Package ‘sigminer’

October 29, 2021

Title Extract, Analyze and Visualize Mutational Signatures for Genomic Variations

Version 2.1.1

Description Genomic alterations including single nucleotide substitution, copy number alteration, etc. are the major force for cancer initialization and development. Due to the specificity of molecular lesions caused by genomic alterations, we can generate characteristic alteration spectra, called ‘signature’ (Wang, Shixiang, et al. (2020) <DOI:10.1371/journal.pgen.1009557> & Alexandrov, Ludmil B., et al. (2020) <DOI:10.1038/s41586-020-1943-3>). This package helps users to extract, analyze and visualize signatures from genomic alteration records, thus providing new insight into cancer study.

License MIT + file LICENSE

URL https://github.com/ShixiangWang/sigminer

BugReports https://github.com/ShixiangWang/sigminer/issues

Depends R (>= 3.5)

Imports cli (>= 2.0.0), cowplot, data.table, dplyr, furrr (>= 0.2.0), future, ggplot2 (>= 3.3.0), ggpubr, maftools, magrittr, methods, NMF, purrr, Rcpp, rlang (>= 0.1.2), stats, tidyR

Suggests Biobase, Biostrings, BSgenome, BSgenome.Hsapiens.UCSC.hg19, circlize, cluster, coor, digest, GenomicRanges, GenSA, ggalluvial, ggcorrplot, ggfittext, ggplotify, ggrepel, IRanges, knitr, lpSolve, markdown, matrixStats, nls, patchwork, pheatmap, quadprog, R.utils, RColorBrewer, reticulate, rmarkdown, roxygen2, scales, synchronicity, testthat, tibble, UCSCXenaTools, copynumber

LinkingTo Rcpp

VignetteBuilder knitr

biocViews

Encoding UTF-8

LazyData true
RoxygenNote 7.1.1.9001

NeedsCompilation yes

Author Shixiang Wang [aut, cre] (<https://orcid.org/0000-0001-9855-7357>),
Ziyu Tao [aut] (<https://orcid.org/0000-0003-3272-1227>),
Huimin Li [aut] (<https://orcid.org/0000-0003-1683-9057>),
Tao Wu [aut] (<https://orcid.org/0000-0002-8999-9628>),
Xue-Song Liu [aut, ctb] (<https://orcid.org/0000-0002-7736-0077>),
Anand Mayakonda [ctb]

Maintainer Shixiang Wang <w_shixiang@163.com>

Repository CRAN

Date/Publication 2021-10-29 05:40:02 UTC

R topics documented:

add_h_arrow .................................................. 4
add_labels .................................................... 5
bp ............................................................... 7
centromeres.hg19 ........................................... 14
centromeres.hg38 ........................................... 14
centromeres.mm10 .......................................... 15
centromeres.mm9 ........................................... 15
chromsize.hg19 ............................................. 16
chromsize.hg38 ............................................. 16
chromsize.mm10 ............................................ 17
chromsize.mm9 ............................................. 17
CN_features .................................................. 18
CopyNumber-class .......................................... 18
cosine ......................................................... 19
cytobands.hg19 ............................................... 19
cytobands.hg38 ............................................. 20
cytobands.mm10 ............................................ 20
cytobands.mm9 ............................................. 21
enrich_component_strand_bias ............................... 21
get_adj_p ...................................................... 22
get_Aneuploidy_score ....................................... 23
get_bayesian_result ......................................... 25
get_cn_freq_table .......................................... 26
get_cn_ploidy ................................................ 26
get_genome_annotation ..................................... 27
get_groups ................................................... 28
get_group_comparison ....................................... 30
get_intersect_size .......................................... 31
get_pLOH_score ............................................... 32
get_shannon_diversity_index ............................... 33
get_sig_cancer_type_index ................................. 34
get_sig_db .................................................... 35
R topics documented:

get_sig_exposure ......................................................... 37
get_sig_feature_association ........................................... 38
get_sig_rec_similarity .................................................. 39
get_sig_similarity ....................................................... 40
get_tidy_association ..................................................... 42
group_enrichment ........................................................ 43
handle_hyper_mutation ................................................... 45
hello ................................................................. 46
MAF-class ............................................................ 46
output_bootstrap ......................................................... 47
output_fit ............................................................. 48
output_sig ............................................................. 49
output_tally ............................................................ 50
read_copynumber ........................................................ 50
read_copynumber_ascat .................................................. 52
read_copynumber_seqz ................................................... 53
read_maf .............................................................. 53
read_sv_as_rs ........................................................... 54
read_vcf .............................................................. 55
read_xena_variants ...................................................... 56
report_bootstrap_p_value ................................................ 57
same_size_clustering ..................................................... 57
scoring ............................................................... 58
show_catalogue .......................................................... 60
show_cn_circos .......................................................... 61
show_cn_components ..................................................... 62
show_cn_distribution ................................................... 63
show_cn_features ........................................................ 64
show_cn_freq_circos ...................................................... 65
show_cn_group_profile .................................................. 67
show_cn_profile ........................................................ 69
show_cor ............................................................. 70
show_cosmic .......................................................... 71
show_cosmic_sig_profile ................................................ 72
show_groups ........................................................... 73
show_group_comparison .................................................. 74
show_group_distribution ................................................ 76
show_group_enrichment ................................................... 78
show_group_mapping ....................................................... 79
show_sig_bootstrap ....................................................... 81
show_sig_consensusmap .................................................. 85
show_sig_exposure ......................................................... 86
show_sig_feature_corplot ................................................ 87
show_sig_fit ........................................................... 89
show_sig_profile ......................................................... 91
show_sig_profile_heatmap ............................................... 94
show_sig_profile_loop ................................................... 96
sigminer .............................................................. 97
Add Horizontal Arrow with Text Label to a ggplot

Description

Add Horizontal Arrow with Text Label to a ggplot

Usage

```r
add_h_arrow(
  p,
  x,
  y,
  label = "optimal number",
  space = 0.01,
  vjust = 0.3,
  seg_len = 0.1,
  arrow_len = unit(2, "mm"),
  arrow_type = c("closed", "open"),
  font_size = 5,
  font_family = c("serif", "sans", "mono"),
  font_face = c("plain", "bold", "italic")
)
```
Arguments

p  a ggplot.
x  position at x axis.
y  position at y axis.
label  text label.
space  a small space between arrow and text.
vjust  vertical adjustment, set to 0 to align with the bottom, 0.5 for the middle, and 1 (the default) for the top.
seg_len  length of the arrow segment.
arrows  length of the arrow.
arrows_type  type of the arrow.
font_size  font size.
font_family  font family.
font_face  font face.

Value

a ggplot object.

Description

Add text labels to a ggplot object, such as the result from show_sig_profile.

Usage

add_labels(
p, 
x, 
y, 
y_end = NULL, 
n_label = NULL, 
labels = NULL, 
revert_order = FALSE, 
font_size = 5, 
font_family = "serif", 
font_face = c("plain", "bold", "italic"), 
...
)
Arguments

- `p`: a ggplot.
- `x`: position at x axis.
- `y`: position at y axis.
- `y_end`: end position of y axis when `n_label` is set.
- `n_label`: the number of label, when this is set, the position of labels at y axis is auto-generated according to `y` and `y_end`.
- `labels`: text labels or a similarity object from `get_sig_similarity`.
- `revert_order`: if TRUE, revert label order.
- `font_size`: font size.
- `font_family`: font family.
- `font_face`: font face.
- `...`: other parameters passing to `ggplot2::annotate`.

Value

a ggplot object.

Examples

```r
# Load mutational signature
load(system.file("extdata", "toy_mutational_signature.RData", package = "sigminer", mustWork = TRUE)
# Show signature profile
p <- show_sig_profile(sig2, mode = "SBS")

# Method 1
p1 <- add_labels(p,
  x = 0.75, y = 0.3, y_end = 0.9, n_label = 3,
  labels = paste0("text", 1:3)
)
p1

# Method 2
p2 <- add_labels(p,
  x = c(0.15, 0.6, 0.75), y = c(0.3, 0.6, 0.9),
  labels = paste0("text", 1:3)
)
p2

# Method 3
sim <- get_sig_similarity(sig2)
p3 <- add_labels(p,
  x = c(0.15, 0.6, 0.75), y = c(0.25, 0.55, 0.8),
  labels = sim, font_size = 2
)
p3
```
A Best Practice for Signature Extraction and Exposure (Activity) Attribution

Description

These functions are combined to provide a best practice for optimally identifying mutational signatures and attributing their activities (exposures) in tumor samples. They are listed in order to use.

- `bp_extract_signatures()` for extracting signatures.
- `bp_show_survey()` for showing measures change under different signature numbers to help user select optimal signature number. At default, an aggregated score (named score) is generated to suggest the best solution.
- `bp_show_survey2()` for showing simplified signature number survey like `show_sig_number_survey()`.
- `bp_get_sig_obj()` for get a (list of) Signature object which is common used in `sigminer` for analysis and visualization.
- `bp_attribute_activity()` for optimizing signature activities (exposures). NOTE: the activities from extraction step may be better! You can also use `sig_extract` to get optimal NMF result from multiple NMF runs. Besides, you can use `sig_fit` to quantify exposures based on signatures extracted from `bp_extract_signatures()`.
- `bp_extract_signatures_iter()` for extracting signature in a iteration way.
- `bp_cluster_iter_list()` for clustering (hclust with average linkage) iterated signatures to help collapse multiple signatures into one. The result cluster can be visualized by `plot()` or `factoextra::fviz_dend()`.
- `bp_get_clustered_sigs()` for getting clustered (grouped) mean signatures from signature clusters.
- Extra: `bp_get_stats()` for obtaining stats for signatures and samples of a solution. These stats are aggregated (averaged) as the stats for a solution (specific signature number).
- Extra: `bp_get_rank_score()` for obtaining rank score for all signature numbers.

Usage

```r
bp_extract_signatures(
    nmf_matrix,
    range = 2:5,
    n_bootstrap = 20L,
    n_nmf_run = 50,
    RTOL = 0.001,
    min_contribution = 0,
    cores = min(4L, future::availableCores()),
    cores_solution = min(cores, length(range)),
    seed = 123456L,
    handle_hyper_mutation = TRUE,
)```
bp_extract_signatures_iter(
    nmf_matrix,
    range = 2:5,
    sim_threshold = 0.95,
    max_iter = 10L,
    n_bootstrap = 20L,
    n_nmf_run = 50,
    RTOL = 0.001,
    min_contribution = 0,
    cores = min(4L, future::availableCores()),
    cores_solution = min(cores, length(range)),
    seed = 123456L,
    handle_hyper_mutation = TRUE,
    report_integer_exposure = FALSE,
    only_core_stats = nrow(nmf_matrix) > 100,
    cache_dir = file.path(tempdir(), "sigminer_bp"),
    keep_cache = FALSE,
    pynmf = FALSE,
    use_conda = TRUE,
    py_path = "/Users/wsx/anaconda3/bin/python"
)

bp_cluster_iter_list(x, k = NULL, include_final_iteration = TRUE)

bp_get_clustered_sigs(SigClusters, cluster_label)

bp_get_sig_obj(obj, signum = NULL)

bp_get_stats(obj)

bp_get_rank_score(obj)

bp_show_survey2(
    obj,
    x = "signature_number",
    left_y = "silhouette",
    right_y = "L2_error",
    left_name = left_y,
    right_name = right_y,

left_color = "black",
right_color = "red",
left_shape = 16,
right_shape = 18,
shape_size = 4,
highlight = NULL
)

bp_show_survey(
  obj,
  add_score = FALSE,
scales = c("free_y", "free"),
  fixed_ratio = TRUE
)

bp_attribute_activity(
  input,
sample_class = NULL,
nmf_matrix = NULL,
method = c("bt", "stepwise"),
bt_use_prop = FALSE,
return_class = c("matrix", "data.table"),
use_parallel = FALSE,
cache_dir = file.path(tempdir(), "sigminer_attribute_activity"),
keep_cache = FALSE
)

Arguments

nmf_matrix a matrix used for NMF decomposition with rows indicate samples and columns indicate components.

range a numeric vector containing the ranks of factorization to try. Note that duplicates are removed and values are sorted in increasing order. The results are notably returned in this order.

n_bootstrap number of bootstrapped (resampling) catalogs used. When it is 0, the original (input) mutation catalog is used for NMF decomposition, this is not recommended, just for testing, user should not set it to 0.

n_nmf_run number of NMF runs for each bootstrapped or original catalog. At default, in total n_bootstrap x n_nmf_run (i.e. 1000) NMF runs are used for the task.

RTOL a threshold proposed by Nature Cancer paper to control how to filter solutions of NMF. Default is 0.1% (from reference #2), only NMF solutions with KLD (KL deviance) <= 100.1% minimal KLD are kept.

min_contribution a component contribution threshold to filter out small contributed components.

cores number of cpu cores to run NMF.
cores_solution cores for processing solutions, default is equal to argument cores.
seed a random seed to make reproducible result.
handle_hyper_mutation
  default is TRUE, handle hyper-mutant samples.

report_integer_exposure
  if TRUE, report integer signature exposure by bootstrapping technique.

only_core_stats
  if TRUE, only calculate the core stats for signatures and samples.

cache_dir
  a directory for keep temp result files.

keep_cache
  if TRUE, keep cache results.

pynmf
  if TRUE, use Python NMF driver Nimfa. The seed currently is not used by this implementation, so the only way to reproduce your result is setting keep_cache = TRUE.

use_conda
  if TRUE, create an independent conda environment to run NMF.

py_path
  path to Python executable file, e.g. '/Users/wsx/anaconda3/bin/python'. In my test, it is more stable than use_conda=TRUE. You can install the Nimfa package by yourself or set use_conda to TRUE to install required Python environment, and then set this option.

sim_threshold
  a similarity threshold for selecting samples to auto-rerun the extraction procedure (i.e. bp_extract_signatures()), default is 0.95.

max_iter
  the maximum iteration size, default is 10, i.e., at most run the extraction procedure 10 times.

x
  result from bp_extract_signatures_iter() or a list of Signature objects.

k
  an integer sequence specifying the cluster number to get silhouette.

include_final_iteration
  if FALSE, exclude final iteration result from clustering for input from bp_extract_signatures_iter(), not applied if input is a list of Signature objects.

SigClusters
  result from bp_cluster_iter_list().

cluster_label
  cluster labels for a specified cluster number, obtain it from SigClusters$sil_df.

obj
  a ExtractionResult object from bp_extract_signatures().

signum
  a integer vector to extract the corresponding Signature object(s). If it is NULL (default), all will be returned.

left_y
  column name for left y axis.

right_y
  column name for right y axis.

left_name
  label name for left y axis.

right_name
  label name for right y axis.

left_color
  color for left axis.

right_color
  color for right axis.

left_shape
  shape setting.

right_shape
  shape setting.

shape_size
  shape setting.

highlight
  a integer to highlight a x.

add_score
  if FALSE, don’t show score and label optimal points by rank score.
scales one of "free_y" (default) and "free" to control the scales of plot facet.

fixed_ratio if TRUE (default), make the x/y axis ratio fixed.

input result from `bp_extract_signatures()` or a Signature object.

sample_class a named string vector whose names are sample names and values are class labels (i.e. cancer subtype). If it is NULL (the default), treat all samples as one group.

method one of 'bt' (use bootstrap exposure median, from reference #2, the most recommended way in my personal view) or stepwise' (stepwise reduce and update signatures then do signature fitting with last signature sets, from reference #2, the result tends to assign the contribution of removed signatures to the remaining signatures, maybe I misunderstand the paper method? PAY ATTENTION).

bt_use_prop this parameter is only used for bt method to reset low contributing signature activity (relative activity <0.01). If TRUE, use empirical P value calculation way (i.e. proportion, used by reference #2), otherwise a t.test is applied.

return_class string, 'matrix' or 'data.table'.

use_parallel if TRUE, use parallel computation based on furrr package. It can also be an integer for specifying cores.

Details

The signature extraction approach is adopted from reference #1, #2, and the whole best practice is adopted from the pipeline used by reference #3. I implement the whole procedure with R code based on the method description of papers. The code is well organized, tested and documented so user will find it pretty simple and useful. Besides, the structure of the results is very clear to see and also visualize like other approaches provided by sigminer.

Value

It depends on the called function.

Measure Explanation in Survey Plot

The survey plot provides a pretty good way to facilitate the signature number selection. A score measure is calculated as the weighted mean of selected measures and visualized as the first sub-plot. The optimal number is highlighted with red color dot and the best values for each measures are also highlighted with orange color dots. The detail of 6 measures shown in plot are explained as below.

- score - an aggregated score based on rank scores from selected measures below. The higher, the better. When two signature numbers have the same score, the larger signature number is preferred (this is a rare situation, you have to double check other measures).
- silhouette - the average silhouette width for signatures, also named as ASW in reference #2. The signature number with silhouette decreases sharply is preferred.
- distance - the average sample reconstructed cosine distance, the lower value is better.
- error - the average sample reconstructed error calculated with L2 formula (i.e. L2 error). This lower value is better. This measure represents a similar concept like distance above, they are all used to quantify how well sample mutation profiles can be reconstructed from signatures, but distance cares the whole mutation profile similarity while error here cares value difference.
- **pos cor** - the average positive signature exposure correlation coefficient. The lower value is better. This measure is constructed based on my understanding about signatures: mutational signatures are typically treated as independent recurrent patterns, so their activities are less correlated.

- **similarity** - the average similarity within a signature cluster. Like silhouette, the point decreases sharply is preferred. In the practice, results from multiple NMF runs are clustered with "clustering with match" algorithm proposed by reference #2. This value indicates if the signature profiles extracted from different NMF runs are similar.

**Author(s)**

Shixiang Wang w_shixiang@163.com

**References**


**See Also**

See sig_estimate, sig_extract, sig_auto_extract, sigprofiler_extract for other approaches.

**Examples**

data("simulated_catalogs")

# Here I reduce the values for n_bootstrap and n_nmf_run
# for reducing the run time.
# In practice, you should keep default or increase the values
# for better estimation.
#
# The input data here is simulated from 10 mutational signatures

# e1 <- bp_extract_signatures(
# t(simulated_catalogs$set1),
# range = 8:12,
# n_bootstrap = 5,
# n_nmf_run = 10
# )
#
# To avoid computation in examples,
# Here just load the result
# (e1$signature and e1$exposure set to NA to reduce package size)
load(system.file("extdata", "e1.RData", package = "sigminer"))
# See the survey for different signature numbers
# The suggested solution is marked as red dot
# with highest integrated score.
p1 <- bp_show_survey(e1)
p1
# You can also exclude plotting and highlighting the score
p2 <- bp_show_survey(e1, add_score = FALSE)
p2

# You can also plot a simplified version
p3 <- bp_show_survey2(e1, highlight = 10)
p3

# Obtain the suggested solution from extraction result
obj_suggested <- bp_get_sig_obj(e1, e1$suggested)
obj_suggested
# If you think the suggested signature number is not right
# Just pick up the solution you want
obj_s8 <- bp_get_sig_obj(e1, 8)

# Track the reconstructed profile similarity
rec_sim <- get_sig_rec_similarity(obj_s8, t(simulated_catalogs$set1))
rec_sim

# After extraction, you can assign the signatures
# to reference COSMIC signatures
# More see ?get_sig_similarity
sim <- get_sig_similarity(obj_suggested)
# Visualize the match result
if (require(pheatmap)) {
  pheatmap::pheatmap(sim$similarity)
}

# You already got the activities of signatures
# in obj_suggested, however, you can still
# try to optimize the result.
# NOTE: the optimization step may not truly optimize the result!
expo <- bp_attribute_activity(e1, return_class = "data.table")
expo$abs_activity

## Not run:
# Iterative extraction:
# This procedure will rerun extraction step
# for those samples with reconstructed catalog similarity
# lower than a threshold (default is 0.95)
e2 <- bp_extract_signatures_iter(
  t(simulated_catalogs$set1),
  range = 9:11,
  n_bootstrap = 5,
  n_nmf_run = 5,
  sim_threshold = 0.99
)
When the procedure run multiple rounds
# you can cluster the signatures from different rounds by
# the following command
# bp_cluster_iter_list(e2)

## Extra utilities
rank_score <- bp_get_rank_score(e1)
rank_score
stats <- bp_get_stats(e2$iter1)
# Get the mean reconstructed similarity
1 - stats$stats_sample$cosine_distance_mean

## End(Not run)

centromeres.hg19  Location of Centromeres at Genome Build hg19

Description
Location of Centromeres at Genome Build hg19

Format
A data.frame

Source
Generate from UCSC gold path

Examples
data(centromeres.hg19)

centromeres.hg38  Location of Centromeres at Genome Build hg38

Description
Location of Centromeres at Genome Build hg38

Format
A data.frame

Source
Generate from Genome Reference Consortium
Examples
data(centromeres.hg38)

---

**centromeres.mm10**  
*Location of Centromeres at Genome Build mm10*

**Description**

Location of Centromeres at Genome Build mm10

**Format**

A data.frame

**Source**

Generate from [https://hgdownload.soe.ucsc.edu/goldenPath/mm10/database/gap.txt.gz](https://hgdownload.soe.ucsc.edu/goldenPath/mm10/database/gap.txt.gz)

Examples
data(centromeres.mm10)

---

**centromeres.mm9**  
*Location of Centromeres at Genome Build mm9*

**Description**

Location of Centromeres at Genome Build mm9

**Format**

A data.frame

**Source**

Generate from [https://hgdownload.soe.ucsc.edu/goldenPath/mm9/database/](https://hgdownload.soe.ucsc.edu/goldenPath/mm9/database/) with code:

```bash
for i in $(seq 1 19) X Y;
do
    wget https://hgdownload.soe.ucsc.edu/goldenPath/mm9/database/chr${i}_gap.txt.gz
done
```

Examples
data(centromeres.mm9)
chromsize.hg19  
*Chromosome Size of Genome Build hg19*

**Description**
Chromosome Size of Genome Build hg19

**Format**
A data.frame

**Source**
Generate from UCSC gold path

**Examples**
data(chromsize.hg19)

chromsize.hg38  
*Chromosome Size of Genome Build hg38*

**Description**
Chromosome Size of Genome Build hg38

**Format**
A data.frame

**Source**
Generate from UCSC gold path

**Examples**
data(chromsize.hg38)
### chromsize.mm10  
**Chromosome Size of Genome Build mm10**

**Description**
Chromosome Size of Genome Build mm10

**Format**
A data.frame

**Source**
Generate from UCSC gold path [http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chrom.sizes](http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chrom.sizes)

**Examples**
data(chromsize.mm10)

### chromsize.mm9  
**Chromosome Size of Genome Build mm9**

**Description**
Chromosome Size of Genome Build mm9

**Format**
A data.frame

**Source**
Generate from UCSC gold path [http://hgdownload.cse.ucsc.edu/goldenPath/mm9/bigZips/mm9.chrom.sizes](http://hgdownload.cse.ucsc.edu/goldenPath/mm9/bigZips/mm9.chrom.sizes)

**Examples**
data(chromsize.mm9)
**CopyNumber-class**

---

### CN.features

*Classification Table of Copy Number Features Devised by Wang et al. for Method 'W'*

---

#### Description

Classification Table of Copy Number Features Devised by Wang et al. for Method 'W'

#### Format

A `data.table` with "sigminer.features" class name

#### Source

Generate from code under `data_raw/`

#### Examples

```r
data(CN.features)
```

---

### CopyNumber-class

*Class CopyNumber*

---

#### Description

S4 class for storing summarized absolute copy number profile.

#### Slots

- `data` `data.table` of absolute copy number calling.
- `summary.per.sample` `data.table` of copy number variation summary per sample.
- `genome_build` genome build version, should be one of 'hg19' or 'hg38'.
- `genome_measure` Set 'called' will use autosomo called segments size to compute total size for CNA burden calculation, this option is useful for WES and target sequencing. Set 'wg' will autosome size from genome build, this option is useful for WGS, SNP etc..
- `annotation` `data.table` of annotation for copy number segments.
- `dropoff.segs` `data.table` of copy number segments dropped from raw input.
Description
Calculate Cosine Measures

Usage
cosine(x, y)

Arguments
x a numeric vector or matrix with column representing vector to calculate similarity.
y must be same format as x.

Value
a numeric value or matrix.

Examples
x <- c(1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
y <- c(0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0)
z1 <- cosine(x, y)
z1
z2 <- cosine(matrix(x), matrix(y))
z2

cytobands.hg19 Location of Chromosome Cytobands at Genome Build hg19

Description
Location of Chromosome Cytobands at Genome Build hg19

Format
A data.frame

Source
from UCSC

Examples
data(cytobands.hg19)
cytobands.hg38  
Location of Chromosome Cytobands at Genome Build hg38

Description
Location of Chromosome Cytobands at Genome Build hg38

Format
A data.frame

Source
from UCSC

Examples

data(cytobands.hg38)

---

cytobands.mm10  
Location of Chromosome Cytobands at Genome Build mm10

Description
Location of Chromosome Cytobands at Genome Build mm10

Format
A data.frame

Source
from UCSC http://hgdownload.cse.ucsc.edu/goldenpath/mm10/database/cytoBand.txt.gz

Examples

data(cytobands.mm10)
cytobands.mm9  Location of Chromosome Cytobands at Genome Build mm9

Description

Location of Chromosome Cytobands at Genome Build mm9

Format

A data.frame

Source

from UCSC http://hgdownload.cse.ucsc.edu/goldenpath/mm9/database/cytoBand.txt.gz

Examples

data(cytobands.mm9)

enrich_component_strand_bias

Performs Strand Bias Enrichment Analysis for a Given Sample-by-Component Matrix

Description

See sig_tally for examples.

Usage

enrich_component_strand_bias(mat)

Arguments

mat a sample-by-component matrix from sig_tally with strand bias labels "T:" and "B:"

Value

a data.table sorted by p_value.
get_adj_p  

Get Adjust P Values from Group Comparison

Description

Setting `aes(label=..p.adj..)` in `ggpubr::compare_means()` does not show adjust p values. The returned result of this function can be combined with `ggpubr::stat_pvalue_manual()` to fix this problem.

Usage

```r
get_adj_p(
  data, 
  .col, 
  .grp = "Sample", 
  comparisons = NULL, 
  method = "wilcox.test", 
  p.adjust.method = "fdr", 
  p.digits = 3L, 
  ...
)
```

Arguments

- `data` a `data.frame` containing column for groups and column for comparison.
- `col` column name for comparison.
- `grp` column name for groups.
- `comparisons` Default is NULL, use all combination in group column. It can be a list of length-2 vectors. The entries in the vector are either the names of 2 values on the x-axis or the 2 integers that correspond to the index of the groups of interest, to be compared.
- `method` a character string indicating which method to be used for comparing means. It can be `’t.test’, ‘wilcox.test’` etc..
- `p.adjust.method` correction method, default is `’fdr’`. Run `p.adjust.methods` to see all available options.
- `p.digits` how many significant digits are to be used.
- `...` other arguments passed to `ggpubr::compare_means()`

Details

More info see `ggpubr::compare_means()`, `ggpubr::stat_compare_means()` and `stats::p.adjust()`.

Value

a `data.frame` containing comparison result
get_Aneuploidy_score

Source

https://github.com/kassambara/ggpubr/issues/143

Examples

library(ggpubr)
# T-test
stat.test <- compare_means(
  len ~ dose,
  data = ToothGrowth,
  method = "t.test",
  p.adjust.method = "fdr"
)
stat.test
# Create a simple box plot
p <- ggboxplot(ToothGrowth, x = "dose", y = "len")
p

# Add p values
my_comparisons <- list(c("0.5", "1"), c("1", "2"), c("0.5", "2"))
p + stat_compare_means(method = "t.test", comparisons = my_comparisons)

# Try adding adjust p values
# proposed by author of ggpubr
# however it does not work
p + stat_compare_means(aes(label = ..p.adj..), method = "t.test", comparisons = my_comparisons)

# Solution:
# calculate adjust p values and their location
# then use stat_pvalue_manual() function
p_adj <- get_adj_p(ToothGrowth, .col = "len", .grp = "dose")
p_adj
p + stat_pvalue_manual(p_adj, label = "p.adj")

# Show selected comparisons
# Of note, p value is adjusted
# for three comparisons, but only
# two are showed in figure
p_adj <- get_adj_p(ToothGrowth,
  .col = "len", .grp = "dose",
  comparisons = list(c("0.5", "1"), c("1", "2"))
)
p + stat_pvalue_manual(p_adj, label = "p.adj")
get_Aneuploidy_score

Description

This implements a Cohen-Sharir method (see reference) like "Aneuploidy Score" computation. You can read the source code to see how it works. Basically, it follows the logic of Cohen-Sharir method but with some difference in detail implementation. Their results should be counterpart, but with no data validation for now. **Please raise an issue if you find problem/bugs in this function.**

Usage

```r
get_Aneuploidy_score(
  data,
  ploidy_df = NULL,
  genome_build = "hg19",
  rm_black_arms = FALSE
)
```

Arguments

- **data**
  - a CopyNumber object or a `data.frame` containing at least 'chromosome', 'start', 'end', 'segVal', 'sample' these columns.
- **ploidy_df**
  - default is NULL, compute ploidy by segment-size weighted copy number across autosome, see `get_cn_ploidy`. You can also provide a `data.frame` with 'sample' and 'ploidy' columns.
- **genome_build**
  - genome build version, should be 'hg19', 'hg38', 'mm9' or 'mm10'.
- **rm_black_arms**
  - if TRUE, remove short arms of chr13/14/15/21/22 from calculation as documented in reference #3.

Value

A `data.frame`

References

- Logic reference: [https://github.com/quevedor2/aneuploidy_score/](https://github.com/quevedor2/aneuploidy_score/)

Examples

```r
# Load copy number object
load(system.file("extdata", "toy_copynumber.RData",
  package = "sigminer", mustWork = TRUE
))

df <- get_Aneuploidy_score(cn)

df
```
df2 <- get_Aneuploidy_score(cn@data)

df3 <- get_Aneuploidy_score(cn@data,  
ploidy_df = get_cn_ploidy(cn@data)
)

df3

get_bayesian_result  Get Specified Bayesian NMF Result from Run

Description

Sometimes, we may want to use or inspect specified run result from sig_auto_extract. This function is designed for this purpose.

Usage

get_bayesian_result(run_info)

Arguments

run_info  a data.frame with 1 row and two necessary columns Run and file.

Value

alist.

Author(s)

Shixiang Wang

Examples

load(system.file("extdata", "toy_copynumber_tally_W.RData",  
  package = "sigminer", mustWork = TRUE
))

res <- sig_auto_extract(cn_tally_W$nmf_matrix, result_prefix = "Test_copynumber", nrun = 1)

# All run info are stored in res$Raw$summary_run
# Obtain result of run 1
res_run1 <- get_bayesian_result(res$Raw$summary_run[1, ])
get_cn_freq_table  Get CNV Frequency Table

Description
Get CNV Frequency Table

Usage
get_cn_freq_table(data,
  genome_build = "hg19",
  cutoff = 2L,
  resolution_factor = 1L
)

Arguments

data: a CopyNumber object or a data.frame containing at least 'chromosome', 'start', 'end', 'segVal', 'sample' these columns.
genome_build: genome build version, used when data is a data.frame, should be 'hg19' or 'hg38'.
cutoff: copy number value cutoff for splitting data into AMP and DEL. The values equal to cutoff are discarded. Default is 2, you can also set a length-2 vector, e.g. c(2,2).
resolution_factor: an integer to control the resolution. When it is 1 (default), compute frequency in each cytoband. When it is 2, use compute frequency in each half cytoband.

Value
a data.table.

get_cn_ploidy  Get Ploidy from Absolute Copy Number Profile

Description
Get Ploidy from Absolute Copy Number Profile

Usage
get_cn_ploidy(data)
Arguments

data a CopyNumber object or a data.frame containing at least 'chromosome', 'start', 'end', 'segVal' these columns.

Value

a value or a data.table

Examples

# Load copy number object
load(system.file("extdata", "toy_copynumber.RData",
    package = "sigminer", mustWork = TRUE
))

df <- get_cn_ploidy(cn)

df

get_genome_annotation Get Genome Annotation

Description

Get Genome Annotation

Usage

gene_genome_annotation(
    data_type = c("chr_size", "centro_loc", "cytobands", "transcript"),
    chrs = paste0("chr", c(1:22, "X", "Y")),
    genome_build = c("hg19", "hg38", "mm10", "mm9")
)

Arguments

data_type 'chr_size' for chromosome size, 'centro_loc' for location of centromeres, 'cytobands' for location of chromosome cytobands and 'transcript' for location of transcripts.

chrs chromosomes start with ‘chr’

genome_build one of ‘hg19’, ‘hg38’

Value

a data.frame containing annotation data
Examples

```r
df1 <- get_genome_annotation()
df1

df2 <- get_genome_annotation(genome_build = "hg38")
df2

df3 <- get_genome_annotation(data_type = "centro_loc")
df3

df4 <- get_genome_annotation(data_type = "centro_loc", genome_build = "hg38")
df4

df5 <- get_genome_annotation(data_type = "cytobands")
df5

df6 <- get_genome_annotation(data_type = "cytobands", genome_build = "hg38")
df6
```

---

**get_groups**

*Get Sample Groups from Signature Decomposition Information*

**Description**

One of key results from signature analysis is to cluster samples into different groups. This function takes Signature object as input and return the membership in each cluster.

**Usage**

```r
get_groups(
  Signature,
  method = c("consensus", "k-means", "exposure", "samples"),
  n_cluster = NULL,
  match_consensus = TRUE
)
```

**Arguments**

- **Signature**
  - a Signature object obtained either from `sig_extract` or `sig_auto_extract`. Now it can be used to relative exposure result in `data.table` format from `sig_fit`.

- **method**
  - grouping method, more see details, could be one of the following:
    - 'consensus' - returns the cluster membership based on the hierarchical clustering of the consensus matrix, it can only be used for the result obtained by `sig_extract()` with multiple runs using NMF package.
    - 'k-means' - returns the clusters by k-means.
    - 'exposure' - assigns a sample into a group whose signature exposure is dominant.
• 'samples' - returns the cluster membership based on the contribution of
signature to each sample, it can only be used for the result obtained by
`sig_extract()` using `NMF` package.

`n_cluster` only used when the method is 'k-means'.

`match_consensus` only used when the method is 'consensus'. If TRUE, the result will match order
as shown in consensus map.

**Details**

Users may find there are bigger differences between using method 'samples' and 'exposure' but
they use a similar idea to find dominant signature, here goes the reason:

Method 'samples' using data directly from NMF decomposition, this means the two matrix $W$ (basis
matrix or signature matrix) and $H$ (coefficient matrix or exposure matrix) are the results of NMF. For
method 'exposure', it uses the signature exposure loading matrix. In this situation, each signature
represents a number of mutations (alterations) about implementation please see source code of
`sig_extract()` function.

**Value**

a data.table object

**See Also**

`NMF::predict()`, `show_groups`.

**Examples**

```r
# Load copy number prepare object
load(system.file("extdata", "toy_copynumber_tally_W.RData",
    package = "sigminer", mustWork = TRUE))

# Extract copy number signatures
library(NMF)
sig <- sig_extract(cn_tally_W$nmf_matrix, 2,
    nrun = 10)

# Methods 'consensus' and 'samples' are from NMF::predict()
g1 <- get_groups(sig, method = "consensus", match_consensus = TRUE)
g1
g2 <- get_groups(sig, method = "samples")
g2

# Use k-means clustering
g3 <- get_groups(sig, method = "k-means")
g3
```
get_group_comparison  Get Comparison Result between Signature Groups

Description

Compare genotypes/phenotypes based on signature groups (samples are assigned to several groups). For categorical type, calculate fisher p value (using stats::fisher.test) and count table. In larger than 2 by 2 tables, compute p-values by Monte Carlo simulation. For continuous type, calculate anova p value (using stats::aov), summary table and Tukey Honest significant difference (using stats::TukeyHSD). The result of this function can be plotted by show_group_comparison().

Usage

get_group_comparison(
  data,
  col_group,
  cols_to_compare,
  type = "ca",
  NAs = NA,
  verbose = FALSE
)

Arguments

data  a data.frame containing signature groups and genotypes/phenotypes (including categorical and continuous type data) want to analyze. User need to construct this data.frame by him/herself.

col_group  column name of signature groups.

cols_to_compare  column names of genotypes/phenotypes want to summarize based on groups.

type  a character vector with length same as cols_to_compare, 'ca' for categorical type and 'co' for continuous type.

NAs  default is NA, filter NAs for categorical columns. Otherwise a value (either length 1 or length same as cols_to_compare) fill NAs.

verbose  if TRUE, print extra information.

Value

a list contains data, summary, p value etc..

Author(s)

Shixiang Wang w_shixiang@163.com
get_intersect_size

Examples

```r
load(system.file("extdata", "toy_copynumber_signature_by_W.RData", 
    package = "sigminer", mustWork = TRUE 
))

# Assign samples to clusters
groups <- get_groups(sig, method = "k-means")

set.seed(1234)

groups$prob <- rnorm(10)
groups$new_group <- sample(c("1", "2", "3", "4", NA), size = nrow(groups), replace = TRUE)

# Compare groups (filter NAs for categorical columns)
groups.cmp <- get_group_comparison(groups[, -1], 
    col_group = "group", 
    cols_to_compare = c("prob", "new_group"), 
    type = c("co", "ca"), verbose = TRUE 
)

# Compare groups (Set NAs of categorical columns to 'Rest')
groups.cmp2 <- get_group_comparison(groups[, -1], 
    col_group = "group", 
    cols_to_compare = c("prob", "new_group"), 
    type = c("co", "ca"), NAs = "Rest", verbose = TRUE 
)
```

get_intersect_size

Get Overlap Size between Interval x and y

Description

Get Overlap Size between Interval x and y

Usage

```r
get_intersect_size(x.start, x.end, y.start, y.end)
```

Arguments

- `x.start`: start position of interval x.
- `x.end`: start position of interval x.
- `y.start`: start position of interval x.
- `y.end`: start position of interval x.
get_pLOH_score

**Value**

a numeric vector.

**Examples**

```r
o1 <- get_intersect_size(1, 5, 3, 20)
o1
o2 <- get_intersect_size(3, 20, 1, 10)
o2
o3 <- get_intersect_size(c(1, 2, 1), c(10, 4, 6), c(4, 2, 5), c(10, 3, 22))
o3
```

**get_pLOH_score**

*Get proportions of pLOH score from Allele Specific Copy Number Profile*

**Description**

pLOH score represents the genome that displayed LOH.

**Usage**

```r
get_pLOH_score(data, rm_chrs = c("chrX", "chrY"), genome_build = "hg19")
```

**Arguments**

- `data` a CopyNumber object or a `data.frame` containing at least 'chromosome', 'start', 'end', 'segVal', "minor_cn", 'sample' these columns.
- `rm_chrs` chromosomes to be removed in calculation. Default is sex chromosomes (recommended).
- `genome_build` genome build version, should be 'hg19', 'hg38', 'mm9' or 'mm10'.

**Value**

A `data.frame`

**References**

Examples

```r
# Load toy dataset of absolute copynumber profile
load(system.file("extdata", "toy_segTab.RData", 
    package = "sigminer", mustWork = TRUE 
))

set.seed(1234)
segTabs$minor_cn <- sample(c(0, 1), size = nrow(segTabs), replace = TRUE)
cn <- read_copynumber(segTabs, 
    seg_cols = c("chromosome", "start", "end", "segVal"), 
    genome_measure = "wg", complement = TRUE, add_loh = TRUE 
)

df <- get_pLOH_score(cn)
df
df2 <- get_pLOH_score(cn@data)
df2
```

get_shannon_diversity_index

*Get Shannon Diversity Index for Signatures*

**Description**

\[ H = - \sum_{i=1}^{n} p_i \ln(p_i) \]

where \( n \) is the number of signatures identified in the signature with exposure > cutoff, and \( p_i \) is the normalized exposure of the \( i \)th signature with exposure > cutoff. Exposures of signatures were normalized to sum to 1.

**Usage**

`get_shannon_diversity_index(rel_expo, cutoff = 0.001)`

**Arguments**

- `rel_expo` a `data.frame` with numeric columns indicating relative signature exposures for each sample. Typically this data can be obtained from `get_sig_exposure()`.
- `cutoff` a relative exposure cutoff for filtering signatures, default is 0.1%.

**Value**

a `data.frame`
get_sig_cancer_type_index

Obtain Signature Index for Cancer Types

Description

Obtain Signature Index for Cancer Types

Usage

get_sig_cancer_type_index(
  sig_type = c("legacy", "SBS", "DBS", "ID"),
  seq_type = c("WGS", "WES"),
  source = c("PCAWG", "TCGA", "nonPCAWG"),
  keyword = NULL
)

Arguments

sig_type signature type.
seq_type sequencing type.
source data source.
keyword keyword to search in the signature index database.

Value

a list.
get_sig_db

Examples

```r
11 <- get_sig_cancer_type_index()
12 <- get_sig_cancer_type_index(sig_type = "SBS")
13 <- get_sig_cancer_type_index(sig_type = "DBS", source = "PCAWG", seq_type = "WGS")
14 <- get_sig_cancer_type_index(sig_type = "ID")
15 <- get_sig_cancer_type_index(keyword = "breast")
```

get_sig_db

Get Curated Reference Signature Database

Description

Reference mutational signatures and their aetiologies, mainly obtained from COSMIC database (SigProfiler results) and cleaned before saving into sigminer package. You can obtain:

- COSMIC legacy SBS signatures.
- COSMIC v3 SBS signatures.
- COSMIC v3 DBS signatures.
- COSMIC v3 ID (indel) signatures.
- SBS and RS (rearrangement) signatures from Nik lab 2020 Nature Cancer paper.
- RS signatures from BRCA560 and USARC cohorts.
- Copy number signatures from USARC cohort and TCGA.

Usage

```r
get_sig_db(sig_db = "legacy")
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig_db</td>
<td>default 'legacy', it can be 'legacy' (for COSMIC v2 'SBS'), 'SBS', 'DBS', 'ID' and 'TSB' (for COSMIC v3 signatures) for small scale mutations. For more specific details, it can also be 'SBS_hg19', 'SBS_hg38', 'SBS_mm9', 'SBS_mm10', 'DBS_hg19', 'DBS_hg38', 'DBS_mm9', 'DBS_mm10' to use COSMIC v3 reference signatures from Alexandrov, Ludmil B., et al. (2020) (reference #1). In addition, it can be one of &quot;SBS_Nik_lab_Organ&quot;, &quot;RS_Nik_lab_Organ&quot;, &quot;SBS_Nik_lab&quot;, &quot;RS_Nik_lab&quot; to refer reference signatures from Degasperi, Andrea, et al. (2020) (reference #2); &quot;RS_BRCA560&quot;, &quot;RS_USARC&quot; to reference signatures from BRCA560 and USARC cohorts; &quot;CNS_USARC&quot; (40 categories), &quot;CNS_TCGA&quot; (48 categories) to reference copy number signatures from USARC cohort and TCGA. UPDATE, the latest version of reference version can be automatically downloaded and loaded from <a href="https://cancer">https://cancer</a>.</td>
</tr>
</tbody>
</table>
sanger.ac.uk/signatures/downloads/ when a option with latest_prefix is specified (e.g. "latest_SBS_GRCh37"). **Note:** the signature profile for different genome builds are basically same. And specific database (e.g. 'SBS_mm10') contains less signatures than all COSMIC signatures (because some signatures are not detected from Alexandrov, Ludmil B., et al. (2020)). For all available options, check the parameter setting.

**Value**

a list.

**References**


**See Also**

get_sig_similarity, sig_fit and show_cosmic_sig_profile.

**Examples**

```r
s1 <- get_sig_db()
s2 <- get_sig_db("SBS")
s3 <- get_sig_db("DBS")
s4 <- get_sig_db("DBS_mm10")
s5 <- get_sig_db("SBS_Nik_lab")
s6 <- get_sig_db("ID")
s7 <- get_sig_db("RS_BRCA560")
s8 <- get_sig_db("RS_USARC")
s9 <- get_sig_db("RS_Nik_lab")
s10 <- get_sig_db("CNS_USARC")
s11 <- get_sig_db("CNS_TCGA")
```
Description

The expected number of mutations (or copy number segment records) with each signature was determined after a scaling transformation \( V \sim WH = W'H' \) where \( W' = WU' \) and \( H' = UH \). The scaling matrix \( U \) is a KxK diagonal matrix (\( K \) is signature number, \( U' \) is the inverse of \( U \)) with the element corresponding to the L1-norm of column vectors of \( W \) (i.e. the sum of the elements of the vector). As a result, the k-th row vector of the final matrix \( H' \) represents the absolute exposure (activity) of the k-th process across samples (e.g., for SBS, the estimated (or expected) number of mutations generated by the k-th process). Of note, for copy number signatures, only components of feature CN was used for calculating \( H' \).

Usage

```r
get_sig_exposure(
  Signature,
  type = c("absolute", "relative"),
  rel_threshold = 0.01
)
```

Arguments

- **Signature**: a Signature object obtained either from `sig_extract` or `sig_auto_extract`, or just a raw exposure matrix with column representing samples (patients) and row representing signatures.
- **type**: 'absolute' for signature exposure and 'relative' for signature relative exposure.
- **rel_threshold**: only used when type is 'relative', relative exposure less than (\( \leq \)) this value will be set to 0 and thus all signature exposures may not sum to 1. This is similar to this argument in `sig_fit`.

Value

a data.table

Author(s)

Shixiang Wang w_shixiang@163.com

References

get_sig_feature_association

Examples

```r
# Load mutational signature
load(system.file("extdata", "toy_mutational_signature.RData", 
    package = "sigminer", mustWork = TRUE))

# Get signature exposure
expo1 <- get_sig_exposure(sig2)
expo1
expo2 <- get_sig_exposure(sig2, type = "relative")
expo2
```

get_sig_feature_association

Calculate Association between Signature Exposures and Other Features

Description

Association of signature exposures with other features will be performed using one of two procedures: for a continuous association variable (including ordinal variable), correlation is performed; for a binary association variable, samples will be divided into two groups and Mann-Whitney U-test is performed to test for differences in signature exposure medians between the two groups. See `get_tidy_association` for cleaning association result.

Usage

```r
get_sig_feature_association(
    data, 
    cols_to_sigs, 
    cols_to_features, 
    type = "ca", 
    method_co = c("spearman", "pearson", "kendall"), 
    method_ca = stats::wilcox.test, 
    min_n = 0.01, 
    verbose = FALSE, 
    ...
)
```

Arguments

data a data.frame contains signature exposures and other features
cols_to_sigs colnames for signature exposure
cols_to_features colnames for other features
type a character vector containing 'ca' for categorical variable and 'co' for continuous variable, it must have the same length as cols_to_features.
**get_sig_rec_similarity**

method_co  method for continuous variable, default is "spearman", could also be "pearson" and "kendall".

method_ca  method for categorical variable, default is "wilcox.test"

min_n  a minimal fraction (e.g. 0.01) or a integer number (e.g. 10) for filtering some variables with few positive events. Default is 0.01.

verbose  if TRUE, print extra message.

...  other arguments passing to test functions, like cor.test.

Value

a list. For 'co' features, 'measure' means correlation coefficient. For 'ca' features, 'measure' means difference in means of signature exposure.

See Also

get_tidy_association

---

**Description**

See bp_extract_signatures for examples.

**Usage**

get_sig_rec_similarity(Signature, nmf_matrix)

**Arguments**

Signature  a Signature object.

nmf_matrix  a matrix used for NMF decomposition with rows indicate samples and columns indicate components.

**Value**

a data.table.
get_sig_similarity

Description

The reference signatures can be either a Signature object specified by Ref argument or known COSMIC signatures specified by sig_db argument. Two COSMIC databases are used for comparisons - "legacy" which includes 30 signatures, and "SBS" - which includes updated/refined 65 signatures. This function is modified from compareSignatures() in maftools package. **NOTE:** all reference signatures are generated from gold standard tool: SigProfiler.

Usage

```r
get_sig_similarity(
  Signature,
  Ref = NULL,
  sig_db = c("legacy", "SBS", "DBS", "ID", "TSB", "SBS_Nik_lab", "RS_Nik_lab", "RS_BRCA560", "RS_USARC", "CNS_USARC", "CNS_TCGA", "SBS_hg19", "SBS_hg38", "SBS_mm9", "SBS_mm10", "DBS_hg19", "DBS_hg38", "DBS_mm9", "DBS_mm10", "SBS_Nik_lab_Organ", "RS_Nik_lab_Organ", "latest_SBS_GRCh37", "latest_DBS_GRCh37", "latest_ID_GRCh37", "latest_SBS_GRCh38", "latest_DBS_GRCh38", "latest_SBS_mm9", "latest_DBS_mm9", "latest_SBS_mm10", "latest_DBS_mm10", "latest_SBS_rn6", "latest_DBS_rn6"),
  db_type = c("", "human-exome", "human-genome"),
  method = "cosine",
  normalize = c("row", "feature"),
  feature_setting = sigminer::CN.features,
  set_order = TRUE,
  pattern_to_rm = NULL,
  verbose = TRUE
)
```

Arguments

- **Signature**
  - a Signature object or a component-by-signature matrix/data.frame (sum of each column is 1) or a normalized component-by-sample matrix/data.frame (sum of each column is 1). More please see examples.
- **Ref**
  - default is NULL, can be a same object as Signature.
- **sig_db**
  - default 'legacy', it can be 'legacy' (for COSMIC v2 'SBS'), 'SBS', 'DBS', 'ID' and 'TSB' (for COSMIV v3.1 signatures) for small scale mutations. For more specific details, it can also be 'SBS_hg19', 'SBS_hg38', 'SBS_mm9', 'SBS_mm10', 'DBS_hg19', 'DBS_hg38', 'DBS_mm9', 'DBS_mm10' to use COSMIC v3 reference signatures from Alexandrov, Ludmil B., et al. (2020) (reference #1). In addition, it can be one of "SBS_Nik_lab_Organ", "RS_Nik_lab_Organ", "SBS_Nik_lab", "RS_Nik_lab" to refer reference signatures from Degasperi, Andrea, et al. (2020) (reference #2); "RS_BRCA560", "RS_USARC" to reference signatures from BRCA560 and USARC cohorts; "CNS_USARC" (40
get_sig_similarity

categories), "CNS_TCGA" (48 categories) to reference copy number signatures from USARC cohort and TCGA. UPDATE, the latest version of reference version can be automatically downloaded and loaded from https://cancer.sanger.ac.uk/signatures/downloads/ when a option with latest_prefix is specified (e.g. "latest_SBS_GRCh37"). Note: the signature profile for different genome builds are basically same. And specific database (e.g. 'SBS_mm10') contains less signatures than all COSMIC signatures (because some signatures are not detected from Alexandrov, Ludmil B., et al. (2020)). For all available options, check the parameter setting.

db_type only used when sig_db is enabled. "" for keeping default, "human-exome" for transforming to exome frequency of component, and "human-genome" for transforming to whole genome frequency of component. Currently only works for 'SBS'.

method default is 'cosine' for cosine similarity.

normalize one of "row" and "feature". "row" is typically used for common mutational signatures. "feature" is designed by me to use when input are copy number signatures.

feature_setting a data.frame used for classification. Only used when method is "Wang" ("W"). Default is CN.features. Users can also set custom input with "feature", "min" and "max" columns available. Valid features can be printed by unique(CN.features$feature).

set_order if TRUE, order the return similarity matrix.

pattern_to_rm patterns for removing some features/components in similarity calculation. A vector of component name is also accepted. The remove operation will be done after normalization. Default is NULL.

verbose if TRUE, print extra info.

Value

a list containing similarities, aetiologies if available, best match and RSS.

Author(s)

Shixiang Wang w_shixiang@163.com

References


Examples

```r
# Load mutational signature
load(system.file("extdata", "toy_mutational_signature.RData",
    package = "sigminer", mustWork = TRUE
))

s1 <- get_sig_similarity(sig2, Ref = sig2)
s1

s2 <- get_sig_similarity(sig2)
s2

s3 <- get_sig_similarity(sig2, sig_db = "SBS")
s3

# Set order for result similarity matrix
s4 <- get_sig_similarity(sig2, sig_db = "SBS", set_order = TRUE)
s4

## Remove some components
## in similarity calculation
s5 <- get_sig_similarity(sig2,
    Ref = sig2,
)
s5

## Same to DBS and ID signatures
x1 <- get_sig_db("DBS_hg19")
x2 <- get_sig_db("DBS_hg38")
s6 <- get_sig_similarity(x1$db, x2$db)
s6
```

---

**get_tidy_association Get Tidy Signature Association Results**

### Description

Get Tidy Signature Association Results

### Usage

```r
get_tidy_association(cor_res, p_adjust = FALSE, method = "fdr")
```
Arguments

cor_res  data returned by `get_sig_feature_association()`
p_adjust logical, if TRUE, adjust p values by data type.
method   p value correction method, see `stats::p.adjust` for more detail.

Value

a data.frame

See Also

`get_sig_feature_association`

Description

This function takes a data.frame as input, compares proportion of positive cases or mean measure in one subgroup and the remaining samples.

Usage

```r
group_enrichment(
  df,
  grp_vars = NULL,
  enrich_vars = NULL,
  cross = TRUE,
  co_method = c("t.test", "wilcox.test")
)
```

Arguments

df          a data.frame.
grp_vars    character vector specifying group variables to split samples into subgroups (at least 2 subgroups, otherwise this variable will be skipped).
enrich_vars character vector specifying measure variables to be compared. If variable is not numeric, only binary cases are accepted in the form of TRUE/FALSE or P/N (P for positive cases and N for negative cases). Of note, NA values set to negative cases.
cross       logical, default is TRUE, combine all situations provided by grp_vars and enrich_vars. For examples, c("A", "B") and c("C", "D") will construct 4 combinations (i.e. "AC", "AD", "BC" and "BD"). A variable can not be in both grp_vars and enrich_vars, such cases will be automatically drop. If FALSE, use pairwise combinations, see section "examples" for use cases.
co_method   test method for continuous variable, default is 't.test'.
Value

A `data.table` with following columns:

- `grp_var`: group variable name.
- `enrich_var`: enrich variable (variable to be compared) name.
- `grp1`: the first group name, should be a member in `grp_var` column.
- `grp2`: the remaining samples, marked as 'Rest'.
- `grp1_size`: sample size for `grp1`.
- `grp1_pos_measure`: for binary variable, it stores the proportion of positive cases in `grp1`; for continuous variable, it stores mean value.
- `grp2_size`: sample size for `grp2`.
- `grp2_pos_measure`: same as `grp1_pos_measure` but for `grp2`.
- `measure_observed`: for binary variable, it stores odds ratio; for continuous variable, it stores scaled mean ratio.
- `measure_tested`: only for binary variable, it stores estimated odds ratio and its 95% CI from `fisher.test()`.
- `p_value`: for binary variable, it stores p value from `fisher.test()`; for continuous variable, it stores value from `wilcox.test()` or `t.test()`.
- `type`: one of "binary" and "continuous".
- `method`: one of "fish.test", "wilcox.test" and "t.test".

See Also

`show_group_enrichment`

Examples

```r
set.seed(1234)
df <- dplyr::tibble(
gen1 = factor(abs(round(rnorm(99, 0, 1)))),
gen2 = rep(LETTERS[1:4], c(50, 40, 8, 1)),
en1 = sample(c("P", "N"), 99, replace = TRUE),
en2 = rnorm(99)
)

print(str(df))
print(head(df))

# Compare gen1:e1, gen1:e2, gen2:e1 and gen2:e2
x1 <- group_enrichment(df, grp_vars = c("gen1", "gen2"), enrich_vars = c("en1", "en2"))
x1

# Only compare gen1:e1, gen2:e2
x2 <- group_enrichment(df, grp_vars = c("gen1", "gen2"), enrich_vars = c("en1", "en2"),
co_method = "wilcox.test",
)```
handle_hyper_mutation

---

**Description**

This can be used for SNV/INDEL count matrix. For copy number analysis, please skip it.

**Usage**

```r
handle_hyper_mutation(nmf_matrix)
```

**Arguments**

- `nmf_matrix` a matrix used for NMF decomposition with rows indicate samples and columns indicate components.

**Value**

- a matrix.

**References**

**MAF-class**

**Description**

Say Hello to Users

**Usage**

```r
hello()
```

**Examples**

```r
hello()
```

---

**MAF-class**  
*Class MAF*

**Description**

S4 class for storing summarized MAF. It is from maftools package.

**Details**

More about MAF object please see maftools.

**Slots**

- `data` : data.table of MAF file containing all non-synonymous variants.
- `variants.per.sample` : table containing variants per sample
- `variant.type.summary` : table containing variant types per sample
- `variant.classification.summary` : table containing variant classification per sample
- `gene.summary` : table containing variant classification per gene
- `summary` : table with basic MAF summary stats
- `maf.silent` : subset of main MAF containing only silent variants
- `clinical.data` : clinical data associated with each sample/Tumor_Sample_Barcode in MAF.
Output Signature Bootstrap Fitting Results

Description

Output Signature Bootstrap Fitting Results

Usage

output_bootstrap(x, result_dir, mut_type = "SBS", sig_db = mut_type)

Arguments

- **x**: result from `sig_fit_bootstrap_batch`.
- **result_dir**: a result directory.
- **mut_type**: one of 'SBS', 'DBS', 'ID' or 'CN'.
- **sig_db**: default 'legacy', it can be 'legacy' (for COSMIC v2 'SBS'), 'SBS', 'DBS', 'ID' and 'TSB' (for COSMIV v3.1 signatures) for small scale mutations. For more specific details, it can also be 'SBS_hg19', 'SBS_hg38', 'SBS_mm9', 'SBS_mm10', 'DBS_hg19', 'DBS_hg38', 'DBS_mm9', 'DBS_mm10' to use COSMIC v3 reference signatures from Alexandrov, Ludmil B., et al. (2020) (reference #1). In addition, it can be one of "SBS_Nik_lab_Organ", "RS_Nik_lab_Organ", "SBS_Nik_lab", "RS_Nik_lab" to refer reference signatures from Degasperi, Andrea, et al. (2020) (reference #2); "RS_BRCA560", "RS_USARC" to refer reference signatures from BRCA560 and USARC cohorts; "CNS_USARC" (40 categories), "CNS_TCGA" (48 categories) to reference copy number signatures from USARC cohort and TCGA. **UPDATE**, the latest version of reference version can be automatically downloaded and loaded from https://cancer.sanger.ac.uk/signatures/downloads/ when a option with latest_prefix is specified (e.g. "latest_SBS_GRCh37"). **Note**: the signature profile for different genome builds are basically same. And specific database (e.g. 'SBS_mm10') contains less signatures than all COSMIC signatures (because some signatures are not detected from Alexandrov, Ludmil B., et al. (2020)). For all available options, check the parameter setting.

Value

Nothing.
output_fit  Output Signature Fitting Results

Description

Output Signature Fitting Results

Usage

```
output_fit(x, result_dir, mut_type = "SBS", sig_db = mut_type)
```

Arguments

- `x`: result from `sig_fit`
- `result_dir`: a result directory.
- `mut_type`: one of 'SBS', 'DBS', 'ID' or 'CN'.
- `sig_db`: default 'legacy', it can be 'legacy' (for COSMIC v2 'SBS'), 'SBS', 'DBS', 'ID' and 'TSB' (for COSMIV v3.1 signatures) for small scale mutations. For more specific details, it can also be 'SBS_hg19', 'SBS_hg38', 'SBS_mm9', 'SBS_mm10', 'DBS_hg19', 'DBS_hg38', 'DBS_mm9', 'DBS_mm10' to use COSMIC v3 reference signatures from Alexandrov, Ludmil B., et al. (2020) (reference #1). In addition, it can be one of "SBS_Nik_lab_Organ", "RS_Nik_lab_Organ", "SBS_Nik_lab", "RS_Nik_lab" to refer reference signatures from Degasperi, Andrea, et al. (2020) (reference #2); "RS_BRCA560", "RS_USARC" to reference signatures from BRCA560 and USARC cohorts; "CNS_USARC" (40 categories), "CNS_TCGA" (48 categories) to reference copy number signatures from USARC cohort and TCGA. UPDATE, the latest version of reference version can be automatically downloaded and loaded from https://cancer.sanger.ac.uk/signatures/downloads/ when a option with latest_prefix is specified (e.g. "latest_SBS_GRCh37"). Note: the signature profile for different genome builds are basically same. And specific database (e.g. 'SBS_mm10') contains less signatures than all COSMIC signatures (because some signatures are not detected from Alexandrov, Ludmil B., et al. (2020)). For all available options, check the parameter setting.

Value

Nothing.
Output Signature Results

Usage

output_sig(sig, result_dir, mut_type = "SBS", sig_db = mut_type)

Arguments

- **sig**: a Signature object.
- **result_dir**: a result directory.
- **mut_type**: one of 'SBS', 'DBS', 'ID' or 'CN'.
- **sig_db**: default 'legacy', it can be 'legacy' (for COSMIC v2 'SBS'), 'SBS', 'DBS', 'ID' and 'TSB' (for COSMIV v3.1 signatures) for small scale mutations. For more specific details, it can also be 'SBS_hg19', 'SBS_hg38', 'SBS_mm9', 'SBS_mm10', 'DBS_hg19', 'DBS_hg38', 'DBS_mm9', 'DBS_mm10' to use COSMIC v3 reference signatures from Alexandrov, Ludmil B., et al. (2020) (reference #1). In addition, it can be one of "SBS_Nik_lab_Organ", "RS_Nik_lab_Organ", "SBS_Nik_lab", "RS_Nik_lab" to refer reference signatures from Degasperi, Andrea, et al. (2020) (reference #2); "RS_BRCA560", "RS_USARC" to refer reference signatures from BRCA560 and USARC cohorts; "CNS_USARC" (40 categories), "CNS_TCGA" (48 categories) to reference copy number signatures from USARC cohort and TCGA. **UPDATE**, the latest version of reference version can be automatically downloaded and loaded from https://cancer.sanger.ac.uk/signatures/downloads/ when a option with latest_prefix is specified (e.g. "latest_SBS_GRCh37"). **Note**: the signature profile for different genome builds are basically same. And specific database (e.g. 'SBS_mm10') contains less signatures than all COSMIC signatures (because some signatures are not detected from Alexandrov, Ludmil B., et al. (2020)). For all available options, check the parameter setting.

Value

Nothing.
output_tally  

Output Tally Result in Barplots

Description

Output Tally Result in Barplots

Usage

```r
output_tally(x, result_dir, mut_type = "SBS")
```

Arguments

- `x`: a matrix with row representing components (motifs) and column representing samples.
- `result_dir`: a result directory.
- `mut_type`: one of 'SBS', 'DBS', 'ID' or 'CN'.

Value

Nothing.

---

read_copynumber  

Read Absolute Copy Number Profile

Description

Read absolute copy number profile for preparing CNV signature analysis. See detail part of `sig_tally()` to see how to handle sex to get correct summary.

Usage

```r
read_copynumber(
  input,
  pattern = NULL,
  ignore_case = FALSE,
  seg_cols = c("Chromosome", "Start.bp", "End.bp", "modal_cn"),
  samp_col = "sample",
  add_loh = FALSE,
  loh_min_len = 10000,
  loh_min_frac = 0.05,
  join_adj_seg = TRUE,
  skip_annotation = FALSE,
  use_all = add_loh,
  min_segnum = 0L,
)```

max_copynumber = 20L,
genome_build = c("hg19", "hg38", "mm10", "mm9"),
genome_measure = c("called", "wg"),
complement = FALSE,
...)

Arguments

input a data.frame or a file or a directory contains copy number profile.
pattern an optional regular expression used to select part of files if input is a directory, more detail please see list.files() function.
ignore_case logical. Should pattern-matching be case-insensitive?
seg_cols four strings used to specify chromosome, start position, end position and copy number value in input, respectively. Default use names from ABSOLUTE calling result.
samp_col a character used to specify the sample column name. If input is a directory and cannot find samp_col, sample names will use file names (set this parameter to NULL is recommended in this case).
add_loh if TRUE, add LOH labels to segments. NOTE a column 'minor_cn' must exist to indicate minor allele copy number value. Sex chromosome will not be labeled.
loh_min_len The length cut-off for labeling a segment as ‘LOH’. Default is 10Kb.
loh_min_frac When join_adj_seg set to TRUE, only the length fraction of LOH region is larger than this value will be labeled as 'LOH'. Default is 30%.
join_adj_seg if TRUE (default), join adjacent segments with same copy number value. This is helpful for precisely count the number of breakpoint. When set use_all=TRUE, the mean function will be applied to extra numeric columns and unique string columns will be pasted by comma for joined records.
skip_annotation if TRUE, skip annotation step, it may affect some analysis and visualization functionality, but speed up reading data.
use_all default is FALSE. If True, use all columns from raw input.
min_segnum minimal number of copy number segments within a sample.
max_copynumber bigger copy number within a sample will be reset to this value.
genome_build genome build version, should be 'hg19', 'hg38', 'mm9' or 'mm10'.
genome_measure default is 'called', can be 'wg' or 'called'. Set 'called' will use called segments size to compute total size for CNA burden calculation, this option is useful for WES and target sequencing. Set 'wg' will use autosome size from genome build, this option is useful for WGS, SNP etc..
complement if TRUE, complement chromosome (except 'Y') does not show in input data with normal copy 2.

... other parameters pass to data.table::fread()
**Value**

A `CopyNumber` object.

**Author(s)**

Shixiang Wang w_shixiang@163.com

**See Also**

`read_maf` for reading mutation data to MAF object.

**Examples**

```R
# Load toy dataset of absolute copynumber profile
load(system.file("extdata", "toy_segTab.RData", 
    package = "sigminer", mustWork = TRUE 
))

cn <- read_copynumber(segTabs, 
    seg_cols = c("chromosome", "start", "end", "segVal"), 
    genome_build = "hg19", complement = FALSE 
)

cn

cn_subset <- subset(cn, sample == "TCGA-DF-A2KN-01A-11D-A17U-01")

# Add LOH
set.seed(1234)
segTabs$minor_cn <- sample(c(0, 1), size = nrow(segTabs), replace = TRUE)

cn <- read_copynumber(segTabs, 
    seg_cols = c("chromosome", "start", "end", "segVal"), 
    genome_measure = "wg", complement = TRUE, add_loh = TRUE 
)

# Use tally method "S" (Steele et al.)
tally_s <- sig_tally(cn, method = "S")

tab_file <- system.file("extdata", "metastatic_tumor.segtab.txt", 
    package = "sigminer", mustWork = TRUE 
)

cn2 <- read_copynumber(tab_file)

cn2
```

---

**read_copynumber_ascat**

Read Copy Number Data from ASCAT Result Files

**Description**

Note, the result is not a `CopyNumber` object, you need to generate it by yourself.

**Usage**

`read_copynumber_ascat(x)`
read_copynumber_seqz

Arguments

- `x` one or more .rds format files which contains ASCAT object from result of ascat.runAscat() in ASCAT package.

Value

a tidy list.

---

read_copynumber_seqz Read Absolute Copy Number Profile from Sequenza Result Directory

Description

Read Absolute Copy Number Profile from Sequenza Result Directory

Usage

```r
read_copynumber_seqz(target_dir, return_df = FALSE, ...)
```

Arguments

- `target_dir` a directory path.
- `return_df` if TRUE, return a data.frame directly, otherwise return a CopyNumber object.
- `...` other parameters passing to `read_copynumber()`.

Value

a data.frame or a CopyNumber object.

---

read_maf Read MAF Files

Description

This function is a wrapper of maftools::read.maf. Useless options in maftools::read.maf are dropped here. You can also use maftools::read.maf to read the data. All reference alleles and mutation alleles should be recorded in positive strand format.

Usage

```r
read_maf(maf, verbose = TRUE)
```
read_sv_as_rs

Arguments

- **maf**: tab delimited MAF file. File can also be gz compressed. Required. Alternatively, you can also provide already read MAF file as a dataframe.
- **verbose**: TRUE logical. Default to be talkative and prints summary.

See Also

read_copynumber for reading copy number data to CopyNumber object.

Examples

```r
laml.maf <- system.file("extdata", "tcga_laml.maf.gz", package = "maftools", mustWork = TRUE)
if (!require("R.utils")) {
  message("Please install 'R.utils' package firstly")
} else {
  laml <- read_maf(maf = laml.maf)
  laml
}
```

---

**read_sv_as_rs**  
*Read Structural Variation Data as RS object*

Description

Read Structural Variation Data as RS object

Usage

`read_sv_as_rs(input)`

Arguments

- **input**: a data.frame or a file with the following columns: "sample", "chr1", "start1", "end1", "chr2", "start2", "end2", "strand1", "strand2", "svclass". NOTE: If column "svclass" already exists in input, "strand1" and "strand2" are optional. If "svclass" is not provided, `read_sv_as_rs()` will compute it by "strand1","strand2"(strand1/strand2),"chr1" and "chr2":
  * translocation, if mates are on different chromosomes.
  * inversion (+/-) and (-/+), if mates on the same chromosome.
  * deletion (+/+), if mates on the same chromosome.
  * tandem-duplication (-/-), if mates on the same chromosome.

Value

a list
**Examples**

sv <- readRDS(system.file("extdata", "toy_sv.rds", package = "sigminer", mustWork = TRUE))
rs <- read_sv_as_rs(sv)
# svclass is optional
rs2 <- read_sv_as_rs(sv[, setdiff(colnames(sv), "svclass")])
identical(rs, rs2)
tally_rs <- sig_tally(rs)

---

**read_vcf**

*Read VCF Files as MAF Object*

**Description**

MAF file is more recommended. In this function, we will mimic the MAF object from the key c(1,2,4,5,7) columns of VCF file.

**Usage**

```r
read_vcf(
  vcfs,
  samples = NULL,
  genome_build = c("hg19", "hg38", "mm10", "mm9"),
  keep_only_pass = FALSE,
  verbose = TRUE
)
```

**Arguments**

- `vcfs` VCF file paths.
- `samples` sample names for VCF files.
- `genome_build` genome build version like "hg19".
- `keep_only_pass` if TRUE, keep only 'PASS' mutation for analysis.
- `verbose` if TRUE, print extra info.

**Value**

a MAF.

**See Also**

read_maf, read_copynumber
read_xena_variants

Examples

vcfs <- list.files(system.file("extdata", package = "sigminer"), 
"*.vcf", full.names = TRUE)

maf <- read_vcf(vcfs)
maf <- read_vcf(vcfs, keep_only_pass = TRUE)

read_xena_variants  Read UCSC Xena Variant Format Data as MAF Object

Description

Read UCSC Xena Variant Format Data as MAF Object

Usage

read_xena_variants(path)

Arguments

path a path to variant file.

Value

a MAF object.

Examples

if (requireNamespace("UCSCXenaTools")) {
  library(UCSCXenaTools)
  options(use_hiplot = TRUE)
  example_file <- XenaGenerate(subset = XenaDatasets == "mc3/ACC_mc3.txt") %>%
    XenaQuery() %>%
    XenaDownload()
  x <- read_xena_variants(example_file$destfiles)
  x@data
  y <- sig_tally(x)
  y
}
report_bootstrap_p_value

Report P Values from bootstrap Results

Description

See examples in sig_fit_bootstrap.

Usage

report_bootstrap_p_value(x, thresholds = c(0.01, 0.05, 0.1))

Arguments

x a (list of) result from sig_fit_bootstrap.
thresholds a vector of relative exposure threshold for calculating p values.

Value

a (list of) matrix

same_size_clustering Same Size Clustering

Description

This is a wrapper for several implementation that classify samples into same size clusters, the details please see this blog. The source code is modified based on code from the blog.

Usage

same_size_clustering(
  mat,
  diss = FALSE,
  clszize = NULL,
  algo = c("nnit", "hcbottom", "kmvar"),
  method = c("maxd", "random", "mind", "elki", "ward.D", "average", "complete", "single")
)

Arguments

mat a data/distance matrix.
diss if TRUE, treat mat as a distance matrix.
clszize integer, number of sample within a cluster.
algo algorithm.
method method.
Value

a vector.

Examples

```r
set.seed(1234L)
x <- rbind(
  matrix(rnorm(100, sd = 0.3), ncol = 2),
  matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2)
)
colnames(x) <- c("x", "y")

y1 <- same_size_clustering(x, clsize = 10)
y11 <- same_size_clustering(as.matrix(dist(x)), clsize = 10, diss = TRUE)
y2 <- same_size_clustering(x, clsize = 10, algo = "hcbottom", method = "ward.D")
y3 <- same_size_clustering(x, clsize = 10, algo = "kmvar")
y33 <- same_size_clustering(as.matrix(dist(x)), clsize = 10, algo = "kmvar", diss = TRUE)
```

scoring

<table>
<thead>
<tr>
<th>scoring</th>
<th>Score Copy Number Profile</th>
</tr>
</thead>
</table>

Description

Returns quantification of copy number profile and events including tandem duplication and Chromothripsis etc. Only copy number data from autosome is used here. Some of the quantification methods are rough, you use at your risk. You should do some extra work to check the result scores.

Usage

```r
scoring(object, TD_size_cutoff = c(1000, 1e+05, 2e+06), TD_cn_cutoff = Inf)
```

Arguments

- **object**: a object of `CopyNumber`.
- **TD_size_cutoff**: a length-3 numeric vector used to specify the start, midpoint, end segment size for determining tandem duplication size range, midpoint is used to split TD into short TD and long TD. Default is 1Kb to 100Kb for short TD, 100Kb to 2Mb for long TD.
- **TD_cn_cutoff**: a number defining the maximum copy number of TD, default is Inf, i.e. no cutoff.
Scoring

A data table with the following scores:

- **cnaBurden**: CNA burden representing the altered genomic fraction as previously reported.
- **cnaLoad**: CNA load representing the quantity of copy number alteration.
- **MACN**: mean altered copy number (MACN) reflecting the property of altered copy number segments, calculated as
  \[ MACN = \frac{\sum_i C N_i}{N_{cnv}} \]
  where \( C N_i \) is the copy number of altered segment \( i \), \( N_{cnv} \) is the number of CNV.
- **weightedMACN**: same as MACN but weighted with segment length.
  \[ MACN_{weighted} = \frac{\sum_i (C N_i \times L_i)}{\sum_i L_i} \]
  where \( L_i \) is the length of altered copy number segment \( i \).
- **Ploidy**: ploidy, the formula is same as weightedMACN but using all copy number segments instead of altered copy number segments.
- **TDP_pnas**: tandem duplication phenotype score from [https://www.pnas.org/content/113/17/E2373](https://www.pnas.org/content/113/17/E2373), the threshold \( k \) in reference is omitted.
  \[ TDP = -\sum_{chr} |TD_{obs} - TD_{exp}| \]
  where \( TD_{total} \) is the number of TD, \( TD_{obs} \) and \( TD_{exp} \) are observed number of TD and expected number of TD for each chromosome.
- **TDP**: tandem duplication score used defined by our group work, TD represents segment with copy number greater than 2.
  \[ TD = \frac{TD_{total}}{\sum_{chr} |TD_{obs} - TD_{exp}| + 1} \]
- **sTDP**: TDP score for short TD.
- **lTDP**: TDP score for long TD.
- **TDP_size**: TDP region size (Mb).
- **sTDP_size**: sTDP region size (Mb).
- **lTDP_size**: lTDP region size (Mb).
- **Chromoth_state**: chromothripsis state score, according to reference doi: [10.1016/j.cell.2013.02.023](https://doi.org/10.1016/j.cell.2013.02.023), chromothripsis frequently leads to massive loss of segments on the affected chromosome with segmental losses being interspersed with regions displaying normal (disomic) copy-number (e.g., copy-number states oscillating between copy-number = 1 and copy-number = 2), forming to hundreds of locally clustered DNA rearrangements. Most of methods use both SV and CNV to infer chromothripsis, here we roughly quantify it with
  \[ \sum_{chr} N_{oscCN}^2 \]
  where \( N_{oscCN} \) is the number of oscillating copy number pattern “2-1-2” for each chromosome.
Examples

# Load copy number object
load(system.file("extdata", "toy_copynumber.RData", 
    package = "sigminer", mustWork = TRUE 
))

d <- scoring(cn)
d

d2 <- scoring(cn, TD_cn_cutoff = 4L)
d2

---

show_catalogue  
Show Alteration Catalogue Profile

Description

Show Alteration Catalogue Profile

Usage

show_catalogue(
  catalogue,  
  mode = c("SBS", "copynumber", "DBS", "ID", "RS"),  
  method = "Wang",  
  normalize = c("raw", "row", "feature"),  
  style = c("default", "cosmic"),  
  samples = NULL,  
  samples_name = NULL,  
  x_lab = "Components",  
  y_lab = "Counts",  
  ...
)

Arguments

catalogue  
result from sig_tally or a matrix with row representing components (motifs) and column representing samples

mode  
signature type for plotting, now supports 'copynumber', 'SBS', 'DBS', 'ID' and 'RS' (genome rearrangement signature).

method  
method for copy number feature classification in sig_tally, can be one of "Wang" ("W"), "S".

normalize  
normalize method.

style  
plot style, one of 'default' and 'cosmic'.

samples  
default is NULL, show sum of all samples in one row. If not NULL, show specified samples.
show_cn_circos

samples_name  set the sample names shown in plot.
x_lab  x axis lab.
y_lab  y axis lab.
...  other arguments passing to show_sig_profile.

Value

a ggplot object

Examples

data("simulated_catalogs")
p <- show_catalogue(simulated_catalogs$set1, style = "cosmic")
p

Description

Another visualization method for copy number profile like show_cn_profile.

Usage

show_cn_circos(
  data,
  samples = NULL,
  show_title = TRUE,
  chrs = paste0("chr", 1:22),
  genome_build = c("hg19", "hg38", "mm10", "mm9"),
  col = NULL,
  side = "inside",
  ...
)

Arguments

data  a CopyNumber object or a data.frame containing at least 'chromosome', 'start', 'end', 'segVal' these columns.
samples  default is NULL, can be a character vector representing multiple samples or number of samples to show. If data argument is a data.frame, a column called sample must exist.
show_title  if TRUE (default), show title with sample ID.
chrs  chromosomes start with 'chr'.
genome_build  genome build version, used when data is a data.frame, should be 'hg19' or 'hg38'.
show_cn_components

Show Copy Number Components

Description

Show classified components ("Wang" ("W") method) for copy number data.

Usage

show_cn_components(
    parameters,  # required
    method = "Wang",  # method of classification
    show_weights = TRUE,  # show weights
    log_y = FALSE,  # log scale on y-axis
    return_plotlist = FALSE,  # return plotlist
    base_size = 12,  # base size for plots
    nrow = 2,  # number of rows in layout
    align = "hv",  # alignment of plots
    ...  # additional arguments
)
show_cn_distribution

**Arguments**

- `parameters` a data.frame contain parameter components, obtain this from `sig_tally` function.
- `method` method for feature classification, can be one of "Wang" ("W"), "S" (for method described in Steele et al. 2019).
- `show_weights` default is TRUE, show weights for each component. Only used when method is "Macintyre".
- `log_y` logical, if TRUE, show log10 based y axis, only works for input from "Wang" ("W") method.
- `return_plotlist` if TRUE, return a list of ggplot objects but a combined plot.
- `base_size` overall font size.
- `nrow` (optional) Number of rows in the plot grid.
- `align` (optional) Specifies whether graphs in the grid should be horizontally ("h") or vertically ("v") aligned. Options are "none" (default), "hv" (align in both directions), "h", and "v".
- `...` other options pass to `plot_grid` function of `cowplot` package.

**Value**

a ggplot object

**Author(s)**

Shixiang Wang w_shixiang@163.com

**Description**

Visually summarize copy number distribution either by copy number segment length or chromosome. Input is a `CopyNumber` object, genome_build option will read from genome_build slot of object.

**Usage**

```r
show_cn_distribution(
  data, 
  rm_normal = TRUE, 
  mode = c("ld", "cd"), 
  fill = FALSE, 
  scale_chr = TRUE, 
  base_size = 14 
)
```
Arguments

- **data**: a CopyNumber object.
- **rm_normal**: logical. Whether remove normal copy (i.e. "segVal" equals 2), default is TRUE.
- **mode**: either "ld" for distribution by CN length or "cd" for distribution by chromosome.
- **fill**: when mode is "cd" and fill is TRUE, plot percentage instead of count.
- **scale_chr**: logical. If TRUE, normalize count to per Megabase unit.
- **base_size**: overall font size.

Value

- a ggplot object

Author(s)

Shixiang Wang w_shixiang@163.com

Examples

```r
# Load copy number object
load(system.file("extdata", "toy_copynumber.RData", package = "sigminer", mustWork = TRUE))
# Plot distribution
p1 <- show_cn_distribution(cn)
p1 <- show_cn_distribution(cn, mode = "cd")
p3 <- show_cn_distribution(cn, mode = "cd", fill = TRUE)
```

Description

Show Copy Number Feature Distributions

Usage

```r
show_cn_features(
  features,
  method = "Wang",
  rm_outlier = FALSE,
  ylab = NULL,
  log_y = FALSE,
  return_plotlist = FALSE,
)```
show_cn_freq_circos

Usage

show_cn_freq_circos(
data,
grids = NULL,  
cutoff = 2L,  
resolution_factor = 1L,  
title = c("AMP", "DEL"),  
chrs = paste0("chr", 1:22),

Arguments

features a feature list generate from sig_tally function.

method method for feature classification, can be one of "Wang" ("W"), "S" (for method described in Steele et al. 2019).

rm_outlier default is FALSE, if TRUE, remove outliers. Only works when method is "Wang" ("W").

ylab lab of y axis.

log_y logical, if TRUE, show log10 based y axis, only works for input from "Wang" ("W") method.

return_plotlist if TRUE, return a list of ggplot objects but a combined plot.

base_size overall font size.

nrow (optional) Number of rows in the plot grid.

align (optional) Specifies whether graphs in the grid should be horizontally ("h") or vertically ("v") aligned. Options are "none" (default), "hv" (align in both directions), "h", and "v".

Value

a ggplot object

Description

Show Copy Number Variation Frequency Profile with Circos

Usage

show_cn_freq_circos(
data,
grids = NULL,  
cutoff = 2L,  
resolution_factor = 1L,  
title = c("AMP", "DEL"),  
chrs = paste0("chr", 1:22),
show_cn_freq_circos

gene_build = c("hg19", "hg38", "mm10", "mm9"),
cols = NULL,
plot_ideogram = TRUE,
track_height = 0.5,
ideogram_height = 1,
...
)

Arguments

data a CopyNumber object or a data.frame containing at least 'chromosome', 'start', 'end', 'segVal', 'sample' these columns.
groups a named list or a column name for specifying groups.
cutoff copy number value cutoff for splitting data into AMP and DEL. The values equal to cutoff are discarded. Default is 2, you can also set a length-2 vector, e.g. c(2,2).
resolution_factor an integer to control the resolution. When it is 1 (default), compute frequency in each cytoband. When it is 2, use compute frequency in each half cytoband.
title length-2 titles for AMP and DEL.
chrs chromosomes start with 'chr'.
genome_build genome build version, used when data is a data.frame, should be 'hg19' or 'hg38'.
cols length-2 colors for AMP and DEL.
plot_ideogram default is TRUE, show ideogram.
track_height track height in mm unit.
ideogram_height ideogram height in mm unit.
... other parameters passing to circlize::circos.genomicLines.

Value

Nothing.

Examples

load(system.file("extdata", "toy_copynumber.RData",
          package = "sigminer", mustWork = TRUE
))

show_cn_freq_circos(cn)
ss <- unique(cn$data$sample)
show_cn_freq_circos(cn, groups = list(a = ss[1:5], b = ss[6:10]), cols = c("red", "green"))
show_cn_group_profile

Description

Show Summary Copy Number Profile for Sample Groups

Usage

show_cn_group_profile(
  data,
  groups = NULL,
  fill_area = TRUE,
  cols = NULL,
  chrs = paste0("chr", c(1:22, "X")),
  genome_build = c("hg19", "hg38", "mm10", "mm9"),
  cutoff = 2L,
  resolution_factor = 1L,
  force_y_limit = TRUE,
  highlight_genes = NULL,
  repel = FALSE,
  nrow = NULL,
  ncol = NULL,
  return_plotlist = FALSE
)

Arguments

data a CopyNumber object or a data.frame containing at least 'chromosome', 'start', 'end', 'segVal', 'sample' these columns.
groups a named list or a column name for specifying groups.
fill_area default is TRUE, fill area with colors.
cols length-2 colors for AMP and DEL.
chrs chromosomes start with 'chr'.
genome_build genome build version, used when data is a data.frame, should be 'hg19' or 'hg38'.
cutoff copy number value cutoff for splitting data into AMP and DEL. The values equal to cutoff are discarded. Default is 2, you can also set a length-2 vector, e.g. c(2,2).
resolution_factor an integer to control the resolution. When it is 1 (default), compute frequency in each cytoband. When it is 2, use compute frequency in each half cytoband.
force_y_limit default is TRUE, force multiple plots
show_cn_group_profile

highlight_genes
gene list to highlight. have same y ranges. You can also set a length-2 numeric value.

repel
if TRUE (default is FALSE), repel highlight genes to avoid overlap.

nrow
number of rows in the plot grid when multiple samples are selected.

ncol
number of columns in the plot grid when multiple samples are selected.

return_plotlist
default is FALSE, if TRUE, return a plot list instead of a combined plot.

Value

a (list of) ggplot object.

Examples

load(system.file("extdata", "toy_copynumber.RData", package = "sigminer", mustWork = TRUE))

p1 <- show_cn_group_profile(cn)
p1

ss <- unique(cn@data$sample)
p2 <- show_cn_group_profile(cn, groups = list(a = ss[1:5], b = ss[6:10]))
p2

p3 <- show_cn_group_profile(cn, 
  groups = list(g1 = ss[1:5], g2 = ss[6:10]),
  force_y_limit = c(-1, 1), nrow = 2)
p3

## Set custom cutoff for custom data
data <- cn@data
data$segVal <- data$segVal - 2L
p4 <- show_cn_group_profile(data, 
  groups = list(g1 = ss[1:5], g2 = ss[6:10]),
  force_y_limit = c(-1, 1), nrow = 2,
  cutoff = c(0, 0))
p4

## Add highlight gene
p5 <- show_cn_group_profile(cn, highlight_genes = c("TP53", "EGFR"))
p5
show_cn_profile

Show Sample Copy Number Profile

Description

Sometimes it is very useful to check details about copy number profile for one or multiple samples. This function is designed to do this job and can be further modified by ggplot2 related packages.

Usage

```
show_cn_profile(
  data,
  samples = NULL,
  show_n = NULL,
  show_title = FALSE,
  show_labels = NULL,
  chrs = paste0("chr", 1:22),
  position = NULL,
  genome_build = c("hg19", "hg38", "mm10", "mm9"),
  ylim = NULL,
  nrow = NULL,
  ncol = NULL,
  return_plotlist = FALSE
)
```

Arguments

data: a CopyNumber object or a data.frame containing at least 'chromosome', 'start', 'end', 'segVal' these columns.
samples: default is NULL, can be a character vector representing multiple samples. If data argument is a data.frame, a column called sample must exist.
show_n: number of samples to show, this is used for checking.
show_title: if TRUE, show title for multiple samples.
show_labels: one of NULL, "s" (for labelling short segments < 1e7) or "a" (all segments).
chrs: chromosomes start with 'chr'.
position: a position range, e.g. "chr1:3218923-116319008". Only data overlaps with this range will be shown.
genome_build: genome build version, used when data is a data.frame, should be 'hg19' or 'hg38'.
ylim: limits for y axis.
nrow: number of rows in the plot grid when multiple samples are selected.
ncol: number of columns in the plot grid when multiple samples are selected.
return_plotlist: default is FALSE, if TRUE, return a plot list instead of a combined plot.
show_cor

A Simple and General Way for Association Analysis

Value

a ggplot object or a list

Examples

# Load copy number object
load(system.file("extdata", "toy_copynumber.RData",
    package = "sigminer", mustWork = TRUE))

p <- show_cn_profile(cn, nrow = 2, ncol = 1)
p

p2 <- show_cn_profile(cn,
    nrow = 2, ncol = 1,
    position = "chr1:3218923-116319008"
)
p2

show_cor

Description

All variables must be continuous. The matrix will be returned as an element of ggplot object. This is basically a wrapper of R package ggcorrplot.

Usage

show_cor(
    data,
    x_vars = colnames(data),
    y_vars = x_vars,
    cor_method = "spearman",
    vis_method = "square",
    lab = TRUE,
    test = TRUE,
    hc_order = FALSE,
    p_adj = NULL,
    ...
)

Arguments

data a data.frame.
x_vars variables/column names shown in x axis.
show_cosmic

### show_cor

- **y_vars**: variables/column names shown in y axis.
- **cor_method**: method for correlation, default is 'spearman'.
- **vis_method**: visualization method, default is 'square', can also be 'circle'.
- **lab**: logical value. If TRUE, add correlation coefficient on the plot.
- **test**: if TRUE, run test for correlation and mark significance.
- **hc_order**: logical value. If TRUE, correlation matrix will be hc.ordered using hclust function.
- **p_adj**: p adjust method, see stats::p.adjust for details.
- **...**: other parameters passing to ggcorrplot::ggcorrplot().

#### Value

A ggplot object

#### See Also

show_sig_feature_corrplot for specific and more powerful association analysis and visualization.

#### Examples

```r
data("mtcars")
p1 <- show_cor(mtcars)
p2 <- show_cor(mtcars, 
               x_vars = colnames(mtcars)[1:4], 
               y_vars = colnames(mtcars)[5:8]
)
p3 <- show_cor(mtcars, vis_method = "circle", p_adj = "fdr")
p1
p1$cor
p2
p3

## Auto detect problem variables
mtcars$xx <- 0L
p4 <- show_cor(mtcars)
p4
```

---

**show_cosmic**

Show Signature Information in Web Browser

### Description

Show Signature Information in Web Browser

#### Usage

```r
show_cosmic(x = "home")
```
**show_cosmic_sig_profile**

**Arguments**

- `x`: a string indicating location ("home" for COSMIC signature home, "legacy" for COSMIC v2 signatures, "SBS" for COSMIC v3 SBS signatures, "DBS" for COSMIC v3 DBS signatures, "ID" for COSMIC v3 INDEL signatures) or signature index (e.g. "SBS1", "DBS2", "ID3").

**Value**

Nothing.

**Examples**

```r
## Not run:
show_cosmic()
show_cosmic("legacy")
show_cosmic("SBS")
show_cosmic("DBS")
show_cosmic("ID")
show_cosmic("SBS1")
show_cosmic("DBS2")
show_cosmic("ID3")

## End(Not run)
```

---

**show_cosmic_sig_profile**

*Plot Reference (Mainly COSMIC) Signature Profile*

**Description**

Plot Reference (Mainly COSMIC) Signature Profile

**Usage**

```r
show_cosmic_sig_profile(
  sig_index = NULL,
  show_index = TRUE,
  sig_db = "legacy",
  ...)
```

**Arguments**

- `sig_index`: a vector for signature index. "ALL" for all signatures.
- `show_index`: if TRUE, show valid indices.
show_groups

Show Signature Contribution in Clusters

Description

See example section in `sig_fit()` for an examples.

Usage

`show_groups(grp_dt, ...)`
show_group_comparison

Arguments

grp_dt               a result data.table from get_groups.
...                  parameters passing to legend(), e.g. x = "topleft".

Value

nothing.

See Also

groups, sig_fit.

show_group_comparison  Plot Group Comparison Result

Description

Using result data from get_group_comparison, this function plots genotypes/phenotypes comparison between signature groups using ggplot2 package and return a list of ggplot object contains individual and combined plots. The combined plot is easily saved to local using cowplot::save_plot(). Of note, default fisher test p values are shown for categorical data and fdr values are shown for continuous data.

Usage

show_group_comparison(
  group_comparison,
  xlab = "group",
  ylab_co = NA,
  legend_title_ca = NA,
  legend_position_ca = "bottom",
  set_ca_sig_yaxis = FALSE,
  set_ca_custom_xlab = FALSE,
  show_pvalue = TRUE,
  ca_p_threshold = 0.01,
  method = "wilcox.test",
  p.adjust.method = "fdr",
  base_size = 12,
  font_size_x = 12,
  text_angle_x = 30,
  text_hjust_x = 0.2,
  ...
)
Arguments

group_comparison
  a list from result of `get_group_comparison` function.

xlab
  lab name of x axis for all plots. if it is NA, remove title for x axis.

ylab_co
  lab name of y axis for plots of continuous type data. Of note, this argument should be a character vector has same length as `group_comparison`, the location for categorical type data should mark with NA.

legend_title_ca
  legend title for plots of categorical type data.

legend_position_ca
  legend position for plots of categorical type data. Of note, this argument should be a character vector has same length as `group_comparison`, the location for continuous type data should mark with NA.

set_ca_sig_yaxis
  if TRUE, use y axis to show signature proportion instead of variable proportion.

set_ca_custom_xlab
  only works when `set_ca_sig_yaxis` is TRUE. If TRUE, set x labels using input xlab, otherwise variable names will be used.

show_pvalue
  if TRUE, show p values.

ca_p_threshold
  a p threshold for categorical variables, default is 0.01. A p value less than 0.01 will be shown as $P < 0.01$.

method
  a character string indicating which method to be used for comparing means. It can be 't.test', 'wilcox.test' etc..

p.adjust.method
  correction method, default is 'fdr'. Run `p.adjust.methods` to see all available options.

base_size
  overall font size.

font_size_x
  font size for x.

text_angle_x
  text angle for x.

text_hjust_x
  adjust x axis text

... other parameters pass to `ggpubr::compare_means()` or `ggpubr::stat_compare_means()` according to the specified method.

Value

  a list of ggplot objects.

Author(s)

Shixiang Wang w_shixiang@163.com
Examples

```r
load(system.file("extdata", "toy_copynumber_signature_by_W.RData", 
    package = "sigminer", mustWork = TRUE
))

# Assign samples to clusters
groups <- get_groups(sig, method = "k-means")

set.seed(1234)

groups$prob <- rnorm(10)
groups$new_group <- sample(c("1", "2", "3", "4", NA), size = nrow(groups), replace = TRUE)

# Compare groups (filter NAs for categorical columns)
groups.cmp <- get_group_comparison(groups[, -1], 
    col_group = "group", 
    cols_to_compare = c("prob", "new_group"), 
    type = c("co", "ca"), verbose = TRUE
)

# Compare groups (Set NAs of categorical columns to 'Rest')
groups.cmp2 <- get_group_comparison(groups[, -1], 
    col_group = "group", 
    cols_to_compare = c("prob", "new_group"), 
    type = c("co", "ca"), NAs = "Rest", verbose = TRUE
)

show_group_comparison(groups.cmp2)

ggcomp$co_comb

show_group_distribution

show_group_distribution

Show Grouped Variable Distribution

Description

This is a general function, it can be used in any proper analysis.

Usage

``show_group_distribution``

data,
gvar,
dvar,
fun = stats::median,
order_by_fun = FALSE,
show_group_distribution

alpha = 0.8,
g_label = "label",
g_angle = 0,
g_position = "top",
point_size = 1L,
segment_size = 1L,
segment_color = "red",
xlab = NULL,
ylab = NULL,
nrow = 1L,
background_color = c("#DCDCDC", "#F5F5F5")
)

Arguments

data               a data.frame.
gvar               a group variable name/index.
dvar               a distribution variable name/index.
fun                 a function to summarize, default is stats::median, can also be mean.
order_by_fun       if TRUE, reorder the groups by summary measure computed by argument fun.
alpha               alpha for points, range from 0 to 1.
g_label             a string 'label' (default) for labeling with sample size, or 'norm' to show just group name, or a named vector to set facet labels.
g_angle             angle for facet labels, default is 0.
g_position          position for facet labels, default is 'top', can also be 'bottom'.
point_size          size of point.
segment_size        size of segment.
segment_color       color of segment.
xlab                title for x axis.
ylab                title for y axis.
nrow                number of row.
background_color    background color for plot panel.

Value

a ggplot object.

Author(s)

Shixiang Wang w_shixiang@163.com
Examples

```r
set.seed(1234)
data <- data.frame(  
yval = rnorm(120),  
gr = c(rep("A", 50), rep("B", 40), rep("C", 30))
)
p <- show_group_distribution(data,  
gvar = 2, dvar = 1,  
g_label = "norm",  
background_color = "grey"
)
p
p2 <- show_group_distribution(data,  
gvar = "gr", dvar = "yval",  
g_position = "bottom",  
order_by_fun = TRUE,  
alpha = 0.3
)
p2

# Set custom group names
p3 <- show_group_distribution(data,  
gvar = 2, dvar = 1,  
g_label = c("A" = "X", "B" = "Y", "C" = "Z")
)
p3
```

show_group_enrichment  

Show Group Enrichment Result

Description

See `group_enrichment` for examples. NOTE the box fill and the box text have different meanings.

Usage

```r
show_group_enrichment(  
  df_enrich,  
  return_list = FALSE,  
  scales = "free",  
  add_text_annotation = TRUE,  
  fill_by_p_value = TRUE,  
  use_fdr = TRUE,  
  cut_p_value = FALSE,  
  cut_breaks = c(-Inf, -5, log10(0.05), -log10(0.05), 5, Inf),  
  cut_labels = c("\downarrow 1e-5", "\downarrow 0.05", "non-significant", "\uparrow 0.05", "\uparrow 1e-5"),  
  fill_scale = scale_fill_gradient2(low = "#08A76B", mid = "white", high = "red",  
                                  midpoint = ifelse(fill_by_p_value, 0, 1)),
)```
cluster_row = FALSE,
}

Arguments

df_enrich    result data.frame from group_enrichment.
return_list  if TRUE, return a list of ggplot object so user can combine multiple plots by
             other R packages like patchwork.
scales       Should scales be fixed ("fixed", the default), free ("free"), or free in one
dimension ("free_x", "free_y")?
add_text_annotation
             if TRUE, add text annotation in box. When show p value with filled color, the
             text indicates relative change; when show relative change with filled color, the
             text indicates p value.
fill_by_p_value
             if TRUE, show log10 based p values with filled color. The +/- of p values indicates
             change direction.
use_fdr      if TRUE, show FDR values instead of raw p-values.
cut_p_value  if TRUE, cut p values into 5 regions for better visualization. Only works when
             fill_by_p_value = TRUE.
cut_breaks   when cut_p_value is TRUE, this option set the (log10 based) breaks.
cut_labels   when cut_p_value is TRUE, this option set the labels.
fill_scale   a Scale object generated by ggplot2 package to set color for continuous values.
center_row   if TRUE, cluster rows with Hierarchical Clustering (’complete’ method).
...          other parameters passing to ggplot2::facet_wrap, only used when return_list
             is FALSE.

Value

a (list of) ggplot object.

show_group_mapping  Map Groups using Sankey

Description

This feature is designed for signature analysis. However, users can also use it in other similar situations.
show_group_mapping

Usage

show_group_mapping(
  data,
  col_to_flow,
  cols_to_map,
  include_sig = FALSE,
  fill_na = FALSE,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  custom_theme = cowplot::theme_minimal_hgrid()
)

Arguments

data a data.frame containing signature group and other categorical groups.

col_to_flow length-1 character showing the column to flow, typically a signature group.

cols_to_map character vector showing colnames of other groups.

include_sig default if FALSE, if TRUE, showing signature group.
fll_na length-1 string to fill NA, default is FALSE.
title the title.
xlab label for x axis.
ylab label for y axis.
custom_theme theme for plotting, default is cowplot::theme_minimal_hgrid().

Value

a ggplot object

Examples

data <- dplyr::tibble(
  Group1 = rep(LETTERS[1:5], each = 10),
  Group2 = rep(LETTERS[6:15], each = 5),
  zzzz = c(rep("xx", 20), rep("yy", 20), rep(NA, 10))
)
p1 <- show_group_mapping(data, col_to_flow = "Group1", cols_to_map = colnames(data)[-1])
p1

p2 <- show_group_mapping(data,
  col_to_flow = "Group1", cols_to_map = colnames(data)[-1],
  include_sig = TRUE
)
p2
show_sig_bootstrap  

Show Signature Bootstrap Analysis Results

Description

See details for description.

Usage

show_sig_bootstrap_exposure(
  bt_result,
  sample = NULL,
  signatures = NULL,
  methods = "QP",
  plot_fun = c("boxplot", "violin"),
  agg_fun = c("mean", "median", "min", "max"),
  highlight = "auto",
  highlight_size = 4,
  palette = "aaas",
  title = NULL,
  xlab = FALSE,
  ylab = "Signature exposure",
  width = 0.3,
  dodge_width = 0.8,
  outlier.shape = NA,
  add = "jitter",
  add.params = list(alpha = 0.3),
...
)

show_sig_bootstrap_error(
  bt_result,
  sample = NULL,
  methods = "QP",
  plot_fun = c("boxplot", "violin"),
  agg_fun = c("mean", "median"),
  highlight = "auto",
  highlight_size = 4,
  palette = "aaas",
  title = NULL,
  xlab = FALSE,
  ylab = "Reconstruction error (L2 norm)",
  width = 0.3,
  dodge_width = 0.8,
  outlier.shape = NA,
  add = "jitter",
  add.params = list(alpha = 0.3),
show_sig_bootstrap_stability(
    bt_result,
    signatures = NULL,
    measure = c("RMSE", "CV", "MAE", "AbsDiff"),
    methods = "QP",
    plot_fun = c("boxplot", "violin"),
    palette = "aaas",
    title = NULL,
    xlab = FALSE,
    ylab = "Signature instability",
    width = 0.3,
    outlier.shape = NA,
    add = "jitter",
    add.params = list(alpha = 0.3),
)

Arguments

bt_result  result object from `sig_fit_bootstrap_batch`.
sample  a sample id.
signatures  signatures to show.
methods  a subset of c("NNLS","QP","SA").
plot_fun  set the plot function.
agg_fun  set the aggregation function when sample is NULL.
highlight  set the color for optimal solution. Default is "auto", which use the same color as bootstrap results, you can set it to color like "red", "gold", etc.
highlight_size  size for highlighting triangle, default is 4.
palette  the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "uiscgb", "uchicago", "simpsons" and "rickandmorty".
title  plot main title.
xlab  character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab  character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
width  numeric value between 0 and 1 specifying box width.
dodge_width  dodge width.
outlier.shape  point shape of outlier. Default is 19. To hide outlier, specify outlier.shape = NA. When jitter is added, then outliers will be automatically hidden.
show_sig_bootstrap

add character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.

add.params parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

... other parameters passing to ggpubr::ggboxplot or ggpubr::ggviolin.

legend character specifying legend position. Allowed values are one of c("top", "bottom", "left", "right", "none"). To remove the legend use legend = "none". Legend position can be also specified using a numeric vector c(x, y); see details section.

measure measure to estimate the exposure instability, can be one of 'RMSE', 'CV', 'MAE' and 'AbsDiff'.

Details

Functions:

- show_sig_bootstrap_exposure - this function plots exposures from bootstrap samples with both dotted boxplot. The optimal exposure (the exposure from original input) is shown as triangle point. Only one sample can be plotted.
- show_sig_bootstrap_error - this function plots decomposition errors from bootstrap samples with both dotted boxplot. The error from optimal solution (the decomposition error from original input) is shown as triangle point. Only one sample can be plotted.
- show_sig_bootstrap_stability - this function plots the signature exposure instability for specified signatures. Currently, the instability measure supports 3 types:
  - 'RMSE' for Mean Root Squared Error (default) of bootstrap exposures and original exposures for each sample.
  - 'CV' for Coefficient of Variation (CV) based on RMSE (i.e. \( \text{RMSE} / \text{btExposure}_\text{mean} \)).
  - 'MAE' for Mean Absolute Error of bootstrap exposures and original exposures for each sample.
  - 'AbsDiff' for Absolute Difference between mean bootstrap exposure and original exposure.

Value

a ggplot object

References


See Also

sig_fit_bootstrap_batch, sig_fit, sig_fit_bootstrap
Examples

```r
if (require("BSgenome.Hsapiens.UCSC.hg19")) {
  laml.maf <- system.file("extdata", "tcga_laml.maf.gz", package = "maftools")
  laml <- read_maf(maf = laml.maf)
  mt_tally <- sig_tally(
    laml,
    ref_genome = "BSgenome.Hsapiens.UCSC.hg19",
    use_syn = TRUE
  )

  library(NMF)
  mt_sig <- sig_extract(mt_tally$nmf_matrix,
    n_sig = 3,
    nrun = 2,
    cores = 1
  )

  mat <- t(mt_tally$nmf_matrix)
  mat <- mat[, colSums(mat) > 0]
  bt_result <- sig_fit_bootstrap_batch(mat, sig = mt_sig, n = 10)
  ## Parallel computation
  ## bt_result = sig_fit_bootstrap_batch(mat, sig = mt_sig, n = 10, use_parallel = TRUE)

  ## At default, mean bootstrap exposure for each sample has been calculated
  p <- show_sig_bootstrap_exposure(bt_result, methods = c("QP"))
  ## Show bootstrap exposure (optimal exposure is shown as triangle)
  p1 <- show_sig_bootstrap_exposure(bt_result, methods = c("QP"), sample = "TCGA-AB-2802")
  p1
  p2 <- show_sig_bootstrap_exposure(bt_result,
    methods = c("QP"),
    sample = "TCGA-AB-3012",
    signatures = c("Sig1", "Sig2")
  )
  p2

  ## Show bootstrap error
  ## Similar to exposure above
  p <- show_sig_bootstrap_error(bt_result, methods = c("QP"))
  p
  p3 <- show_sig_bootstrap_error(bt_result, methods = c("QP"), sample = "TCGA-AB-2802")
  p3

  ## Show exposure (in)stability
  p4 <- show_sig_bootstrap_stability(bt_result, methods = c("QP"))
  p4
  p5 <- show_sig_bootstrap_stability(bt_result, methods = c("QP"), measure = "MAE")
  p5
  p6 <- show_sig_bootstrap_stability(bt_result, methods = c("QP"), measure = "AbsDiff")
  p6
  p7 <- show_sig_bootstrap_stability(bt_result, methods = c("QP"), measure = "CV")
  p7
}```
This function is a wrapper of \texttt{NMF::consensusmap()}. 

**Usage**

```r
show_sig_consensusmap(
    sig, 
    main = "Consensus matrix",
    tracks = c("consensus:", "silhouette:"),
    lab_row = NA,
    lab_col = NA,
    ...
)
```

**Arguments**

- `sig`: a Signature object obtained from \texttt{sig_extract}. 
- `main`: Main title as a character string or a grob. 
- `tracks`: Special additional annotation tracks to highlight associations between basis components and sample clusters: 
  - `basis`: matches each row (resp. column) to the most contributing basis component in `basismap` (resp. `coefmap`). In `basismap` (resp. `coefmap`), adding a track `':basis'` to `annCol` (resp. `annRow`) makes the column (resp. row) corresponding to the component being also highlighted using the matching colours. 
- `lab_row`: labels for the rows. 
- `lab_col`: labels for the columns. 
- `...`: other parameters passing to \texttt{NMF::consensusmap()}. 

**Value**

nothing
show_sig_exposure

Plot Signature Exposure

Description
Currently support copy number signatures and mutational signatures.

Usage

```r
show_sig_exposure(
  Signature,
  sig_names = NULL,
  groups = NULL,
  grp_order = NULL,
  grp_size = NULL,
  cutoff = NULL,
  style = c("default", "cosmic"),
  palette = use_color_style(style),
  base_size = 12,
  font_scale = 1,
  rm_space = FALSE,
  rm_grid_line = TRUE,
  rm_panel_border = FALSE,
  hide_samps = TRUE,
  legend_position = "top"
)
```

Arguments

- **Signature**: a Signature object obtained either from `sig_extract` or `sig_auto_extract`, or just a raw **absolute** exposure matrix with column representing samples (patients) and row representing signatures (signature names must end with different digital numbers, e.g. Sig1, Sig10, x12). If you named signatures with letters, you can specify them by `sig_names` parameter.

- **sig_names**: set name of signatures, can be a character vector.

- **groups**: sample groups, default is NULL.

- **grp_order**: order of groups, default is NULL.

- **grp_size**: font size of groups.

- **cutoff**: a cutoff value to remove hyper-mutated samples.

- **style**: plot style, one of 'default' and 'cosmic', works when parameter `set_gradient_color` is FALSE.

- **palette**: palette used to plot, default use a built-in palette according to parameter `style`.

- **base_size**: overall font size.

- **font_scale**: a number used to set font scale.
show_sig_feature_corrplot

- **rm_space**
  - default is FALSE. If TRUE, it will remove border color and expand the bar width to 1. This is useful when the sample size is big.

- **rm_grid_line**
  - default is FALSE. If TRUE, remove grid lines of plot.

- **rm_panel_border**
  - default is TRUE for style 'cosmic', remove panel border to keep plot tight.

- **hide_samps**
  - if TRUE, hide sample names.

- **legend_position**
  - position of legend, default is 'top'.

**Value**

a ggplot object

**Author(s)**

Shixiang Wang

**Examples**

```r
# Load mutational signature
load(system.file("extdata", "toy_mutational_signature.RData", 
  package = "sigminer", mustWork = TRUE
))
# Show signature exposure
p1 <- show_sig_exposure(sig2)
p1

# Load copy number signature
load(system.file("extdata", "toy_copynumber_signature_by_W.RData", 
  package = "sigminer", mustWork = TRUE
))
# Show signature exposure
p2 <- show_sig_exposure(sig)
p2
```

**Description**

This function is for association visualization. Of note, the parameters `p_val` and `drop` will affect the visualization of association results under p value threshold.
show_sig_feature_corrplot

Usage

show_sig_feature_corrplot(
  tidy_cor,
  feature_list,
  sort_features = FALSE,
  sig_orders = NULL,
  drop = TRUE,
  return_plotlist = FALSE,
  p_val = 0.05,
  xlab = "Signatures",
  ylab = "Features",
  co_gradient_colors = scale_color_gradient2(low = "blue", mid = "white", high = "red",
                                              midpoint = 0),
  ca_gradient_colors = co_gradient_colors,
  plot_ratio = "auto",
  breaks_count = NULL
)

Arguments

tidy_cor        data returned by `get_tidy_association`.
feature_list    a character vector contains features want to be plotted. If missing, all features will be used.
sort_features   default is FALSE, use feature order obtained from the previous step. If TRUE, sort features as feature_list.
sig_orders      signature levels for ordering.
drop            if TRUE, when a feature has no association with all signatures (p value larger than threshold set by p_val), this feature will be removed from the plot. Otherwise, this feature (a row) will keep with all blank white.
return_plotlist if TRUE, return as a list of ggplot objects.
p_val           p value threshold. If p value larger than this threshold, the result becomes blank white.
               xlab               label for x axis.
               ylab               label for y axis.
co_gradient_colors a Scale object representing gradient colors used to plot for continuous features.
ca_gradient_colors a Scale object representing gradient colors used to plot for categorical features.
plot_ratio       a length-2 numeric vector to set the height/width ratio.
breaks_count     breaks for sample count. If set it to NULL, ggplot bin scale will be used to automatically determine the breaks. If set it to NA, aes for sample will be not used.
show_sig_fit

Value

a ggplot2 object

See Also

get_tidy_association and get_sig_feature_association

Examples

# The data is generated from Wang, Shixiang et al.
load(system.file("extdata", "asso_data.RData",
package = "sigminer", mustWork = TRUE))

p <- show_sig_feature_corrplot(
  tidy_data.seqz.feature,
  p_val = 0.05,
  breaks_count = c(0L, 200L, 400L, 600L, 800L, 1020L))

p
Arguments

- **fit_result**: result object from `sig_fit`.
- **samples**: samples to show, if NULL, all samples are used.
- **signatures**: signatures to show.
- **plot_fun**: set the plot function.
- **palette**: the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmörty".
- **title**: plot main title.
- **xlab**: character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
- **ylab**: character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
- **legend**: character specifying legend position. Allowed values are one of c("top", "bottom", "left", "right", "none"). To remove the legend use legend = "none". Legend position can be also specified using a numeric vector c(x, y); see details section.
- **width**: numeric value between 0 and 1 specifying box width.
- **outlier.shape**: point shape of outlier. Default is 19. To hide outlier, specify outlier.shape = NA. When jitter is added, then outliers will be automatically hidden.
- **add**: character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.
- **add.params**: parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").
- **...**: other arguments to be passed to `geom_boxplot`, `ggpar` and `facet`.

Value

a `ggplot` object.

See Also

- `sig_fit`, `show_sig_bootstrap_exposure`, `sig_fit_bootstrap`, `sig_fit_bootstrap_batch`
show_sig_profile

show_sig_profile  Show Signature Profile

Description
Who don’t like to show a barplot for signature profile? This is for it.

Usage

```r
show_sig_profile(
  Signature,
  mode = c("SBS", "copynumber", "DBS", "ID", "RS"),
  method = "Wang",
  by_context = FALSE,
  normalize = c("row", "column", "raw", "feature"),
  y_tr = NULL,
  filters = NULL,
  feature_setting = sigminer::CN.features,
  style = c("default", "cosmic"),
  palette = use_color_style(style, ifelse(by_context, "SBS", mode), method),
  set_gradient_color = FALSE,
  free_space = "free_x",
  rm_panel_border = style == "cosmic",
  rm_grid_line = style == "cosmic",
  rm_axis_text = FALSE,
  bar_border_color = ifelse(style == "default", "grey50", "white"),
  bar_width = 0.7,
  paint_axis_text = TRUE,
  x_label_angle = ifelse(mode == "copynumber" & !(startsWith(method, "T") | method == "X"), 60, 90),
  x_label_vjust = ifelse(mode == "copynumber" & !(startsWith(method, "T") | method == "X"), 1, 0.5),
  x_label_hjust = 1,
  x_lab = "Components",
  y_lab = "auto",
  y_limits = NULL,
  params = NULL,
  show_cv = FALSE,
  params_label_size = 3,
  params_label_angle = 60,
  y_expand = 1,
  digits = 2,
  base_size = 12,
  font_scale = 1,
  sig_names = NULL,
  sig_orders = NULL,
  check_sig_names = TRUE
)```
Arguments

**Signature**
a Signature object obtained either from `sig_extract` or `sig_auto_extract`, or just a raw signature matrix with row representing components (motifs) and column representing signatures (column names must start with 'Sig').

**mode**
signature type for plotting, now supports 'copynumber', 'SBS', 'DAS', 'ID' and 'RS' (genome rearrangement signature).

**method**
method for copy number feature classification in `sig_tally`, can be one of "Wang" ("W"), "S".

**by_context**
for specific use.

**normalize**
one of 'row', 'column', 'raw' and "feature", for row normalization (signature), column normalization (component), raw data, row normalization by feature, respectively. Of note, "feature" only works when the mode is 'copynumber'.

**y_tr**
a function (e.g. `log10`) to transform y axis before plotting.

**filters**
a pattern used to select components to plot.

**feature_setting**
a data.frame used for classification. **Only used when method is "Wang"** ("W"). Default is `CN.features`. Users can also set custom input with "feature", "min" and "max" columns available. Valid features can be printed by `unique(CN.features$feature)`.

**style**
plot style, one of 'default' and 'cosmic', works when parameter `set_gradient_color` is FALSE.

**palette**
palette used to plot when `set_gradient_color` is FALSE, default use a built-in palette according to parameter `style`.

**set_gradient_color**
default is FALSE, if TRUE, use gradient colors to fill bars.

**free_space**
default is 'free_x'. If "fixed", all panels have the same size. If "free_y" their height will be proportional to the length of the y scale; if "free_x" their width will be proportional to the length of the x scale; or if "free" both height and width will vary. This setting has no effect unless the appropriate scales also vary.

**rm_panel_border**
default is TRUE for style 'cosmic', remove panel border to keep plot tight.

**rm_grid_line**
default is FALSE, if TRUE, remove grid lines of plot.

**rm_axis_text**
default is FALSE, if TRUE, remove component texts. This is useful when multiple signature profiles are plotted together.

**bar_border_color**
the color of bar border.

**bar_width**
bar width. By default, set to 70% of the resolution of the data.

**paint_axis_text**
if TRUE, color on text of x axis.

**x_label_angle**
font angle for x label.
show_sig_profile

x_label_vjust  font vjust for x label.
x_label_hjust  font hjust for x label.
x_lab           x axis lab.
y_lab           y axis lab.
y_limits        limits to expand in y axis. e.g., 0.2, c(0, 0.3).
params          params data.frame of components, obtained from sig_tally.
show_cv         default is FALSE, if TRUE, show coefficient of variation when params is not NULL.
params_label_size font size for params label.
params_label_angle font angle for params label.
y_expand        y expand height for plotting params of copy number signatures.
digits          digits for plotting params of copy number signatures.
base_size       overall font size.
font_scale       a number used to set font scale.
sig_names        subset signatures or set name of signatures, can be a character vector. Default is NULL, prefix ‘Sig’ plus number is used.
sig_orders       set order of signatures, can be a character vector. Default is NULL, the signatures are ordered by alphabetical order. If an integer vector set, only specified signatures are plotted.
check_sig_names  if TRUE, check signature names when input is a matrix, i.e., all signatures (col-names) must start with 'Sig'.

Value

a ggplot object

Author(s)
Shixiang Wang

See Also

show_sig_profile_loop, show_sig_profile_heatmap

Examples

# Load SBS signature
load(system.file("extdata", "toy_mutational_signature.RData",
    package = "sigminer", mustWork = TRUE
))
# Show signature profile
p1 <- show_sig_profile(sig2, mode = "SBS")
p1
# Use 'y_tr' option to transform values in y axis
p11 <- show_sig_profile(sig2, mode = "SBS", y_tr = function(x) x * 100)

# Load copy number signature from method "W"
load(system.file("extdata", "toy_copynumber_signature_by_W.RData", 
    package = "sigminer", mustWork = TRUE
))

# Show signature profile
p2 <- show_sig_profile(sig, 
    style = "cosmic", 
    mode = "copynumber", 
    method = "W", 
    normalize = "feature"
)

# Visualize rearrangement signatures
s <- get_sig_db("RS_Nik_lab")
ss <- s$db[, 1:3]
colnames(ss) <- c("Sig1", "Sig2", "Sig3")
p3 <- show_sig_profile(ss, mode = "RS", style = "cosmic")
p3

show_sig_profile_heatmap

Show Signature Profile with Heatmap

Description
This is a complementary function to show_sig_profile(), it is used for visualizing some big signatures, i.e. SBS-1536, not all signatures are supported. See details for current supported signatures.

Usage

show_sig_profile_heatmap(
    Signature,
    mode = c("SBS", "DBS"),
    normalize = c("row", "column", "raw"),
    filters = NULL,
    x_lab = NULL,
    y_lab = NULL,
    legend_name = "auto",
    palette = "red",
    x_label_angle = 90,
    x_label_vjust = 1,
    x_label_hjust = 0.5,
    y_label_angle = 0,
)
show_sig_profile_heatmap

```r
y_label_vjust = 0.5,
y_label_hjust = 1,
flip_xy = FALSE,
sig_names = NULL,
sig_orders = NULL,
check_sig_names = TRUE
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>a Signature object obtained either from <code>sig_extract</code> or <code>sig_auto_extract</code>, or just a raw signature matrix with row representing components (motifs) and column representing signatures (column names must start with 'Sig').</td>
</tr>
<tr>
<td>mode</td>
<td>one of &quot;SBS&quot; and &quot;DBS&quot;.</td>
</tr>
<tr>
<td>normalize</td>
<td>one of 'row', 'column', 'raw' and &quot;feature&quot;, for row normalization (signature), column normalization (component), raw data, row normalization by feature, respectively. Of note, 'feature' only works when the mode is 'copynumber'.</td>
</tr>
<tr>
<td>filters</td>
<td>a pattern used to select components to plot.</td>
</tr>
<tr>
<td>x_lab</td>
<td>x label.</td>
</tr>
<tr>
<td>y_lab</td>
<td>y label.</td>
</tr>
<tr>
<td>legend_name</td>
<td>name of figure legend.</td>
</tr>
<tr>
<td>palette</td>
<td>color for value.</td>
</tr>
<tr>
<td>x_label_angle</td>
<td>angle for x axis text.</td>
</tr>
<tr>
<td>x_label_vjust</td>
<td>vjust for x axis text.</td>
</tr>
<tr>
<td>x_label_hjust</td>
<td>hjust for x axis text.</td>
</tr>
<tr>
<td>y_label_angle</td>
<td>angle for y axis text.</td>
</tr>
<tr>
<td>y_label_vjust</td>
<td>vjust for y axis text.</td>
</tr>
<tr>
<td>y_label_hjust</td>
<td>hjust for y axis text.</td>
</tr>
<tr>
<td>flip_xy</td>
<td>if TRUE, flip x axis and y axis.</td>
</tr>
<tr>
<td>sig_names</td>
<td>subset signatures or set name of signatures, can be a character vector. Default is NULL, prefix 'Sig' plus number is used.</td>
</tr>
<tr>
<td>sig_orders</td>
<td>set order of signatures, can be a character vector. Default is NULL, the signatures are ordered by alphabetical order. If an integer vector set, only specified signatures are plotted.</td>
</tr>
<tr>
<td>check_sig_names</td>
<td>if TRUE, check signature names when input is a matrix, i.e., all signatures (column names) must start with 'Sig'.</td>
</tr>
</tbody>
</table>

**Details**

Support:

- SBS-24
- SBS-96
show_sig_profile_loop

- SBS-384
- SBS-1536
- SBS-6144
- DBS-78
- DBS-186

Value

a ggplot object.

Examples

```r
# Load SBS signature
load(system.file("extdata", "toy_mutational_signature.RData",
   package = "sigminer", mustWork = TRUE
))
# Show signature profile
p1 <- show_sig_profile_heatmap(sig2, mode = "SBS")
p1
```

Description

Show Signature Profile with Loop Way

Usage

```r
show_sig_profile_loop(
   Signature,
   sig_names = NULL,
   ncol = 1,
   nrow = NULL,
   x_lab = "Components",
   ...
)
```

Arguments

- **Signature**: a Signature object obtained either from `sig_extract` or `sig_auto_extract`, or just a raw signature matrix with row representing components (motifs) and column representing signatures (column names must start with 'Sig').
- **sig_names**: subset signatures or set name of signatures, can be a character vector. Default is `NULL`, prefix 'Sig' plus number is used.
- **ncol**: (optional) Number of columns in the plot grid.
nrow  (optional) Number of rows in the plot grid.
x_lab  x axis lab.
...  other parameters but sig_order passing to show_sig_profile.

Value

a ggplot result from cowplot::plot_grid().

See Also

show_sig_profile

Examples

load(system.file("extdata", "toy_mutational_signature.RData", package = "sigminer", mustWork = TRUE))
# Show signature profile
p1 <- show_sig_profile_loop(sig2, mode = "SBS")
p1
p2 <- show_sig_profile_loop(sig2, mode = "SBS", style = "cosmic", sig_names = c("A", "B", "C"))
p2

Description

- Author: Shixiang Wang (w_shixiang@163.com)
- Please go to https://shixiangwang.github.io/sigminer-doc/ for full vignette.
- Result visualization for MAF is provide by maftools package, please read its vignette.
**Description**

This function provides an interface to software SigProfiler. More please see https://github.com/AlexandrovLab/SigProfilerExtractor. Typically, a reference genome is not required because the input is a matrix (my understanding).

**Usage**

```
sigprofiler_extract(
    nmf_matrix, 
    output, 
    range = 2:5, 
    nrun = 10L, 
    refit = FALSE, 
    refit_plot = FALSE, 
    is_exome = FALSE, 
    init_method = c("nndsvd_min", "random", "alexandrov-lab-custom", "nndsvd", "nndsvda", 
                     "nndsvdar"), 
    cores = -1L, 
    genome_build = c("hg19", "hg38", "mm10", "mm9"), 
    use_conda = FALSE, 
    py_path = NULL, 
    sigprofiler_version = "1.1.3"
)
```

```
sigprofiler_import(
    output, 
    order_by_expo = FALSE, 
    type = c("suggest", "refit", "all")
)
```

**Arguments**

- **nmf_matrix**: a matrix used for NMF decomposition with rows indicate samples and columns indicate components.
- **output**: output directory.
- **range**: signature number range, i.e. 2:5.
- **nrun**: the number of iteration to be performed to extract each signature number.
- **refit**: if TRUE, then refit the denovo signatures with nrls. Same meaning as optimize option in sig_extract or sig_auto_extract.
- **refit_plot**: if TRUE, SigProfiler will make denovo to COSMIC signatures decomposition plots. However, this may fail due to some matrix cannot be identified by SigProfiler plot program.
is_exome if TRUE, the exomes will be extracted.
init_method the initialization algorithm for W and H matrix of NMF. Options are 'random', 'nndsvd', 'nndsvda', 'nndsvdar', 'alexandrov-lab-custom' and 'nndsvd_min'.
cores number of cores used for computation.
genome_build I think this option is useless when input is matrix, keep it in case it is useful.
use_conda if TRUE, create an independent conda environment to run SigProfiler.
py_path path to Python executable file, e.g. '/Users/wsx/anaconda3/bin/python'.
sigprofiler_version version of SigProfilerExtractor. If this package is not installed, the specified package will be installed. If this package is installed, this option is useless.
order_by_expo if TRUE, order the import signatures by their exposures, e.g. the signature contributed the most exposure in all samples will be named as Sig1.
type one of 'suggest' (for suggested solution), 'refit' (for refit solution) or 'all' (for all solutions).

Value

For sigprofiler_extract(), returns nothing. See output directory.
For sigprofiler_import(), a list containing Signature object.

Examples

if (FALSE) {
  load(system.file("extdata", "toy_copynumber_tally_W.RData", package = "sigminer", mustWork = TRUE
})

reticulate::conda_list()

sigprofiler_extract(cn_tally_W$nmf_matrix, "~/test/test_sigminer", use_conda = TRUE
)

sigprofiler_extract(cn_tally_W$nmf_matrix, "~/test/test_sigminer", use_conda = FALSE, py_path = "/Users/wsx/anaconda3/bin/python"
)
Description

A bayesian variant of NMF algorithm to enable optimal inferences for the number of signatures through the automatic relevance determination technique. This functions delevers highly interpretable and sparse representations for both signature profiles and attributions at a balance between data fitting and model complexity (this method may introduce more signatures than expected, especially for copy number signatures (thus I don’t recommend you to use this feature to extract copy number signatures)). See detail part and references for more.

Usage

```r
sig_auto_extract(
  nmf_matrix = NULL,
  result_prefix = "BayesNMF",
  destdir = tempdir(),
  method = c("L1W.L2H", "L1KL", "L2KL"),
  strategy = c("stable", "optimal", "ms"),
  ref_sigs = NULL,
  K0 = 25,
  nrun = 10,
  niter = 2e+05,
  tol = 1e-07,
  cores = 1,
  optimize = FALSE,
  skip = FALSE,
  recover = FALSE
)
```

Arguments

- `nmf_matrix`: a matrix used for NMF decomposition with rows indicate samples and columns indicate components.
- `result_prefix`: prefix for result data files.
- `destdir`: path to save data runs, default is `tempdir()`.
- `method`: default is "L1W.L2H", which uses an exponential prior for W and a half-normal prior for H (This method is used by PCAWG project, see reference #3). You can also use "L1KL" to set exponential priors for both W and H, and "L2KL" to set half-normal priors for both W and H. The latter two methods are originally implemented by SignatureAnalyzer software.
- `strategy`: the selection strategy for returned data. Set 'stable' for getting optimal result from the most frequent K. Set 'optimal' for getting optimal result from all Ks. Set 'ms' for getting result with maximum mean cosine similarity with provided reference signatures. See `ref_sigs` option for details. If you want select other solution, please check `get_bayesian_result`.
- `ref_sigs`: A Signature object or matrix or string for specifying reference signatures, only used when `strategy = 'ms'`. See `Signature` and `sig_db` options in `get_sig_similarity` for details.
- `K0`: number of initial signatures.
number of independent simulations.
niter the maximum number of iterations.
tol tolerance for convergence.
cores number of cpu cores to run NMF.
optimize if TRUE, then refit the denovo signatures with QP method, see sig_fit.
skip if TRUE, it will skip running a previous stored result. This can be used to extend run times, e.g. you try running 10 times firstly and then you want to extend it to 20 times.
recover if TRUE, try to recover result from previous runs based on input result_prefix, destdir and nrun. This is pretty useful for reproducing result. Please use skip if you want to recover an unfinished job.

Details

There are three methods available in this function: "L1W.L2H", "L1KL" and "L2KL". They use different priors for the bayesian variant of NMF algorithm (see method parameter) written by reference #1 and implemented in SignatureAnalyzer software (reference #2).

I copied source code for the three methods from Broad Institute and supplementary files of reference #3, and wrote this higher function. It is more friendly for users to extract, visualize and analyze signatures by combining with other powerful functions in sigminer package. Besides, I implemented parallel computation to speed up the calculation process and a similar input and output structure like sig_extract().

Value

a list with Signature class.

Author(s)

Shixiang Wang

References


See Also

sig_tally for getting variation matrix, sig_extract for extracting signatures using NMF package, sig_estimate for estimating signature number for sig_extract.
**Examples**

```r
load(system.file("extdata", "toy_copynumber_tally_W.RData", 
    package = "sigminer", mustWork = TRUE
))
res <- sig_auto_extract(cn_tally_W$nmf_matrix, result_prefix = "Test_copynumber", nrun = 1)
# At default, all run files are stored in tempdir()
dir(tempdir(), pattern = "Test_copynumber")

laml.maf <- system.file("extdata", "tcga_laml.maf.gz", package = "maftools")
laml <- read_maf(maf = laml.maf)
mt_tally <- sig_tally(
    laml,
    ref_genome = "BSgenome.Hsapiens.UCSC.hg19",
    use_syn = TRUE
)
x <- sig_auto_extract(mt_tally$nmf_matrix, 
    strategy = "ms", nrun = 3, ref_sigs = "legacy"
)x
```

---

**sig_convert**

*Convert Signatures between different Genomic Distribution of Components*

**Description**

Converts signatures between two representations relative to different sets of mutational opportunities. Currently, only SBS signature is supported.

**Usage**

```r
sig_convert(sig, from = "human-genome", to = "human-exome")
```

**Arguments**

- **sig**
  - a Signature object obtained either from `sig_extract` or `sig_auto_extract`, or just a raw signature matrix/data.frame with row representing components (motifs) and column representing signatures.

- **from**
  - either one of "human-genome" and "human-exome" or an opportunity matrix (repeated n columns with each row represents the total number of mutations for a component, n is the number of signature).

- **to**
  - same as from.

**Details**

The default opportunity matrix for "human-genome" and "human-exome" comes from COSMIC signature database v2 and v3.
**Value**

a matrix.

**References**

convert_signatures function from sigfit package.

**Examples**

```r
# Load SBS signature
load(system.file("extdata", "toy_mutational_signature.RData",
    package = "sigminer", mustWork = TRUE
))
# Exome-relative to Genome-relative
sig_converted <- sig_convert(sig2,
    from = "human-exome",
    to = "human-genome"
)
sig_converted

show_sig_profile(sig2, style = "cosmic")
show_sig_profile(sig_converted, style = "cosmic")
```

**Description**

Use NMF package to evaluate the optimal number of signatures. This is used along with `sig_extract`. Users should library(NMF) firstly. If NMF objects are returned, the result can be further visualized by NMF plot methods like NMF::consensusmap() and NMF::basismap().

`sig_estimate()` shows comprehensive rank survey generated by NMF package, sometimes it is hard to consider all measures. `show_sig_number_survey()` provides a one or two y-axis visualization method to help users determine the optimal signature number (showing both stability ("cophenetic") and error (RSS) at default). Users can also set custom measures to show.

`show_sig_number_survey2()` is modified from NMF package to better help users to explore survey of signature number.

**Usage**

```r
sig_estimate(
    nmf_matrix,
    range = 2:5,
    nrun = 10,
    use_random = FALSE,
    method = "brunet",
    seed = 123456,
```
cores = 1,
keep_nmfObj = FALSE,
save_plots = FALSE,
plot_basename = file.path(tempdir(), "nmf"),
what = "all",
verbose = FALSE
)

show_sig_number_survey(
  object,
  x = "rank",
  left_y = "cophenetic",
  right_y = "rss",
  left_name = left_y,
  right_name = toupper(right_y),
  left_color = "black",
  right_color = "red",
  left_shape = 16,
  right_shape = 18,
  shape_size = 4,
  highlight = NULL
)

show_sig_number_survey2(
  x,
  y = NULL,
  what = c("all", "cophenetic", "rss", "residuals", "dispersion", "evar", "sparseness",
            "sparseness.basis", "sparseness.coef", "silhouette", "silhouette.coef",
            "silhouette.basis", "silhouette.consensus"),
  na.rm = FALSE,
  xlab = "Total signatures",
  ylab = "",
  main = "Signature number survey using NMF package"
)

Arguments

nmf_matrix a matrix used for NMF decomposition with rows indicate samples and columns indicate components.
range a numeric vector containing the ranks of factorization to try. Note that duplicates are removed and values are sorted in increasing order. The results are notably returned in this order.
nrun a numeric giving the number of run to perform for each value in range, nrun set to 30-50 is enough to achieve robust result.
use_random Should generate random data from input to test measurements. Default is TRUE.
method specification of the NMF algorithm. Use 'brunet' as default. Available methods for NMF decompositions are 'brunet', 'lee', 'ls-nmf', 'nsNMF', 'offset'.
seed: specification of the starting point or seeding method, which will compute a starting point, usually using data from the target matrix in order to provide a good guess.

cores: number of cpu cores to run NMF.

keep_nmfObj: default is FALSE, if TRUE, keep NMF objects from runs, and the result may be huge.

save_plots: if TRUE, save signature number survey plot to local machine.

plot_basename: when save plots, set custom basename for file path.

what: a character vector whose elements partially match one of the following item, which correspond to the measures computed by summary() on each – multi-run – NMF result: 'all', 'cophenetic', 'rss', 'residuals', 'dispersion', 'evar', 'silhouette' (and more specific *.coef, *.basis, *.consensus), 'sparseness' (and more specific *.coef, *.basis). It specifies which measure must be plotted (what='all' plots all the measures).

verbose: if TRUE, print extra message.

object: a Survey object generated from sig_estimate, or a data.frame contains at least rank columns and columns for one measure.

x: a data.frame or NMF.rank object obtained from sig_estimate().

left_y: column name for left y axis.

right_y: column name for right y axis.

left_name: label name for left y axis.

right_name: label name for right y axis.

left_color: color for left axis.

right_color: color for right axis.

left_shape, right_shape, shape_size: shape setting.

highlight: a integer to highlight a x.

y: for random simulation, a data.frame or NMF.rank object obtained from sig_estimate().

na.rm: single logical that specifies if the rank for which the measures are NA values should be removed from the graph or not (default to FALSE). This is useful when plotting results which include NAs due to error during the estimation process. See argument stop for nmfEstimateRank.

xlab: x-axis label

ylab: y-axis label

main: main title

Details

The most common approach is to choose the smallest rank for which cophenetic correlation coefficient starts decreasing (Used by this function). Another approach is to choose the rank for which the plot of the residual sum of squares (RSS) between the input matrix and its estimate shows an inflection point. More custom features please directly use NMF::nmfEstimateRank.
sig_estimate

Value

- sig_estimate: a list contains information of NMF run and rank survey.
- show_sig_number_survey: a ggplot object
- show_sig_number_survey2: a ggplot object

Author(s)

Shixiang Wang

References


See Also

sig_extract for extracting signatures using NMF package, sig_auto_extract for extracting signatures using automatic relevance determination technique. sig_estimate for estimating signature number for sig_extract, show_sig_number_survey2 for more visualization method.

Examples

```r
load(system.file("extdata", "toy_copynumber_tally_W.RData", package = "sigminer", mustWork = TRUE))
library(NMF)

cn_estimate <- sig_estimate(cn_tally_W$nmf_matrix, cores = 1, nrun = 5, verbose = TRUE)

p <- show_sig_number_survey2(cn_estimate$survey)
p

# Show two measures
show_sig_number_survey(cn_estimate)

# Show one measure
p1 <- show_sig_number_survey(cn_estimate, right_y = NULL)
p1

p2 <- add_h_arrow(p, x = 4.1, y = 0.953, label = "selected number")
p2

# Show data from a data.frame
p3 <- show_sig_number_survey(cn_estimate$survey)
p3

# Show other measures
head(cn_estimate$survey)
```
sig_extract

Extract Signatures through NMF

Description
Do NMF de-composition and then extract signatures.

Usage
sig_extract(nmf_matrix, n_sig, nrun = 10, cores = 1, method = "brunet", optimize = FALSE, pynmf = FALSE, use_conda = TRUE, py_path = "/Users/wsx/anaconda3/bin/python", seed = 123456, ...)

Arguments

- **nmf_matrix**
a matrix used for NMF decomposition with rows indicate samples and columns indicate components.

- **n_sig**
  number of signature. Please run `sig_estimate` to select a suitable value.

- **nrun**
a numeric giving the number of run to perform for each value in range, nrun set to 30-50 is enough to achieve robust result.

- **cores**
  number of cpu cores to run NMF.

- **method**
specification of the NMF algorithm. Use 'brunet' as default. Available methods for NMF decompositions are 'brunet', 'lee', 'ls-nmf', 'nsNMF', 'offset'.

- **optimize**
  if TRUE, then refit the denovo signatures with QP method, see `sig_fit`. 
if TRUE, use Python NMF driver Nimfa. The seed currently is not used by this implementation.

if TRUE, create an independent conda environment to run NMF.

path to Python executable file, e.g. '/Users/wsx/anaconda3/bin/python'. In my test, it is more stable than use_conda=TRUE. You can install the Nimfa package by yourself or set use_conda to TRUE to install required Python environment, and then set this option.

specification of the starting point or seeding method, which will compute a starting point, usually using data from the target matrix in order to provide a good guess.

other arguments passed to NMF::nmf().

a list with Signature class.

Shixiang Wang


sig_tally for getting variation matrix, sig_estimate for estimating signature number for sig_extract, sig_auto_extract for extracting signatures using automatic relevance determination technique.

load(system.file("extdata", "toy_copynumber_tally_W.RData", package = "sigminer", mustWork = TRUE))

# Extract copy number signatures
res <- sig_extract(cn_tally_W$nmf_matrix, 2, nrun = 1)
*sig_fit*  

**Fit Signature Exposures with Linear Combination Decomposition**

**Description**

The function performs a signatures decomposition of a given mutational catalogue \( V \) with known signatures \( W \) by solving the minimization problem \( \min(||W*H - V||) \) where \( W \) and \( V \) are known.

**Usage**

```r
sig_fit(
  catalogue_matrix, 
  sig, 
  sig_index = NULL, 
  sig_db = c("legacy", "SBS", "DBS", "ID", "TSB", "SBS_Nik_lab", "RS_Nik_lab", 
  "RS_BRCA560", "RS_USARC", "CNS_USARC", "CNS_TCGA", "SBS_hg19", "SBS_hg38", "SBS_mm9", 
  "SBS_mm10", "DBS_hg19", "DBS_hg38", "DBS_mm9", "DBS_mm10", "SBS_Nik_lab_Organ", 
  "RS_Nik_lab_Organ", "latest_SBS_GRCh37", "latest_DBS_GRCh37", "latest_ID_GRCh37", 
  "latest_SBS_GRCh38", "latest_DBS_GRCh38", "latest_SBS_mm9", "latest_DBS_mm9", 
  "latest_SBS_mm10", "latest_DBS_mm10", "latest_SBS_rn6", "latest_DBS_rn6"), 
  db_type = c("", "human-exome", "human-genome"), 
  show_index = TRUE, 
  method = c("QP", "NNLS", "SA"), 
  auto_reduce = FALSE, 
  type = c("absolute", "relative"), 
  return_class = c("matrix", "data.table"), 
  return_error = FALSE, 
  rel_threshold = 0, 
  mode = c("SBS", "DBS", "ID", "copynumber"), 
  true_catalog = NULL, 
  ... 
)
```

**Arguments**

- `catalogue_matrix`  
  a numeric matrix \( V \) with row representing components and columns representing samples, typically you can get \( \text{nmf_matrix} \) from \text{sig_tally}() and transpose it by `t()`.

- `sig`  
  a `Signature` object obtained either from \text{sig_extract} or \text{sig_auto_extract}, or just a raw signature matrix/data.frame with row representing components (motifs) and column representing signatures.

- `sig_index`  
  a vector for signature index. "ALL" for all signatures.

- `sig_db`  
  default 'legacy', it can be 'legacy' (for \text{COSMIC v2 'SBS'}, 'SBS', 'DBS', 'ID' and 'TSB' (for \text{COSMIV v3.1 signatures}) for small scale mutations. For more specific details, it can also be 'SBS_hg19', 'SBS_hg38', 'SBS_mm9',
'SBS_mm10', 'DBS_hg19', 'DBS_hg38', 'DBS_mm9', 'DBS_mm10' to use COSMIC v3 reference signatures from Alexandrov, Ludmil B., et al. (2020) (reference #1). In addition, it can be one of "SBS_Nik_lab_Organ", "RS_Nik_lab_Organ", "SBS_Nik_lab", "RS_Nik_lab" to refer reference signatures from Degasperi, Andrea, et al. (2020) (reference #2); "RS_BRCA560", "RS_USARC" to reference signatures from BRCA560 and USARC cohorts; "CNS_USARC" (40 categories), "CNS_TCGA" (48 categories) to reference copy number signatures from USARC cohort and TCGA. UPDATE, the latest version of reference version can be automatically downloaded and loaded from https://cancer.sanger.ac.uk/signatures/downloads/ when a option with latest_ prefix is specified (e.g. "latest_SBS_GRCh37"). Note: the signature profile for different genome builds are basically same. And specific database (e.g. 'SBS_mm10') contains less signatures than all COSMIC signatures (because some signatures are not detected from Alexandrov, Ludmil B., et al. (2020)). For all available options, check the parameter setting.

**db_type**
only used when sig_db is enabled. "" for keeping default, "human-exome" for transforming to exome frequency of component, and "human-genome" for transforming to whole genome frequency of component. Currently only works for 'SBS'.

**show_index**
if TRUE, show valid indices.

**method**
method to solve the minimazation problem. 'NNLS' for non-negative least square; 'QP' for quadratic programming; 'SA' for simulated annealing.

**auto_reduce**
if TRUE, try reducing the input reference signatures to increase the cosine similarity of reconstructed profile to observed profile.

**type**
'absolute' for signature exposure and 'relative' for signature relative exposure.

**return_class**
string, 'matrix' or 'data.table'.

**return_error**
if TRUE, also return sample error (Frobenius norm) and cosine similarity between observed sample profile (asa. spectrum) and reconstructed profile. NOTE: it is better to obtain the error when the type is 'absolute', because the error is affected by relative exposure accuracy.

**rel_threshold**
numeric vector, a signature with relative exposure lower than (equal is included, i.e. <=) this value will be set to 0 (both absolute exposure and relative exposure). In this case, sum of signature contribution may not equal to 1.

**mode**
signature type for plotting, now supports 'copynumber', 'SBS', 'DBS', 'ID' and 'RS' (genome rearrangement signature).

**true_catalog**
used by sig_fit_bootstrap, user never use it.

**control**
control parameters passing to argument control in GenSA function when use method 'SA'.

**Details**
The method 'NNLS' solves the minimization problem with nonnegative least-squares constraints. The method 'QP' and 'SA' are modified from SignatureEstimation package. See references for details. Of note, when fitting exposures for copy number signatures, only components of feature CN is used.
**Value**

The exposure result either in matrix or data.table format. If `return_error` set `TRUE`, a list is returned.

**References**


**See Also**

`sig_extract`, `sig_auto_extract`, `sig_fit_bootstrap`, `sig_fit_bootstrap_batch`

**Examples**

```r
W <- matrix(c(1, 2, 3, 4, 5, 6), ncol = 2)
colnames(W) <- c("sig1", "sig2")
W <- apply(W, 2, function(x) x / sum(x))

H <- matrix(c(2, 5, 3, 6, 1, 9, 1, 2), ncol = 4)
colnames(H) <- paste0("samp", 1:4)

V <- W %*% H
V

if (requireNamespace("quadprog", quietly = TRUE)) {
  H_infer <- sig_fit(V, W, method = "QP")
  H_infer
  H

  H_dt <- sig_fit(V, W, method = "QP", auto_reduce = TRUE, return_class = "data.table")
  H_dt

  ## Show results
  show_sig_fit(H_infer)
  show_sig_fit(H_dt)

  ## Get clusters/groups
  H_dt_rel <- sig_fit(V, W, return_class = "data.table", type = "relative")
  z <- get_groups(H_dt_rel, method = "k-means")
  show_groups(z)
}

# if (requireNamespace("GenSA", quietly = TRUE)) {
#   H_infer <- sig_fit(V, W, method = "SA")
#   H_infer
#   H
```
### sig_fit_bootstrap

Obtain Bootstrap Distribution of Signature Exposures of a Certain Tumor Sample

**Description**

This can be used to obtain the confidence of signature exposures or search the suboptimal decomposition solution.

**Usage**

```r
sig_fit_bootstrap(
  catalog,
  sig,
  n = 100L,
  sig_index = NULL,
  sig_db = "legacy",
  db_type = c("", "human-exome", "human-genome"),
  show_index = TRUE,
  method = c("QP", "NNLS", "SA"),
  auto_reduce = FALSE,
  SA_not_bootstrap = FALSE,
  type = c("absolute", "relative"),
  rel_threshold = 0,
  mode = c("SBS", "DBS", "ID", "copynumber"),
  find_suboptimal = FALSE,
  suboptimal_ref_error = NULL,
  suboptimal_factor = 1.05,
  ...
)
```

**Arguments**

- `catalog` a named numeric vector or a numeric matrix with dimension N x 1. N is the number of component, 1 is the sample.
**sig_fit_bootstrap**

sig  
a Signature object obtained either from `sig_extract` or `sig_auto_extract`, or just a raw signature matrix/data.frame with row representing components (motifs) and column representing signatures.

n  
the number of bootstrap replicates.

sig_index  
a vector for signature index. "ALL" for all signatures.

sig_db  
default 'legacy', it can be 'legacy' (for COSMIC v2 'SBS'), 'SBS', 'DBS', 'ID' and 'TSB' (for COSMIV v3.1 signatures) for small scale mutations. For more specific details, it can also be 'SBS_hg19', 'SBS_hg38', 'SBS_mm9', 'SBS_mm10', 'DBS_hg19', 'DBS_hg38', 'DBS_mm9', 'DBS_mm10' to use COSMIC v3 reference signatures from Alexandrov, Ludmil B., et al. (2020) (reference #1). In addition, it can be one of "SBS_Nik_lab_Organ", "RS_Nik_lab_Organ", "SBS_Nik_lab", "RS_Nik_lab" to refer reference signatures from Degasperi, Andrea, et al. (2020) (reference #2); "RS_BRCA560", "RS_USARC" to reference signatures from BRCA560 and USARC cohorts; "CNS_USARC" (40 categories), "CNS_TCGA" (48 categories) to reference copy number signatures from USARC cohort and TCGA. **UPDATE**, the latest version of reference version can be automatically downloaded and loaded from [https://cancer.sanger.ac.uk/signatures/downloads/](https://cancer.sanger.ac.uk/signatures/downloads/) when a option with latest_prefix is specified (e.g. "latest_SBS_GRCh37"). Note: the signature profile for different genome builds are basically same. And specific database (e.g. 'SBS_mm10') contains less signatures than all COSMIC signatures (because some signatures are not detected from Alexandrov, Ludmil B., et al. (2020)). For all available options, check the parameter setting.

db_type  
only used when `sig_db` is enabled. "" for keeping default, "human-exome" for transforming to exome frequency of component, and "human-genome" for transforming to whole genome frequency of component. Currently only works for 'SBS'.

show_index  
if TRUE, show valid indices.

method  
method to solve the minimazation problem. 'NNLS' for non-negative least square; 'QP' for quadratic programming; 'SA' for simulated annealing.

auto_reduce  
if TRUE, try reducing the input reference signatures to increase the cosine similarity of reconstructed profile to observed profile.

SA_not_bootstrap  
if TRUE, directly run 'SA' multiple times with original input instead of bootstrap samples.

type  
'absolute' for signature exposure and 'relative' for signature relative exposure.

rel_threshold  
numeric vector, a signature with relative exposure lower than (equal is included, i.e. <=) this value will be set to 0 (both absolute exposure and relative exposure). In this case, sum of signature contribution may not equal to 1.

mode  
signature type for plotting, now supports 'copynumber', 'SBS', 'DBS', 'ID' and 'RS' (genome rearrangement signature).

find_suboptimal  
logical, if TRUE, find suboptimal decomposition with slightly higher error than the optimal solution by method 'SA'. This is useful to explore hidden dependencies between signatures. More see reference.
suboptimal_ref_error
baseline error used for finding suboptimal solution. if it is NULL, then use 'SA'
method to obtain the optimal error.

suboptimal_factor
suboptimal factor to get suboptimal error, default is 1.05, i.e., suboptimal error
is 1.05 times baseline error.

... control parameters passing to argument control in GenSA function when use
method 'SA'.

Value

a list

References

Huang X, Wojtowicz D, Przytycka TM. Detecting presence of mutational signatures in cancer with

See Also

report_bootstrap_p_value, sig_fit, sig_fit_bootstrap_batch

Examples

W <- matrix(c(1, 2, 3, 4, 5, 6), ncol = 2)
colnames(W) <- c("sig1", "sig2")
W <- apply(W, 2, function(x) x / sum(x))

H <- matrix(c(2, 5, 3, 6, 1, 9, 1, 2), ncol = 4)
colnames(H) <- paste0("samp", 1:4)

V <- W %*% H

if (requireNamespace("quadprog", quietly = TRUE)) {
  H_bootstrap <- sig_fit_bootstrap(V[, 1], W, n = 10, type = "absolute"
  ## Typically, you have to run many times to get close to the answer
  boxplot(t(H_bootstrap$expo))
  H[, 1]
}

  ## Return P values
  ## In practice, run times >= 100
  ## is recommended
  report_bootstrap_p_value(H_bootstrap)
  ## For multiple samples
  ## Input a list
  report_bootstrap_p_value(list(samp1 = H_bootstrap, samp2 = H_bootstrap))

  #   ## Find suboptimal decomposition
  # H_suboptimal <- sig_fit_bootstrap(V[, 1], W,
  #   n = 10,}
# sig_fit_bootstrap_batch

```r
# type = "absolute",
# method = "SA",
# find_suboptimal = TRUE
# }
```

---

**Exposure Instability Analysis of Signature Exposures with Bootstrap-ping**

---

**Description**

Read `sig_fit_bootstrap` for more option setting.

**Usage**

```r
sig_fit_bootstrap_batch(
catalogue_matrix,
methods = c("QP"),
n = 100L,
min_count = 1L,
p_val_thresholds = c(0.05),
use_parallel = FALSE,
seed = 123456L,
job_id = NULL,
result_dir = tempdir(),
...
)
```

**Arguments**

- **catalogue_matrix**
  - a numeric matrix $V$ with row representing components and columns representing samples, typically you can get `nmf_matrix` from `sig_tally()` and transpose it by `t()`.

- **methods**
  - a subset of c("NNLS","QP","SA").

- **n**
  - the number of bootstrap replicates.

- **min_count**
  - minimal exposure in a sample, default is 1. Any patient has total exposure less than this value will be filtered out.

- **p_val_thresholds**
  - a vector of relative exposure threshold for calculating p values.

- **use_parallel**
  - if TRUE, use parallel computation based on `furrr` package. It can also be an integer for specifying cores.

- **seed**
  - random seed to reproduce the result.
Obtain or Modify Signature Information

Description

Obtain or Modify Signature Information

Usage

sig_names(sig)

sig_modify_names(sig, new_names)

sig_number(sig)

sig_attrs(sig)

Value

a list of data.table.

See Also

sig_fit, sig_fit_bootstrap

Examples

```r
W <- matrix(c(1, 2, 3, 4, 5, 6), ncol = 2)
colnames(W) <- c("sig1", "sig2")
W <- apply(W, 2, function(x) x / sum(x))

H <- matrix(c(2, 5, 3, 6, 1, 9, 1, 2), ncol = 4)
colnames(H) <- paste0("samp", 1:4)

V <- W %*% H
V

if (requireNamespace("quadprog")) {
  z10 <- sig_fit_bootstrap_batch(V, sig = W, n = 10)
  z10
}
```
sig_signature(sig, normalize = c("row", "column", "raw", "feature"))

sig_exposure(sig, type = c("absolute", "relative"))

Arguments

- **sig**: a Signature object obtained either from `sig_extract` or `sig_auto_extract`.
- **new_names**: new signature names.
- **normalize**: one of 'row', 'column', 'raw' and "feature", for row normalization (signature), column normalization (component), raw data, row normalization by feature, respectively.
- **type**: one of 'absolute' and 'relative'.

Value

a Signature object or data.

Examples

```r
## Operate signature names
load(system.file("extdata", "toy_mutational_signature.RData", 
    package = "sigminer", mustWork = TRUE 
))
sig_names(sig2)
cc <- sig_modify_names(sig2, new_names = c("Sig2", "Sig1", "Sig3"))
sig_names(cc)

# The older names are stored in tags.
print(attr(cc, "tag"))
## Get signature number
sig_number(sig2)
## Get signature attributes
sig_number(sig2)
## Get signature matrix
z <- sig_signature(sig2)
z <- sig_signature(sig2, normalize = "raw")
## Get exposure matrix
## Of note, this is different from get_sig_exposure()
## it returns a matrix instead of data table.
z <- sig_exposure(sig2)  # it is same as sig$Exposure
z <- sig_exposure(sig2, type = "relative")  # it is same as sig2$Exposure.norm
```

**Tally a Genomic Alteration Object**

sig_tally

```r
tally
```

```r
tally
```
**Description**

Tally a variation object like MAF, CopyNumber and return a matrix for NMF de-composition and more. This is a generic function, so it can be further extended to other mutation cases. **Please read details about how to set sex for identifying copy number signatures.** Please read [https://osf.io/s93d5/](https://osf.io/s93d5/) for the generation of SBS, DBS and ID (INDEL) components.

**Usage**

```r
sig_tally(object, ...)  
## S3 method for class 'CopyNumber'
  sig_tally(
    object,
    method = "Wang",
    ignore_chrs = NULL,
    indices = NULL,
    add_loh = FALSE,
    feature_setting = sigminer::CN.features,
    cores = 1,
    keep_only_matrix = FALSE,
    ...
  )
  
## S3 method for class 'RS'
  sig_tally(object, keep_only_matrix = FALSE, ...)
  
## S3 method for class 'MAF'
  sig_tally(
    object,
    mode = c("SBS", "DBS", "ID", "ALL"),
    ref_genome = "BSgenome.Hsapiens.UCSC.hg19",
    genome_build = NULL,
    add_trans_bias = FALSE,
    ignore_chrs = NULL,
    use_syn = TRUE,
    keep_only_matrix = FALSE,
    ...
  )
```

**Arguments**

- `object`: a CopyNumber object or MAF object or SV object (from read_sv_as_rs).
- `...`: custom setting for operating object. Detail see S3 method for corresponding class (e.g. CopyNumber).
- `method`: method for feature classification, can be one of "Wang" ("W"), "S" (for method described in Steele et al. 2019).
- `ignore_chrs`: Chromosomes to ignore from analysis. e.g. chrX and chrY.
**sig_tally**

- **indices**: integer vector indicating segments to keep.
- **add_loh**: flag to add LOH classifications.
- **feature_setting**: a data.frame used for classification. **Only used when method is "Wang"** ("W"). Default is `CN.features`. Users can also set custom input with "feature", "min" and "max" columns available. Valid features can be printed by `unique(CN.features$feature)`.
- **cores**: number of computer cores to run this task. You can use `future::availableCores()` function to check how many cores you can use.
- **keep_only_matrix**: if TRUE, keep only matrix for signature extraction. For a MAF object, this will just return the most useful matrix.
- **mode**: type of mutation matrix to extract, can be one of 'SBS', 'DBS' and 'ID'.
- **ref_genome**: 'BSgenome.Hsapiens.UCSC.hg19', 'BSgenome.Hsapiens.UCSC.hg38', 'BSgenome.Mmusculus.UCSC.mm10', 'BSgenome.Mmusculus.UCSC.mm9', etc.
- **genome_build**: genome build 'hg19', 'hg38', 'mm9' or "mm10", if not set, guess it by ref_genome.
- **add_trans_bias**: if TRUE, consider transcriptional bias categories. 'T:' for Transcribed (the variant is on the transcribed strand); 'U:' for Un-transcribed (the variant is on the untranscribed strand); 'B:' for Bi-directional (the variant is on both strand and is transcribed either way); 'N:' for Non-transcribed (the variant is in a non-coding region and is untranslated); 'Q:' for Questionable. **NOTE**: the result counts of 'B' and 'N' labels are a little different from SigProfilerMatrixGenerator, the reason is unknown (may be caused by annotation file).
- **use_syn**: Logical. If TRUE, include synonymous variants in analysis.

**Details**

For identifying copy number signatures, we have to derive copy number features firstly. Due to the difference of copy number values in sex chromosomes between male and female, we have to do an extra step **if we don’t want to ignore them**.

I create two options to control this, the default values are shown as the following, you can use the same way to set (per R session).

```r
options(sigminer.sex = "female", sigminer.copynumber.max = NA_integer_)
```

- If your cohort are all females, you can totally ignore this.
- If your cohort are all males, set `sigminer.sex` to 'male' and `sigminer.copynumber.max` to a proper value (the best is consistent with `read_copynumber`).
- If your cohort contains both males and females, set `sigminer.sex` as a data.frame with two columns "sample" and "sex". And set `sigminer.copynumber.max` to a proper value (the best is consistent with `read_copynumber`).

**Value**

a list contains a matrix used for NMF de-composition.
Methods (by class)

- CopyNumber: Returns copy number features, components and component-by-sample matrix
- RS: Returns genome rearrangement sample-by-component matrix
- MAF: Returns SBS mutation sample-by-component matrix and APOBEC enrichment

Author(s)

Shixiang Wang

References


See Also

sig_estimate for estimating signature number for sig_extract, sig_auto_extract for extracting signatures using automatic relevance determination technique.

Examples

```r
# Load copy number object
load(system.file("extdata", "toy_copynumber.RData",
                package = "sigminer", mustWork = TRUE))

# Use method designed by Wang, Shixiang et al.
cn_tally_W <- sig_tally(cn, method = "W")

# Use method designed by Steele et al.
# See example in read_copynumber

# Prepare SBS signature analysis
laml.maf <- system.file("extdata", "tcga_laml.maf.gz", package = "maftools")
laml <- read_maf(maf = laml.maf)
if (require("BSgenome.Hsapiens.UCSC.hg19")) {
  mt_tally <- sig_tally(laml,
```
An Unified Interface to Extract Signatures

Description

This function provides an unified interface to signature extractor implemented in sigminer. If you determine a specific approach, please also read the documentation of corresponding extractor. See "Arguments" part.

Usage

```r
sig_unify_extract(
  nmf_matrix,
  range = 2:5,
  nrun = 10,
  approach = c("bayes_nmf", "repeated_nmf", "bootstrap_nmf", "sigprofiler"),
  cores = 1L,
  ...
)
```

Arguments

- `nmf_matrix`: a matrix used for NMF decomposition with rows indicate samples and columns indicate components.
- `range`: signature number range, i.e. 2:5.
- `nrun`: the number of iteration to be performed to extract each signature number.
- `approach`: approach name.
  - "repeated_nmf" - sig_extract
simulated_catalogs

- "bayes_nmf" - sig_auto_extract
- "bootstrap_nmf" - bp_extract_signatures
- "sigprofiler" - sigprofiler

Value

Result dependent on the approach setting.

See Also

sig_extract, sig_auto_extract, bp_extract_signatures, sigprofiler

Examples

load(system.file("extdata", "toy_copynumber_tally_W.RData", 
  package = "sigminer", mustWork = TRUE
))
# Extract signatures
# It is same as sig_extract(cn_tally_W$nmf_matrix, 2, nrun = 1)
res <- sig_unify_extract(cn_tally_W$nmf_matrix, 2,
  nrun = 1,
  approach = "repeated_nmf"
)
# Auto-extract signatures based on bayesian NMF
res2 <- sig_unify_extract(cn_tally_W$nmf_matrix,
  nrun = 1,
  approach = "bayes_nmf"
)

simulated_catalogs

A List of Simulated SBS-96 Catalog Matrix

Description

Data from doi: 10.1038/s4301802000275. 5 simulated mutation catalogs are used by the paper but only 4 are available. The data are simulated from COSMIC mutational signatures 1, 2, 3, 5, 6, 8, 12, 13, 17 and 18. Each sample is a linear combination of 5 randomly selected signatures with the addiction of Poisson noise. The number of mutation in each sample is randomly selected between 1,000 and 50,000 mutations, in log scale so that a lower number of mutations is more likely to be selected. The proportion of each signature in each sample is also random.

Format

A list of matrix
**Simulation**

**Source**
Generate from code under data_raw/

**Examples**
data(simulated_catalogs)

---

**Simulation Analysis**

**Description**
- `simulate_signature()` - Simulate signatures from signature pool.
- `simulate_catalogue()` - Simulate catalogs from signature/catalog pool.
- `simulate_catalogue_matrix()` - Simulate a bootstrapped catalog matrix.

**Usage**
simulate_signature(x, weights = NULL)
simulate_catalogue(x, n, weights = NULL)
simulate_catalogue_matrix(x)

**Arguments**

- **x**
a numeric vector representing a signature/catalog or matrix with rows representing signatures/samples and columns representing components.

- **weights**
a numeric vector for weights.

- **n**
an integer indicating mutation number to be generated in a catalog.

**Value**
a matrix.

**Examples**

```r
# Generate a catalog
set.seed(1234)
catalog <- as.integer(table(sample(1:96, 1000, replace = TRUE)))
names(catalog) <- paste0("comp", 1:96)
# Generate a signature
sig <- catalog / sum(catalog)

# Simulate catalogs
x1 <- simulate_catalogue(catalog, 10) # 10 mutations
x1
```
x2 <- simulate_catalogue(catalog, 100) # 100 mutations
x3 <- simulate_catalogue(catalog, 1000) # 1000 mutations
# Similar with a signature
x4 <- simulate_catalogue(sig, 10) # 10 mutations

# Load SBS signature
load(system.file("extdata", "toy_mutational_signature.RData",
    package = "sigminer", mustWork = TRUE
))
s <- t(sig2$Signature.norm)
# Generate a signature from multiple signatures/catalogs
s1 <- simulate_signature(s)
s1
s2 <- simulate_signature(s, weights = 1:3)
s2
# Generate a catalog from multiple signatures/catalogs
c1 <- simulate_catalogue(s, 100, weights = 1:3)
c1

copynumber

### Description

Subsetting CopyNumber object

### Usage

```r
# S3 method for class 'CopyNumber'
subset(x, subset = TRUE, ...)
```

### Arguments

- `x`: a CopyNumber object to be subsetted.
- `subset`: logical expression indicating rows to keep.
- `...`: further arguments to be passed to or from other methods. Useless here.

### Value

- a CopyNumber object

### Author(s)

Shixiang Wang
**transcript.hg19**

**Merged Transcript Location at Genome Build hg19**

**Description**

Merged Transcript Location at Genome Build hg19

**Format**

A data.table

**Source**

from GENCODE release v33.

**Examples**

```r
data(transcript.hg19)
```

---

**transcript.hg38**

**Merged Transcript Location at Genome Build hg38**

**Description**

Merged Transcript Location at Genome Build hg38

**Format**

A data.table

**Source**

from GENCODE release v33.

**Examples**

```r
data(transcript.hg38)
```
transcript.mm10  
*Merged Transcript Location at Genome Build mm10*

**Description**

Merged Transcript Location at Genome Build mm10

**Format**

A data.table

**Source**

from GENCODE release M25.

**Examples**

```r
data(transcript.mm10)
```

---

transcript.mm9  
*Merged Transcript Location at Genome Build mm9*

**Description**

Merged Transcript Location at Genome Build mm9

**Format**

A data.table

**Source**

from UCSC [http://hgdownload.cse.ucsc.edu/goldenPath/mm9/database/transcriptome.txt.gz](http://hgdownload.cse.ucsc.edu/goldenPath/mm9/database/transcriptome.txt.gz)

**Examples**

```r
data(transcript.mm9)
```
transform_seg_table  Transform Copy Number Table

Description
Transform Copy Number Table

Usage
transform_seg_table(
data,
genome_build = c("hg19", "hg38", "mm10", "mm9"),
ref_type = c("cytoband", "gene"),
values_fill = NA,
values_fn = function(x, ...) { round(mean(x, ...)) },
resolution_factor = 1L
)

Arguments

data                    a CopyNumber object or a data.frame containing at least 'chromosome', 'start', 'end', 'segVal', 'sample' these columns.
genome_build             genome build version, used when data is a data.frame, should be 'hg19' or 'hg38'.
ref_type                 annotation data type used for constructing matrix.
values_fill              Optionally, a (scalar) value that specifies what each value should be filled in with when missing. This can be a named list if you want to apply different aggregations to different value columns.
values_fn                Optionally, a function applied to the value in each cell in the output. You will typically use this when the combination of id_cols and value column does not uniquely identify an observation. This can be a named list if you want to apply different aggregations to different value columns.
resolution_factor        an integer to control the resolution. When it is 1 (default), compute frequency in each cytoband. When it is 2, use compute frequency in each half cytoband.

Value
a data.table.
Examples

```r
load(system.file("extdata", "toy_copynumber.RData", 
    package = "sigminer", mustWork = TRUE
))
# Compute the mean segVal in each cytoband
x <- transform_seg_table(cn, resolution_factor = 1)
x
# Compute the mean segVal in each half-cytoband
x2 <- transform_seg_table(cn, resolution_factor = 2)
x2
```

---

**Description**

Set Color Style for Plotting

**Usage**

```r
use_color_style(
    style, 
    mode = c("SBS", "copynumber", "DBS", "ID", "RS"),
    method = "Wang"
)
```

**Arguments**

- `style`: one of 'default' and 'cosmic'.
- `mode`: only used when the style is 'cosmic', can be one of "SBS", "copynumber", "DBS", "ID".
- `method`: used to set a more custom palette for different methods.

**Value**

color values.

**Examples**

```r
use_color_style("default")
use_color_style("cosmic")
```
Index

* **bootstrap**
  - `sig_fit_bootstrap`, 112

  - `add_h_arrow`, 4
  - `add_labels`, 5
  - `bp`, 7
  - `bp_attribute_activity(bp)`, 7
  - `bp_cluster_iter_list(bp)`, 7
  - `bp_cluster_iter_list()`, 10
  - `bp_extract_signatures`, 39, 122
  - `bp_extract_signatures(bp)`, 7
  - `bp_extract_signatures()`, 10, 11
  - `bp_extract_signatures_iter(bp)`, 7
  - `bp_extract_signatures_iter()`, 10
  - `bp_get_clustered_sigs(bp)`, 7
  - `bp_get_rank_score(bp)`, 7
  - `bp_get_sigs_obj(bp)`, 7
  - `bp_get_stats(bp)`, 7
  - `bp_show_survey(bp)`, 7
  - `bp_show_survey2(bp)`, 7

  - `centromeres.hg19`, 14
  - `centromeres.hg38`, 14
  - `centromeres.mm10`, 15
  - `centromeres.mm9`, 15
  - `chromsize.hg19`, 16
  - `chromsize.hg38`, 16
  - `chromsize.mm10`, 17
  - `chromsize.mm9`, 17
  - `circlize::circos.genomicHeatmap`, 62
  - `circlize::circos.genomicLines`, 66
  - `CN.features`, 18, 41, 92, 119
  - `CopyNumber`, 27, 52–54, 58, 61, 63, 64, 69, 118, 124
  - `CopyNumber-Class`, 18
  - `cosine`, 19
  - `cowplot::save_plot()`, 74
  - `cytobands.hg19`, 19
  - `cytobands.hg38`, 20
  - `cytobands.mm10`, 20
  - `cytobands.mm9`, 21
  - `data.table::fread()`, 51
  - `enrich::component_strand_bias`, 21
  - `facet`, 90
  - `future::availableCores()`, 119
  - `geom_boxplot`, 90
  - `get_adj_p`, 22
  - `get_Aneuploidy_score`, 23
  - `get_bayesian_result`, 25, 100
  - `get_cn_freq_table`, 26
  - `get_cn_ploidy`, 24, 26
  - `get_genome_annotation`, 27
  - `get_group_comparison`, 30, 74, 75
  - `get_groups`, 28, 74
  - `get_intersect_size`, 31
  - `get_pLOH_score`, 32
  - `get_shannon_diversity_index`, 33
  - `get_sig_cancer_type_index`, 34
  - `get_sig_db`, 35
  - `get_sig_exposure`, 37
  - `get_sig_exposure()`, 33
  - `get_sig_feature_association`, 38, 43, 89
  - `get_sig_feature_association()`, 43
  - `get_sig_rec_similarity`, 39
  - `get_sig_similarity`, 6, 36, 40, 100
  - `get_tidy_association`, 38, 39, 42, 88, 89
  - `gpar`, 90
  - `ggplot2::annotate`, 6
  - `ggplot2::facet_wrap`, 79
  - `ggpubr::compare_means()`, 22, 75
  - `ggpubr::ggbetweenplot`, 83
  - `ggpubr::ggviolin`, 83
  - `ggpubr::stat_compare_means()`, 22, 75
  - `ggpubr::stat_pvalue_manual()`, 22
group_enrichment, 43, 78, 79
handle_hyper_mutation, 45
hello, 46
legend(), 74
list.files(), 51
MAF, 52, 55, 97, 118
MAF (MAF-class), 46
MAF-class, 46
maftools::read.maf, 53
mean, 77
NMF::nmf(), 108
NMF::nmfEstimateRank, 105
NMF::predict(), 29
output_bootstrap, 47
output_fit, 48
output_sig, 49
output_tally, 50
plot_grid, 63, 65
read_copynumber, 50, 54, 55, 119
read_copynumber(), 53
read_copynumber_ascat, 52
read_copynumber_seqz, 53
read_maf, 52, 53, 55
read_sv_as_rs, 54, 118
read_vcf, 55
read_xena_variants, 56
report_bootstrap_p_value, 57, 114
same_size_clustering, 57
scoring, 58
show_catalogue, 60
show_cn_circos, 61
show_cn_components, 62
show_cn_distribution, 63
show_cn_features, 64
show_cn_freq_circos, 65
show_cn_group_profile, 67
show_cn_profile, 61, 69
show_cor, 70
show_cosmic, 71
show_cosmic_sig_profile, 36, 72
show_group_comparison, 74
show_group_comparison(), 30
show_group_distribution, 76
show_group_enrichment, 44, 78
show_group_mapping, 79
show_groups, 29, 73
show_sig_bootstrap, 81
show_sig_bootstrap_error, 83
show_sig_bootstrap_error(show_sig_bootstrap), 81
show_sig_bootstrap_exposure, 83, 90
show_sig_bootstrap_exposure(show_sig_bootstrap), 81
show_sig_bootstrap_stability, 83
show_sig_bootstrap_stability(show_sig_bootstrap), 81
show_sig_consensusmap, 85
show_sig_exposure, 86
show_sig_feature_corrplot, 71, 87
show_sig_fit, 89
show_sig_number_survey(sig_estimate), 103
show_sig_number_survey(), 7
show_sig_number_survey2, 106
show_sig_number_survey2(sig_estimate), 103
show_sig_profile, 5, 61, 73, 91, 97
show_sig_profile(), 94
show_sig_profile_heatmap, 93, 94
show_sig_profile_loop, 93, 96
sig_attrs(sig_operation), 116
sig_auto_extract, 12, 25, 28, 37, 86, 92, 95, 96, 98, 99, 102, 106, 108, 109, 111, 113, 117, 120, 122
sig_convert, 102
sig_estimate, 12, 101, 103, 105–108, 120
sig_estimate(), 105
sig_exposure(sig_operation), 116
sig_extract(), 28, 29, 101
sig_fit, 7, 28, 36, 37, 48, 74, 83, 89, 90, 101, 107, 109, 114, 116
sig_fit(), 73
sig_fit_bootstrap, 57, 83, 90, 110, 111, 112, 115, 116
sig_fit_bootstrap_batch, 47, 82, 83, 90, 111, 114, 115
sig_modify_names(sig_operation), 116
INDEX

sig_names(sig_operation), 116
sig_number(sig_operation), 116
sig_operation, 116
sig_signature(sig_operation), 116
sig_tally, 21, 60, 63, 65, 92, 93, 101, 108, 117
sig_tally(), 50
sig_unify_extract, 121
sigminer, 97
sigprofiler, 98, 122
sigprofiler_extract, 12
sigprofiler_extract(sigprofiler), 98
sigprofiler_import(sigprofiler), 98
simulate_catalogue(simulation), 123
simulate_catalogue_matrix(simulation), 123
simulate_signature(simulation), 123
simulated_catalogs, 122
simulation, 123
stats::aov, 30
stats::fisher.test, 30
stats::median, 77
stats::p.adjust, 43, 71
stats::p.adjust(), 22
stats::TukeyHSD, 30
subset.CopyNumber, 124

transcript.hg19, 125
transcript.hg38, 125
transcript.mm10, 126
transcript.mm9, 126
transform_seg_table, 127

use_color_style, 128