_example_2_path)
    )

junctions
```

---

outlier_aggregate *Processing outliers*

Description

The set of functions prefixed with "outlier_" are used to detect outliers. They are designed to be run after you have extracted your junctions and coverage based features, in the order outlier_detect, outlier_aggregate. Or, alternatively the wrapper function outlier_process can be used to run the 2 functions stated above in one go. For more details of the individual functions, see "Details".

Usage

```
outlier_aggregate(
  junctions,
  samp_id_col = "samp_id",
  bp_param = BiocParallel::SerialParam()
)

outlier_detect(
  junctions,
  feature_names = c("score", "coverage_score"),
  bp_param = BiocParallel::SerialParam(),
  ...
)

outlier_process(
  junctions,
  feature_names = c("score", "coverage_score"),
  samp_id_col = "samp_id",
  bp_param = BiocParallel::SerialParam(),
  ...
)
```

Arguments

junctions	junction data as a RangedSummarizedExperiment-class object.
samp_id_col	name of the column in the SummarizedExperiment that details the sample ids.
bp_param	a BiocParallelParam-class instance denoting whether to parallelise the calculating of outlier scores across samples.
feature_names	names of assays in junctions that are to be used as input into the outlier detection model.
...	additional arguments passed to the outlier detection model (isolation forest) for setting parameters.

Details

outlier_process wraps all "outlier_" prefixed functions in dasper. This is designed to simplify processing of the detecting outlier junctions for those familiar or uninterested with the intermediates.

outlier_detect will use the features in assays named feature_names as input into an unsupervised outlier detection algorithm to score each junction based on how outlier-y it looks in relation to other junctions in the patient. The default expected score and coverage_score features can be calculated using the junction_process and coverage_process respectively.

outlier_aggregate will aggregate the outlier scores into a cluster-level. It will then rank each cluster based on this aggregated score and annotate each cluster with it's associated gene and transcript.

Value

DataFrame with one row per cluster detailing each cluster's associated junctions, outlier scores, ranks and genes.

Functions

- outlier_aggregate: Aggregate outlier scores from per junction to cluster-level
- outlier_detect: Detecting outlier junctions

See Also

for more details on the isolation forest model used: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Isolati

Examples

```
##### Set up txdb #####

# use GenomicState to load txdb (GENCODE v31)
ref <- GenomicState::GenomicStateHub(
    version = "31",
    genome = "hg38",
    filetype = "TxDb"
)[[1]]

##### Set up BigWig #####

# obtain path to example bw on recount2
bw_path <- recount::download_study(
    project = "SRP012682",
    type = "samples",
    download = FALSE
)[[1]]

# cache the bw for speed in later
# examples/testing during R CMD Check
bw_path <- dasper:::.file_cache(bw_path)


##### junction_process #####

junctions_processed <- junction_process(
    junctions_example,
    ref,
    types = c("ambig_gene", "unannotated"),
)

##### coverage_process #####

junctions_w_coverage <- coverage_process(
```

```
    junctions_processed,
    ref,
    coverage_paths_case = rep(bw_path, 2),
    coverage_paths_control = rep(bw_path, 3)
)

##### outlier_detect #####

junctions_w_outliers <- outlier_detect(junctions_w_coverage)

##### outlier_aggregate #####

outlier_scores <- outlier_aggregate(junctions_w_outliers)

##### outlier_process #####

# this wrapper will obtain outlier scores identical to those
# obtained through running the individual wrapped functions shown below
outlier_processed <- outlier_process(junctions_w_coverage)

# the two objects are equivalent
all.equal(outlier_processed, outlier_scores, check.attributes = FALSE)
```

---

plot_sashimi *Visualise RNA-seq data in a the form of a sashimi plot*

Description

plot_sashimi plots the splicing events and coverage over a specific genes/transcript and/or genomic region of interest. The plotting are built from ggplot2 functions.

Usage

```
plot_sashimi(
  junctions,
  ref,
  gene_tx_id,
  case_id = NULL,
  sum_func = mean,
  region = NULL,
  annot_colour = c(ggpubr::get_palette("jco", 1), ggpubr::get_palette("npg", 7)[c(1, 3,
    2, 5, 6)], ggpubr::get_palette("jco", 6)[c(3)]),
  digits = 2,
  count_label = TRUE
)
```

Arguments

junctions	junction data as a RangedSummarizedExperiment-class object.
ref	either path to gtf/gff3 or object of class TxDb-class.
gene_tx_id	gene name as ensembl id, ensembl transcript id or gene name.
case_id	list of one element. This must be a character vector containing the
sum_func	list of one element. This must be a character vector containing the
region	a GenomicRanges of length 1 that is used to filter the exons/junctions. Only those that overlap this region are plotted.
annot_colour	character vector of colours for junction types. One value must be supplied labelling each of the 7 possible types.
digits	used in round, specifying the number of digits to round the junction counts to for visualisation purposes.
count_label	logical value specifying whether to add label the count of each junction.

Value

ggplot displaying the splicing (and coverage) surrounding the transcript/region of interest.

Examples

```
# use GenomicState to load txdb (GENCODE v31)
ref <- GenomicState::GenomicStateHub(
    version = "31",
    genome = "hg38",
    filetype = "TxDb"
)[[1]]

junctions_processed <- junction_process(
    junctions_example,
    ref,
    types = c("ambig_gene", "unannotated")
)

sashimi_plot <- plot_sashimi(
    junctions = junction_filter(junctions_processed),
    ref = ref,
    gene_tx_id = "ENSG00000142156.14",
    sum_func = NULL
)
```

Index