Package 'EvoTraceR'

December 18, 2024

Version 1.0.2

Date 2023-01-09

Title EvoTraceR

Depends R (>= 4.4.0)

Config/reticulate list(

```
packages = list(
list(package = 'numpy', version = '1.21.5', pip = TRUE),
list(package = 'numba', pip = TRUE)
)
)
```

Config/testthat/edition 3

Imports ggplot2,
dplyr,
cli,
Biostrings,
pwalign,
benthos,
scales,
utils,
tidyr,
tibble,
methods,
stringr,
forcats,
data.table,
pheatmap,
ggtree,
plyr,
ape,
foreach,
purrr,
reticulate,
parallel,
doParallel

Suggests BiocGenerics,
BiocStyle,
testthat,
knitr,

Name EvoTraceR

Description Package to identifiy and analyse Amplicon Sequence Variants after performing an experiment through the EVoBC kit.

Encoding UTF-8

LazyData TRUE

License file LICENSE

URL `https://github.com/Nowak-Lab/EvoTraceR`

BugReports `https://github.com/Nowak-Lab/EvoTraceR`

biocViews BiomedicalInformatics,
SingleCell,
SomaticMutation

RoxygenNote 7.3.1

VignetteBuilder knitr

Additional_repositories `https://bioconductor.org/packages/release/bioc`

Contents

`analyse_mutations` *analyse_mutations*

Description

This function considers the pairwise alignment output performed with `asv_analysis` and computes the overall frequency in each sample of each mutation in every barcode position.

Usage

```
analyse_mutations(EvoTraceR_object)
```

Arguments

`EvoTraceR_object`
EvoTraceR object.

Value

EvoTraceR object with the field `alignment`, updated with `mutations_df`: dataframe containing for each position in each ASV the corresponding mutation state and the frequency of that mutation in all sequences.

Examples

```
## Not run:
data(EvoTraceR_analyzed)
EvoTraceR_object = analyse_mutations(EvoTraceR_object)

## End(Not run)
```

`asv_analysis` *ASV_analysis*

Description

This function performs the analysis on ASV sequences identified by the previous steps and aligns them to the reference sequence. First, it pools together those sequences characterized by a Hamming distance equals or lower than 2, summing their counts. To perform this step, EvoTraceR uses the sequence clusering agorithm impemented in the python package UMI-tools Then it performs pairwise alignment using Needleman-Wunsch global alignment algorithm implemented in function `pairwiseAlignment` in package `pwalign`, aligning each sequence to the original barcode considered in the analysis. (See the pwalign documentation here for more details). After identifying all indels (insertions are identified by their start position and number of nucleoides inserted, while deletions are identified by their start and end position), it removes the ones that are too small and don't span any cut site: to perform this filter, it exapnds the start and end position by the number of bases specified in the parameter `cleaning_window`, it counts the number of cut sites spanned after the expantion and it removes those that span 0 sites. Next, it pools together those sequences that differ by one another only by substitutions. Then it computes the normalized counts of each ASV in each sample, dividing each count by the total number of sequences for each sample and multiplying by 1e6, yelding Counts Per Million (CPM). After normalization, counts are summed in each ASV, and those sequences showing a frequency lower than the threshold specified in parameter `asv_count_cutoff` are removed. Finally, it performs filtering of the sequences based on their flanking sequences.

Usage

```
asv_analysis(
  EvoTraceR_object,
  ref_name = "BC10v0",
  ref_seq =
    "TCTACACGCGCGTTCAACCGAGGAAAACTACACACACGTTCAACCACGGTTTTTTACACACGCATTCAACCACGGACT
  ref_flank_left = "^TCTAC",
  ref_flank_right = "CCCGTGGATC$",
  flanking_filtering = "right",
  ref_cut_sites = c(17, 43, 69, 95, 121, 147, 173, 199, 225, 251),
  ref_border_sites = c(1, 26, 52, 78, 104, 130, 156, 182, 208, 234),
  output_figures = TRUE,
  asv_count_cutoff = 2,
  pwa_gapOpening = -25,
  pwa_gapExtension = 0,
  pwa_match = 15,
  pwa_mismatch = -4,
  pwa_type = "global",
  cleaning_window = c(3, 3),
  batch_size = 100,
  cores = parallel::detectCores()
)
```

Arguments

`EvoTraceR_object`	(Required). Object of class EvoTraceR, result of the function `initialize_EvoTraceR`
`ref_name`	String indicating the ID of the reference sequence used in the experiment. Default is 'BC10v0',
`ref_seq`	String indicating the reference sequence used in the experimenti. Default is 'TC-TACACGCGCGTTCAACCGAGGAAAACTACACACACGTTCAACCACGGTTTTTT-TACACACGCATTCAACCACGGACTGCTACACACGCACTCAACCGTGGATATTTACATACTCGTT
`ref_flank_left`	String indicating the first nucleotides of the reference sequence that never mutate over the course of the experiment. Default is "^TCTAC",
`ref_flank_right`	String indicating the first nucleotides of the reference sequence that never mutate over the course of the experiment. Default is "CCCGTGGATC$",
`flanking_filtering`	Which among the flaning regions to use to filter out contaminated sequences. Must be one of c('left', 'right', 'both', 'either'), default is 'both'
`ref_cut_sites`	Positions in the reference sequence of the cutting sites. Default is c(17, 42, 68, 94, 120, 146, 171, 198, 224, 251),
`ref_border_sites`	c(26, 52, 78, 104, 130, 156, 182, 208, 234).
`output_figures`	(Optional). Default TRUE: Boolean indicating whether a user whishes to store a figure indicating the number of ASV tracked during the different steps of the analysis.

`asv_count_cutoff`	(Optional). Default to 2. Minimum number of counts in Counts Per Million (CPM) for an ASV to be considered in the statistics.
`pwa_gapOpening`	(Optional). Default is -25. Parameter `gapOpening` passed to `pairwiseAlignment` from `pwalign` (See description).
`pwa_gapExtension`	(Optional). Default is 0. Parameter `gapExtension` passed to `pairwiseAlignment` from `pwalign` (See description). Default is 0.
`pwa_match`	(Optional). Default is 15. Parameter indicating the score for matches during pairwise alignment with `pwalign`. This parameter, together with the previous one, are used to construct the substitution matrix used by the function `pairwiseAlignment`.
`pwa_mismatch`	(Optional). Default is -4. Parameter indicating the penalty for mismatch events during pairwise alignment with `pwalign`.
`pwa_type`	(Optional). Parameter indicating the type of pairwise alignment. Must be one of One of "global", "local", "overlap", "global-local", and "local-global". For more details see original documentation
`cleaning_window`	(Optional). Default is c(3,3). Vector containing the number of nucleotides that we use for extending respectively the start and end position of each indel to determine the ones that don't span any cut sites and thus get removed. (See description for more information).
`batch_size`	(Optional). Default is 100. Number of batches to split reads into for parallel execution of `pairwiseAlignment`. This parameter can be tuned together with the cores parameter to optimize the speed of alignment.
`cores`	(Optional). Default is parallel::detectCores(). Number of cores to use for pairwise alignment.

Details

Then, using the CPM it computes different statistics for the final ASVS, storing: the relative frequency of all ASVs in each sample. the relative frequency of each ASV in the samples. the counts for each ASV normalized to the counts of the sample with maximum frequency. the frequency of the different ASVs in each sample.

Value

The EvoTraceR object passed as a parameter with the following new fields:

- `clean_asv_dataframe`: ASV sequences identified post-filtering (contamination removed, sequences with a similarity higher than `pid_cutoff_nmbc` to the original barcode aggregated to it and ASVs named in increasing order (ASV01, ASV02, etc.) according to their total counts.
- `reference`: Info about the reference sequence used for the current analysis.
- `statistics`: another list with the following sub-fields:

- `asv_df_percentages`: dataframe with six columns. `asv_names` is the name of the ASV. `sample` is the sample identifier (e.g. ID of an organ or, in case of longitudinal data, of the timepoint); `count`: total counts per million for a specific ASV in a specific sample; `perc_in_sample`: CPM normalized to the total counts in the corresponding sample; `perc_asv`: CPM normalized to the total counts for the corresponding ASV; `perc_fold_to_max`: CPM normalized to the maximum counts observed for the corresponding ASV in a sample.
- `asv_totalCounts`: for each ASV, total counts and number of samples in which it was detected.
- `sample_totalcounts`: for each sample, total counts and number of distinct ASVs detected.
- `asv_diversity_persample`: measures of clonal richness and measures of heterogeneity computed for each sample based on the ASVs detected.
- `asv_persample_frequency`: counts for each ASV in each sample.
- `asv_persample_detection`: binary matrix indicating whether a sequence has been detected in the corresponding sample.
- `asv_toBarcode_similarity`: edit distance, percentage similarity and alignment score of each ASV compared to the original barcode.
- `all_asv_statistics`: all the statistics computed on each ASV grouped together in the same tibble.

• `alignment`, another list with the following fields:
  - `Binary_mutation_matrix`: binary matrix encoding the presence/absence of a mutation in an ASV.
  - `asv_barcode_alignment`: tibble where each line corresponds to a position in a ASV, and the columns encode the following information:
    * asv_names: name of the ASV
    * sample: sample identifier
    * position_bc260: position of the alteration in the original barcode. Note that insertions are assigned to the position that coincides with their beginning.
    * alt: type of alteration. wt = Wild Type (i.e. non-mutated position). sub = substitution. del = deletion. ins = insertion.
    * ref_asv,
    * read_asv: respectively, the reference nucleotide observed in the original barcode and the one observed on the sequence.
  - `mutations_coordinates`: dataframe containing a list of all mutations, with their start and end position.
  - `ASV_alterations_width`: dataframe containing the number of nucleotides affected by each type of mutation in each ASV.

This function saves the figure `asv_filtering_freq.pdf`, which is a barplot indicating the change in the number of sequences during all the processing steps.

Examples

```
## Not run:
data(EvoTraceR_object)
```

```
EvoTraceR_object = asv_analysis(EvoTraceR_object = EvoTraceR_object)

## End(Not run)
```

`create_df_summary` *create_df_summary*

Description

This function combines in one dataframe all the data that have been computed for the phylogenetic analysis.

Usage

```
create_df_summary(EvoTraceR_object)
```

Arguments

`EvoTraceR_object`	(Required).

Value

EvoTraceR_object with a field named plot_summary containing dataframe df_to_plot_final. This dataframe containins all the information computed within the package for each ASV in each sample.

Examples

```
## Not run:
EvoTraceR_object = create_df_summary(EvoTraceR_object)

## End(Not run)
```

`EvoTraceR_object` *example data obtained from running function* `asv_analys`*, where the original barcode is identified and all ASVs are aligned to it.*

Description

example data obtained from running function `asv_analys`, where the original barcode is identified and all ASVs are aligned to it.

Usage

```
data(EvoTraceR_object)
```

Format

EvoTraceR object updated with the following fields:

`clean_asv_dataframe`**:** ASV sequences identified post-filtering (contamination removed, sequences with a similarity higher than `pid_cutoff_nmbc` to the original barcode aggregated to it and ASVs named in increasing order (ASV01, ASV02, etc.) according to their total counts.

`reference`**:** Info about the reference sequence used for the current analysis.

`statistics`**:** another list with the following sub-fields:

- `asv_df_percentages`**:** dataframe with six columns. `asv_names` is the name of the ASV. `sample` is the sample identifier (e.g. ID of an organ or, in case of longitudinal data, of the timepoint); `count`: total counts per million for a specific ASV in a specific sample; `perc_in_sample`: CPM normalized to the total counts in the corresponding sample; `perc_asv`: CPM normalized to the total counts for the corresponding ASV; `perc_fold_to_max`: CPM normalized to the maximum counts observed for the corresponding ASV in a sample.
- `asv_totalCounts`**:** for each ASV, total counts and number of samples in which it was detected.
- `sample_totalcounts`**:** for each sample, total counts and number of distinct ASVs detected.
- `asv_diversity_persample`**:** measures of clonal richness and measures of heterogeneity computed for each sample based on the ASVs detected.
- `asv_persample_frequency`**:** counts for each ASV in each sample.
- `asv_persample_detection`**:** binary matrix indicating whether a sequence has been detected in the corresponding sample.
- `asv_toBarcode_similarity`**:** edit distance, percentage similarity and alignment score of each ASV compared to the original barcode.
- `all_asv_statistics`**:** all the statistics computed on each ASV grouped together in the same tibble.

`alignment`**,** another list with the following fields:

- `Binary_mutation_matrix`**:** binary matrix encoding the presence/absence of a mutation in an ASV.
- `asv_barcode_alignment`**:** tibble where each line corresponds to a position in a ASV, and the columns encode the following information:
  - **asv_names:** name of the ASV
  - **sample:** sample identifier
  - **position_bc260:** position of the alteration in the original barcode. Note that insertions are assigned to the position that coincides with their beginning.
  - **alt:** type of alteration. wt = Wild Type (i.e. non-mutated position). sub = substitution. del = deletion. ins = insertion.
  - **ref_asv and read_asv** : respectively, the reference nucleotide observed in the original barcode and the one observed on the sequence.
- `mutations_coordinates`**:** dataframe containing a list of all mutations, with their start and end position.
- `ASV_alterations_width`**:** dataframe containing the number of nucleotides affected by each type of mutation in each ASV.

`infer_phylogeny` *infer_phylogeny*

Description

This function reconstructs the phylogenetic tree using the Cassiopeia suite. It uses the greedy algorithm, that iteratively splits the set of ASVs in clusters based on the most common mutation at each iteration. Note that this requires installing cassiopeia in the anaconda/virtual enviroment used by reticulate.

Usage

```
infer_phylogeny(EvoTraceR_object, mutations_use = "del_ins")
```

Arguments

`EvoTraceR_object`	(Required)
`mutations_use`	(optional). Default = 'del_ins' A string indicating what type of mutations to use for phylogeny reconstruction. Can be one of c('del', 'del_ins'). The first corresponds to using only cleaned deletions. The second refers to the case where cleaned deletions and insertions are considered.

Value

EvoTraceR object with a new field called phylogeny, that stores:

- `mutations_in_phylogeny`: string indicating which mutations were used for phylogeny reconstruction.
- `tree`: dataframe storing the tree returned b y Cassiopeia. This dataframe has a column "group" that indicates the clonal population to which the node belongs to (a clonal population is a cluster returned by Cassiopeia).
- `tree_phylo`: `phylo` object containing the tree returned by Cassiopeia.
- `tree_uncollapsed`: `phylo` object containing the tree returned by Cassiopeia prior to the collapsing of mutationless edges.

Examples

```
## Not run:
EvoTraceR_object = infer_phylogeny(EvoTraceR_object)

## End(Not run)
```

initialize_EvoTraceR

initialize_EvoTraceR

Description

This function initializes the EvoTraceR object, by computing the set of Amplicon Sequence Variants. It calls trimmomatic and flash to perform adapters trimming, discard low quality bases and merge forward and reverse reads.

Usage

```
initialize_EvoTraceR(
  output_dir,
  trimmomatic_path,
  flash_path,
  input_dir = NULL,
  map_file_sample = NULL,
  sample_order = "alphabetical"
)
```

Arguments

`output_dir`	(Required). Path to the directory where all output files will be stored. This function will output the file `quality_track_reads.csv`, containing a track of the number of sequences during the different steps.
`trimmomatic_path`	(Required). Local path to the executable of `Trimmomatic`. For details about the download please see here.
`flash_path`	(Required). Local path to the executable of `Flash`. For details about the download please see here.
`input_dir`	Path to the directory containing `.fastq` files for forward and reverse reads. This folder should contain the fastq files (2 for each sample) with the following name pattern: FILEPREFIX_SAMPLE_BARCODEVERSION_R1.fastq FILEPREFIX_SAMPLE_BARCODEVERSION_R1.fastq. SAMPLE refers to either an organ (in case multiple organs were sequenced) or timepoint (if longitudinal data are provided). Note that EvoTraceR does not support mixed sample types (i.e. samples must be either all from organs or all from timepoints).
`map_file_sample`	(Optional). In case fastq files names are not in the format FILEPREFIX_SAMPLE_FILESUFFIX_RX.fastq), then users should provide a list that associates each filename (without the suffix _R1 and _R2) to the corresponding organ/day. (e.g., if we have forward and reverse files named file1_R1.fastq and file1_R2.fastq that correspond to organ PRL (code for Prostate Left), than the parameter should be set as: `map_file_sample = c("file1" = "PRL"))`.

`sample_order` (Optional). Vector containing the order in which the user wants samples to appear in all plots. If `NULL` the alphabetical order will be set.

Value

An object of type EvoTraceR, which is a list that will contain the following fields:

- `fastq_directory`: directory where the input fastq files are located.
- `output_directory`: directory where all the output files are being stored.
- `map_file_sample`: dataframe has as many rows as the input datasets, and for each input stores the sample (e.g. organ or day for longitudinal data) to which it is associated.
- `asv_prefilter`: dataframe that stores all sequences detected after these preliminary steps. Note that these sequences still need to be filtered (see also `asv_analysis`).

. This function also saves the `.csv` file `quality_track_reads.csv`: track of the number of sequences during `trimming and merging` steps.

Examples

```
## Not run:
input_dir = system.file("extdata", "input", package = "EvoTraceR")
output_dir = system.file("extdata", "output", package = "EvoTraceR")
EvoTraceR_object = initialize_EvoTraceR(input_dir = input_dir, output_dir = output_dir, trim

## End(Not run)
```

`setup_reticulate` *setup_reticulate*

Description

This function loads reticulate and needs to be run after the installation of the package

Usage

```
setup_reticulate()
```

Index