Package 'scPipe'

May 25, 2024

Title Pipeline for single cell multi-omic data pre-processing

Date 2022-10-12

Version 2.4.0

Type Package

biocViews ImmunoOncology, Software, Sequencing, RNASeq, GeneExpression, SingleCell, Visualization, SequenceMatching, Preprocessing, QualityControl, GenomeAnnotation, DataImport

Description A preprocessing pipeline for single cell RNA-seq/ATAC-seq data that starts from the fastq files and produces a feature count matrix with associated quality control information. It can process fastq data generated by CEL-seq, MARS-seq, Drop-seq, Chromium 10x and SMART-seq protocols.

Depends R (>= 4.2.0), SingleCellExperiment

LinkingTo Rcpp, Rhtslib (>= 1.13.1), zlibbioc, testthat

Imports AnnotationDbi, basilisk, BiocGenerics, biomaRt, Biostrings, data.table, dplyr, DropletUtils, flexmix, GenomicRanges, GenomicAlignments, GGally, ggplot2, glue (>= 1.3.0), grDevices, graphics, hash, IRanges, magrittr, MASS, Matrix (>= 1.5.0), mclust, methods, MultiAssayExperiment, org.Hs.eg.db, org.Mm.eg.db, purrr, Rcpp (>= 0.11.3), reshape, reticulate, Rhtslib, rlang, robustbase, Rsamtools, Rsubread, rtracklayer, SummarizedExperiment, S4Vectors, scales, stats, stringr, tibble, tidyr, tools, utils, vctrs (>= 0.5.2)

SystemRequirements C++11, GNU make

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

URL https://github.com/LuyiTian/scPipe

BugReports https://github.com/LuyiTian/scPipe

Suggests BiocStyle, DT, GenomicFeatures, grid, igraph, kableExtra, knitr, locStra, plotly, rmarkdown, RColorBrewer, readr, reshape2, RANN, shiny, scater (>= 1.11.0), testthat, xml2, umap

VignetteBuilder knitr

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

git_branch RELEASE_3_19

git_last_commit 1f02d19

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-24

Author Luyi Tian [aut],
Shian Su [aut, cre],
Shalin Naik [ctb],
Shani Amarasinghe [aut],
Oliver Voogd [aut],
Phil Yang [aut],
Matthew Ritchie [ctb]

Maintainer Shian Su <su.s@wehi.edu.au>

Contents

.qq_outliers_robust *Detect outliers based on robust linear regression of QQ plot*

Description

Detect outliers based on robust linear regression of QQ plot

Usage

```
.qq_outliers_robust(x, df, conf)
```

Arguments

x	a vector of mahalanobis distance
df	degree of freedom for chi-square distribution
conf	confidence for linear regression

Value

cell names of outliers

anno_import *Import gene annotation*

Description

Because of the variations in data format depending on annotation source, this function has only been tested with human annotation from ENSEMBL, RefSeq and Gencode. If it behaves unexpectedly with any annotation please submit an issue at www.github.com/LuyiTian/scPipe with details.

Usage

```
anno_import(filename)
```

Arguments

filename	The name of the annotation gff3 or gtf file. File can be gzipped.

Details

Imports and GFF3 or GTF gene annotation file and transforms it into a SAF formatted data.frame. SAF described at http://bioinf.wehi.edu.au/featureCounts/. SAF contains positions for exons, strand and the GeneID they are associated with.

Value

data.frame containing exon information in SAF format

Examples

```
ens_chrY <- anno_import(system.file("extdata", "ensembl_hg38_chrY.gtf.gz", package = "scPipe"))
```

`anno_to_saf` *Convert annotation from GenomicRanges to Simple Annotation Format (SAF)*

Description

This function converts a GRanges object into a data.frame of the SAF format for scPipe's consumption. The GRanges object should contain a "type" column where at least some features are annotated as "exon", in addition there should be a gene_id column specifying the gene to which the exon belongs. In the SAF only the gene ID, chromosome, start, end and strand are recorded, this is a gene-exon centric format, with all entries containing the same gene ID treated as exons of that gene. It is possible to count alternative features by setting the gene_id column to an arbitrary feature name and having alternative features in the SAF table, the main caveat is that the features are still treated as exons, and the mapping statistics for exon and intron will not reflect biological exons and introns but rather the annotation features.

Usage

```
anno_to_saf(anno)
```

Arguments

`anno`	The GRanges object containing exon information

Details

Convert a GRanges object containing type and gene_id information into a SAF format data.frame. SAF described at http://bioinf.wehi.edu.au/featureCounts/. SAF contains positions for exons, strand and the GeneID they are associated with.

Value

data.frame containing exon information in SAF format

Examples

```
 ## Not run:
     anno <- system.file("extdata", "ensembl_hg38_chrY.gtf.gz", package = "scPipe")
     saf_chrY <- anno_to_saf(rtracklayer::import(anno))

## End(Not run)
```

`calculate_QC_metrics` *Calculate QC metrics from gene count matrix*

Description

Calculate QC metrics from gene count matrix

Usage

```
calculate_QC_metrics(sce)
```

Arguments

`sce`	a `SingleCellExperiment` object containing gene counts

Details

get QC metrics using gene count matrix. The QC statistics added are

- number_of_genes number of genes detected.
- total_count_per_cell sum of read number after UMI deduplication.
- non_mt_percent 1 - percentage of mitochondrial gene counts. Mitochondrial genes are retrived by GO term GO:0005739
- non_ERCC_percent ratio of exon counts to ERCC counts
- non_ribo_percent 1 - percentage of ribosomal gene counts ribosomal genes are retrived by GO term GO:0005840.

Value

an `SingleCellExperiment` with updated QC metrics

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce <- SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) <- "mmusculus_gene_ensembl"
gene_id_type(sce) <- "ensembl_gene_id"
QC_metrics(sce) <- sc_sample_qc
```

```
demultiplex_info(sce) <- cell_barcode_matching
UMI_dup_info(sce) <- UMI_duplication

# The sample qc data already run through function `calculate_QC_metrics`.
# So we delete these columns and run `calculate_QC_metrics` to get them again:
colnames(colnames(QC_metrics(sce)))
QC_metrics(sce) <- QC_metrics(sce)[,c("unaligned","aligned_unmapped","mapped_to_exon")]
sce = calculate_QC_metrics(sce)
colnames(QC_metrics(sce))
```

`cell_barcode_matching` *cell barcode demultiplex statistics for a small sample scRNA-seq dataset to demonstrate capabilities of scPipe*

Description

This data.frame contains cell barcode demultiplex statistics with several rows:

- barcode_unmatch_ambiguous_mapping is the number of reads that do not match any barcode, but aligned to the genome and mapped to multiple features.
- barcode_unmatch_mapped_to_intron is the number of reads that do not match any barcode, but aligned to the genome and mapped to intron.
- barcode_match is the number of reads that match the cell barcodes
- barcode_unmatch_unaligned is the number of reads that do not match any barcode, and not aligned to the genome
- barcode_unmatch_aligned is the number of reads that do not match any barcode, but aligned to the genome and do not mapped to any feature
- barcode_unmatch_mapped_to_exon is the number of reads that do not match any barcode, but aligned to the genome and mapped to the exon

Format

a data.frame instance, one row per cell.

Value

NULL, but makes a data frame with cell barcode demultiplex statistics

Author(s)

Luyi Tian

Source

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts =as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

demultiplex_info(sce)
```

---

check_barcode_start_position

Check Valid Barcode Start Position

Description

Checks to see if the given barcode start position (`bstart`) is valid for the fastq file. If the found barcode percentage is less than the given `threshold`, a new barcode start position is searched for by checking every position from the start of each read to 10 bases after the `bstart`

Usage

```
check_barcode_start_position(
  fastq,
  barcode_file,
  barcode_file_realname,
  bstart,
  blength,
  search_lines,
  threshold
)
```

Arguments

`fastq`	file containing reads
`barcode_file`	csv file
`barcode_file_realname`	the real name of the csv file
`bstart`	the start position for barcodes in the given reads
`blength`	length of each barcode
`search_lines`	the number of fastq lines to use for the check
`threshold`	the minimum percentage of found barcodes to accept for the program to continue

Value

Boolean; TRUE if program can continue execution, FALSE otherwise.

`convert_geneid` *convert the gene ids of a SingleCellExperiment object*

Description

convert the gene ids of a SingleCellExperiment object

Usage

```
convert_geneid(sce, returns = "external_gene_name", all = TRUE)
```

Arguments

`sce`	a SingleCellExperiment object
`returns`	the gene id which is set as return. Default to be 'external_gene_name' A possible list of attributes can be retrieved using the function `listAttributes` from `biomaRt` package. The commonly used id types are 'external_gene_name', 'ensembl_gene_id' or 'entrezgene'.
`all`	logic. For genes that cannot covert to new gene id, keep them with the old id or delete them. The default is keep them.

Details

convert the gene id of all datas in the SingleCellExperiment object

Value

sce with converted id

Examples

```
# the gene id in example data are `external_gene_name`
# the following example will convert it to `external_gene_name`.
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
head(rownames(sce))
sce = convert_geneid(sce, return="external_gene_name")
head(rownames(sce))
```

create_processed_report

create_processed_report

Description

Create an HTML report summarising pro-processed data. This is an alternative to the more verbose `create_report` that requires only the processed counts and stats folders.

Usage

```
create_processed_report(
  outdir = ".",
  organism,
  gene_id_type,
  report_name = "report"
)
```

Arguments

`outdir`	output folder.
`organism`	the organism of the data. List of possible names can be retrieved using the function 'listDatasets'from 'biomaRt' package. (e.g. 'mmusculus_gene_ensembl' or 'hsapiens_gene_ensembl').
`gene_id_type`	gene id type of the data A possible list of ids can be retrieved using the function 'listAttributes' from 'biomaRt' package. the commonly used id types are 'external_gene_name', 'ensembl_gene_id' or 'entrezgene'.
`report_name`	the name of the report .Rmd and .html files.

Value

file path of the created compiled document.

Examples

```
## Not run:
create_report(
       outdir="output_dir_of_scPipe",
       organism="mmusculus_gene_ensembl",
       gene_id_type="ensembl_gene_id")

## End(Not run)
```

create_report *create_report*

Description

create an HTML report using data generated by proprocessing step.

Usage

```
create_report(
  sample_name,
  outdir,
  r1 = "NA",
  r2 = "NA",
  outfq = "NA",
  read_structure = list(bs1 = 0, bl1 = 0, bs2 = 0, bl2 = 0, us = 0, ul = 0),
  filter_settings = list(rmlow = TRUE, rmN = TRUE, minq = 20, numbq = 2),
  align_bam = "NA",
  genome_index = "NA",
  map_bam = "NA",
  exon_anno = "NA",
  stnd = TRUE,
  fix_chr = FALSE,
  barcode_anno = "NA",
  max_mis = 1,
  UMI_cor = 1,
  gene_fl = FALSE,
  organism,
  gene_id_type
)
```

Arguments

sample_name	sample name
outdir	output folder
r1	file path of read1
r2	file path of read2 default to be NULL
outfq	file path of the output of sc_trim_barcode
read_structure	a list contains read structure configuration. For more help see '?sc_trim_barcode'
filter_settings	a list contains read filter settings for more help see '?sc_trim_barcode'
align_bam	the aligned bam file
genome_index	genome index used for alignment
map_bam	the mapped bam file

exon_anno	the gff exon annotation used. Can have multiple files
stnd	whether to perform strand specific mapping
fix_chr	add ‘chr‘ to chromosome names, fix inconsistent names.
barcode_anno	cell barcode annotation file path.
max_mis	maximum mismatch allowed in barcode. Default to be 1
UMI_cor	correct UMI sequence error: 0 means no correction, 1 means simple correction and merge UMI with distance 1.
gene_fl	whether to remove low abundant gene count. Low abundant is defined as only one copy of one UMI for this gene
organism	the organism of the data. List of possible names can be retrieved using the function ‘listDatasets‘from ‘biomaRt‘ package. (i.e ‘mmusculus_gene_ensembl‘ or ‘hsapiens_gene_ensembl‘)
gene_id_type	gene id type of the data A possible list of ids can be retrieved using the function ‘listAttributes‘ from ‘biomaRt‘ package. the commonly used id types are ‘external_gene_name‘, ‘ensembl_gene_id‘ or ‘entrezgene‘

Value

no return

Examples

```
## Not run:
create_report(sample_name="sample_001",
       outdir="output_dir_of_scPipe",
       r1="read1.fq",
       r2="read2.fq",
       outfq="trim.fq",
       read_structure=list(bs1=-1, bl1=2, bs2=6, bl2=8, us=0, ul=6),
       filter_settings=list(rmlow=TRUE, rmN=TRUE, minq=20, numbq=2),
       align_bam="align.bam",
       genome_index="mouse.index",
       map_bam="aligned.mapped.bam",
       exon_anno="exon_anno.gff3",
       stnd=TRUE,
       fix_chr=FALSE,
       barcode_anno="cell_barcode.csv",
       max_mis=1,
       UMI_cor=1,
       gene_fl=FALSE,
       organism="mmusculus_gene_ensembl",
       gene_id_type="ensembl_gene_id")

## End(Not run)
```

`create_sce_by_dir` *create a SingleCellExperiment object from data folder generated by preprocessing step*

Description

after we run `sc_gene_counting` and finish the preprocessing step. `create_sce_by_dir` can be used to generate the SingleCellExperiment object from the folder that contains gene count matrix and QC statistics. it can also generate the html report based on the gene count and quality control statistics

Usage

```
create_sce_by_dir(
  datadir,
  organism = NULL,
  gene_id_type = NULL,
  pheno_data = NULL,
  report = FALSE
)
```

Arguments

`datadir`	the directory that contains all the data and ‘stat‘ subfolder.
`organism`	the organism of the data. List of possible names can be retrieved using the function ‘listDatasets‘from ‘biomaRt‘ package. (i.e ‘mmusculus_gene_ensembl‘ or ‘hsapiens_gene_ensembl‘)
`gene_id_type`	gene id type of the data A possible list of ids can be retrieved using the function ‘listAttributes‘ from ‘biomaRt‘ package. the commonly used id types are ‘external_gene_name‘, ‘ensembl_gene_id‘ or ‘entrezgene‘
`pheno_data`	the external phenotype data that linked to each single cell. This should be an `AnnotatedDataFrame` object
`report`	whether to generate the html report in the data folder

Details

after we run `sc_gene_counting` and finish the preprocessing step. `create_sce_by_dir` can be used to generate the SingleCellExperiment object from the folder that contains gene count matrix and QC statistics.

Value

a SingleCellExperiment object

Examples

```
## Not run:
# the sce can be created fron the output folder of scPipe
# please refer to the vignettes
sce = create_sce_by_dir(datadir="output_dir_of_scPipe",
    organism="mmusculus_gene_ensembl",
    gene_id_type="ensembl_gene_id")

## End(Not run)
# or directly from the gene count and quality control matrix:
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
dim(sce)
```

---

demultiplex_info *demultiplex_info*

Description

Get or set cell barcode demultiplex results in a SingleCellExperiment object

Usage

```
demultiplex_info(object)

demultiplex_info(object) <- value

demultiplex_info.sce(object)

## S4 method for signature 'SingleCellExperiment'
demultiplex_info(object)

## S4 replacement method for signature 'SingleCellExperiment'
demultiplex_info(object) <- value
```

Arguments

object	A SingleCellExperiment object.
value	Value to be assigned to corresponding object.

Value

a dataframe of cell barcode demultiplex information

A DataFrame of cell barcode demultiplx results.

Author(s)

Luyi Tian

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

demultiplex_info(sce)
```

---

detect_outlier *Detect outliers based on QC metrics*

Description

This algorithm will try to find comp number of components in quality control metrics using a Gaussian mixture model. Outlier detection is performed on the component with the most genes detected. The rest of the components will be considered poor quality cells. More cells will be classified low quality as you increase comp.

Usage

```
detect_outlier(
  sce,
  comp = 1,
  sel_col = NULL,
  type = c("low", "both", "high"),
  conf = c(0.9, 0.99),
  batch = FALSE
)
```

Arguments

sce	a SingleCellExperiment object containing QC metrics.
comp	the number of component used in GMM. Depending on the quality of the experiment.
sel_col	a vector of column names which indicate the columns to use for QC. By default it will be the statistics generated by 'calculate_QC_metrics()'
type	only looking at low quality cells ('low') or possible doublets ('high') or both ('both')
conf	confidence interval for linear regression at lower and upper tails.Usually, this is smaller for lower tail because we hope to pick out more low quality cells than doublets.
batch	whether to perform quality control separately for each batch. Default is FALSE. If set to TRUE then you should have a column called 'batch' in the 'colData(sce)'.

Details

detect outlier using Mahalanobis distances

Value

an updated SingleCellExperiment object with an 'outlier' column in colData

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
# the sample qc data already run through function `calculate_QC_metrics`
# for a new sce please run `calculate_QC_metrics` before `detect_outlier`
sce = detect_outlier(sce)
table(QC_metrics(sce)$outliers)
```

---

feature_info *Get or set* feature_info *from a SingleCellExperiment object*

Description

Get or set feature_info from a SingleCellExperiment object

Usage

```
feature_info(object)

feature_info(object) <- value

feature_info.sce(object)

## S4 method for signature 'SingleCellExperiment'
feature_info(object)

## S4 replacement method for signature 'SingleCellExperiment'
feature_info(object) <- value
```

Arguments

object	A SingleCellExperiment object.
value	Value to be assigned to corresponding object.

Value

a dataframe of feature info for scATAC-seq data

A DataFrame of feature information

Author(s)

Shani Amarasinghe

---

feature_type *Get or set* feature_type *from a SingleCellExperiment object*

Description

Get or set feature_type from a SingleCellExperiment object

Usage

```
feature_type(object)

feature_type(object) <- value

feature_type.sce(object)

## S4 method for signature 'SingleCellExperiment'
feature_type(object)

## S4 replacement method for signature 'SingleCellExperiment'
feature_type(object) <- value
```

Arguments

object	A SingleCellExperiment object.
value	Value to be assigned to corresponding object.

Value

the feature type used in feature counting for scATAC-Seq data

A string representing the feature type

Author(s)

Shani Amarasinghe

---

gene_id_type *Get or set* gene_id_type *from a SingleCellExperiment object*

Description

Get or set gene_id_type from a SingleCellExperiment object

Usage

```
gene_id_type(object)

gene_id_type(object) <- value

gene_id_type.sce(object)

## S4 method for signature 'SingleCellExperiment'
gene_id_type(object)

## S4 replacement method for signature 'SingleCellExperiment'
gene_id_type(object) <- value
```

Arguments

object	A SingleCellExperiment object.
value	Value to be assigned to corresponding object.

Value

the gene id type used by Biomart

gene id type string

Author(s)

Luyi Tian

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

gene_id_type(sce)
```

---

get_chromosomes *Get Chromosomes*

Description

Gets a list of NamedList of chromosomes and the reference length acquired through the bam index file.

Usage

```
get_chromosomes(bam, keep_contigs = "^chr")
```

Arguments

bam	file path to the bam file to get data from
keep_contigs	regular expression used with grepl to filter reference names

Value

a named list where element names are chromosomes reference names and elements are integer lengths

---

get_ercc_anno *Get ERCC annotation table*

Description

Helper function to retrieve ERCC annotation as a dataframe in SAF format

Usage

```
get_ercc_anno()
```

Value

data.frame containing ERCC annotation

Examples

```
ercc_anno <- get_ercc_anno()
```

`get_genes_by_GO` *Get genes related to certain GO terms from biomart database*

Description

Get genes related to certain GO terms from biomart database

Usage

```
get_genes_by_GO(
  returns = "ensembl_gene_id",
  dataset = "mmusculus_gene_ensembl",
  go = NULL
)
```

Arguments

`returns`	the gene id which is set as return. Default to be ensembl id A possible list of attributes can be retrieved using the function `listAttributes` from `biomaRt` package. The commonly used id types are 'external_gene_name', 'ensembl_gene_id' or 'entrezgene'.
`dataset`	Dataset you want to use. List of possible datasets can be retrieved using the function `listDatasets` from `biomaRt` package.
`go`	a vector of GO terms

Details

Get genes related to certain GO terms from biomart database

Value

a vector of gene ids.

Examples

```
# get all genes under GO term GO:0005739 in mouse, return ensembl gene id
get_genes_by_GO(returns="ensembl_gene_id",
    dataset="mmusculus_gene_ensembl",
    go=c('GO:0005739'))
```

get_read_str *Get read structure for particular scRNA-seq protocol*

Description

The supported protocols are:

- CelSeq
- CelSeq2
- DropSeq
- 10x (also called ChromiumV1)

If you know the structure of a specific protocol and would like it supported, please leave a issue post at www.github.com/luyitian/scPipe.

Usage

```
get_read_str(protocol)
```

Arguments

protocol	name of the protocol

Value

list of UMI and Barcode locations for use in other scPipe functions

Examples

```
get_read_str("celseq")
```

organism.sce *Get or set* organism *from a SingleCellExperiment object*

Description

Get or set organism from a SingleCellExperiment object

Usage

```
organism.sce(object)

## S4 method for signature 'SingleCellExperiment'
organism(object)

## S4 replacement method for signature 'SingleCellExperiment'
organism(object) <- value
```

Arguments

object	A SingleCellExperiment object.
value	Value to be assigned to corresponding object.

Value

organism string

Author(s)

Luyi Tian

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

organism(sce)
```

---

plot_demultiplex *plot_demultiplex*

Description

Plot cell barcode demultiplexing result for the SingleCellExperiment. The barcode demultiplexing result is shown using a barplot, with the bars indicating proportions of total reads. Barcode matches and mismatches are summarised along with whether or not the read mapped to the genome. High proportion of genome aligned reads with no barcode match may indicate barcode integration failure.

Usage

```
plot_demultiplex(sce)
```

Arguments

sce	a SingleCellExperiment object

Value

a ggplot2 bar chart

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

plot_demultiplex(sce)
```

---

plot_mapping *Plot mapping statistics for* SingleCellExperiment *object.*

Description

Plot mapping statistics for SingleCellExperiment object.

Usage

```
plot_mapping(sce, sel_col = NULL, percentage = FALSE, dataname = "")
```

Arguments

sce	a SingleCellExperiment object
sel_col	a vector of column names, indicating the columns to use for plot. by default it will be the mapping result.
percentage	TRUE to convert the number of reads to percentage
dataname	the name of this dataset, used as plot title

Value

a ggplot2 object

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

plot_mapping(sce,percentage=TRUE,dataname="sc_sample")
```

plot_QC_pairs *Plot GGAlly pairs plot of QC statistics from* SingleCellExperiment *object*

Description

Plot GGAlly pairs plot of QC statistics from SingleCellExperiment object

Usage

```
plot_QC_pairs(sce, sel_col = NULL)
```

Arguments

sce	a SingleCellExperiment object
sel_col	a vector of column names which indicate the columns to use for plot. By default it will be the statistics generated by 'calculate_QC_metrics()'

Value

a ggplot2 object

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
sce = detect_outlier(sce)

plot_QC_pairs(sce)
```

plot_UMI_dup *Plot UMI duplication frequency*

Description

Plot the UMI duplication frequency.

Usage

```
plot_UMI_dup(sce, log10_x = TRUE)
```

Arguments

sce	a SingleCellExperiment object
log10_x	whether to use log10 scale for x axis

Value

a line chart of the UMI duplication frequency

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

plot_UMI_dup(sce)
```

QC_metrics *Get or set quality control metrics in a SingleCellExperiment object*

Description

Get or set quality control metrics in a SingleCellExperiment object

Usage

```
QC_metrics(object)

QC_metrics(object) <- value

QC_metrics.sce(object)

## S4 method for signature 'SingleCellExperiment'
QC_metrics(object)

## S4 replacement method for signature 'SingleCellExperiment'
QC_metrics(object) <- value
```

Arguments

object	A SingleCellExperiment object.
value	Value to be assigned to corresponding object.

Value

a dataframe of quality control matrics

A DataFrame of quality control metrics.

Author(s)

Luyi Tian

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
QC_metrics(sce) = sc_sample_qc

head(QC_metrics(sce))
```

---

read_cells *Read Cell barcode file*

Description

Read Cell barcode file

Usage

```
read_cells(cells)
```

Arguments

cells	the file path to the barcode file. Assumes one barcode per line or barcode csv. Or, cells can be a comma delimited string of barcodes

Value

a character vector of the provided barcodes

remove_outliers *Remove outliers in* SingleCellExperiment

Description

Removes outliers flagged by detect_outliers()

Usage

```
remove_outliers(sce)
```

Arguments

sce	a SingleCellExperiment object

Value

a SingleCellExperiment object without outliers

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
sce = detect_outlier(sce)
dim(sce)
sce = remove_outliers(sce)
dim(sce)
```

scPipe *scPipe - single cell RNA-seq pipeline*

Description

The scPipe will do cell barcode demultiplexing, UMI deduplication and quality control on fastq data generated from all protocols

Author(s)

Luyi Tian <tian.l@wehi.edu.au>; Shian Su <su.s@wehi.edu.au>

sc_aligning *aligning the demultiplexed FASTQ reads using the Rsubread:align()*

Description

after we run the sc_trim_barcode or sc_atac_trim_barcode to demultiplex the fastq files, we are using this function to align those fastq files to a known reference.

Usage

```
sc_aligning(
  R1,
  R2 = NULL,
  tech = "atac",
  index_path = NULL,
  ref = NULL,
  output_folder = NULL,
  output_file = NULL,
  input_format = "FASTQ",
  output_format = "BAM",
  type = "dna",
  nthreads = 1
)
```

Arguments

R1	a mandatory character vector including names of files that include sequence reads to be aligned. For paired-end reads, this gives the list of files including first reads in each library. File format is FASTQ/FASTA by default.
R2	a character vector, the second fastq file, which is required if the data is paired-end
tech	a character string giving the sequencing technology. Possible value includes "atac" or "rna"
index_path	character string specifying the path/basename of the index files, if the Rsubread genome build is available
ref	a character string specifying the path to reference genome file (.fasta, .fa format)
output_folder	a character string, the name of the output folder
output_file	a character vector specifying names of output files. By default, names of output files are set as the file names provided in R1 added with an suffix string
input_format	a string indicating the input format
output_format	a string indicating the output format
type	type of sequencing data (‘RNA‘ or ‘DNA‘)
nthreads	numeric value giving the number of threads used for mapping.

Value

the file path of the output aligned BAM file

Examples

```
## Not run:
sc_aligning(index_path,
    tech = 'atac',
    R1,
    R2,
    nthreads  = 6)

## End(Not run)
```

`sc_atac_bam_tagging` *BAM tagging*

Description

Demultiplexes the reads

Usage

```
sc_atac_bam_tagging(
  inbam,
  output_folder = NULL,
  bc_length = NULL,
  bam_tags = list(bc = "CB", mb = "OX"),
  nthreads = 1
)
```

Arguments

`inbam`	The input BAM file
`output_folder`	The path of the output folder
`bc_length`	The length of the cellular barcodes
`bam_tags`	The BAM tags
`nthreads`	The number of threads

Details

sc_atac_bam_tagging()

Value

file path of the resultant demmultiplexed BAM file.

Examples

```
r1 <- system.file("extdata", "small_chr21_R1.fastq.gz", package="scPipe")
r2 <- system.file("extdata", "small_chr21_R3.fastq.gz", package="scPipe")
barcode_fastq <- system.file("extdata", "small_chr21_R2.fastq.gz", package="scPipe")
out <- tempdir()

sc_atac_trim_barcode(r1=r1, r2=r2, bc_file=barcode_fastq, output_folder=out)

demux_r1 <- file.path(out, "demux_completematch_small_chr21_R1.fastq.gz")
demux_r2 <- file.path(out, "demux_completematch_small_chr21_R3.fastq.gz")
reference <- system.file("extdata", "small_chr21.fa", package="scPipe")

aligned_bam <- sc_aligning(ref=reference, R1=demux_r1, R2=demux_r2, nthreads=6, output_folder=out)

out_bam <- sc_atac_bam_tagging(
   inbam = aligned_bam,
   output_folder = out,
   nthreads = 6)
```

---

sc_atac_cell_calling *identifying true vs empty cells*

Description

the methods to call true cells are of various ways. implement (i.e. `filtering` from `scATAC-Pro` as default

Usage

```
sc_atac_cell_calling(
  mat,
  cell_calling = "filter",
  output_folder,
  genome_size = NULL,
  cell_qc_metrics_file = NULL,
  lower = NULL,
  min_uniq_frags = 3000,
  max_uniq_frags = 50000,
  min_frac_peak = 0.3,
  min_frac_tss = 0,
  min_frac_enhancer = 0,
  min_frac_promoter = 0.1,
  max_frac_mito = 0.15
)
```

Arguments

mat	the feature by cell matrix.
cell_calling	the cell calling approach, possible options were "emptydrops" , "cellranger" and "filter". But we opten to using "filter" as it was most robust. "emptydrops" is still an opition for data with large umber of cells.
output_folder	output directory for the cell called matrix.
genome_size	genome size for the data in feature by cell matrix.
cell_qc_metrics_file	quality per barcode file for the barcodes in the matrix if using the `cellranger` or `filter` options.
lower	the lower threshold for the data if using the `emptydrops` function for cell calling.
min_uniq_frags	The minimum number of required unique fragments required for a cell (used for `filter` cell calling)
max_uniq_frags	The maximum number of required unique fragments required for a cell (used for `filter` cell calling)
min_frac_peak	The minimum proportion of fragments in a cell to overlap with a peak (used for `filter` cell calling)
min_frac_tss	The minimum proportion of fragments in a cell to overlap with a tss (used for `filter` cell calling)
min_frac_enhancer	The minimum proportion of fragments in a cell to overlap with a enhancer sequence (used for `filter` cell calling)
min_frac_promoter	The minimum proportion of fragments in a cell to overlap with a promoter sequence (used for `filter` cell calling)
max_frac_mito	The maximum proportion of fragments in a cell that are mitochondrial (used for `filter` cell calling)

Examples

```
## Not run:
sc_atac_cell_calling <- function(mat,
 cell_calling,
 output_folder,
 genome_size    = NULL,
 cell_qc_metrics_file = NULL,
 lower          = NULL)

## End(Not run)
```

sc_atac_create_cell_qc_metrics

generating a file useful for producing the qc plots

Description

uses the peak file and annotation files for features

Usage

```
sc_atac_create_cell_qc_metrics(
  frags_file,
  peaks_file,
  promoters_file,
  tss_file,
  enhs_file,
  output_folder
)
```

Arguments

frags_file	The fragment file
peaks_file	The peak file
promoters_file	The path of the promoter annotation file
tss_file	The path of the tss annotation file
enhs_file	The path of the enhs annotation file
output_folder	The path of the output folder for resultant files

Value

Nothing (Invisible 'NULL')

sc_atac_create_fragments

Generating the popular fragments for scATAC-Seq data

Description

Takes in a tagged and sorted BAM file and outputs the associated fragments in a .bed file

Usage

```
sc_atac_create_fragments(
  inbam,
  output_folder = "",
  min_mapq = 30,
  nproc = 1,
  cellbarcode = "CB",
  chromosomes = "^chr",
  readname_barcode = NULL,
  cells = NULL,
  max_distance = 5000,
  min_distance = 10,
  chunksize = 5e+05
)
```

Arguments

inbam	The tagged, sorted and duplicate-free input BAM file
output_folder	The path of the output folder
min_mapq	: int Minimum MAPQ to retain fragment
nproc	: int, optional Number of processors to use. Default is 1.
cellbarcode	: str Tag used for cell barcode. Default is CB (used by cellranger)
chromosomes	: str, optional Regular expression used to match chromosome names to include in the output file. Default is "(?i)^chr" (starts with "chr", case-insensitive). If None, use all chromosomes in the BAM file.
readname_barcode	: str, optional Regular expression used to match cell barocde stored in read name. If None (default), use read tags instead. Use "[^:]*" to match all characters before the first colon (":").
cells	: str File containing list of cell barcodes to retain. If None (default), use all cell barcodes found in the BAM file.
max_distance	: int, optional Maximum distance between integration sites for the fragment to be retained. Allows filtering of implausible fragments that likely result from incorrect mapping positions. Default is 5000 bp.
min_distance	: int, optional Minimum distance between integration sites for the fragment to be retained. Allows filtering implausible fragments that likely result from incorrect mapping positions. Default is 10 bp.
chunksize	: int Number of BAM entries to read through before collapsing and writing fragments to disk. Higher chunksize will use more memory but will be faster.

Value

returns NULL

sc_atac_create_report *HTML report generation*

Description

Generates a HTML report using the output folder produced by the pipeline

Usage

```
sc_atac_create_report(
  input_folder,
  output_folder = NULL,
  organism = NULL,
  sample_name = NULL,
  feature_type = NULL,
  n_barcode_subset = 500
)
```

Arguments

input_folder	The path of the folder produced by the pipeline
output_folder	The path of the output folder to store the HTML report in
organism	A string indicating the name of the organism being analysed
sample_name	A string indicating the name of the sample
feature_type	A string indicating the type of the feature (‘genome_bin‘ or ‘peak‘)
n_barcode_subset	if you require only to visualise stats for a sample of barcodes to improve processing time (integer)

Value

the path of the output file

sc_atac_create_sce *sc_atac_create_sce()*

Description

sc_atac_create_sce()

Usage

```
sc_atac_create_sce(
  input_folder = NULL,
  organism = NULL,
  sample_name = NULL,
  feature_type = NULL,
  pheno_data = NULL,
  report = FALSE
)
```

Arguments

input_folder	The output folder produced by the pipeline
organism	The type of the organism
sample_name	The name of the sample
feature_type	The type of the feature
pheno_data	The pheno data
report	Whether or not a HTML report should be produced

Value

a SingleCellExperiment object created from the scATAC-Seq data provided

Examples

```
## Not run:
sc_atac_create_sce(
   input_folder = input_folder,
   organism = "hg38",
   feature_type = "peak",
   report = TRUE)

## End(Not run)
```

---

sc_atac_emptydrops_cell_calling

empty drops cell calling

Description

The empty drops cell calling method

Usage

```
sc_atac_emptydrops_cell_calling(mat, output_folder, lower = NULL)
```

Arguments

mat	The input matrix
output_folder	The path of the output folder
lower	The lower threshold for the data if using the emptydrops function for cell calling.

---

sc_atac_feature_counting

generating the feature by cell matrix

Description

feature matrix is created using a given demultiplexed BAM file and a selected feature type

Usage

```
sc_atac_feature_counting(
  fragment_file,
  feature_input = NULL,
  bam_tags = list(bc = "CB", mb = "OX"),
  feature_type = "peak",
  organism = "hg38",
  cell_calling = "filter",
  sample_name = "",
  genome_size = NULL,
  promoters_file = NULL,
  tss_file = NULL,
  enhs_file = NULL,
  gene_anno_file = NULL,
  pheno_data = NULL,
  bin_size = NULL,
  yieldsize = 1e+06,
  n_filter_cell_counts = 200,
  n_filter_feature_counts = 10,
  exclude_regions = FALSE,
  excluded_regions_filename = NULL,
  output_folder = NULL,
  fix_chr = "none",
  lower = NULL,
  min_uniq_frags = 3000,
  max_uniq_frags = 50000,
  min_frac_peak = 0.3,
  min_frac_tss = 0,
  min_frac_enhancer = 0,
  min_frac_promoter = 0.1,
  max_frac_mito = 0.15,
```

```
  create_report = FALSE
)
```

Arguments

fragment_file	The fragment file
feature_input	The feature input data e.g. the .narrowPeak file for peaks of a bed file format
bam_tags	The BAM tags
feature_type	The type of feature
organism	The organism type (contains hg19, hg38, mm10)
cell_calling	The desired cell calling method; either cellranger, emptydrops or filter.
sample_name	The sample name to identify which is the data is analysed for.
genome_size	The size of the genome (used for the cellranger cell calling method)
promoters_file	The path of the promoter annotation file (if the specified organism isn't recognised).
tss_file	The path of the tss annotation file (if the specified organism isn't recognised).
enhs_file	The path of the enhs annotation file (if the specified organism isn't recognised).
gene_anno_file	The path of the gene annotation file (gtf or gff3 format).
pheno_data	The phenotypic data as a data frame
bin_size	The size of the bins
yieldsize	The yield size
n_filter_cell_counts	An integer value to filter the feature matrix on the number of reads per cell (default = 200)
n_filter_feature_counts	An integer value to filter the feature matrix on the number of reads per feature (default = 10).
exclude_regions	Whether or not the regions (specified in the file) should be excluded
excluded_regions_filename	The filename of the file containing the regions to be excluded
output_folder	The output folder
fix_chr	Whether chr should be fixed or not
lower	the lower threshold for the data if using the emptydrops function for cell calling
min_uniq_frags	The minimum number of required unique fragments required for a cell (used for filter cell calling)
max_uniq_frags	The maximum number of required unique fragments required for a cell (used for filter cell calling)
min_frac_peak	The minimum proportion of fragments in a cell to overlap with a peak (used for filter cell calling)
min_frac_tss	The minimum proportion of fragments in a cell to overlap with a tss (used for filter cell calling)

`min_frac_enhancer`
: The minimum proportion of fragments in a cell to overlap with a enhancer sequence (used for `filter` cell calling)

`min_frac_promoter`
: The minimum proportion of fragments in a cell to overlap with a promoter sequence (used for `filter` cell calling)

`max_frac_mito`
: The maximum proportion of fragments in a cell that are mitochondrial (used for `filter` cell calling)

`create_report`
: Logical value to say whether to create the report or not (default = TRUE).

Value

None (invisible 'NULL')

Examples

```
## Not run:
sc_atac_feature_counting(
   fragment_file = fragment_file,
   cell_calling = "filter",
   exclude_regions = TRUE,
   feature_input = feature_file)

## End(Not run)
```

---

sc_atac_filter_cell_calling

filter cell calling

Description

specify various qc cutoffs to select the desired cells

Usage

```
sc_atac_filter_cell_calling(
  mtx,
  cell_qc_metrics_file,
  min_uniq_frags = 0,
  max_uniq_frags = 50000,
  min_frac_peak = 0.05,
  min_frac_tss = 0,
  min_frac_enhancer = 0,
  min_frac_promoter = 0,
  max_frac_mito = 0.2
)
```

Arguments

mtx	The input matrix
cell_qc_metrics_file	A file containing qc statistics for each cell
min_uniq_frags	The minimum number of required unique fragments required for a cell
max_uniq_frags	The maximum number of required unique fragments required for a cell
min_frac_peak	The minimum proportion of fragments in a cell to overlap with a peak
min_frac_tss	The minimum proportion of fragments in a cell to overlap with a tss
min_frac_enhancer	The minimum proportion of fragments in a cell to overlap with a enhancer sequence
min_frac_promoter	The minimum proportion of fragments in a cell to overlap with a promoter sequence
max_frac_mito	The maximum proportion of fragments in a cell that are mitochondrial

---

sc_atac_peak_calling *sc_atac_peak_calling()*

Description

sc_atac_peak_calling()

Usage

```
sc_atac_peak_calling(
  inbam,
  ref = NULL,
  genome_size = NULL,
  output_folder = NULL
)
```

Arguments

inbam	The input tagged, sorted, duplicate-free input BAM file
ref	The reference genome file
genome_size	The size of the genome
output_folder	The path of the output folder

Value

None (invisible 'NULL')

Examples

```
## Not run:
sc_atac_peak_calling(
    inbam,
    reference)

## End(Not run)
```

---

sc_atac_pipeline *A convenient function for running the entire pipeline*

Description

A convenient function for running the entire pipeline

Usage

```
sc_atac_pipeline(
  r1,
  r2,
  bc_file,
  valid_barcode_file = "",
  id1_st = -0,
  id1_len = 16,
  id2_st = 0,
  id2_len = 16,
  rmN = TRUE,
  rmlow = TRUE,
  organism = NULL,
  reference = NULL,
  feature_type = NULL,
  remove_duplicates = FALSE,
  samtools_path = NULL,
  genome_size = NULL,
  bin_size = NULL,
  yieldsize = 1e+06,
  exclude_regions = TRUE,
  excluded_regions_filename = NULL,
  fix_chr = "none",
  lower = NULL,
  cell_calling = "filter",
  promoters_file = NULL,
  tss_file = NULL,
  enhs_file = NULL,
  gene_anno_file = NULL,
  min_uniq_frags = 3000,
  max_uniq_frags = 50000,
```

```
  min_frac_peak = 0.3,
  min_frac_tss = 0,
  min_frac_enhancer = 0,
  min_frac_promoter = 0.1,
  max_frac_mito = 0.15,
  report = TRUE,
  nthreads = 12,
  output_folder = NULL
)
```

Arguments

r1	The first read fastq file
r2	The second read fastq file
bc_file	the barcode information, can be either in a fastq format (e.g. from 10x-ATAC) or from a .csv file (here the barcode is expected to be on the second column). Currently, for the fastq approach, this can be a list of barcode files.
valid_barcode_file	optional file path of the valid (expected) barcode sequences to be found in the bc_file (.txt, can be txt.gz). Must contain one barcode per line on the second column separated by a comma (default =""). If given, each barcode from bc_file is matched against the barcode of best fit (allowing a hamming distance of 1). If a FASTQ bc_file is provided, barcodes with a higher mapping quality, as given by the fastq reads quality score are prioritised.
id1_st	barcode start position (0-indexed) for read 1, which is an extra parameter that is needed if the bc_file is in a .csv format.
id1_len	barcode length for read 1, which is an extra parameter that is needed if the bc_file is in a .csv format.
id2_st	barcode start position (0-indexed) for read 2, which is an extra parameter that is needed if the bc_file is in a .csv format.
id2_len	barcode length for read 2, which is an extra parameter that is needed if the bc_file is in a .csv format.
rmN	ogical, whether to remove reads that contains N in UMI or cell barcode.
rmlow	logical, whether to remove reads that have low quality barcode sequences.
organism	The name of the organism e.g. hg38
reference	The reference genome file
feature_type	The feature type (either ‘genome_bin‘ or ‘peak‘)
remove_duplicates	Whether or not to remove duplicates (samtools is required)
samtools_path	A custom path of samtools to use for duplicate removal
genome_size	The size of the genome (used for the cellranger cell calling method)
bin_size	The size of the bins for feature counting with the ‘genome_bin‘ feature type
yieldsize	The number of reads to read in for feature counting

exclude_regions
: Whether or not the regions should be excluded

excluded_regions_filename
: The filename of the file containing the regions to be excluded

fix_chr
: Specify 'none', 'exclude_regions', 'feature' or 'both' to prepend the string "chr" to the start of the associated file

lower
: the lower threshold for the data if using the emptydrops function for cell calling.

cell_calling
: The desired cell calling method either cellranger, emptydrops or filter

promoters_file
: The path of the promoter annotation file (if the specified organism isn't recognised)

tss_file
: The path of the tss annotation file (if the specified organism isn't recognised)

enhs_file
: The path of the enhs annotation file (if the specified organism isn't recognised)

gene_anno_file
: The path of the gene annotation file (gtf or gff3 format)

min_uniq_frags
: The minimum number of required unique fragments required for a cell (used for filter cell calling)

max_uniq_frags
: The maximum number of required unique fragments required for a cell (used for filter cell calling)

min_frac_peak
: The minimum proportion of fragments in a cell to overlap with a peak (used for filter cell calling)

min_frac_tss
: The minimum proportion of fragments in a cell to overlap with a tss (used for filter cell calling)

min_frac_enhancer
: The minimum proportion of fragments in a cell to overlap with a enhancer sequence (used for filter cell calling)

min_frac_promoter
: The minimum proportion of fragments in a cell to overlap with a promoter sequence (used for filter cell calling)

max_frac_mito
: The maximum proportion of fragments in a cell that are mitochondrial (used for filter cell calling)

report
: Whether or not a HTML report should be produced

nthreads
: The number of threads to use for alignment (sc_align) and demultiplexing (sc_atac_bam_tagging)

output_folder
: The path of the output folder

Value

None (invisible 'NULL')

Examples

```
data.folder <- system.file("extdata", package = "scPipe", mustWork = TRUE)
r1      <- file.path(data.folder, "small_chr21_R1.fastq.gz")
r2      <- file.path(data.folder, "small_chr21_R3.fastq.gz")

# Using a barcode fastq file:
```

```
# barcodes in fastq format
barcode_fastq      <- file.path(data.folder, "small_chr21_R2.fastq.gz")

## Not run:
sc_atac_pipeline(
  r1 = r1,
  r2 = r2,
  bc_file = barcode_fastq
)

## End(Not run)
```

`sc_atac_pipeline_quick_test` *A function that tests the pipeline on a small test sample (without duplicate removal)*

Description

A function that tests the pipeline on a small test sample (without duplicate removal)

Usage

```
sc_atac_pipeline_quick_test()
```

Value

None (invisible 'NULL')

`sc_atac_plot_cells_per_feature` *A histogram of the log-number of cells per feature*

Description

A histogram of the log-number of cells per feature

Usage

```
sc_atac_plot_cells_per_feature(sce)
```

Arguments

`sce`	The SingleExperimentObject produced by the sc_atac_create_sce function at the end of the pipeline

Value

returns NULL

`sc_atac_plot_features_per_cell`

A histogram of the log-number of features per cell

Description

A histogram of the log-number of features per cell

Usage

```
sc_atac_plot_features_per_cell(sce)
```

Arguments

`sce`	The SingleExperimentObject produced by the sc_atac_create_sce function at the end of the pipeline

Value

returns NULL

`sc_atac_plot_features_per_cell_ordered`

Plot showing the number of features per cell in ascending order

---

Description

Plot showing the number of features per cell in ascending order

Usage

```
sc_atac_plot_features_per_cell_ordered(sce)
```

Arguments

`sce`	The SingleExperimentObject produced by the sc_atac_create_sce function at the end of the pipeline

Value

returns NULL

sc_atac_plot_fragments_cells_per_feature

A scatter plot of the log-number of fragments and log-number of cells per feature

Description

A scatter plot of the log-number of fragments and log-number of cells per feature

Usage

```
sc_atac_plot_fragments_cells_per_feature(sce)
```

Arguments

sce	The SingleExperimentObject produced by the sc_atac_create_sce function at the end of the pipeline

Value

returns NULL

sc_atac_plot_fragments_features_per_cell

A scatter plot of the log-number of fragments and log-number of features per cell

Description

A scatter plot of the log-number of fragments and log-number of features per cell

Usage

```
sc_atac_plot_fragments_features_per_cell(sce)
```

Arguments

sce	The SingleExperimentObject produced by the sc_atac_create_sce function at the end of the pipeline

Value

returns NULL

`sc_atac_plot_fragments_per_cell`
A histogram of the log-number of fragments per cell

Description

A histogram of the log-number of fragments per cell

Usage

```
sc_atac_plot_fragments_per_cell(sce)
```

Arguments

`sce`	The SingleExperimentObject produced by the sc_atac_create_sce function at the end of the pipeline

Value

returns NULL

`sc_atac_plot_fragments_per_feature`
A histogram of the log-number of fragments per feature

Description

A histogram of the log-number of fragments per feature

Usage

```
sc_atac_plot_fragments_per_feature(sce)
```

Arguments

`sce`	The SingleExperimentObject produced by the sc_atac_create_sce function at the end of the pipeline

Value

returns NULL

sc_atac_remove_duplicates *Removing PCR duplicates using samtools*

Description

Takes in a BAM file and removes the PCR duplicates using the samtools markdup function. Requires samtools 1.10 or newer for statistics to be generated.

Usage

```
sc_atac_remove_duplicates(inbam, samtools_path = NULL, output_folder = NULL)
```

Arguments

inbam	The tagged, sorted and duplicate-free input BAM file
samtools_path	The path of the samtools executable (if a custom installation is to be specified)
output_folder	The path of the output folder

Value

file path to a bam file created from samtools markdup

sc_atac_tfidf *generating the UMAPs for sc-ATAC-Seq preprocessed data*

Description

Takes the binary matrix and generate a TF-IDF so the clutering can take place on the reduced dimentions.

Usage

```
sc_atac_tfidf(binary.mat, output_folder = NULL)
```

Arguments

binary.mat	The final, filtered feature matrix in binary format
output_folder	The path of the output folder

Value

None (invisible 'NULL')

Examples

```
## Not run:
sc_atac_tfidf(binary.mat = final_binary_matrix)

## End(Not run)
```

`sc_atac_trim_barcode` *demultiplex raw single-cell ATAC-Seq fastq reads*

Description

single-cell data need to be demultiplexed in order to retain the information of the cell barcodes the data belong to. Here we reformat fastq files so barcode/s (and if available the UMI sequences) are moved from the sequence into the read name. Since scATAC-Seq data are mostly paired-end, both 'r1' and 'r2' are demultiplexed in this function.

Usage

```
sc_atac_trim_barcode(
  r1,
  r2,
  bc_file = NULL,
  valid_barcode_file = "",
  output_folder = "",
  umi_start = 0,
  umi_length = 0,
  umi_in = "both",
  rmN = FALSE,
  rmlow = FALSE,
  min_qual = 20,
  num_below_min = 2,
  id1_st = -0,
  id1_len = 16,
  id2_st = 0,
  id2_len = 16,
  no_reverse_complement = FALSE
)
```

Arguments

`r1`	read one for pair-end reads.
`r2`	read two for pair-end reads, NULL if single read.
`bc_file`	the barcode information, can be either in a `fastq` format (e.g. from 10x-ATAC) or from a `.csv` file (here the barcode is expected to be on the second column). Currently, for the fastq approach, this can be a list of barcode files.

`valid_barcode_file`
: optional file path of the valid (expected) barcode sequences to be found in the bc_file (.txt, can be txt.gz). Must contain one barcode per line on the second column separated by a comma (default =""). If given, each barcode from bc_file is matched against the barcode of best fit (allowing a hamming distance of 1). If a FASTQ `bc_file` is provided, barcodes with a higher mapping quality, as given by the fastq reads quality score are prioritised.

`output_folder`
: the output dir for the demultiplexed fastq file, which will contain fastq files with reformatted barcode and UMI into the read name. Files ending in `.gz` will be automatically compressed.

`umi_start`
: if available, the start position of the molecular identifier.

`umi_length`
: if available, the start position of the molecular identifier.

`umi_in`
: umi_in

`rmN`
: logical, whether to remove reads that contains N in UMI or cell barcode.

`rmlow`
: logical, whether to remove reads that have low quality barcode sequences

`min_qual`
: the minimum base pair quality that is allowed (default = 20).

`num_below_min`
: the maximum number of base pairs below the quality threshold.

`id1_st`
: barcode start position (0-indexed) for read 1, which is an extra parameter that is needed if the `bc_file` is in a `.csv` format.

`id1_len`
: barcode length for read 1, which is an extra parameter that is needed if the `bc_file` is in a `.csv` format.

`id2_st`
: barcode start position (0-indexed) for read 2, which is an extra parameter that is needed if the `bc_file` is in a `.csv` format.

`id2_len`
: barcode length for read 2, which is an extra parameter that is needed if the `bc_file` is in a `.csv` format.

`no_reverse_complement`
: specifies if the reverse complement of the barcode sequence should be used for barcode error correction (only when barcode sequences are provided as fastq files). FALSE (default) lets the function decide whether to use reverse complement, and TRUE forces the function to use the forward barcode sequences.

Value

None (invisible 'NULL')

Examples

```
data.folder <- system.file("extdata", package = "scPipe", mustWork = TRUE)
r1      <- file.path(data.folder, "small_chr21_R1.fastq.gz")
r2      <- file.path(data.folder, "small_chr21_R3.fastq.gz")

# Using a barcode fastq file:

# barcodes in fastq format
barcode_fastq      <- file.path(data.folder, "small_chr21_R2.fastq.gz")
```

```
sc_atac_trim_barcode (
r1            = r1,
r2            = r2,
bc_file       = barcode_fastq,
rmN           = TRUE,
rmlow         = TRUE,
output_folder = tempdir())

# Using a barcode csv file:

# barcodes in .csv format
barcode_1000       <- file.path(data.folder, "chr21_modified_barcode_1000.csv")

## Not run:
sc_atac_trim_barcode (
r1            = r1,
r2            = r2,
bc_file       = barcode_1000,
id1_st        = 0,
rmN           = TRUE,
rmlow         = TRUE,
output_folder = tempdir())

## End(Not run)
```

`sc_correct_bam_bc` *sc_correct_bam_bc*

Description

update the cell barcode tag in bam file with corrected barcode output to a new bam file. the function will be useful for methods that use the cell barcode information from bam file, such as 'Demuxlet'

Usage

```
sc_correct_bam_bc(
  inbam,
  outbam,
  bc_anno,
  max_mis = 1,
  bam_tags = list(am = "YE", ge = "GE", bc = "BC", mb = "OX"),
  mito = "MT",
  nthreads = 1
)
```

Arguments

`inbam`	input bam file. This should be the output of `sc_exon_mapping`
`outbam`	output bam file with updated cell barcode

bc_anno	barcode annotation, first column is cell id, second column is cell barcode sequence
max_mis	maximum mismatch allowed in barcode. (default: 1)
bam_tags	list defining BAM tags where mapping information is stored. • "am": mapping status tag • "ge": gene id • "bc": cell barcode tag • "mb": molecular barcode tag
mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
nthreads	number of threads to use. (default: 1)

Value

no return

Examples

```
data_dir="celseq2_demo"
barcode_annotation_fn = system.file("extdata", "barcode_anno.csv",
    package = "scPipe")
## Not run:
# refer to the vignettes for the complete workflow
...
sc_correct_bam_bc(file.path(data_dir, "out.map.bam"),
    file.path(data_dir, "out.map.clean.bam"),
    barcode_annotation_fn)
...

## End(Not run)
```

---

sc_count_aligned_bam *sc_count_aligned_bam*

Description

Wrapper to run sc_exon_mapping, sc_demultiplex and sc_gene_counting with a single command

Usage

```
sc_count_aligned_bam(
  inbam,
  outbam,
  annofn,
```

```
  bam_tags = list(am = "YE", ge = "GE", bc = "BC", mb = "OX"),
  bc_len = 8,
  UMI_len = 6,
  stnd = TRUE,
  fix_chr = FALSE,
  outdir,
  bc_anno,
  max_mis = 1,
  mito = "MT",
  has_UMI = TRUE,
  UMI_cor = 1,
  gene_fl = FALSE,
  keep_mapped_bam = TRUE,
  nthreads = 1
)
```

Arguments

inbam	input aligned bam file. can have multiple files as input
outbam	output bam filename
annofn	single string or vector of gff3 annotation filenames, data.frame in SAF format or GRanges object containing complete gene_id metadata column.
bam_tags	list defining BAM tags where mapping information is stored. • "am": mapping status tag • "ge": gene id • "bc": cell barcode tag • "mb": molecular barcode tag
bc_len	total barcode length
UMI_len	UMI length
stnd	TRUE to perform strand specific mapping. (default: TRUE)
fix_chr	TRUE to add 'chr' to chromosome names, MT to chrM. (default: FALSE)
outdir	output folder
bc_anno	barcode annotation, first column is cell id, second column is cell barcode sequence
max_mis	maximum mismatch allowed in barcode. (default: 1)
mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
has_UMI	whether the protocol contains UMI (default: TRUE)
UMI_cor	correct UMI sequencing error: 0 means no correction, 1 means simple correction and merge UMI with distance 1. 2 means merge on both UMI alignment position match.
gene_fl	whether to remove low abundance genes. A gene is considered to have low abundance if only one copy of one UMI is associated with it.
keep_mapped_bam	TRUE if feature mapped bam file should be retained.
nthreads	number of threads to use. (default: 1)

Value

no return

Examples

```
## Not run:
sc_count_aligned_bam(
  inbam = "aligned.bam",
  outbam = "mapped.bam",
  annofn = c("MusMusculus-GRCm38p4-UCSC.gff3", "ERCC92_anno.gff3"),
  outdir = "output",
  bc_anno = "barcodes.csv"
)

## End(Not run)
```

`sc_demultiplex` *sc_demultiplex*

Description

Process bam file by cell barcode, output to outdir/count/[cell_id].csv. the output contains information for all reads that can be mapped to exons. including the gene id, UMI of that read and the distance to transcript end position.

Usage

```
sc_demultiplex(
  inbam,
  outdir,
  bc_anno,
  max_mis = 1,
  bam_tags = list(am = "YE", ge = "GE", bc = "BC", mb = "OX"),
  mito = "MT",
  has_UMI = TRUE,
  nthreads = 1
)
```

Arguments

`inbam`	input bam file. This should be the output of `sc_exon_mapping`
`outdir`	output folder
`bc_anno`	barcode annotation, first column is cell id, second column is cell barcode sequence
`max_mis`	maximum mismatch allowed in barcode. (default: 1)
`bam_tags`	list defining BAM tags where mapping information is stored.

- "am": mapping status tag
- "ge": gene id
- "bc": cell barcode tag
- "mb": molecular barcode tag

mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
has_UMI	whether the protocol contains UMI (default: TRUE)
nthreads	number of threads to use. (default: 1)

Value

no return

Examples

```
data_dir="celseq2_demo"
barcode_annotation_fn = system.file("extdata", "barcode_anno.csv",
    package = "scPipe")
## Not run:
# refer to the vignettes for the complete workflow
...
sc_demultiplex(file.path(data_dir, "out.map.bam"),
    data_dir,
    barcode_annotation_fn,has_UMI=FALSE)
...

## End(Not run)
```

---

sc_demultiplex_and_count

sc_demultiplex_and_count

Description

Wrapper to run sc_demultiplex and sc_gene_counting with a single command

Usage

```
sc_demultiplex_and_count(
  inbam,
  outdir,
  bc_anno,
  max_mis = 1,
  bam_tags = list(am = "YE", ge = "GE", bc = "BC", mb = "OX"),
  mito = "MT",
  has_UMI = TRUE,
```

```
  UMI_cor = 1,
  gene_fl = FALSE,
  nthreads = 1
)
```

Arguments

inbam	input bam file. This should be the output of sc_exon_mapping
outdir	output folder
bc_anno	barcode annotation, first column is cell id, second column is cell barcode sequence
max_mis	maximum mismatch allowed in barcode. (default: 1)
bam_tags	list defining BAM tags where mapping information is stored. • "am": mapping status tag • "ge": gene id • "bc": cell barcode tag • "mb": molecular barcode tag
mito	mitochondrial chromosome name. This should be consistent with the chromosome names in the bam file.
has_UMI	whether the protocol contains UMI (default: TRUE)
UMI_cor	correct UMI sequencing error: 0 means no correction, 1 means simple correction and merge UMI with distance 1. 2 means merge on both UMI alignment position match.
gene_fl	whether to remove low abundance genes. A gene is considered to have low abundance if only one copy of one UMI is associated with it.
nthreads	number of threads to use. (default: 1)

Value

no return

Examples

```
## Not run:
refer to the vignettes for the complete workflow, replace demultiplex and
count with single command:
...
sc_demultiplex_and_count(
   file.path(data_dir, "out.map.bam"),
   data_dir,
   barcode_annotation_fn,
   has_UMI = FALSE
)
...

## End(Not run)
```

sc_detect_bc *sc_detect_bc*

Description

Detect cell barcode and generate the barcode annotation

Usage

```
sc_detect_bc(
  infq,
  outcsv,
  prefix = "CELL_",
  bc_len,
  max_reads = 1e+06,
  min_count = 10,
  number_of_cells = 10000,
  max_mismatch = 1,
  white_list_file = NULL
)
```

Arguments

infq	input fastq file, should be the output file of sc_trim_barcode
outcsv	output barcode annotation
prefix	the prefix of cell name (default: 'CELL_')
bc_len	the length of cell barcode, should be consistent with bl1+bl2 in sc_trim_barcode
max_reads	the maximum of reads processed (default: 1,000,000)
min_count	minimum counts to keep, barcode will be discarded if it has lower count. Default value is 10. This should be set according to max_reads.
number_of_cells	number of cells kept in result. (default: 10000)
max_mismatch	the maximum mismatch allowed. Barcodes within this number will be considered as sequence error and merged. (default: 1)
white_list_file	a file that list all the possible barcodes each row is a barcode sequence. the list for 10x can be found at: https://community.10xgenomics.com/t5/Data-Sharing/List-of-valid-cellular-barcodes/td-p/527 (default: NULL)

Value

no return

Examples

```
## Not run:
# `sc_detect_bc`` should run before `sc_demultiplex` for
# Drop-seq or 10X protocols
sc_detect_bc("input.fastq","output.cell_index.csv",bc_len=8)
sc_demultiplex(...,"output.cell_index.csv")

## End(Not run)
```

`sc_exon_mapping` *sc_exon_mapping*

Description

Map aligned reads to exon annotation. The result will be written into optional fields in bam file with different tags. Following this link for more information regarding to bam file format: http://samtools.github.io/hts-specs

The function can accept multiple bam file as input, if multiple bam file is provided and the 'bc_len' is zero, then the function will use the barcode in the 'barcode_vector' to insert into the 'bc' bam tag. So the length of 'barcode_vector' and the length of 'inbam' should be the same If this is the case then the 'max_mis' argument in 'sc_demultiplex' should be zero. If 'be_len' is larger than zero, then the function will still seek for barcode in fastq headers with given length. In this case each bam file is not treated as from a single cell.

Usage

```
sc_exon_mapping(
  inbam,
  outbam,
  annofn,
  bam_tags = list(am = "YE", ge = "GE", bc = "BC", mb = "OX"),
  bc_len = 8,
  barcode_vector = "",
  UMI_len = 6,
  stnd = TRUE,
  fix_chr = FALSE,
  nthreads = 1
)
```

Arguments

`inbam`	input aligned bam file. can have multiple files as input
`outbam`	output bam filename
`annofn`	single string or vector of gff3 annotation filenames, data.frame in SAF format or GRanges object containing complete gene_id metadata column.

bam_tags	list defining BAM tags where mapping information is stored. • "am": mapping status tag • "ge": gene id • "bc": cell barcode tag • "mb": molecular barcode tag
bc_len	total barcode length
barcode_vector	a list of barcode if each individual bam is a single cell. (default: NULL). The barcode should be of the same length for each cell.
UMI_len	UMI length
stnd	TRUE to perform strand specific mapping. (default: TRUE)
fix_chr	TRUE to add 'chr' to chromosome names, MT to chrM. (default: FALSE)
nthreads	number of threads to use. (default: 1)

Value

generates a bam file with exons assigned

Examples

```
data_dir="celseq2_demo"
ERCCanno_fn = system.file("extdata", "ERCC92_anno.gff3",
    package = "scPipe")
## Not run:
# for the complete workflow, refer to the vignettes
...
sc_exon_mapping(file.path(data_dir, "out.aln.bam"),
                file.path(data_dir, "out.map.bam"),
                ERCCanno_fn)
...

## End(Not run)
```

---

sc_gene_counting *sc_gene_counting*

Description

Generate gene counts matrix with UMI deduplication

Usage

```
sc_gene_counting(outdir, bc_anno, UMI_cor = 2, gene_fl = FALSE)
```

Arguments

outdir	output folder containing sc_demultiplex output
bc_anno	barcode annotation comma-separated-values, first column is cell id, second column is cell barcode sequence
UMI_cor	correct UMI sequencing error: 0 means no correction, 1 means simple correction and merge UMI with distance 1. 2 means merge on both UMI alignment position match.
gene_fl	whether to remove low abundance genes. A gene is considered to have low abundance if only one copy of one UMI is associated with it.

Value

no return

Examples

```
data_dir="celseq2_demo"
barcode_annotation_fn = system.file("extdata", "barcode_anno.csv",
package = "scPipe")
## Not run:
# refer to the vignettes for the complete workflow
...
sc_gene_counting(data_dir, barcode_annotation_fn)
...

## End(Not run)
```

---

sc_get_umap_data *Generates UMAP data from sce object*

Description

Produces a DataFrame containing the UMAP dimensions, as well as all the colData of the sce object for each cell

Usage

```
sc_get_umap_data(sce, n_neighbours = 30)
```

Arguments

sce	The SingleCellExperiment object
n_neighbours	No. of neighbours for KNN

Value

A dataframe containing the UMAP dimensions, as well as all the colData of the sce object for each cell

sc_integrate *Integrate multi-omic scRNA-Seq and scATAC-Seq data into a MultiAssayExperiment*

Description

Generates an integrated SCE object with scRNA-Seq and scATAC-Seq data produced by the scPipe pipelines

Usage

```
sc_integrate(
  sce_list,
  barcode_match_file = NULL,
  sce_column_to_barcode_files = NULL,
  rev_comp = NULL,
  cell_line_info = NULL,
  output_folder = NULL
)
```

Arguments

sce_list	A list of SCE objects, named with the corresponding technologies
barcode_match_file	A .csv file with columns corresponding to the barcodes for each tech
sce_column_to_barcode_files	A list of files containing the barcodes for each tech (if not needed then give a 'NULL' entry)
rev_comp	A named list of technologies and logical flags specifying if reverse complements should be applyed for sequences (if not needed then provide a 'NULL' entry)
cell_line_info	A list of files, each of which contains 2 columns corresponding to the barcode and cell line for each tech (if not needed then provide a 'NULL' entry)
output_folder	The path to the output folder

Value

Returns a MultiAssayExperiment containing the scRNA-Seq and scATAC-Seq data produced by the scPipe pipelines

Examples

```
## Not run:
sc_integrate(
   sce_list = list("RNA" = sce.rna, "ATAC" = sce.atac),
   barcode_match_file = bc_match_file,
   sce_column_to_barcode_files = list("RNA" = rna_bc_anno, "ATAC" = NULL),
```

```
    rev_comp = list("RNA" = FALSE, "ATAC" = TRUE),
    cell_line_info = list("RNA" = rna_cell_line_info, "ATAC" = atac_cell_line_info,)
    output_folder = output_folder
    )

## End(Not run)
```

`sc_interactive_umap_plot` *Produces an interactive UMAP plot via Shiny*

Description

Can colour the UMAP by any of the colData columns in the SCE object

Usage

```
sc_interactive_umap_plot(sce)
```

Arguments

`sce`	The SingleCellExperiment object

Value

A shiny object which represents the app. Printing the object or passing it to 'shiny::runApp(...)' will run the app.

`sc_mae_plot_umap` *Generates UMAP of multiomic data*

Description

Uses feature count data from multiple experiment objects to produce UMAPs for each assay and then overlay them on the same pair of axes

Usage

```
sc_mae_plot_umap(mae, by = NULL, output_file = NULL)
```

Arguments

`mae`	The MultiAssayExperiment object
`by`	What to colour the points by. Needs to be in colData of all experiments.
`output_file`	The path of the output file

Value

A ggplot2 object representing the UMAP plot

`sc_sample_data` *a small sample scRNA-seq counts dataset to demonstrate capabilities of scPipe*

Description

This data set contains counts for high variable genes for 100 cells. The cells have different cell types. The data contains raw read counts. The cells are chosen randomly from 384 cells and they did not go through quality controls. The rows names are Ensembl gene ids and the columns are cell names, which is the wall position in the 384 plates.

Format

a matrix instance, one row per gene.

Value

NULL, but makes a matrix of count data

Author(s)

Luyi Tian

Source

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

Examples

```
# use the example dataset to perform quality control
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication
sce = detect_outlier(sce)

plot_QC_pairs(sce)
```

`sc_sample_qc` *quality control information for a small sample scRNA-seq dataset to demonstrate capabilities of scPipe.*

Description

This data.frame contains cell quality control information for the 100 cells. For each cell it has:

- unaligned the number of unaligned reads.
- aligned_unmapped the number of reads that aligned to genome but fail to map to any features.
- mapped_to_exon is the number of reads that mapped to exon.
- mapped_to_intron is the number of reads that mapped to intron.
- ambiguous_mapping is the number of reads that mapped to multiple features. They are not considered in the following analysis.
- mapped_to_ERCC is the number of reads that mapped to ERCC spike-in controls.
- mapped_to_MT is the number of reads that mapped to mitochondrial genes.
- total_count_per_cell is the number of reads that mapped to exon after UMI deduplication. In contrast, 'mapped_to_exon' is the number of reads mapped to exon before UMI deduplication.
- number_of_genes is the number of genes detected for each cells
- non_ERCC_percent is 1 - (percentage of ERCC reads). Reads are UMI deduplicated.
- non_mt_percent is 1 - (percentage of mitochondrial reads). Reads are UMI deduplicated.
- non_ribo_percent is 1- (percentage of ribosomal reads). Reads are UMI deduplicated.

Format

a data.frame instance, one row per cell.

Value

NULL, but makes a data frame with cell quality control data.frame

Author(s)

Luyi Tian

Source

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
head(QC_metrics(sce))
plot_mapping(sce,percentage=TRUE,dataname="sc_sample")
```

---

sc_trim_barcode *sc_trim_barcode*

Description

Reformat fastq files so barcode and UMI sequences are moved from the sequence into the read name.

Usage

```
sc_trim_barcode(
  outfq,
  r1,
  r2 = NULL,
  read_structure = list(bs1 = -1, bl1 = 0, bs2 = 6, bl2 = 8, us = 0, ul = 6),
  filter_settings = list(rmlow = TRUE, rmN = TRUE, minq = 20, numbq = 2)
)
```

Arguments

`outfq`	the output fastq file, which reformat the barcode and UMI into the read name. Files ending in `.gz` will be automatically compressed.
`r1`	read one for pair-end reads. This read should contain the transcript.
`r2`	read two for pair-end reads, NULL if single read. (default: NULL)
`read_structure`	a list containing the read structure configuration: • bs1: starting position of barcode in read one. -1 if no barcode in read one. • bl1: length of barcode in read one, if there is no barcode in read one this number is used for trimming beginning of read one. • bs2: starting position of barcode in read two • bl2: length of barcode in read two • us: starting position of UMI • ul: length of UMI
`filter_settings`	A list contains read filter settings:

- rmlow whether to remove the low quality reads.
- rmN whether to remove reads that contains N in UMI or cell barcode.
- minq the minimum base pair quality that we allowed
- numbq the maximum number of base pair that have quality below numbq

Details

Positions used in this function are 0-indexed, so they start from 0 rather than 1. The default read structure in this function represents CEL-seq paired-ended reads. This contains a transcript in the first read, a UMI in the first 6bp of the second read followed by a 8bp barcode. So the read structure will be : `list(bs1=-1, bl1=0, bs2=6, bl2=8, us=0, ul=6)`. `bs1=-1, bl1=0` indicates negative start position and zero length for the barcode on read one, this is used to denote "no barcode" on read one. `bs2=6, bl2=8` indicates there is a barcode in read two that starts at the 7th base with length 8bp. `us=0, ul=6` indicates a UMI from first base of read two and the length in 6bp.

For a typical Drop-seq experiment the read structure will be `list(bs1=-1, bl1=0, bs2=0, bl2=12, us=12, ul=8)`, which means the read one only contains transcript, the first 12bp in read two are cell barcode, followed by a 8bp UMI.

Value

generates a trimmed fastq file named `outfq`

Examples

```
data_dir="celseq2_demo"
## Not run:
# for the complete workflow, refer to the vignettes
...
sc_trim_barcode(file.path(data_dir, "combined.fastq"),
   file.path(data_dir, "simu_R1.fastq"),
   file.path(data_dir, "simu_R2.fastq"))
...

## End(Not run)
```

TF.IDF.custom *Returns the TF-IDF normalised version of a binary matrix*

Description

Returns the TF-IDF normalised version of a binary matrix

Usage

```
TF.IDF.custom(binary.mat, verbose = TRUE)
```

Arguments

binary.mat	The binary matrix
verbose	boolean flag to print status messages

Value

Returns the TF-IDF normalised version of a binary matrix

UMI_duplication *UMI duplication statistics for a small sample scRNA-seq dataset to demonstrate capabilities of scPipe*

Description

This data.frame contains UMI duplication statistics, where the first column is the number of duplication, and the second column is the count of UMIs.

Format

a data.frame instance, one row per cell.

Value

NULL, but makes a data frame with UMI dulication statistics

Author(s)

Luyi Tian

Source

Christin Biben (WEHI). She FACS sorted cells from several immune cell types including B cells, granulocyte and some early progenitors.

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts =as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

head(UMI_dup_info(sce))
```

UMI_dup_info *Get or set UMI duplication results in a SingleCellExperiment object*

Description

Get or set UMI duplication results in a SingleCellExperiment object

Usage

```
UMI_dup_info(object)

UMI_dup_info(object) <- value

UMI_dup_info.sce(object)

## S4 method for signature 'SingleCellExperiment'
UMI_dup_info(object)

## S4 replacement method for signature 'SingleCellExperiment'
UMI_dup_info(object) <- value
```

Arguments

object	A SingleCellExperiment object.
value	Value to be assigned to corresponding object.

Value

a dataframe of cell UMI duplication information

A DataFrame of UMI duplication results.

Author(s)

Luyi Tian

Examples

```
data("sc_sample_data")
data("sc_sample_qc")
sce = SingleCellExperiment(assays = list(counts = as.matrix(sc_sample_data)))
organism(sce) = "mmusculus_gene_ensembl"
gene_id_type(sce) = "ensembl_gene_id"
QC_metrics(sce) = sc_sample_qc
demultiplex_info(sce) = cell_barcode_matching
UMI_dup_info(sce) = UMI_duplication

head(UMI_dup_info(sce))
```

Index