Package: SelectSim (via r-universe)

October 20, 2026

Title Selected Events Linked by Evolutionary Conditions in Cancer

Version 0.1.6

Description Implements the 'SelectSim' methodology for identifying patterns of co-occurrence and mutual exclusivity between functional genomic alterations in cancer cohorts. The package processes mutation annotation data, constructs alteration matrices, estimates expected alteration-pair frequencies, and quantifies deviations associated with selective interactions. The methodology is described in Iyer et al. (2026) <doi:10.1038/s41588-026-02661-4>.

License MIT + file LICENSE

URL https://csogroup.github.io/SelectSim/

BugReports https://github.com/CSOgroup/SelectSim/issues

Depends R (>= 3.5)

Imports doParallel, doRNG, dplyr, foreach, ggplot2, ggpubr, ggridges, Matrix, parallel, Rcpp, Rfast, stats

Suggests knitr, rmarkdown, testthat (>= 3.0.0), tictoc

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

LazyDataCompression xz

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Language en-US

Config/pak/sysreqs cmake make libicu-dev

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

Date/Publication 2026-07-12 07:24:53 UTC

RemoteUrl https://github.com/csogroup/selectsim

RemoteRef HEAD

RemoteSha 54c06ec5f77441937726ca62c10781b3775d6890

Contents

add . . . 3
al.pairwise.alteration.stats . . . 4
al.stats . . . 4
am.pairwise.alteration.coverage . . . 5
am.pairwise.alteration.overlap . . . 6
am.stats . . . 7
am.weight.pairwise.alteration.overlap . . . 7
binary.yule . . . 8
effectSize . . . 9
estimate_p_val . . . 9
estimate_pairwise_p . . . 10
estimateFDR2 . . . 11
filter_maf_column . . . 11
filter_maf_complex . . . 12
filter_maf_gene.name . . . 13
filter_maf_ignore . . . 14
filter_maf_missense . . . 14
filter_maf_mutation.type . . . 15
filter_maf_mutations . . . 16
filter_maf_sample . . . 17
filter_maf_schema . . . 17
filter_maf_truncating . . . 18
generateS . . . 19
generateW_block . . . 19
generateW_mean_tmb . . . 20
GENIE_maf_schema . . . 21
get.blocks . . . 21
interaction.table . . . 22
luad_maf . . . 23
luad_result . . . 24
luad_run_data . . . 24
maf2gam . . . 25
mutation_type . . . 26
new.AL.general . . . 26
new.ALS . . . 28
new.AMS . . . 28
null_model_parallel . . . 29
obs_exp_scatter . . . 30
oncokb_genes . . . 30
oncokb_truncating_genes . . . 31
overlap_pair_extract . . . 31

r.am.pairwise.alteration.overlap 32
r.effectSize 33
retrieveOutliers 34
ridge_plot_ed 34
ridge_plot_ed_compare 35
selectX 36
stat_maf_column 38
stat_maf_gene 38
stat_maf_sample 39
TCGA_maf_schema 40
template.obj.gen 40
theme_Publication 41
variant_catalogue 42
w.r.am.pairwise.alteration.overlap 42

Index **44**

---

add *Sum a list of matrices element-wise*

Description

Sum a list of matrices element-wise

Usage

```
add(x)
```

Arguments

x	List of matrices of identical dimensions

Value

Single matrix that is the element-wise sum of all matrices in x

Examples

```
mats <- list(matrix(1:4, 2, 2), matrix(1:4, 2, 2))
add(mats)
```

al.pairwise.alteration.stats *Compute pairwise alteration statistics for an alteration landscape*

Description

Compute pairwise alteration statistics for an alteration landscape

Usage

```
al.pairwise.alteration.stats(al, als = NULL, do.blocks = FALSE)
```

Arguments

al	SelectX object (list containing al, W, etc.)
als	Alteration landscape stats (from al.stats); computed internally if NULL
do.blocks	Whether to also compute block-level pairwise stats

Value

List with overlap and w_overlap matrices, plus optional sample.blocks entries.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
al.pairwise.alteration.stats(result$obj)
```

al.stats *Compute alteration landscape statistics*

Description

Compute alteration landscape statistics

Usage

```
al.stats(al)
```

Arguments

al	SelectX object (list containing al, W, etc. as returned by selectX)

Value

ALS object with overall and per-block alteration statistics.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
stats <- al.stats(result$obj)
```

---

am.pairwise.alteration.coverage

Compute pairwise alteration coverage statistics

Description

Compute pairwise alteration coverage statistics

Usage

```
am.pairwise.alteration.coverage(overlap_M, M.stats, w_overlap_M)
```

Arguments

overlap_M	The pairwise overlap matrix
M.stats	The alteration matrix stats (from am.stats)
w_overlap_M	The weighted pairwise overlap matrix

Value

List with overlap and w_overlap sparse matrices.

Examples

```
am <- matrix(c(0,1,1,0,1,1), nrow = 2,
             dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
W  <- matrix(1, nrow = 2, ncol = 3,
             dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
ovlp  <- am.pairwise.alteration.overlap(am)
wovlp <- am.weight.pairwise.alteration.overlap(am, W)
stats <- am.stats(am)
am.pairwise.alteration.coverage(ovlp, stats, wovlp)
```

---

am.pairwise.alteration.overlap

Compute pairwise alteration co-occurrence counts

Description

Compute pairwise alteration co-occurrence counts

Usage

```
am.pairwise.alteration.overlap(am)
```

Arguments

am	Binary alteration matrix (features x samples)

Value

Square matrix of pairwise co-occurrence counts (features x features).

Examples

```
am <- matrix(c(0,1,1,0,1,1), nrow = 2,
             dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
am.pairwise.alteration.overlap(am)
```

am.stats *Compute summary statistics for a binary alteration matrix*

Description

Compute summary statistics for a binary alteration matrix

Usage

```
am.stats(am)
```

Arguments

am	Binary alteration matrix (features x samples)

Value

AMS object with basic counts: n.samples, n.alterations, n.occurrences, per-sample and per-feature counts.

Examples

```
am <- matrix(c(0,1,1,0,1,1), nrow = 2,
             dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
am.stats(am)
```

am.weight.pairwise.alteration.overlap *Compute TMB-weighted pairwise alteration overlap*

Description

Compute TMB-weighted pairwise alteration overlap

Usage

```
am.weight.pairwise.alteration.overlap(am, W)
```

Arguments

am	Binary alteration matrix (features x samples)
W	Weight matrix (features x samples) with per-sample TMB weights

Value

Weighted pairwise overlap matrix (features x features).

Examples

```
am <- matrix(c(0,1,1,0,1,1), nrow = 2,
             dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
W  <- matrix(1, nrow = 2, ncol = 3,
             dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
am.weight.pairwise.alteration.overlap(am, W)
```

`binary.yule` *Compute Yule Q coefficient for all gene pairs*

Description

Compute Yule Q coefficient for all gene pairs

Usage

```
binary.yule(overlap, mat)
```

Arguments

`overlap`	The pairwise overlap matrix
`mat`	The binary GAM (features x samples)

Value

Matrix of Yule Q coefficients

Examples

```
am   <- matrix(c(0,1,1,0,1,1), nrow = 2,
               dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
ovlp <- am.pairwise.alteration.overlap(am)
binary.yule(ovlp, am)
```

effectSize *Compute effect size between observed and expected overlap*

Description

Compute effect size between observed and expected overlap

Usage

```
effectSize(obs, exp)
```

Arguments

obs	The observed overlap values
exp	The expected (null model mean) overlap values

Value

Effect size value(s)

Examples

```
effectSize(obs = 5, exp = 3)
effectSize(obs = c(5, 2, 0), exp = c(3, 3, 1))
```

estimate_p_val *Compute empirical two-sided p-value for a gene pair*

Description

Compute empirical two-sided p-value for a gene pair

Usage

```
estimate_p_val(robs_co, obs.co, gene1, gene2)
```

Arguments

robs_co	List of null model overlap matrices (one per permutation)
obs.co	Observed pairwise overlap matrix
gene1	Name of the first gene/alteration
gene2	Name of the second gene/alteration

Value

Two-sided empirical p-value

Examples

```
set.seed(1)
genes <- c("geneA", "geneB")
robs_co <- lapply(seq_len(20), function(i) {
  m <- matrix(sample(0:5, 4, replace = TRUE), 2, 2,
              dimnames = list(genes, genes))
  m
})
obs_co <- matrix(c(5, 3, 3, 4), 2, 2, dimnames = list(genes, genes))
estimate_p_val(robs_co, obs_co, "geneA", "geneB")
```

---

estimate_pairwise_p *Compute p-values for all gene pairs in a results table*

Description

Compute p-values for all gene pairs in a results table

Usage

```
estimate_pairwise_p(obs, exp, results, nSim)
```

Arguments

obs	Observed pairwise overlap matrix
exp	List of null model overlap matrices (one per permutation)
results	Results data frame with SFE_1 and SFE_2 columns
nSim	Number of permutations

Value

Vector of p-values, one per row in results.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE, estimate_pairwise = TRUE)
head(result$result[, c("SFE_1","SFE_2","pairwise_p")])
```

`estimateFDR2` *Estimate FDR by scanning observed vs null effect sizes*

Description

Estimate FDR by scanning observed vs null effect sizes

Usage

```
estimateFDR2(obs, exp, nSim, maxFDR = 0.25)
```

Arguments

`obs`	Vector of observed effect sizes
`exp`	Vector of null model effect sizes (all permutations concatenated)
`nSim`	Number of permutations used to generate exp
`maxFDR`	FDR cutoff; scanning stops once FDR exceeds this value

Value

Vector of FDR values, one per element of obs.

Examples

```
set.seed(1)
obs <- c(0.8, 0.5, 0.3, 0.1)
exp <- runif(400)
estimateFDR2(obs, exp, nSim = 100)
```

`filter_maf_column` *Filter maf function*

Description

`filter_maf_column()` takes a maf file and filters a MAF dataframe by retaining only the rows with a column value included in the values list

Usage

```
filter_maf_column(maf, values, column, inclusive = TRUE, fixed = TRUE, ...)
```

Arguments

maf	a maf as dataframe
values	a list containing the elements to filter
column	column in maf file to filter
inclusive	a boolean to include or exclude the data frame with values in list provided
fixed	a grep argument to specify if grep use the argument as string or not
...	Other options

Value

filtered_maf a filtered maf file

Examples

```
data(luad_maf, package = "SelectSim")
filter_maf_column(luad_maf, values = "Missense_Mutation",
                  column = "Variant_Classification")
```

---

filter_maf_complex *Filter a MAF dataframe by a combination of column values*

Description

Filter a MAF dataframe by a combination of column values

Usage

```
filter_maf_complex(maf, values, ...)
```

Arguments

maf	A MAF dataframe
values	A dataframe of (column, value) pairs to match against
...	Additional arguments passed to merge

Value

Filtered MAF dataframe containing only rows matching the value combinations.

Examples

```
data(luad_maf, package = "SelectSim")
combos <- data.frame(Hugo_Symbol = "TP53",
                     Variant_Classification = "Missense_Mutation")
filter_maf_complex(luad_maf, combos,
                   by.x = c("Hugo_Symbol","Variant_Classification"),
                   by.y = c("Hugo_Symbol","Variant_Classification"))
```

---

filter_maf_gene.name *Filter a MAF dataframe by gene name*

Description

Filter a MAF dataframe by gene name

Usage

```
filter_maf_gene.name(maf, genes, gene.col = "Hugo_Symbol", ...)
```

Arguments

maf	A MAF dataframe
genes	Vector of gene names to retain
gene.col	Column name containing gene symbols
...	Additional arguments passed to filter_maf_column

Value

Filtered MAF dataframe.

Examples

```
data(luad_maf, package = "SelectSim")
filter_maf_gene.name(luad_maf, genes = c("TP53", "KRAS"))
```

filter_maf_ignore — *This function filters a MAF dataframe by retaining (or discarding) ignore mutations*

Description

filter_maf_ignore() takes a maf file and filters a MAF dataframe by retaining only the rows with a column value included in the values list

Usage

```
filter_maf_ignore(maf, schema = TCGA_maf_schema, ...)
```

Arguments

maf	a maf as dataframe
schema	a data-frame schema; see TCGA_maf_schema for an example
...	Other options

Value

filtered_maf a filtered maf file

Examples

```
data(luad_maf, package = "SelectSim")
filter_maf_ignore(luad_maf)
```

filter_maf_missense — *This function filters a MAF dataframe by retaining (or discarding) missense mutations*

Description

filter_maf_missense() takes a maf file and filters a MAF dataframe by retaining only the rows with a column value included in the values list

Usage

```
filter_maf_missense(maf, schema = TCGA_maf_schema, ...)
```

Arguments

maf	a maf as dataframe
schema	a data-frame schema; see TCGA_maf_schema for an example
...	Other options

Value

filtered_maf a filtered maf file

Examples

```
data(luad_maf, package = "SelectSim")
filter_maf_missense(luad_maf)
```

`filter_maf_mutation.type` *Filter a MAF dataframe by mutation type*

Description

Filter a MAF dataframe by mutation type

Usage

```
filter_maf_mutation.type(
  maf,
  variants,
  variant.col = "Variant_Classification",
  ...
)
```

Arguments

`maf`	A MAF dataframe
`variants`	Vector of variant classification values to retain
`variant.col`	Column name containing variant classifications
`...`	Additional arguments passed to filter_maf_column

Value

Filtered MAF dataframe.

Examples

```
data(luad_maf, package = "SelectSim")
filter_maf_mutation.type(luad_maf, variants = "Missense_Mutation")
```

`filter_maf_mutations` *Filter a MAF dataframe by specific gene-mutation combinations*

Description

Filter a MAF dataframe by specific gene-mutation combinations

Usage

```
filter_maf_mutations(
  maf,
  values,
  maf.col = c("Hugo_Symbol", "HGVSp_Short"),
  values.col = maf.col,
  ...
)
```

Arguments

`maf`	A MAF dataframe
`values`	Dataframe of allowed (gene, mutation) combinations
`maf.col`	Columns in maf to join on
`values.col`	Corresponding columns in values to join on
`...`	Additional arguments passed to filter_maf_complex

Value

Filtered MAF dataframe containing only rows matching the allowed combinations.

Examples

```
data(luad_maf, package = "SelectSim")
allowed <- data.frame(Hugo_Symbol  = c("TP53", "KRAS"),
                      HGVSp_Short  = c("p.R175H", "p.G12C"))
filter_maf_mutations(luad_maf, allowed)
```

filter_maf_sample *Filter a MAF dataframe by sample ID*

Description

Filter a MAF dataframe by sample ID

Usage

```
filter_maf_sample(maf, samples, sample.col = "Tumor_Sample_Barcode", ...)
```

Arguments

maf	A MAF dataframe
samples	Vector of sample IDs to retain
sample.col	Column name containing sample IDs
...	Additional arguments passed to filter_maf_column

Value

Filtered MAF dataframe.

Examples

```
data(luad_maf, package = "SelectSim")
ids <- unique(luad_maf$Tumor_Sample_Barcode)[1:5]
filter_maf_sample(luad_maf, samples = ids)
```

filter_maf_schema *This function filters a MAF dataframe by sample id*

Description

`filter_maf_schema()` takes a maf file and filters a MAF dataframe by retaining only the rows with a column value included in the values list

Usage

```
filter_maf_schema(maf, schema = TCGA_maf_schema, column, values, ...)
```

Arguments

maf	a maf as dataframe
schema	a data-frame schema; see TCGA_maf_schema for an example
column	column in maf file to filter
values	a list containing the elements to file
...	Other options

Value

filtered_maf a filtered maf file

Examples

```
data(luad_maf, package = "SelectSim")
filter_maf_schema(luad_maf, TCGA_maf_schema,
                  column = "mutation.type",
                  values = TCGA_maf_schema$mutation.type$truncating)
```

---

filter_maf_truncating *This function filters a MAF dataframe by retaining (or discarding) truncating mutations*

Description

filter_maf_truncating() takes a maf file and filters a MAF dataframe by retaining only the rows with a column value included in the values list

Usage

```
filter_maf_truncating(maf, schema = TCGA_maf_schema, ...)
```

Arguments

maf	a maf as dataframe
schema	a data-frame schema; see TCGA_maf_schema for an example
...	Other options

Value

filtered_maf a filtered maf file

Examples

```
data(luad_maf, package = "SelectSim")
filter_maf_truncating(luad_maf)
```

generateS *Generate S matrix*

Description

Generate S matrix

Usage

```
generateS(gam, sample.weights, upperBound = 1)
```

Arguments

gam	the gam with genes*samples
sample.weights	the samples weights
upperBound	clip the values greater than 1 to keep it bounded between 0 to 1

Value

S the S matrix

Examples

```
gam <- matrix(c(0,1,1,0,1,1), nrow = 2,
              dimnames = list(c("geneA","geneB"), c("s1","s2","s3")))
generateS(gam, sample.weights = c(s1 = 1, s2 = 1, s3 = 1))
```

generateW_block *Generate block-aware sample weight matrix*

Description

Computes a sample weight matrix that accounts for sample-class covariates (blocks). Within each block the reference TMB is the block median; the overall reference used by `generateW_mean_tmb` is the mean of those block medians.

Usage

```
generateW_block(al, lambda, tau)
```

Arguments

al	Alteration landscape object (from `new.AL.general`).
lambda	Numeric penalty factor for the weight computation.
tau	Numeric fold-change threshold below which no penalty is applied.

Value

List with W_block (per-block weight matrices), W (full concatenated weight matrix), W_median (per-block median TMBs), and mean_TMB (mean of block medians).

Examples

```
data(luad_run_data, package = "SelectSim")
al <- new.AL.general(luad_run_data$M,
                     feat.covariates  = luad_run_data$alteration.class,
                     sample.covariates = luad_run_data$sample.class,
                     min.freq = 10)
generateW_block(al, lambda = 0.3, tau = 1)
```

---

generateW_mean_tmb *Generate sample weight matrix from TMB values*

Description

Computes a per-sample weight matrix based on the ratio of each sample's TMB to the expected (mean) TMB. Samples with higher-than-expected TMB receive lower weights via a penalty lambda, controlled by the fold-change threshold tau.

Usage

```
generateW_mean_tmb(
  tmb,
  mean_tmb,
  ngenes,
  lambda = 0.3,
  tau = 1,
  discrete = TRUE
)
```

Arguments

tmb	Numeric vector of per-sample TMB values.
mean_tmb	Numeric scalar; the reference (expected) TMB used to compute fold changes.
ngenes	Integer; number of genes (rows) in the output weight matrix.
lambda	Numeric; weight penalty factor. Higher values penalise high-TMB samples more strongly (default 0.3).
tau	Numeric; fold-change threshold below which no penalty is applied (default 1).
discrete	Logical; if TRUE, fold changes are rounded up before applying the penalty (default TRUE).

Value

Numeric matrix of sample weights (ngenes x length(tmb)).

Examples

```
tmb      <- c(s1 = 10, s2 = 50, s3 = 20)
mean_tmb <- 25
generateW_mean_tmb(tmb, mean_tmb, ngenes = 3)
```

GENIE_maf_schema *GENIE_maf_schema: schema for GENIE maf file to process the mutations*

Description

GENIE_maf_schema: schema for GENIE maf file to process the mutations

Usage

```
GENIE_maf_schema
```

Value

A list defining the expected GENIE MAF column schema.

Examples

```
str(GENIE_maf_schema)
```

---

get.blocks *Get sample/alteration blocks*

Description

Get sample/alteration blocks

Usage

```
get.blocks(al)
```

Arguments

al	The alteration landscape

Value

Classification of samples and alterations in blocks.

Examples

```
data(luad_run_data, package = "SelectSim")
al <- new.AL.general(luad_run_data$M,
                     feat.covariates  = luad_run_data$alteration.class,
                     sample.covariates = luad_run_data$sample.class,
                     min.freq = 10)
get.blocks(al)
```

---

interaction.table *Build the full interaction results table from selectX outputs*

Description

Build the full interaction results table from selectX outputs

Usage

```
interaction.table(
  al,
  als,
  obs,
  wobs,
  r.obs = NULL,
  r.wobs = NULL,
  null,
  maxFDR = 0.2,
  n.cores = 1,
  estimate_pairwise = FALSE,
  n.permut = 1000
)
```

Arguments

al	Alteration landscape object (al$am, etc.)
als	Alteration landscape stats (from al.stats)
obs	Observed pairwise overlap matrix
wobs	Weighted observed pairwise overlap matrix
r.obs	List of null model overlap matrices
r.wobs	List of null model weighted overlap matrices
null	Null model list (simulated binary matrices)

maxFDR	FDR cutoff for calling significant results
n.cores	Number of cores
estimate_pairwise	Whether to compute per-pair empirical p-values
n.permut	Number of permutations

Value

Data frame with one row per gene pair and columns for overlap, effect sizes, FDR, and interaction type.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
head(result$result)
```

---

luad_maf *Lung adenocarcinoma MAF from TCGA cohort*

Description

MAF file of LUAD from TCGA

Usage

```
data(luad_maf)
```

Format

A data frame

Value

A data frame containing TCGA LUAD mutation data.

Examples

```
data(luad_maf)
dim(luad_maf)
head(luad_maf)
```

luad_result *Lung adenocarcinoma from TCGA cohort as SelectSim run results*

Description

Result of LUAD generated by SelectSim with SelectSim run done at min.freq=10.

Usage

```
data(luad_result)
```

Format

A data frame

Value

A data frame containing SelectSim results for the LUAD cohort.

Examples

```
data(luad_result)
dim(luad_result)
head(luad_result)
```

luad_run_data *Lung adenocarcinoma from TCGA cohort as SelectSim run object*

Description

Preprocessed TCGA LUAD data ready to pass directly to `selectX()`.

Usage

```
data(luad_run_data)
```

Format

A named list with four elements:

M A named list containing:

- **M** A named list of binary alteration matrices (genes x samples), one per alteration type (e.g., `missense`, `truncating`).
- **tmb** A named list of data frames, one per alteration type, each with columns `sample` (character) and `mutation` (integer TMB count).

sample.class Named character vector of sample-type annotations (length = number of samples). Names are sample IDs.

alteration.class Named character vector of alteration-type annotations (length = number of genes). Names are gene symbols.

Value

A named list containing preprocessed LUAD input data for `selectX()`.

Examples

```
data(luad_run_data)
names(luad_run_data)
str(luad_run_data, max.level = 1)
```

`maf2gam` *Generate gam from the maf file*

Description

`maf2gam()` takes a maf file and converts into gam

Usage

```
maf2gam(
  maf,
  sample.col = "Tumor_Sample_Barcode",
  gene.col = "Hugo_Symbol",
  value.var = "HGVSp_Short",
  samples = NULL,
  genes = NULL,
  fun.aggregate = length,
  binarize = TRUE,
  fill = NA
)
```

Arguments

`maf`	A MAF dataframe
`sample.col`	Column name for sample IDs
`gene.col`	Column name for gene symbols
`value.var`	Column used as the value to aggregate
`samples`	Vector of sample IDs to include; NULL keeps all samples present in the MAF
`genes`	Vector of gene names to include; NULL keeps all genes present in the MAF
`fun.aggregate`	Aggregation function applied per (sample, gene) cell
`binarize`	If TRUE, convert aggregated counts to binary presence/absence
`fill`	Value used for missing (sample, gene) combinations

Value

Numeric matrix (samples x genes) representing the gene alteration matrix.

Examples

```
data(luad_maf, package = "SelectSim")
small_maf <- luad_maf[
  luad_maf$Tumor_Sample_Barcode %in%
    unique(luad_maf$Tumor_Sample_Barcode)[1:5],
]
gam <- maf2gam(small_maf)
dim(gam)
```

---

mutation_type *Mutation list object*

Description

Mutation list object

Usage

```
mutation_type
```

Value

A list containing the supported mutation-type classifications.

Examples

```
str(mutation_type)
```

---

new.AL.general *Create an Alteration Landscape (AL) object*

Description

Builds an Alteration Landscape object from a list of genome alteration matrices and their corresponding tumor mutation burdens.

Usage

```
new.AL.general(
  am,
  feat.covariates = NULL,
  sample.covariates = NULL,
  min.freq,
  verbose = FALSE
)
```

Arguments

`am`	A named list with two required elements: `M` (a named list of binary alteration matrices, each genes x samples) and `tmb` (a named list of data frames, one per matrix in `M`, each with columns `sample` and `mutation`). The names of `M` and `tmb` must match. All matrices in `M` must have identical row and column names.
`feat.covariates`	Named character vector of alteration-type annotations, one entry per feature (gene). Names must match rownames of the matrices in `M`. If `NULL`, all features are labelled `"MUT"`.
`sample.covariates`	Named character vector of sample-type annotations, one entry per sample. Names must match colnames of the matrices in `M`. If `NULL`, all samples are labelled `"sample"`.
`min.freq`	Minimum number of samples a gene must be mutated in (strictly greater than) to be retained. Features with `rowSums <= min.freq` are dropped.
`verbose`	Logical; print progress messages.

Value

An Alteration Landscape (AL) object (list of class `"AL"`) containing the filtered alteration matrices, TMB vectors, and covariate assignments.

Examples

```
data(luad_run_data, package = "SelectSim")
al <- new.AL.general(
  am              = luad_run_data$M,
  feat.covariates  = luad_run_data$alteration.class,
  sample.covariates = luad_run_data$sample.class,
  min.freq        = 10
)
```

new.ALS *Initialize an Alteration Landscape Stats (ALS) container*

Description

Initialize an Alteration Landscape Stats (ALS) container

Usage

```
new.ALS(al)
```

Arguments

al	The alteration landscape object (checked for NULL)

Value

Empty ALS list object.

Examples

```
new.ALS(list())
```

new.AMS *Initialize an Alteration Matrix Stats (AMS) container*

Description

Initialize an Alteration Matrix Stats (AMS) container

Usage

```
new.AMS(am)
```

Arguments

am	The alteration matrix (checked for NULL)

Value

Empty AMS list object.

Examples

```
new.AMS(matrix(0, 2, 2))
```

`null_model_parallel` *Generating the null_simulation matrix*

Description

Generating the null_simulation matrix

Usage

```
null_model_parallel(al, temp_mat, W, n.cores = 1, n.permut, seed = 42)
```

Arguments

`al`	Alteration landscape object
`temp_mat`	template matrices
`W`	weight matrix
`n.cores`	Number of cores
`n.permut`	Number of simulations
`seed`	Random seed

Value

List of simulated binary matrices (one per permutation)

Examples

```
data(luad_run_data, package = "SelectSim")
al       <- new.AL.general(luad_run_data$M,
                           feat.covariates  = luad_run_data$alteration.class,
                           sample.covariates = luad_run_data$sample.class,
                           min.freq = 10)
temp_obj <- template.obj.gen(al)
W        <- generateW_block(al, lambda = 0.3, tau = 1)
sims     <- null_model_parallel(al, temp_obj$temp_mat, W$W,
                                n.cores = 1, n.permut = 10)
length(sims)
```

obs_exp_scatter *Scatter plot of observed vs expected weighted co-mutation*

Description

Plots each gene pair as a point with observed weighted co-mutation on the y-axis and expected (null model mean) on the x-axis. Significant co-mutations (CO) and mutual exclusivities (ME) are coloured; non-significant pairs are grey.

Usage

```
obs_exp_scatter(result, title)
```

Arguments

result	Result table from selectX()$result.
title	Plot title.

Value

A ggplot2 object.

Examples

```
data(luad_result, package = "SelectSim")
obs_exp_scatter(result = luad_result, title = "TCGA LUAD")
```

oncokb_genes *OncoKB v3.9 cancer genes*

Description

cancer genes list

Usage

```
data(oncokb_genes)
```

Format

A list

Value

An object containing cancer-associated genes annotated by OncoKB.

Examples

```
data(oncokb_genes)
length(oncokb_genes)
head(oncokb_genes)
```

`oncokb_truncating_genes` *OncoKB v3.9 cancer genes consider for truncating mutations*

Description

cancer genes list consider for truncating mutations

Usage

```
data(oncokb_truncating_genes)
```

Format

A list

Value

An object containing genes considered for truncating-mutation analyses.

Examples

```
data(oncokb_truncating_genes)
length(oncokb_truncating_genes)
head(oncokb_truncating_genes)
```

`overlap_pair_extract` *Extract null-model weighted overlap distribution for a gene pair*

Description

Returns the vector of weighted co-mutation values from all null-model permutations for a specific pair of genes.

Usage

```
overlap_pair_extract(gene1, gene2, obj)
```

Arguments

gene1	Name of the first gene/alteration.
gene2	Name of the second gene/alteration.
obj	SelectX object (list returned by selectX()$obj).

Value

Numeric vector of length obj$nSim with the null-model weighted overlap values for the pair.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
genes <- rownames(result$obj$al$am$full)[1:2]
overlap_pair_extract(genes[1], genes[2], result$obj)
```

---

r.am.pairwise.alteration.overlap

Compute null overlap matrix

Description

Compute null overlap matrix

Usage

```
r.am.pairwise.alteration.overlap(null, n.permut, n.cores = 1)
```

Arguments

null	The null model (list of simulated binary matrices)
n.permut	Accepted for API compatibility; currently unused internally.
n.cores	Accepted for API compatibility; currently unused internally.

Value

Overlap matrix summed across all permutations.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
r.am.pairwise.alteration.overlap(result$obj$null, n.permut = result$obj$nSim)
```

---

r.effectSize *Compute effect sizes for null model permutations*

Description

Compute effect sizes for null model permutations

Usage

```
r.effectSize(null_overlap, mean_mat, n.permut = 1000, n.cores = 1)
```

Arguments

null_overlap	List of null model overlap matrices (one per permutation)
mean_mat	Mean overlap matrix across all permutations
n.permut	Number of permutations
n.cores	Number of cores (currently unused; sequential only)

Value

List of effect size vectors, one per permutation.

Examples

```
data(luad_run_data, package = "SelectSim")
result  <- selectX(M = luad_run_data$M,
                   sample.class = luad_run_data$sample.class,
                   alteration.class = luad_run_data$alteration.class,
                   n.cores = 1, min.freq = 10, n.permut = 10,
                   verbose = FALSE)
null_ov  <- r.am.pairwise.alteration.overlap(result$obj$null,
                                             n.permut = result$obj$nSim)
mean_mat <- Reduce("+", null_ov) / result$obj$nSim
r.effectSize(null_ov, mean_mat, n.permut = result$obj$nSim)
```

retrieveOutliers *Identify outlier null-model matrices*

Description

Flags simulations whose mean per-gene and per-sample deviation from the observed counts falls in the top 10%, indicating numerical instability. These are removed before computing effect sizes to prevent them from inflating the null distribution.

Usage

```
retrieveOutliers(obj, nSim = 1000)
```

Arguments

obj	SelectX object (list with al and null fields)
nSim	Number of simulations in the null model

Value

Logical vector; TRUE for each null matrix flagged as an outlier.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
outliers <- retrieveOutliers(result$obj, nSim = result$obj$nSim)
sum(outliers)
```

ridge_plot_ed *Ridge plot of null-model background distribution for significant gene pairs*

Description

For each significant evolutionary dependency in `result_df`, plots the null-model weighted-overlap distribution as a ridge, with vertical lines marking the observed overlap (red) and mean background (blue).

Usage

```
ridge_plot_ed(result_df, obj)
```

Arguments

result_df	Subset of the result table from selectX()$result containing only the pairs you want to display (e.g., significant hits).
obj	SelectX object (selectX()$obj).

Value

A ggplot2 ridge-plot object.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
sig_pairs <- head(result$result[result$result$FDR, ], 3)
if (nrow(sig_pairs) > 0) ridge_plot_ed(sig_pairs, result$obj)
```

---

ridge_plot_ed_compare *Ridge plot comparing null-model distributions for two datasets*

Description

For each gene pair in result_df, overlays the null-model weighted-overlap distributions from two separate selectX runs, allowing visual comparison of evolutionary dependencies across cohorts or conditions.

Usage

```
ridge_plot_ed_compare(result_df, obj1, obj2, name1, name2)
```

Arguments

result_df	A data frame of gene pairs to display, with columns SFE_1, SFE_2, name, type, dataset1_w_overlap, and dataset2_w_overlap.
obj1	SelectX object for dataset 1 (selectX()$obj).
obj2	SelectX object for dataset 2 (selectX()$obj).
name1	Display label for dataset 1.
name2	Display label for dataset 2.

Value

A ggplot2 ridge-plot object.

Examples

```
data(luad_run_data, package = "SelectSim")
r1 <- selectX(M = luad_run_data$M,
              sample.class = luad_run_data$sample.class,
              alteration.class = luad_run_data$alteration.class,
              n.cores = 1, min.freq = 10, n.permut = 10, verbose = FALSE)
r2 <- selectX(M = luad_run_data$M,
              sample.class = luad_run_data$sample.class,
              alteration.class = luad_run_data$alteration.class,
              n.cores = 1, min.freq = 10, n.permut = 10, verbose = FALSE)
common <- head(r1$result, 3)
common$dataset1_w_overlap <- common$w_overlap
common$dataset2_w_overlap <- common$w_overlap
ridge_plot_ed_compare(common, r1$obj, r2$obj, "Run1", "Run2")
```

---

selectX *SelectX main function from SelectSim to create alteration object with background model*

Description

`selectX()` takes a list object which consist of genome alteration matrix and tumor mutation burden.

Usage

```
selectX(
  M,
  sample.class,
  alteration.class,
  n.cores = 1,
  min.freq = 10,
  n.permut = 1000,
  lambda = 0.3,
  tau = 1,
  save.object = FALSE,
  folder = "./",
  verbose = TRUE,
  estimate_pairwise = FALSE,
  maxFDR = 0.25,
  seed = 42
)
```

Arguments

M	a list data object which consist of gams and tmbs.
sample.class	sample covariates as named list.
alteration.class	alteration covariates as named list.
n.cores	no of cores.
min.freq	number of samples for features to be at least mutated in.
n.permut	number of simulations.
lambda	lambda parameter.
tau	tau (fold change) parameter.
save.object	store the SelectX object.
folder	folder path to store the results.
verbose	print the time and each steps.
estimate_pairwise	Compute pairwise p-value
maxFDR	FDR value
seed	a random seed

Value

result a SelectSim object with background model and other info along with result table

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(
  M               = luad_run_data$M,
  sample.class    = luad_run_data$sample.class,
  alteration.class = luad_run_data$alteration.class,
  n.cores         = 1,
  min.freq        = 10,
  n.permut        = 10,
  verbose         = FALSE
)
head(result$result)
```

stat_maf_column *Summary functions for MAF file*

Description

stat_maf_column() takes a maf file and filters a MAF dataframe by retaining only the rows with a column value included in the values list

Usage

```
stat_maf_column(maf, column, ...)
```

Arguments

maf	a maf as dataframe
column	a data-frame schema; see TCGA_maf_schema for an example
...	Other options

Value

filtered_maf a filtered maf file

Examples

```
data(luad_maf, package = "SelectSim")
stat_maf_column(luad_maf, column = "Variant_Classification")
```

stat_maf_gene *Count mutations per gene in a MAF file*

Description

Count mutations per gene in a MAF file

Usage

```
stat_maf_gene(maf, column = "Hugo_Symbol", ...)
```

Arguments

maf	A MAF dataframe
column	Column name containing gene symbols
...	Additional arguments passed to stat_maf_column

Value

Table of mutation counts per gene.

Examples

```
data(luad_maf, package = "SelectSim")
head(stat_maf_gene(luad_maf))
```

`stat_maf_sample` *Count mutations per sample in a MAF file*

Description

Count mutations per sample in a MAF file

Usage

```
stat_maf_sample(maf, column = "Tumor_Sample_Barcode", ...)
```

Arguments

`maf`	A MAF dataframe
`column`	Column name containing sample IDs
`...`	Additional arguments passed to stat_maf_column

Value

Table of mutation counts per sample.

Examples

```
data(luad_maf, package = "SelectSim")
head(stat_maf_sample(luad_maf))
```

TCGA_maf_schema *TCGA_maf_schema: schema for TCGA maf file to process the mutations*

Description

TCGA_maf_schema: schema for TCGA maf file to process the mutations

Usage

```
TCGA_maf_schema
```

Value

A list defining the expected TCGA MAF column schema.

Examples

```
str(TCGA_maf_schema)
```

template.obj.gen *Generate the template matrix*

Description

Computes the expected mutation probability matrices (one per GAM type, per sample block) used as the simulation background in `null_model_parallel`.

Usage

```
template.obj.gen(al)
```

Arguments

`al`	Alteration landscape object

Value

A list with `template.obj` (per-block S matrices) and `temp_mat` (full concatenated template matrices, one per GAM type).

Examples

```
data(luad_run_data, package = "SelectSim")
al <- new.AL.general(luad_run_data$M,
                     feat.covariates  = luad_run_data$alteration.class,
                     sample.covariates = luad_run_data$sample.class,
                     min.freq = 10)
template.obj.gen(al)
```

---

theme_Publication *A clean ggplot2 theme for publication-quality plots*

Description

A clean ggplot2 theme for publication-quality plots

Usage

```
theme_Publication(base_size = 14, base_family = "sans")
```

Arguments

base_size	size of fonts
base_family	type of font

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
ggplot(data.frame(x = 1:3, y = 1:3), aes(x, y)) +
  geom_point() + theme_Publication()
```

variant_catalogue *OncoKB v3.9 cancer genes*

Description

A dataframe cancer genes with missense mutation annotations

Usage

```
data(variant_catalogue)
```

Format

A data frame

Value

A data frame containing cancer-gene and variant annotations.

Examples

```
data(variant_catalogue)
dim(variant_catalogue)
head(variant_catalogue)
```

w.r.am.pairwise.alteration.overlap *Compute null weighted overlap matrix*

Description

Compute null weighted overlap matrix

Usage

```
w.r.am.pairwise.alteration.overlap(null, W, n.permut, n.cores = 1)
```

Arguments

null	The null model (list of simulated binary matrices)
W	The weight matrix (features x samples)
n.permut	Accepted for API compatibility; currently unused internally.
n.cores	Accepted for API compatibility; currently unused internally.

Value

Weighted overlap matrix summed across all permutations.

Examples

```
data(luad_run_data, package = "SelectSim")
result <- selectX(M = luad_run_data$M,
                  sample.class = luad_run_data$sample.class,
                  alteration.class = luad_run_data$alteration.class,
                  n.cores = 1, min.freq = 10, n.permut = 10,
                  verbose = FALSE)
w.r.am.pairwise.alteration.overlap(result$obj$null, W = result$obj$W$W,
                                   n.permut = result$obj$nSim)
```

Index

∗ **datasets**
    luad_maf, 23
    luad_result, 24
    luad_run_data, 24
    oncokb_genes, 30
    oncokb_truncating_genes, 31
    variant_catalogue, 42

add, 3
al.pairwise.alteration.stats, 4
al.stats, 4
am.pairwise.alteration.coverage, 5
am.pairwise.alteration.overlap, 6
am.stats, 7
am.weight.pairwise.alteration.overlap, 7

binary.yule, 8

effectSize, 9
estimate_p_val, 9
estimate_pairwise_p, 10
estimateFDR2, 11

filter_maf_column, 11
filter_maf_complex, 12
filter_maf_gene.name, 13
filter_maf_ignore, 14
filter_maf_missense, 14
filter_maf_mutation.type, 15
filter_maf_mutations, 16
filter_maf_sample, 17
filter_maf_schema, 17
filter_maf_truncating, 18

generateS, 19
generateW_block, 19
generateW_mean_tmb, 20
GENIE_maf_schema, 21
get.blocks, 21

interaction.table, 22

luad_maf, 23
luad_result, 24
luad_run_data, 24

maf2gam, 25
mutation_type, 26

new.AL.general, 26
new.ALS, 28
new.AMS, 28
null_model_parallel, 29

obs_exp_scatter, 30
oncokb_genes, 30
oncokb_truncating_genes, 31
overlap_pair_extract, 31

r.am.pairwise.alteration.overlap, 32
r.effectSize, 33
retrieveOutliers, 34
ridge_plot_ed, 34
ridge_plot_ed_compare, 35

selectX, 36
stat_maf_column, 38
stat_maf_gene, 38
stat_maf_sample, 39

TCGA_maf_schema, 40
template.obj.gen, 40
theme_Publication, 41

variant_catalogue, 42

w.r.am.pairwise.alteration.overlap, 42