Package 'CNVgears'

October 18, 2022

Type Package

Title A Framework of Functions to Combine, Analize and Interpret CNVs Calling Results

Version 1.4.0

Description This package contains a set of functions to perform several type of processing and analysis on CNVs calling pipelines/algorithms results in an integrated manner and regardless of the raw data type (SNPs array or NGS). It provides functions to combine multiple CNV calling results into a single object, filter them, compute CNVRs (CNV Regions) and inheritance patterns, detect genic load, and more. The package is best suited for studies in human family-based cohorts.

Depends R (>= 4.1), data.table

Imports ggplot2

Suggests VariantAnnotation, DelayedArray, knitr, biomaRt, evobiR, rmarkdown, devtools, cowplot, usethis, scales, testthat, GenomicRanges, cn.mops, R.utils

License GPL-3

Encoding UTF-8

LazyData true

biocViews Software, WorkflowStep, Preprocessing

BugReports https://github.com/SinomeM/CNVgears/issues

RoxygenNote 7.1.1

VignetteBuilder knitr

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

git_branch RELEASE_3_15

git_last_commit 84c8857

git_last_commit_date 2022-04-26

Date/Publication 2022-10-18

Author Simone Montalbano [cre, aut]

Maintainer Simone Montalbano <simone.montalbano@protonmail.com>

R topics documented:

`chr_uniform` *Uniform chromosome notation*

Description

This is a function for internal use in the package, it handles the standardize process of chromosome notation within the other functions.

Usage

```
chr_uniform(DT_in)
```

Arguments

DT_in,	a data.table with a columns named "chr"

Value

the same data.table in input with the "chr" uniformed to the notation "1", "2", ... , "23", "24" for the chromosomes 1:22, X and Y

Examples

```
DT <- data.table::data.table(chr= c("chr1", "chrX", "chr20"))
DT <- chr_uniform(DT)
DT
```

---

cleaning_filter *Filter CNVs calls based on several parameters*

Description

cleaning_filter "clean" a CNV calls dataset based on measures such as length, number of calls per sample and more.

Usage

```
cleaning_filter(
  results,
  min_len = 10000,
  min_NP = 10,
  blacklists = NULL,
  blacklist_samples = NULL,
  blacklist_chrs = NULL
)
```

Arguments

results,	a data.table, the output of read_results.
min_len,	minimum CNVs length, any shorter event will be filtered out. Default is 10000.
min_NP,	minimum CNVs points, any shorter event will be filtered out. Default is 10.
blacklists,	blacklist, a list of data.table or data.frame containing at least the following columns: "chr", "start", "end". Any event in a blacklists' region will be filtered out.
blacklist_samples,	character vector containing samples ID to filter out.
blacklist_chrs,	character vector containing chromosomes names in the package format, 1:22 for autosomes and 23 24 for chr X and Y.

Details

This function can be used together with summary in order to clean the dataset from possible noise and unwanted calls. It is generally recommended to briefly explore the data using summary and then proceeding to filter out any unwanted group of events. Mandatory arguments of the function are "results" and "min_len"/"min_NP", default values are the authors suggested minimal filtering step, however its quite common to filter anything shorter than 10 or even 50 kb, and/or any call made by less than 10 points. The use of blacklist of any kind is optional and should be done with caution, as it can filter potential biologically relevant events. Over-segmented samples the user wishes to exclude can be specified via blacklist_samples. Immunoglobulin regions can be generated with the function immuno_regions, while telomeric and/or centromeric regions can be obtained with telom_centrom.

Value

the CNVresults object results after the selected filters have been applied.

Examples

```
DT <- cleaning_filter(penn_22, min_len = 50000)
```

---

cnmops_to_CNVresults *Convert cn.mops results into* CNVgears *format*

Description

Convert cn.mops results into CNVgears format

Usage

```
cnmops_to_CNVresults(cnRes, sample_list, markers)
```

Arguments

cnRes	cn.mops results (after integer CN calling)
sample_list	minimal cohort metadata, a data.table produced by the function read_metadt.
markers	a data.table containing the marker list, the output read_finalreport_snps with DT_type set to "markers" or read_NGS_intervals.

Value

the input object cnRes converted into CNVresults

Examples

```
library(cn.mops)
data(cn.mops)
resCNMOPS <- cn.mops(XRanges)
resCNMOPS <- calcIntegerCopyNumbers(resCNMOPS)
resCNMOPS_cnvs <- cnvs(resCNMOPS)
# cnmops_calls <- cnmops_to_CNVresults(resCNMOPS_cnvs, sample_list, markers)
```

`CNVgears` *CNVgears: A package to analyze CNVs calling/segmentation results*

Description

`CNvgears` provides several functions to analyze the results of CNVs calling and/or segmentation on SNPs arrays or NGS data.

Details

The CNVgears package provides several functions useful in order to perform a series of analysis the result of CNVs calling or segmentation pipelines or algorithms, on both Ilummina SNP array (e.g. PennCNV, iPattern or EnsembleCNV) and NGS data (e.g. ModSeg and gCNV pipelines from GATK), in an integrated framework. To do so all the data is imported in a standardized manner, allowing the user to perform analysis and data manipulation regardless of the initial raw data type, from (among the others) CNVRs creation and exclusion of immunoglobulin regions, to de novo CNVs discovery and genic content annotation.

It has been originally developed for the CNVs characterization of the Italian Autism Network (ITAN) collection (DOI: 10.1186/s12888-018-1937-y).

Analysis pipelines examples

Here are briefly illustrated some workflow examples that can be done either interactively on sequentially. See the vignettes for further details.

Staring from the results of gCNV and ModSeg pipelines on WES data in a cohort of families:

1. load the intervals list (using `read_NGS_intervals`);
2. load samples table with minimal metadata (sample ID, sex, role, family ID);
3. load the segmentation results of all the samples in the cohort, for each pipeline separately;
4. merge adjacent segments (with equal CN);
5. filter out CNVs in immunoglobulin (IG) regions;
6. find eventual oversegmented samples (can be marked or excluded from the analysis);
7. find replicated segments in the pipelines and merge the results into a single `data.table`;
8. create the CNVRs;
9. exclude common CNVs based on the CNVRs frequency;
10. annotate genic contents of the CNVs

11. find the inheritance pattern of a selected subset of events (or the whole dataset) in the offspring, based on the segments of the parents;
12. fine-screen putative de novo calls using the per-interval raw data (copy ratio or LRR like) of the trio;
13. visualize the good de novo candidate per point raw data in the family to visually confirm the inheritance pattern.

CNVgears functions

The CNVgears functions are organized in groups:

- input/output
- filtering
- CNVRs
- inter results comparison and merging
- de novo discovery/inheritance pattern detection
- plotting

CNVresults_to_GRanges *CNVresults to GRanges*

Description

`CNVresults_to_GRanges` convert `CNVresults` objects into `GRanges`

Usage

```
CNVresults_to_GRanges(DT)
```

Arguments

`DT`	a `CNVresults` object.

Details

A simple wrapper for the function `GenomicRanges::makeGRangesFromDataFrame`. Retained metadata columns are: sample_ID, GT and meth_ID.

Value

the input object `DT` converted into `GRanges`.

Examples

```
GR <- CNVresults_to_GRanges(penn_22)
```

cnvrs_create *Compute Copy Number Variable Regions (CNVRs)*

Description

`cnvrs_create` compute CNVRs on a `CNVResutls` object, typically the output of `inter_res_merge`.

Usage

```
cnvrs_create(cnvs, chr_arms, prop = 0.3)
```

Arguments

`cnvs`	a `CNVresults` produced by `read_results`.
`chr_arms`	a `data.table` containing chromosomal arms locations. They are bundled in the package for hg18, hg19 and hg38 (`hgXX_chr_arms`).
`prop`	reciprocal overlap proportion, default 0.3 (30%).

Details

Copy Number Variable Regions (CNVRs) are defined as groups of reciprocal overlapping CNVs. This function first try to assign every call to a CNVR (or create a new one if it is not possible), then check if adjacent CNVRs cam be merged, and finally recheck all the CNVs in each CNVRs and un-assign them if the reciprocal overlap is no longer fulfilled with all the member of the CNVR. If any event is touched by this last step, a new cycle begins, this until no more CNVs can be removed from the assigned CNVR.

Value

a `list` of two elements. The first element is a `data.table` that contains the actual CNVR information, genomic location and frequency in the cohort. The second element is the `CNVresults`

Examples

```
DT <- cnvrs_create(penn_22, hg19_chr_arms)
```

cnvs_inheritance *Compute CNVs inheritance*

Description

`cnvs_inheritance` compute CNVs inheritance pattern and search good de novo CNVs candidates

Usage

```
cnvs_inheritance(
  sample_list,
  markers,
  results,
  raw_path,
  mmethod = 1,
  alfa = 0.05,
  min_NP = 10,
  adjust_pval = TRUE,
  reciprocal_overlap = 0.3
)
```

Arguments

sample_list	a data.table, the output of read_metadt.
markers	a data.table, the output of read_NGS_intervals or read_finalreport_snps, depending on the initial data type.
results	a CNVresults object, the output of read_results or inter_res_merge.
raw_path	character, the path where raw data processed by read_NGS_raw or read_finalreport_raw.
mmethod	fine-screening method.
alfa	p value alfa value, default in 0.05.
min_NP	minimum number of points to try fine screening a de novo candidate
adjust_pval	logical compute adjusted p-value?
reciprocal_overlap	minimum reciprocal overlap for inherited CNVs detection

Details

Given a trio (mother, father, offspring) this function computes inheritance patterns of the offspring's CNVs. This is done both by comparing the actual calls in the trio (CNVs-level) and the raw data (markers-level). For this reason it is suggested to use it separately on each method and then combine the results. As an example if the focus are de novo CNVs, select only those and then combine the outputs using inter_res_merge. In this way it also possible to further increase the confidence in the results, e.g. by filtering out all de novo candidates called by a single method.

Internally the function is structured in several steps. [...]

Value

a CNVresults object with some additional columns: inheritance and related p-values.

cohort_examples *Sample list for the runnable examples. It comes from CNV calling on array data from a subset of the 1000 Genomes project, see vignettes for more details.*

Description

Sample list for the runnable examples. It comes from CNV calling on array data from a subset of the 1000 Genomes project, see vignettes for more details.

Usage

```
cohort_examples
```

Format

An object of class `data.table` (inherits from `data.frame`) with 23 rows and 4 columns.

genic_load *Annotate genic load*

Description

Annotate genic load

Usage

```
genic_load(DT_in, biotypes = "protein_coding", mart = NULL)
```

Arguments

`DT_in,`	a `data.table` consisting of CNVs calling results.
`biotypes,`	character vector of Genecode biotypes, default value is "protein_coding".
`mart,`	user specified `biomaRt::useMart()` object. Used if DT_in is assembly (e.g. hg19) require older Ensembl releases. If NA the latest release available by `biomaRt::useMart()` is used. This function takes a `CNVresults` object as input and add two additional columns representing the genic content of each call, i.e. "genes" and "n_genes". The genes considered can be changed using the `biotypes` parameter depending on which types of genes is the user interested in. At the moment, genes a stored as a "-" separated list of Ensembl IDs.

Value

the `CNVresults` object `DT_in` with additional columns: genes and n_genes.

Examples

```
## Not run:
DT <- genic_load(penn_22)

## End(Not run)
```

---

genomic_locus *Rapid genomic locus annotator for CNV calls*

Description

genomic_locus add the locus information to a data.table containing CNV calls-like data

Usage

```
genomic_locus(
  DT_in,
  remote_cytobands = TRUE,
  bands,
  assembly = "hg19",
  keep_str_end = TRUE
)
```

Arguments

DT_in,	a data.table, usually read_results output.
remote_cytobands,	logical, indicates whether the function should download the CytoBands file or use a local object.
bands,	the local object if remote_cytobands is set to FALSE.
assembly,	character, specify the genomic assembly. Can be either "hg18", "hg19" or "hg38".
keep_str_end,	logical, specify if intermediate columns (locus_start and locus_end) must be kept or discarded.

Details

This function takes a CNVresults object and add a columns containing the genomic locus information. By default the file containing the CytoBands location for the specified assembly are downloaded from UCSC website, but also a local object (as data.table or data.frame) can be used via the argument bands.

Value

a CNVresults, DT_in with the additional column "locus".

Examples

```
## Not run:
DT <- genomic_locus(penn_22)

## End(Not run)
```

`hg18_chr_arms` *Chromosomal location of the genomic arms for the assembly hg18*

Description

Chromosomal location of the genomic arms for the assembly hg18

Usage

```
hg18_chr_arms
```

Format

A `data.table` with four columns:

chr chromosome name (1:24 format)

arm_ID genomic arm name

start genomic location of the arm start

end genomic location of the arm end

`hg18_start_end_centromeres` *Start, end and centrosome location of each chromosome for the assembly hg18*

Description

Start, end and centrosome location of each chromosome for the assembly hg18

Usage

```
hg18_start_end_centromeres
```

Format

A `data.table` with four columns:

chr chromosome name (1:24 format)

start start

end end

centromere centromere

hg19_chr_arms *Chromosomal location of the genomic arms for the assembly hg19*

Description

Chromosomal location of the genomic arms for the assembly hg19

Usage

```
hg19_chr_arms
```

Format

A `data.table` with four columns:

chr chromosome name (1:24 format)

arm_ID genomic arm name

start genomic location of the arm start

end genomic location of the arm end

hg19_start_end_centromeres *Start, end and centrosome location of each chromosome for the assembly hg19*

Description

Start, end and centrosome location of each chromosome for the assembly hg19

Usage

```
hg19_start_end_centromeres
```

Format

A `data.table` with four columns:

chr chromosome name (1:24 format)

start start

end end

centromere centromere

hg38_chr_arms *Chromosomal location of the genomic arms for the assembly hg38*

Description

Chromosomal location of the genomic arms for the assembly hg38

Usage

```
hg38_chr_arms
```

Format

A `data.table` with four columns:

chr chromosome name (1:24 format)

arm_ID genomic arm name

start genomic location of the arm start

end genomic location of the arm end

hg38_start_end_centromeres *Start, end and centrosome location of each chromosome for the assembly hg38*

Description

Start, end and centrosome location of each chromosome for the assembly hg38

Usage

```
hg38_start_end_centromeres
```

Format

A `data.table` with four columns:

chr chromosome name (1:24 format)

start start

end end

centromere centromere

immuno_regions *Retrieve genomic regions of consecutive immunoglobulin genes*

Description

Retrieve genomic regions of consecutive immunoglobulin genes

Usage

```
immuno_regions(biotype = NULL, n_genes = 5, mart = NULL)
```

Arguments

biotype,	vector of character, is used to filter the genes based on gene_biotype from to biomaRt::getBM. By default all the immunoglobulin biotypes al listed in Genecode <https://www.gencodegenes.org/pages/biotypes.html>. If the user specify one or more values only those are used.
n_genes,	integer, number of minimum consecutive genes required to construct an "immunoglobulin region", default is 5.
mart,	user provided mart (from biomaRt::useMart function). For older assemblies the user must manually retrieve the correct mart via biomaRt.
	immuno_regions is used to construct a data.table of genomic regions where all the consecutive gene have an immunoglobulin (or user specified). The output can be used as a blacklist in cleaning_filter.

Value

a list with two element, the first is a data.table containing the actual regions (to be passed to cleaning_filter for example), the second is the full genes list (for the required biotypes), again as data.table.

Examples

```
## Not run:
DT <- immuno_regions(biotype = c("IG_C_gene", "IG_D_gene"))

## End(Not run)
```

inter_res_merge *Combine the results from multiple methods in a single object*

Description

inter_res_merge combines the results of multiple CNV calling methods imported via read_results into a single CNVresults object.

Usage

```
inter_res_merge(
  res_list,
  sample_list,
  chr_arms,
  prop = 0.3,
  inner_outer = "outer"
)
```

Arguments

res_list	a list of CNVresults.
sample_list	cohort information
chr_arms	a data.table containing chromosomal arms locations. They are bundled in the package for hg18, hg19 and hg38 (hgXX_chr_arms).
prop	proportion of reciprocal overlap to define two calls as "replicated".
inner_outer	keep "inner" or "outer" borders? If NA all columns will be kept.

Details

Multiple CNVresults objects are combined into a single object and replicated calls are merged. The amount of reciprocal overlap necessary to define two calls as "replicated" is controlled with the prop parameter. When merging two or more events there will be two borders, inner and outer. By default the outer border are kept as "start" and "end" of the final table. If the user want to keep all the information inner_outer can be set to NA. In this case "start" and "end" will also be the outer borders. This is done because a CNVresults object need explicit "start" and "end" columns.

Value

a CNVresults containing the merge of the one provided via res_list.

Examples

```
DT <- inter_res_merge(res_list = list(penn_22, quanti_22),
sample_list= cohort_examples, chr_arms= hg19_chr_arms)
```

lrr_trio_plot *Plot markers raw data in a CNV region for a trio*

Description

Plot markers raw data in a CNV region for a trio

Usage

```
lrr_trio_plot(cnv, raw_path, sample_list, results)
```

Arguments

cnv,	a data.table containig one single line of CNVs calling results.
raw_path,	character, the path where raw data processed by read_NGS_raw or read_finalreport_raw.
sample_list,	minimal cohort metadata, a data.table produced by the function read_metadt.
results,	a data.table, the output of read_results.
	This function plot the raw data in the regions of interest in order to visually confirm the presence of a good de novo CNV candidate.

Value

a list containing the ggplot2 object.

markers_examples *Markers file for the runnable examples It comes from CNV calling on array data from a subset of the 1000 Genomes project, see vignettes for more details.*

Description

Markers file for the runnable examples It comes from CNV calling on array data from a subset of the 1000 Genomes project, see vignettes for more details.

Usage

```
markers_examples
```

Format

An object of class data.table (inherits from data.frame) with 115377 rows and 5 columns.

merge_calls *Merge adjacent CNV with equal Copy Number*

Description

merge_calls screen the segments of a sample and merge adjacent calls with equal GT if close enough

Usage

```
merge_calls(DT_in, prop = 0.3)
```

Arguments

DT_in	a data.table in the format of read_results output.
prop	numeric value that multiplied by the length of a call gives the range where adjacent calls are searched.

Details

This function takes a CVNResults object and try to merge adjacent calls (having equal GT and being close enough). It is designed to process one sample at the time and is integrated in the function read_results. It is not suggested to use it stand-alone. The merging is done starting from the larger calls, on adjacent call with equal GT in a range of +/- length*prop, trough several iterations until no more calls can be merged.

Value

a CNVresults, DT_in processing result.

Examples

```
# with merge
DT <- read_results(DT_path = system.file("extdata", "chrs_14_22_cnvs_penn.txt",
package = "CNVgears"), res_type = "file", DT_type = "TSV/CSV", chr_col = "chr",
start_col = "posStart", end_col = "posEnd", CN_col = "CN", samp_ID_col = "Sample_ID",
sample_list = cohort_examples, markers = markers_examples, method_ID = "P")
# without
DT <- read_results(do_merge = FALSE, DT_path = system.file("extdata", "chrs_14_22_cnvs_penn.txt",
package = "CNVgears"), res_type = "file", DT_type = "TSV/CSV", chr_col = "chr",
start_col = "posStart", end_col = "posEnd", CN_col = "CN", samp_ID_col = "Sample_ID",
sample_list = cohort_examples, markers = markers_examples, method_ID = "P")
```

penn_22 *Chromosome 22 PennCNV example results for the runnable examples The results of CNV calling using the program PennCNV on array data from a subset of the 1000 Genomes project, see vignettes for more details. Only chromosome 22 was kept in order to keep the object small.*

Description

Chromosome 22 PennCNV example results for the runnable examples The results of CNV calling using the program PennCNV on array data from a subset of the 1000 Genomes project, see vignettes for more details. Only chromosome 22 was kept in order to keep the object small.

Usage

```
penn_22
```

Format

an object of `CNVresults` class.

quanti_22 *Chromosome 22 QuantiSNP example results for the runnable examples. The results of CNV calling using the program QuantiSNP on array data from a subset of the 1000 Genomes project, see vignettes for more details. Only chromosome 22 was kept in order to keep the object small.*

Description

Chromosome 22 QuantiSNP example results for the runnable examples. The results of CNV calling using the program QuantiSNP on array data from a subset of the 1000 Genomes project, see vignettes for more details. Only chromosome 22 was kept in order to keep the object small.

Usage

```
quanti_22
```

Format

an object of `CNVresults` class.

`read_finalreport_raw` *Read Illumina array raw data*

Description

`read_finalreport` handles inputs of data in FinalReport-like format

Usage

```
read_finalreport_raw(
  DT_path,
  rds_path,
  pref,
  suff,
  sample_list,
  markers,
  chr_col = "Chr",
  pos_col = "Position",
  LRR_col = "Log R Ratio",
  BAF_col = "B Allele Freq"
)
```

Arguments

`DT_path,`	character, the path to the directory with the raw data (i.e. the splitted final reports)
`rds_path,`	path to the directory where RDS should be stored.
`pref, suff,`	eventual prefix an suffix (e.g. ".txt"). If not necessary must be set to `NA`.
`sample_list,`	a `data.table`, the output of `read_metadt`.
`markers,`	a `data.table`, the output of `read_NGS_intervals` or `read_finalreport_snps`, depending on the initial data type.
`chr_col,`	name of the column containing the chromosome information in the input file. Default is `"Chr"`.
`pos_col,`	name of the column containing the SNPs position information in the input file. Default is `"Position"`.
`LRR_col,`	name of the column containing the LRR information in the input file. Default is `"Log R Ratio"`.
`BAF_col,`	name of the column containing the BAF information in the input file. Default is `"B Allele Freq"`.

Details

This function handles the input, pre-processing and temporary storage (as RDS files) of the the markers-level raw data for each sample starting from FinalReport-like files (any plain text file with columns header can be read). Input must be one file per sample.

Value

nothing, this function saves the results on disk.

Examples

```
read_finalreport_raw(DT_path = system.file("extdata", package = "CNVgears"),
                     pref = "Final_report_example", suff = ".txt",
                     rds_path = file.path("tmp_RDS"),
                     markers = markers_examples,
                     sample_list = cohort_examples[sample_ID == "NA12878", ])
```

---

read_finalreport_snps *Read Illumina array raw data*

Description

`read_finalreport` handles inputs of data in FinalReport-like format

Usage

```
read_finalreport_snps(
  DT_path,
  mark_ID_col = "SNP Name",
  chr_col = "Chr",
  pos_col = "Position"
)
```

Arguments

`DT_path,`	path to the input file.
`mark_ID_col,`	name of the column containing the SNP ID information in the input file. Default is `"SNP Name"`.
`chr_col,`	name of the column containing the chromosome information in the input file. Default is `"Chr"`.
`pos_col,`	name of the column containing the SNPs position information in the input file. Default is `"Position"`.

Details

This function is used to load data in FinalReport like format into a `data.table` containing the SNPs markers information (i.e. chromosome and position). The function expect a single file where each markers is present one single time. Similar files are often required or produced by the calling algorithm/pipeline, e.g. the PFB file in PennCNV can be used here.

Value

a `data.table`, will be of `Markers` class in future versions.

Examples

```
DT <- read_finalreport_snps(system.file("extdata", "SNP.pfb", package = "CNVgears"),
mark_ID_col = "Name", chr_col = "Chr", pos_col = "Position")
```

`read_metadt` *Read sample file with minimal metadata*

Description

`read_medaDT` handles the input of the sample table (sampleID, sex, role, famID) in a standardized format.

Usage

```
read_metadt(
  DT_path,
  sample_ID_col = "sample_ID",
  sex_col = "sex",
  role_col = "role",
  fam_ID_col = "fam_ID"
)
```

Arguments

`DT_path`	path to the input file.
`sample_ID_col`	name of the columns containing the sample ID in the original file;
`sex_col`	name of the columns containing the sex information in the original file;
`role_col`	name of the columns containing the role information ID in the original file;
`fam_ID_col`	name of the columns containing the family ID in the original file;

Details

This function is needed in the first step of virtually every analysis. The input data must have at least the following columns:

- sample ID, self describing;
- sex, ideally in 1/2 format, for males and females, however also "male"/"female" or "Male"/"Female" are accepted;
- role, role of the sample in the family, either "father", "mother", "proband" or "sibling";
- family ID, self describing.

Actual name and order of the columns in the file is not relevant since they are passed to the function via parameter. Since the function in this package are optimized for family based studies, family ID and role information for each sample are required, however if the user is interested only, as an example, in CNVRs computation, genic content annotation or identification of calls in IG regions and does not have such information "role" and famID can be "NA". Note that doing so some functions won't be usable.

Value

cohort metatadata object, a `data.table`. Will be of the `SampleList` class in future versions.

Examples

```
DT <- read_metadt(DT_path = system.file("extdata", "cohort.ped", package = "CNVgears"),
sample_ID_col = "Individual ID", fam_ID_col = "Family ID", sex_col = "Gender",
role_col = "Role")
```

`read_NGS_intervals` *Read genomic intervals*

Description

`read_intervals` handles inputs of data used as the markers in CNVs calling/segmentation using NGS data (WES or WGS)

Usage

```
read_NGS_intervals(
  DT_path,
  chr_col = "CONTIG",
  start_col = "START",
  end_col = "END"
)
```

Arguments

`DT_path,`	path to the input file.
`chr_col,`	name of the column containing the chromosome information in the input file.
`start_col,`	name of the column containing the start information in the input file.
`end_col,`	name of the column containing the end information in the input file.

Details

This function is used to load data in interval list or BED like formats into a `data.table` that integrates with the other functions of the package. This is usually done at the beginning of a project involving CNVs calling/segmentation on NGS data (WES or WGS) pipelines' results analysis. The function should automatically skip any eventual header. The parameters default values are for file in GATK interval list like format.

Value

a `data.table`, will be of `Markers` class in future versions.

Examples

```
read_NGS_intervals(DT_path = system.file("extdata", "markers_WES_example.tsv",
package = "CNVgears"), chr_col = "chr", start_col = "start", end_col = "end")
```

`read_NGS_raw` *Read raw copyratio/LRR data for NGS intervals*

Description

`read_NGS_raw`

Usage

```
read_NGS_raw(
  DT_path,
  rds_path,
  pref,
  suff,
  chr_col,
  start_col,
  end_col,
  raw_col,
  raw_type,
  sample_list,
  markers
)
```

Arguments

`DT_path,`	path to the input file.
`rds_path,`	path to the directory where RDS should be stored.
`pref, suff,`	eventual prefix an suffix (e.g. ".txt"). If not necessary must be set to `NA`.
`chr_col,`	name of the column containing the chromosome information in the input file.
`start_col,`	name of the column containing the start information in the input file.
`end_col,`	name of the column containing the end information in the input file.
`raw_col,`	name of the column containing the marker-level raw data in the input file.
`raw_type,`	character, it describes the format in which the LRR-like data is give. Can be either "log2R", "numeric_CN" or "copyratio", see description.
`sample_list,`	a `data.table`, the output of `read_metadt`.
`markers,`	a `data.table`, the output of `read_NGS_intervals` or `read_finalreport_snps`, depending on the initial data type.

Details

This function handles the input, pre-processing and temporary storage (as RDS files) of the the markers-level raw data for each sample. Those are mostly required for CNVs inheritance detection. Any marker-level log2R-like measure is supported, given that, for somatic chromosomes: $$LRR = log2R = log2(CopyRatio) = log2(numeric_CN/2*)$$ Input must be one file per sample.

Value

nothing, this function saves the results on disk.

`read_results` *Read CNVs calling or semgentation results*

Description

`read_results` takes the results of a CNVs calling pipeline and return them in a standardized object.

Usage

```
read_results(
  DT_path,
  res_type,
  DT_type,
  pref = NA,
  suff = NA,
  sample_list,
  markers,
  chr_col,
  start_col,
  end_col,
  CN_col,
  samp_ID_col,
  end_vcf = "END",
  CN_vcf = "CN",
  do_merge = TRUE,
  merge_prop = 0.5,
  method_ID
)
```

Arguments

`DT_path,`	path to the directory containing the individual files, if `res_type` is set to "directory" or to the single file if `res_type` is set to "file".
`res_type,`	can be either "directory" or "file", indicates whether the function must expect a single file for all samples or one file per sample.
`DT_type,`	can be either "VCF" or "TSV/CSV", indicate the file type.

pref, suff,	eventual prefix an suffix (e.g. ".txt") to the files to be used when res_type is set to "directory". If not necessary must be set to NA.
sample_list,	minimal cohort metadata, a data.table produced by the function read_metadt.
markers,	a data.table containing the marker list, the output read_finalreport_snps with DT_type set to "markers" or read_NGS_intervals.
chr_col,	name of the column containing the chromosome information in the input data.
start_col,	name of the column containing the start information in the input data.
end_col,	name of the column containing the end information in the input data.
CN_col,	name of the column containing the Copy Number information in the input data.
samp_ID_col,	name of the column containing the sample ID information in the input file, required if res_type is set to "file".
end_vcf,	name of the field containing the segment end information in the VCF file(s), passed to the function read_vcf.
CN_vcf,	name of the field containing the segment copy number information in the VCF file(s), passed to the function read_vcf.
do_merge,	logical, indicates whether the function merge_calls should be automatically called for each sample (strongly suggested).
merge_prop	minimum reciprocal overlap proportion in order to merge.
method_ID,	character identifying the method (algorithms/pipeline), one letter code is strongly encouraged (e.g. "P" for PennCNV and "M" for GATK ModSeg). Numeric are converted to character.

Details

This function aims to convert a variety of possible types of CNVs calling/segmentation pipelines and/or algorithms results into a standardized format in order to easily integrate with the other functions in this package. Currently two main files type and two main file-organization structures are considered, for a total of four generic situations:

- VCF files, one per sample (e.g. the results of GATK gCNV pipeline);
- VCF file, all sample of a cohort in the same file (not yet fully implemented);
- TSV/CSV file, one file per sample (e.g. the results of GATK ModSeg pipeline, or the results of running "manually" PennCNV);
- TSV/CSV file, all samples of a cohort in the same file (e.g. the results of EnsembleCNV).

If multiple files containing results for multiple samples are present (e.g. the results of PennCNV joint calling on trios) at the moment it is recommended that the user concatenated those individual file in a single one prior loading them with read_results. Note that any line occurring before the columns header are automatically skipped by fread.

Value

a CNVresults object.

Examples

```
DT <- read_results(DT_path = system.file("extdata", "chrs_14_22_cnvs_penn.txt",
package = "CNVgears"), res_type = "file", DT_type = "TSV/CSV", chr_col = "chr",
start_col = "posStart", end_col = "posEnd", CN_col = "CN", samp_ID_col = "Sample_ID",
sample_list = cohort_examples, markers = markers_examples, method_ID = "P")
```

---

read_vcf *Convert a VCF file of genomics segments into a* data.table

---

Description

read_vcf read a VCF file into a data.table

Usage

```
read_vcf(
  DT_path,
  end_vcf = "END",
  CN_vcf = "CN",
  samples = NA,
  explore = FALSE
)
```

Arguments

DT_path	path to the file.
end_vcf	name of the field containing the segment end information.
CN_vcf	name of the field containing the segment copy number information.
samples	NA by default, if a character vector is provided is used to identify and select samples in a VCF containing multiple ones.
explore	logic, FALSE by default. If TRUE the file in DT_path is not loaded, instead, several info about the VCF fields are printed.

Details

This function use readVcf from VariantAnnotation to read VCF files, then it select only the necessary columns (for the purpose of CNVs calling results analysis) and convert it to a data.table. Can also be used to check the names of the necessary fields (end and copy number) if not already known, using the parameter explore. By default it expect a file containing data for a single sample (e.g. the results of gCNV from GATK), but it can process files containing multiple samples if a character vector containing the IDs is given to the parameter samples.

Value

a CNVresults from the VCF results conversion.

Examples

```
read_vcf(DT_path = system.file("extdata", "VCF_res_example.vcf", package = "CNVgears"))
read_vcf(DT_path = system.file("extdata", "VCF_res_example.vcf", package = "CNVgears"),
explore = TRUE)
```

`summary.CNVresults` *Explore CNV calling results prior filtering*

Description

Explore CNV calling results prior filtering

Usage

```
## S3 method for class 'CNVresults'
summary(object, sample_list, markers, plots_path = NA, ...)
```

Arguments

`object,`	a `data.table`, the output of `read_results`.
`sample_list,`	a `data.table`, the output of `read_metadt`.
`markers,`	a `data.table`, the output of `read_NGS_intervals` or `read_finalreport_snps`, depending on the initial data type.
`plots_path,`	path where the plots should be saved, if `NA` no plot is produced.
`...`	compatibility

This function produce several summary statistics on the CNVs results in input. Ideally, it should be used interactively, together with `cleaning_filter`. Some information is printed on the console (mostly `summary()` on several characteristics of the results and the cohort), several plots can be produced and saved in the user specified location. Finally the function also return a `data.table` of sample-level summary.

Value

a `data.table` with summary statistics for the samples in the cohort.

Examples

```
DT <- summary(penn_22, cohort_examples, markers_examples)
```

`telom_centrom` *Generate blacklist for telomeric and centromeric regions*

Description

Generate blacklist for telomeric and centromeric regions

Usage

```
telom_centrom(DT_in, telom = TRUE, centrom = TRUE, region = 50000)
```

Arguments

`DT_in,`	a `data.table` with start, and centromere position for each chromosome. For the assemblies hg18, hg19 and hg38 this is provided in the package.
`telom,`	logical, should be telomeric blacklist be produced?
`centrom,`	logical, should be centromeric blacklist be produced?
`region,`	integer. How many basepairs large should the regions be? The centromeric region will be twice this large.

Value

, a `data.table` that can be passed to `cleaning_filter` as blacklist.

Examples

```
DT <- telom_centrom(hg19_start_end_centromeres)
```

Index