Package: MIPanalyzer (via r-universe)

June 25, 2024

Type Package

Title Filtering and analysis of MIP data

Version 1.1.0

Description Filtering and analysis of MIP data.

License MIT + file LICENSE

Encoding UTF-8

LinkingTo Rcpp

Imports Rcpp, vcfR, dplyr, reshape2, RColorBrewer, plotly, tibble, ggplot2, ape, stringr, methods, rworldmap, coda, data.table

RoxygenNote 7.2.3

SystemRequirements C++11

Suggests testthat, knitr, rmarkdown, here, kableExtra, tidyverse

VignetteBuilder knitr

Repository https://plasmogenepi.r-universe.dev

RemoteUrl https://github.com/mrc-ide/MIPanalyzer

RemoteRef HEAD

RemoteSha 72f692e8c70b607f8ae65dfaf3d2e417acb8582d

Contents

`check_MIPanalyzer_loaded` *Check that MIPanalyzer package has loaded successfully*

Description

Simple function to check that MIPanalyzer package has loaded successfully.

Usage

```
check_MIPanalyzer_loaded()
```

`explore_filter_coverage_loci` *Explore locus coverage prior to filtering*

Description

Explore what effect the `filter_coverage_loci()` function will have on the data without actually applying any filters. Can be used to set coverage thresholds.

Usage

```
explore_filter_coverage_loci(
  x,
  min_coverage = 5,
  max_low_coverage = 50,
  breaks = 100
)
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`min_coverage`	the coverage threshold below which data is deemed to be low-coverage.
`max_low_coverage`	(percentage). Loci are not allowed to contain more than this many low-coverage samples. In the `filter_coverage_loci()` function, any locus with more than `max_low_coverage` low-coverage samples will be dropped.
`breaks`	number of breaks spanning the range `[0,100]`.

`explore_filter_coverage_samples` *Explore sample coverage prior to filtering*

Description

Explore what effect the `filter_coverage_samples()` function will have on the data without actually applying any filters. Can be used to set coverage thresholds.

Usage

```
explore_filter_coverage_samples(
  x,
  min_coverage = 5,
  max_low_coverage = 50,
  breaks = 100
)
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`min_coverage`	the coverage threshold below which data is deemed to be low-coverage.
`max_low_coverage`	(percentage). Samples are not allowed to contain more than this many low-coverage loci. In the `filter_coverage_samples()` function, any sample with more than `max_low_coverage` low-coverage loci will be dropped.
`breaks`	number of breaks spanning the range `[0,100]`.

`filter_counts` *Filter alleles based on raw counts*

Description

Drop any allele for which the number of read counts is below a given threshold. Coverage is adjusted to account for dropped reads.

Usage

```
filter_counts(
  x,
  count_min = 2,
  description = "filter individual allele counts"
)
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`count_min`	alleles with fewer than this many counts are dropped.
`description`	brief description of the filter, to be saved in the filter history.

`filter_coverage_loci` *Filter loci based on coverage*

Description

Set a coverage threshold: any coverage value below this threshold is deemed to be low-coverage. Then set a maximum percent low-coverage samples per locus: any locus with greater than this percentage low-coverage samples is dropped. Note that threshold values can be explored without applying any filtering using the `explore_filter_coverage_loci()` function.

Usage

```
filter_coverage_loci(
  x,
  min_coverage = 5,
  max_low_coverage = 50,
  replace_low_coverage = FALSE,
  description = "filter loci based on coverage"
)
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`min_coverage`	the coverage threshold below which data is deemed to be low-coverage.
`max_low_coverage`	any locus with more than `max_low_coverage` percent of low-coverage samples will be dropped.
`replace_low_coverage`	(Boolean). If `TRUE` then any remaining low-coverage loci will be replaced with `NA`.
`description`	brief description of the filter, to be saved in the filter history.

`filter_coverage_samples` *Filter samples based on coverage*

Description

Set a coverage threshold: any coverage value below this threshold is deemed to be low-coverage. Then set a maximum percent low-coverage loci per sample: any sample with greater than this percentage low-coverage loci is dropped. Note that threshold values can be explored without applying any filtering using the `explore_filter_coverage_samples()` function.

Usage

```
filter_coverage_samples(
  x,
  min_coverage = 5,
  max_low_coverage = 50,
  replace_low_coverage = FALSE,
  description = "filter samples based on coverage"
)
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`min_coverage`	the coverage threshold below which data is deemed to be low-coverage.
`max_low_coverage`	any sample with more than `max_low_coverage` percent of low-coverage loci will be dropped.
`replace_low_coverage`	(Boolean). If `TRUE` then any remaining low-coverage loci will be replaced with `NA`.
`description`	brief description of the filter, to be saved in the filter history.

`filter_loci` *Filter out some loci*

Description

Filter out some loci.

Usage

```
filter_loci(x, locus_filter, description = "")
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`locus_filter`	boolean vector specifying whether to keep (`TRUE`) or drop (`FALSE`) each locus.
`description`	brief description of the filter, to be saved in the filter history.

`filter_loci_invariant` *Filter loci to drop invariant sites*

Description

Filter loci to drop invariant sites.

Usage

```
filter_loci_invariant(x, description = "filter loci to drop invariant sites")
```

Arguments

`x`	object of class `mipanalyzer_multiallelic`.
`description`	brief description of the filter, to be saved in the filter history.

`filter_overcounts` *Filter out over-counts*

Description

Filter out over-counts, defined as count > coverage. Replace any such element with NA.

Usage

```
filter_overcounts(x, description = "replace overcounts with NA")
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`description`	brief description of the filter, to be saved in the filter history.

`filter_samples` *Filter out some samples*

Description

Filter out some samples.

Usage

```
filter_samples(x, sample_filter, description = "")
```

Arguments

`x`	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.
`sample_filter`	boolean vector specifying whether to keep (`TRUE`) or drop (`FALSE`) each sample.
`description`	brief description of the filter, to be saved in the filter history.

`filter_wsaf` *Filter alleles based on within-sample allele frequencies*

Description

Drop any allele for which the within-sample allele frequency (WSAF) is below a givin threshold. Thresholds apply in both directions, for example if `wsaf_min = 0.01` then alleles with a WSAF less than 0.01 *or* greater than 0.99 will be rounded to 0 or 1, respectively. Coverage is adjusted to account for dropped reads.

Usage

```
filter_wsaf(x, wsaf_min = 0.01, description = "filter individual allele WSAF")
```

Arguments

`x`	object of class `mipanalyzer_multiallelic`.
`wsaf_min`	alleles with counts that make a WSAF less than this threshold are dropped.
`description`	brief description of the filter, to be saved in the filter history.

`get_genomic_distance` *Get genomic distance between samples*

Description

Get genomic distance between samples using a distance metric that allows for mixed infections and takes account of linkage (see references for details).

Usage

```
get_genomic_distance(x, cutoff = 0.1, report_progress = TRUE)
```

Arguments

`x`	object of class `mipanalyzer_biallelic`.
`cutoff`	when calculating weights, correlations below this value are ignored (see references).
`report_progress`	if `TRUE` then a progress bar is printed to the console.

References

MalariaGEN Plasmodium falciparum Community Project. "Genomic epidemiology of artemisinin resistant malaria". eLIFE (2016).

`get_IBS_distance` *Get identity by state (IBS) distance*

Description

Get identity by state (IBS) distance, computed as the proportion of sites that are identical between samples. If `ignore_het` = `TRUE` then heterozygous sites are ignored, otherwise the major strain is called at every locus.

Usage

```
get_IBS_distance(x, ignore_het = TRUE, diagonal = NULL, report_progress = TRUE)
```

Arguments

`x`	object of class `mipanalyzer_biallelic`.
`ignore_het`	whether to ignore heterozygous comparisons, or alternatively call the major allele at every locus (see details).
`diagonal`	Should the diagonal of the distance matrix be changed to a given value. Default = `NULL`, which cause no changes.
`report_progress`	if `TRUE` then a progress bar is printed to the console.

`get_IB_mixture` *Get identity by mixture*

Description

Get identity by "mixture distance". The mixture distance between two samples is the proportion of loci that have identical within-sample allele frequencies (WSAFs), or alternatively have WSAFs within a given tolerance. This extends the idea of identity by state (IBS) to continuous WSAFs rather than categorical genotypes.

Usage

```
get_IB_mixture(x, tol = 0, diagonal = NULL, report_progress = TRUE)
```

Arguments

`x`	object of class `mipanalyzer_biallelic`.
`tol`	tolerance on mixture comparisons. Default = 0.
`diagonal`	Should the diagonal of the distance matrix be changed to a given value. Default = NULL, which cause no changes.
`report_progress`	if `TRUE` then a progress bar is printed to the console.

get_spatial_distance *Get great circle distance between spatial points*

Description

Get great circle distance between spatial points.

Usage

```
get_spatial_distance(lat, long)
```

Arguments

`lat`	vector of latitudes.
`long`	vector of longitudes.

get_wsaf *Get within-sample allele frequencies*

Description

Get within-sample allele frequencies from coverage and count data. Missing values can optionally be imputed by applying a summary function to the non NA values at each locus. The default summary function takes the mean of the non NA values.

Usage

```
get_wsaf(x, impute = TRUE, FUN = median, ...)
```

Arguments

`x`	object of class `mipanalyzer_biallelic`.
`impute`	whether to impute missing values.
`FUN`	function used to impute missing values. Default = ‘median‘
`...`	other arguments to pass to `FUN`.

`inbreeding_mle` *Estimate pairwise inbreeding coefficient F by maximum likelihood*

Description

Estimates the inbreeding coefficient between all pairs of samples by maximum likelihood.

Usage

```
inbreeding_mle(
  x,
  f = seq(0, 1, l = 11),
  ignore_het = FALSE,
  report_progress = TRUE
)
```

Arguments

`x`	object of class `mipanalyzer_biallelic`.
`f`	values of f that are explored.
`ignore_het`	whether to ignore heterzygous comparisons, or alternatively call the major allele at every locus (see details).
`report_progress`	if `TRUE` then a progress bar is printed to the console.

Details

The probability of seeing the same or different alleles at a locus can be written in terms of the global allele frequency p and the inbreeding coefficient f, for example the probability of seeing the same REF allele is $(1-f)*p^2+f*p$. This formula can be multiplied over all loci to arrive at the overall likelihood of each value of f, which can then be chosen by maximum likelihood. This function carries out this comparison between all pairwise samples, passed in as a matrix. The formula above only applies when comparing homozygous calls - for homo/het or het/het comparisons we can either ignore these loci (the default) or convert hets to homo by calling the major allele at every locus.

`is.mipanalyzer_biallelic` *Determine if object is of class mipanalyzer_biallelic*

Description

Determine if object is of class `mipanalyzer_biallelic`.

Usage

```
is.mipanalyzer_biallelic(x)
```

Arguments

`x`	object of class `mipanalyzer_biallelic`

`is.mipanalyzer_multiallelic` *Determine if object is of class mipanalyzer_multiallelic*

Description

Determine if object is of class `mipanalyzer_multiallelic`.

Usage

```
is.mipanalyzer_multiallelic(x)
```

Arguments

`x`	object of class `mipanalyzer_multiallelic`

`lonlat_to_bearing` *Calculate great circle distance and bearing between coordinates*

Description

Calculate great circle distance and bearing between spatial coordinates.

Usage

```
lonlat_to_bearing(origin_lon, origin_lat, dest_lon, dest_lat)
```

Arguments

`origin_lon`	The origin longitude
`origin_lat`	The origin latitude
`dest_lon`	The destination longitude
`dest_lat`	The destination latitude

Examples

```
# one degree longitude should equal approximately 111km at the equator
lonlat_to_bearing(0, 0, 1, 0)
```

`MIPanalyzer` *MIPanalyzer*

Description

This package can be used to read in raw molecular inversion probe (MIP) data from vcf into a format that is convenient to work with. Data can be filtered based on counts, frequencies, missingness or other criteria. Filtered data can be analysed by common methods including PCA and various pairwise genetic metrics, and can be visualised in multiple ways. This package is intended to evolve as new MIP analyses are needed, thereby making it easy to repeat common analyses as new data becomes available.

Filtering and analysis of MIP data.

Author(s)

Maintainer: Bob Verity `<r.verity@imperial.ac.uk>`

`mipanalyzer_biallelic_to_vcfR` *Converts a MIPanalyzer biallelic object to vcfR*

Description

Converts an object of class `mipanalyzer_biallelic` to `vcfR` format.

Usage

```
mipanalyzer_biallelic_to_vcfR(input = NULL, cutoff = 0.1)
```

Arguments

`input`	an object of class `mipanalyzer_biallelic`.
`cutoff`	the within-sample non-referent allele frequency cutoff to transform your biallelic site to a genotype matrix.

`mipanalyzer_file` *Load system file*

Description

Load a file from within the inst/extdata folder of the MIPanalyzer package. File extension must be one of .csv, .txt, or .rds.

Usage

```
mipanalyzer_file(name)
```

Arguments

`name`	the name of a file within the inst/extdata folder.

`pca_wsaf` *PCA of within-sample allele frequencies*

Description

Conduct principal components analysis (PCA) on a matrix of within-sample allele frequencies (WSAF). Missing values must have been already imputed. Output includes the raw components, the variance in the data explained by each component, and the loadings of each component also returned.

Usage

```
pca_wsaf(x)
```

Arguments

`x`	a matrix of within-sample allele frequencies, as produced by the function `get_wsaf()`.

Details

Contributions of each variable are computed from the loading values (stored as "rotation" within the `prcomp` object). The percent contribution of a variable is defined as the absolute loading value for this variable, divided by the sum of loadings over all variables and multiplied by 100.

Value

Invisibly returns a list of class ‘prcomp‘ with the following components

- "sdev" the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix).
- "rotation" the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). The function `princomp` returns this in the element `loadings`.
- "center, scale" the centering and scaling used.
- "x" the value of the rotated data (the centred data multiplied by the rotation matrix). Hence, `cov(x)` is the diagonal matrix `diag(sdev^2)`.
- "var" the variance in the data explained by each component.
- "contribution" the percent contribution of a variable (i.e. a locus) to the overall variation.

`pcoa_genomic_distance` *PCoA of genomic distances between samples*

Description

Conduct principal coordinate analysis (PCoA) on a matrix of genomic distances.

Usage

```
pcoa_genomic_distance(x)
```

Arguments

`x`	matrix of genomic distances, as produced by the function `get_genomic_distance()`.

`Pf_chrom_lengths` *Get dataframe of P.falciparum chromosome lengths*

Description

Get dataframe of P.falciparum chromosome lengths

Usage

```
Pf_chrom_lengths()
```

plot_coverage *Plot coverage matrix*

Description

Plot matrix of coverage over all samples and loci.

Usage

```
plot_coverage(x)
```

Arguments

x	object of class `mipanalyzer_biallelic` or `mipanalyzer_multiallelic`.

plot_distance *Plot a distance matrix*

Description

Simple image plot of a matrix of pairwise distances.

Usage

```
plot_distance(m, col_pal = "plasma")
```

Arguments

m	square matrix of pairwise distances.
col_pal	which viridis colour pallet to use. Options are "viridis", "plasma", "magma" or "inferno".

`plot_map` *Produce ggplot map*

Description

Produce ggplot map.

Usage

```
plot_map(
  x_limits = c(12, 35),
  y_limits = c(-13, 5),
  col_country = grey(0.3),
  col_country_border = grey(0.5),
  size_country_border = 0.5,
  col_sea = grey(0.1),
  resolution = "coarse"
)
```

Arguments

`x_limits`	longitude limits of map.
`y_limits`	latitude limits of map.
`col_country`	fill colour of countries.
`col_country_border`	colour of country borders.
`size_country_border`	size of country borders.
`col_sea`	fill colour of sea.
`resolution`	the resolution of the underlying map. Must be one of "coarse", "low", "less", "islands", "li", "high".

`plot_pca` *Plot PCA*

Description

Plots either the first 2 or 3 principal components.

Usage

```
plot_pca(
  pca,
  num_components = 2,
  col = NULL,
  col_palette = NULL,
  ggplot = FALSE
)
```

Arguments

`pca`	output of `pca_wsaf()` function.
`num_components`	numeric for number of components used. Default = 2.
`col`	vector by which samples are coloured.
`col_palette`	vector of colours for each group.
`ggplot`	boolean for plotting using ggplot. Default = FALSE

`plot_pca_contribution` *Plot PCA contribution of each variable*

Description

Plot PCA contribution of each variable.

Usage

```
plot_pca_contribution(
  pca,
  component = 1,
  chrom,
  pos,
  locus_type = NULL,
  y_buffer = 0
)
```

Arguments

`pca`	output of `pca_wsaf()` function.
`component`	which component to plot.
`chrom`	the chromosome corresponding to each contribution value.
`pos`	the genomic position corresponding each contribution value.
`locus_type`	defines the colour of each bar.
`y_buffer`	(percent). A buffer added to the bottom of each y-axis, making room for other annotations to be added.

`plot_pca_variance` *Plot variance explained by PCA components*

Description

Plot the variance explained by each PCA component. The number of components shown is controlled by `num_components`, with up to the first 10 componenets shown by default. If less than the requested number of components exist, then all the components will be shown.

Usage

```
plot_pca_variance(pca, num_components = 10)
```

Arguments

`pca`	output of `pca_wsaf()` function.
`num_components`	maximum components to be shown.

`plot_pcoa` *Plot PCoA*

Description

Plots either the first 2 or 3 vectors of PCoA.

Usage

```
plot_pcoa(pcoa, num_components = 2, col = NULL, col_palette = NULL)
```

Arguments

`pcoa`	object of class "pcoa", as produced by `pcoa_wsaf()` function.
`num_components`	numeric for number of components used. Default = 2.
`col`	vector by which samples are coloured.
`col_palette`	vector of colours for each group.

`plot_wsaf` *Plot within-sample allele frequencies*

Description

Simple image plot of within-sample allele frequencies. The top row of the plot corresponds to the first row of the input matrix.

Usage

```
plot_wsaf(x, col_pal = "plasma")
```

Arguments

`x`	matrix of within-sample allele frequencies, with samples in rows and loci in columns.
`col_pal`	which viridis colour pallet to use. Options are "viridis", "plasma", "magma" or "inferno".

`print.mipanalyzer_biallelic` *Custom print function for class mipanalyzer_biallelic*

Description

Custom print function for class `mipanalyzer_biallelic`, printing a summary of the key elements (also equivalent to `summary(x)`). To do an ordinary `print()`, use the `print_full()` function.

Usage

```
## S3 method for class 'mipanalyzer_biallelic'
print(x, ...)
```

Arguments

`x`	object of class `mipanalyzer_biallelic`
`...`	other arguments (ignored)

`print.mipanalyzer_multiallelic` *Custom print function for class mipanalyzer_multiallelic*

Description

Custom print function for class `mipanalyzer_multiallelic`, printing a summary of the key elements (also equivalent to `summary(x)`). To do an ordinary `print()`, use the `print_full()` function.

Usage

```
## S3 method for class 'mipanalyzer_multiallelic'
print(x, ...)
```

Arguments

`x`	object of class `mipanalyzer_multiallelic`
`...`	other arguments (ignored)

`rbetabinom` *Draw from Beta-binomial distribution*

Description

Draw from Beta-binomial distribution.

Usage

```
rbetabinom(n = 1, k = 10, alpha = 1, beta = 1)
```

Arguments

`n`	number of draws.
`k`	number of binomial trials.
`alpha`	first shape parameter of beta distribution.
`beta`	second shape parameter of beta distribution.

`rdirichlet` *Draw from Dirichlet distribution*

Description

Draw from Dirichlet distribution given a vector of shape parameters.

Usage

```
rdirichlet(shape = rep(1, 3))
```

Arguments

`shape`	vector of shape parameters.

`sim_biallelic` *Simulate biallelic data*

Description

Simulate biallelic data from a simple statistical model. Inputs include the complexity of infection (COI), population-level allele frequencies (PLAF) and some parameters dicating skew and error distributions. Outputs include the phased haplotypes and the un-phased read count and coverage data.

Usage

```
sim_biallelic(
  COI = 3,
  PLAF = runif(10, 0, 0.5),
  coverage = 100,
  alpha = 1,
  overdispersion = 0,
  epsilon = 0
)
```

Arguments

`COI`	complexity of infection.
`PLAF`	vector of population-level allele frequencies at each locus.
`coverage`	coverage at each locus. If a single value then the same coverage is applied over all loci.
`alpha`	shape parameter of the symmetric Dirichlet prior on strain proportions.

`overdispersion`	the extent to which counts are over-dispersed relative to the binomial distribution. Counts are Beta-binomially distributed, with the beta distribution having shape parameters `p/overdispersion` and `(1-p)/overdispersion`.
`epsilon`	the probability of a single read being mis-called as the other allele. Applies in both directions.

Details

Simulated data are drawn from a simple statistical model:

1. Strain proportions are drawn from a symmetric Dirichlet distribution with shape parameter `alpha`.
2. Phased haplotypes are drawn at every locus, one for each `COI`. The allele at each locus is drawn from a Bernoulli distribution with probability given by the `PLAF`.
3. The "true" within-sample allele frequency at every locus is obtained by multiplying haplotypes by their strain proportions, and summing over haplotypes. Errors are introduced through the equation
$$wsaf_error = wsaf * (1 - e) + (1 - wsaf) * e$$
where $wsaf$ is the WSAF without error and e is the error parameter `epsilon`.
4. Final read counts are drawn from a beta-binomial distribution with expectation w_error. The raw number of draws is given by the `coverage`, and the skew of the distribution is given by the `overdispersion` parameter. If `overdispersion = 0` then the distribution is binomial, rather than beta-binomial.

`summary.mipanalyzer_biallelic`
Print summary for class mipanalyzer_biallelic

Description

Custom summary function for class `mipanalyzer_biallelic`.

Usage

```
## S3 method for class 'mipanalyzer_biallelic'
summary(object, ...)
```

Arguments

`object`	object of class `mipanalyzer_biallelic`
`...`	other arguments (ignored)

`summary.mipanalyzer_multiallelic` *Print summary for class mipanalyzer_multiallelic*

Description

Custom summary function for class `mipanalyzer_multiallelic`.

Usage

```
## S3 method for class 'mipanalyzer_multiallelic'
summary(object, ...)
```

Arguments

`object`	object of class `mipanalyzer_multiallelic`
`...`	other arguments (ignored)

`vcf_to_mipanalyzer_biallelic` *Convert vcf to biallelic mipanalyzer data class*

Description

Convert vcf to biallelic mipanalyzer data class.

Usage

```
vcf_to_mipanalyzer_biallelic(file = NULL, vcfR = NULL, verbose = TRUE)
```

Arguments

`file`	path to vcf file.
`vcfR`	object of class `vcfR`.
`verbose`	if reading from file, whether to read in verbose manner.

`vcf_to_mipanalyzer_multiallelic` *Convert vcf to multiallelic mipanalyzer data class*

Description

Convert vcf to multiallelic mipanalyzer data class.

Usage

```
vcf_to_mipanalyzer_multiallelic(file = NULL, vcfR = NULL, verbose = TRUE)
```

Arguments

`file`	path to vcf file.
`vcfR`	object of class `vcfR`.
`verbose`	if reading from file, whether to read in verbose manner.

Index