Package 'mutationtypes'

July 23, 2025

Title Validate and Convert Mutational Impacts Using Standard Genomic Dictionaries

Version 0.0.1

Description Check concordance of a vector of mutation impacts with standard dictionaries such as Sequence Ontology (SO) <http://www.sequenceontology.org/>, Mutation Annotation Format (MAF) <https://docs.gdc.cancer.gov/Encyclopedia/pages/Mutation_Annotation_Format_TCGAv2/> or Prediction and Annotation of Variant Effects (PAVE) <https://github.com/hartwigmedical/hmftools/tree/master/pave>. It enables conversion between SO/PAVE and MAF terms and selection of the most severe consequence where multiple ampersand (&) delimited impacts are given.

License LGPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.3

Imports assertions, cli, data.table, stats, utils

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/selkamand/mutationtypes

BugReports https://github.com/selkamand/mutationtypes/issues

NeedsCompilation no

Author Sam El-Kamand [aut, cre] (ORCID: <https://orcid.org/0000-0003-2270-8088>), Children's Cancer Institute Australia [cph]

Maintainer Sam El-Kamand <sam.elkamand@gmail.com>

Repository CRAN

Date/Publication 2023-12-14 16:10:05 UTC

Contents

`mutation_types_convert_pave_to_maf`
Convert PAVE Mutation Types to MAF

Description

Convert PAVE Mutation Types to MAF

Usage

```
mutation_types_convert_pave_to_maf(
  pave_mutation_types,
  variant_type = NULL,
  split_on_ampersand = TRUE,
  missing_to_silent = FALSE,
  verbose = TRUE
)
```

Arguments

`pave_mutation_types`	a vector of PAVE terms you want to convert to MAF variant classifications (character)
`variant_type`	a vector describing each mutations type. Valid elements include: "SNP", "DNP", "TNP", "ONP", "DEL", "INS". Used to map frameshift_variant to more specific MAF columns (character)
`split_on_ampersand`	should '&' separated PAVE terms be automatically converted to single PAVE terms based on highest severity? (flag)
`missing_to_silent`	should missing (NA) or empty ('') mutation types be converted to 'Silent' mutations?
`verbose`	verbose (flag)

Value

matched MAF variant classification terms (character)

Examples

```
mutation_types_convert_pave_to_maf(
  c('upstream_gene_variant', 'stop_lost', 'splice_acceptor_variant')
)
```

`mutation_types_convert_so_to_maf`
Convert SO Mutation Types to MAF

Description

Convert SO Mutation Types to MAF

Usage

```
mutation_types_convert_so_to_maf(
  so_mutation_types,
  variant_type = NULL,
  inframe = NULL,
  split_on_ampersand = TRUE,
  missing_to_silent = FALSE,
  verbose = TRUE
)
```

Arguments

`so_mutation_types`	a vector of SO terms you want to convert to MAF variant classifications (character)
`variant_type`	a vector describing each mutations type. Valid elements include: "SNP", "DNP", "TNP", "ONP", "DEL", "INS". Used to map frameshift_variant to more specific MAF columns (character)
`inframe`	is the mutation inframe? (logical). Used to map protein_altering_variant to valid MAF columns
`split_on_ampersand`	should '&' separated SO terms be automatically converted to single SO terms based on highest severity? (flag)
`missing_to_silent`	should missing (NA) or empty ('') mutation types be converted to 'Silent' mutations?
`verbose`	verbose (flag)

Value

matched MAF variant classification terms (character)

Examples

```
mutation_types_convert_so_to_maf(c('INTRAGENIC', 'INTRAGENIC', 'intergenic_region'))
```

`mutation_types_identify` *Identify Mutation Dictionary Used*

Description

Looks at variant consequence terms and guesses what mutation dictionary was used. SO and PAVE dictionaries overlap, meaning an observed set of terms can perfectly match both ontologies. If this happens, we assume they are SO terms.

Usage

```
mutation_types_identify(
  mutation_types,
  split_on_ampersand = TRUE,
  verbose = TRUE,
  ignore_missing = FALSE
)
```

Arguments

`mutation_types`	mutation types to test (character)
`split_on_ampersand`	split mutation types in a single string separated by ampersand (&) into 2 distinct mutation type columns (flag)
`verbose`	verbose (flag)
`ignore_missing`	should we ignore missing (NA) or empty (") mutation_types when identifying a classification scheme (flag)

Value

one of c('SO', 'MAF', 'UNKNOWN'). Will return 'UNKNOWN' unless ALL mutation types fit with one of the supported dictionaries

Examples

```
mutation_types_identify(c('bob', 'billy', 'missense_variant'))
```

mutation_types_maf *Dictionary of MAF terms*

Description

Dictionary of MAF terms

Usage

```
mutation_types_maf()
```

Value

valid MAF terms (character)

Examples

```
mutation_types_maf()
```

mutation_types_maf_palette *Palettes: MAF*

Description

Palettes: MAF

Usage

```
mutation_types_maf_palette()
```

Value

named vector. Names are MAF terms. Values are colors

Examples

```
mutation_types_maf_palette()
```

mutation_types_pave *Dictionary of PAVE terms*

Description

PAVE is a newer annotation software that supports annotaiton of mainly just a subset of SO terms, but with a couple of important additions to indicate when a non-obvious consequence can be found thanks to phasing.

Usage

```
mutation_types_pave()
```

Value

valid PAVE terms (character)

Examples

```
mutation_types_pave()
```

mutation_types_pave_palette *Palettes: PAVE*

Description

Palettes: PAVE

Usage

```
mutation_types_pave_palette()
```

Value

named vector. Names are PAVE terms. Values are colors

Examples

```
mutation_types_pave_palette()
```

mutation_types_so *Dictionary of So terms*

Description

Dictionary of So terms

Usage

```
mutation_types_so()
```

Value

valid SO terms (character)

Examples

```
mutation_types_so()
```

mutation_types_so_palette *Palettes: SO*

Description

Palettes: SO

Usage

```
mutation_types_so_palette()
```

Value

named vector. Names are SO terms. Values are colors

Examples

```
mutation_types_so_palette()
```

`select_most_severe_consequence_pave` *Select the most severe consequence (PAVE)*

Description

Take a character vector which may contain multiple PAVE mutation types separated by '&' And choose only the most severe consequence

Usage

```
select_most_severe_consequence_pave(
  pave_mutation_types,
  missing_is_valid = FALSE
)
```

Arguments

`pave_mutation_types`
: a character vector of PAVE terms, where multiple pave_mutation_types per field are & delimited, and you want to choose the most severe consequence .

`missing_is_valid`
: should NA values be considered valid mutation classes or should they throw an error? (flag)

Value

the most severe consequence for each element in pave_mutation_types

Examples

```
select_most_severe_consequence_pave(
 c(
   "upstream_gene_variant&phased_synonymous&5_prime_UTR_variant",
   "missense_variant&frameshift_variant"
 )
)
#> Result:
#> c("phased_synonymous", "frameshift_variant")
```

`select_most_severe_consequence_so` *Select the most severe consequence (SO)*

Description

Take a character vector which may contain multiple so mutation types separated by '&' And choose only the most severe consequence

Usage

```
select_most_severe_consequence_so(so_mutation_types, missing_is_valid = FALSE)
```

Arguments

`so_mutation_types`	a character vector of SO terms, where multiple so_mutation_types per field are & delimited, and you want to choose the most severe consequence .
`missing_is_valid`	should NA values be considered valid mutation classes or should they throw an error? (flag)

Value

the most severe consequence for each element in so_mutation_types

Examples

```
select_most_severe_consequence_so(
 c(
   "intergenic_variant&feature_truncation&splice_acceptor_variant",
   "initiator_codon_variant&inframe_insertion"
 )
)
#> Result:
#> c("splice_acceptor_variant", "initiator_codon_variant")
```

Index