Package: CaDrA (via r-universe)

May 24, 2026

Type Package

Title Candidate Driver Analysis

Version 1.10.0

Description Performs both stepwise and backward heuristic search for candidate (epi)genetic drivers based on a binary multi-omics dataset. CaDrA's main objective is to identify features which, together, are significantly skewed or enriched pertaining to a given vector of continuous scores (e.g. sample-specific scores representing a phenotypic readout of interest, such as protein expression, pathway activity, etc.), based on the union occurence (i.e. logical OR) of the events.

Depends R (>= 4.4.0)

LazyData false

License GPL-3 + file LICENSE

URL https://github.com/montilab/CaDrA/

Encoding UTF-8

RoxygenNote 7.3.3

Imports doParallel, ggplot2, gplots, graphics, grid, gtable, knnmi, MASS, methods, misc3d, plyr, ppcor, R.cache, reshape2, stats, SummarizedExperiment

Suggests BiocManager, devtools, knitr, pheatmap, rmarkdown, testthat (>= 3.1.6)

Config/testthat/edition 3

biocViews Microarray, RNASeq, GeneExpression, Software, FeatureExtraction

VignetteBuilder knitr

BugReports https://github.com/montilab/CaDrA/issues

Config/pak/sysreqs libicu-dev zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-04-28 13:01:04 UTC

RemoteUrl https://github.com/bioc/CaDrA

RemoteRef RELEASE_3_23

RemoteSha 17d98f67f391fd6b755dbe0615cfa92be62f5a67

Contents

`BRCA_GISTIC_MUT_SIG` *Genomic Data from TCGA BRCA MUT + GISTIC*

Description

A SummarizedExperiment object consists of 16,873 genomic features across 951 samples.

Usage

```
data(BRCA_GISTIC_MUT_SIG)
```

Format

An object of class `SummarizedExperiment` from `SummarizedExperiment` package containing an assay of 16,873 rows (features) and 951 columns (samples) see `SummarizedExperiment` for more details.

Value

a SummarizedExperiment object

References

Kartha VK, Kern JG, Sebastiani P, Zhang L, Varelas X, Monti S (2019) CaDrA: A computational framework for performing candidate driver analyses using binary genomic features. (Frontiers in Genetics)

`CaDrA` *CaDrA Search*

Description

Perform permutation-based testings on a sample of permuted input scores using `candidate_search` as the main iterative function for each run.

Usage

```
CaDrA(
  FS,
  input_score,
 method = c("ks_pval", "ks_score", "wilcox_pval", "wilcox_score", "revealer", "knnmi",
    "correlation", "custom"),
  method_alternative = c("less", "greater", "two.sided"),
  cmethod = c("spearman", "pearson"),
  custom_function = NULL,
  custom_parameters = NULL,
  weights = NULL,
  search_start = NULL,
  top_N = 1,
  search_method = c("both", "forward"),
  max_size = 7,
  n_perm = 1000,
  perm_alternative = c("one.sided", "two.sided"),
  obs_best_score = NULL,
  smooth = TRUE,
  plot = FALSE,
  ncores = 1,
  cache = FALSE,
  cache_path = NULL,
  verbose = FALSE
)
```

Arguments

`FS`	a matrix of binary features or a SummarizedExperiment class object from SummarizedExperiment package where rows represent features of interest (e.g. genes, transcripts, exons, etc...) and columns represent the samples. The assay of FS contains binary (1/0) values indicating the presence/absence of omics features.

`input_score` a vector of continuous scores representing a phenotypic readout of interest such as protein expression, pathway activity, etc.

NOTE: `input_score` object must have names or labels that match the column names of `FS` object.

`method` a character string specifies a scoring method that is used in the search. There are 6 options: (`"ks_pval"` or `ks_score` or `"wilcox_pval"` or `wilcox_score` or `"revealer"` (conditional mutual information from REVEALER) or `"knnmi"` (K-Nearest Neighbor Mutual Information Estimator from knnmi) or `"correlation"` or `"custom"` (a user-defined scoring method)). Default is `ks_pval`.

`method_alternative`

a character string specifies an alternative hypothesis testing (`"two.sided"` or `"greater"` or `"less"`). Default is `less` for left-skewed significance testing.

NOTE: This argument only applies to `ks_pval` and `wilcox_pval` method

`cmethod` correlation method to use - spearman or pearson. Default is "spearman".

NOTE: This argument only applies to `correlation` method

`custom_function`

If method is `"custom"`, specifies a user-defined function here. Default is `NULL`.

NOTE: `custom_function` must take `FS` and `input_score` as its input arguments and its final result must return a vector of row-wise scores where its labels or names match the row names of `FS` object.

`custom_parameters`

If method is `"custom"`, specifies a list of additional arguments (excluding `FS` and `input_score`) to be passed to `custom_function`. For example: custom_parameters = list(alternative = "less"). Default is `NULL`.

`weights` If method is `ks_score` or `ks_pval`, specifying a vector of weights will perform a weighted-KS testing. Default is `NULL`.

NOTE: `weights` must have names or labels that match the labels of `input_score`.

`search_start` a vector of character strings (separated by commas) specifies feature names in the FS object to start the search with. If `search_start` is provided, then `top_N` parameter will be ignored and vice versa. Default is `NULL`.

`top_N` an integer specifies the number of features to start the search over. By default, it starts with the feature that has the highest best score (top_N = 1).

NOTE: If `top_N` is provided, then `search_start` parameter will be ignored and vice versa. If top_N > 10, it may result in a longer search time.

`search_method` a character string specifies an algorithm to filter out the best candidates (`"forward"` or `"both"`). Default is `both` (i.e., backward and forward).

`max_size` an integer specifies a maximum size that a meta-feature can extend to do for a given search. Default is `7`.

`n_perm` an integer specifies the number of permutations to perform. Default is `1000`.

`perm_alternative`

an alternative hypothesis type for calculating permutation-based p-value. Options: one.sided, two.sided. Default is `one.sided`.

`obs_best_score` a numeric value corresponding to the best observed score. This value is used to compare against the `n_perm` calculated best scores. Default is `NULL`. If set to NULL, we will compute the observed best score based on the given parameters.

smooth	a logical value indicates whether or not to add a smoothing factor of 1 to the calculation of permutation-based p-value. This option is used to avoid a returned p-value of 0. Default is TRUE.
plot	a logical value indicates whether or not to plot the empirical null distribution of the permuted best scores. Default is FALSE.
ncores	an integer specifies the number of cores to perform parallelization for permutation-based testing. Default is 1.
cache	a logical value determines whether or not to cache the permuted best scores. This helps to save time for future loading instead of re-computing the permutation-based testing every time. Default is FALSE.
cache_path	a path to cache permuted best scores. Default is NULL. If NULL, the cache path is set to system home directory (e.g. $HOME/.Rcache) for future loading.
verbose	a logical value indicates whether or not to print the diagnostic messages. Default is FALSE.

Value

a list of 4 objects: key, perm_best_scores, obs_best_score, perm_pval

-key: a list of parameters that was used to cache the results of the permutation-based testing. This is useful as the permuted best scores can be recycled to save time for future loading.

-perm_best_scores: a vector of permuted best scores obtained by performing candidate_search over n_perm iterations of permuted input scores.

-obs_best_score: a user-provided best score or an observed best score obtained by performing candidate_search on a given dataset and input parameters. This value is later used to compare against the permuted best scores (perm_best_scores).

perm_pval: a permutation-based p-value obtained by calculating sum(perm_best_scores > obs_best_score)/n_perm

NOTE: If smooth = TRUE, a smoothing factor of 1 will be added to the calculation of perm_pval.

e.g. (sum(perm_best_scores > obs_best_score) + 1) / (n_perm + c)

This is just to not return a p-value of 0

Examples

```
# Load pre-computed feature set
data(sim_FS)

# Load pre-computed input-score
data(sim_Scores)

# Set seed for permutation
set.seed(21)

# Define additional parameters and start the search function
cadra_result <- CaDrA(
  FS = sim_FS, input_score = sim_Scores, method = "ks_pval",
  weights = NULL, method_alternative = "less", top_N = 1,
  search_start = NULL, search_method = "both", max_size = 7,
```

```
  n_perm = 10, perm_alternative = "one.sided", plot = FALSE,
  smooth = TRUE, obs_best_score = NULL,
  ncores = 1, cache = FALSE, cache_path = NULL
)
```

`calc_rowscore` *Calculate row-wise scores of a given binary feature set based on a given scoring method*

Description

Calculate row-wise scores of a given binary feature set based on a given scoring method

Usage

```
calc_rowscore(
  FS,
  input_score,
  meta_feature = NULL,
 method = c("ks_pval", "ks_score", "wilcox_pval", "wilcox_score", "revealer", "knnmi",
    "correlation", "custom"),
  method_alternative = c("less", "greater", "two.sided"),
  cmethod = c("spearman", "pearson"),
  custom_function = NULL,
  custom_parameters = NULL,
  weights = NULL,
  do_check = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`FS`	a matrix of binary features or a SummarizedExperiment class object from SummarizedExperiment package where rows represent features of interest (e.g. genes, transcripts, exons, etc...) and columns represent the samples. The assay of FS contains binary (1/0) values indicating the presence/absence of omics features.
`input_score`	a vector of continuous scores representing a phenotypic readout of interest such as protein expression, pathway activity, etc. NOTE: `input_score` object must have names or labels that match the column names of `FS` object.
`meta_feature`	a vector of one or more features representing known causes of activation or features associated with a response of interest (`e.g. input_score`). Default is NULL.

`method` a character string specifies a scoring method that is used in the search. There are 6 options: ("ks_pval" or ks_score or "wilcox_pval" or wilcox_score or "revealer" (conditional mutual information from REVEALER) or "knnmi" (k-Nearest Neighbor Mutual Information Estimator from knnmi) or "correlation" (based on simple correlation - pearson or spearman) or "custom" (a user-defined scoring method)). Default is ks_pval.

`method_alternative`
a character string specifies an alternative hypothesis testing ("two.sided" or "greater" or "less"). Default is less for left-skewed significance testing.

NOTE: This argument only applies to ks_pval and wilcox_pval method

`cmethod` correlation method to use - spearman or pearson. Default is "spearman" #' NOTE: This argument only applies to correlation method only

`custom_function`
if method is "custom", specifies a user-defined function here. Default is NULL.

NOTE: custom_function must take FS and input_score as its input arguments, and its final result must return a vector of row-wise scores where its labels or names matched the row names of FS object.

`custom_parameters`
if method is "custom", specifies a list of additional arguments (excluding FS and input_score) to be passed to custom_function. For example: custom_parameters = list(alternative = "less"). Default is NULL.

`weights` If method is ks_score or ks_pval, specifying a vector of weights will perform a weighted-KS testing. Default is NULL.

NOTE: weights must have names or labels that match the names or labels of input_score.

`do_check` a logical value indicates whether or not to validate if the given parameters (FS and input_score) are valid inputs. Default is TRUE.

`verbose` a logical value indicates whether or not to print the diagnostic messages. Default is FALSE.

`...` additional parameters to be passed to custom_function

Value

return a vector of row-wise positive scores where it is ordered from most significant to least significant (e.g. from highest to lowest values) and its labels or names must match the row names of FS object

Examples

```
# Create a feature matrix
mat <- matrix(c(1,0,1,0,0,0,0,0,1,0,
                0,0,1,0,1,0,1,0,0,0,
                0,0,0,0,1,0,1,0,1,0), nrow=3)

colnames(mat) <- 1:10
row.names(mat) <- c("TP_1", "TP_2", "TP_3")
```

```
# Create a vector of observed input scores
set.seed(42)
input_score = rnorm(n = ncol(mat))
names(input_score) <- colnames(mat)

# Run the ks method
ks_rowscore_result <- calc_rowscore(
  FS = mat,
  input_score = input_score,
  meta_feature = NULL,
  method = "ks_pval",
  method_alternative = "less",
  weights = NULL
)

# Run the wilcoxon method
wilcox_rowscore_result <- calc_rowscore(
  FS = mat,
  input_score = input_score,
  meta_feature = NULL,
  method = "wilcox_pval",
  method_alternative = "less"
)

# Run the revealer method
revealer_rowscore_result <- calc_rowscore(
  FS = mat,
  input_score = input_score,
  meta_feature = NULL,
  method = "revealer"
)

# Run the revealer method
knnmi_rowscore_result <- calc_rowscore(
  FS = mat,
  input_score = input_score,
  meta_feature = NULL,
  method = "knnmi"
)

# Run the correlation method
 corr_result <- calc_rowscore(
   FS = mat,
   input_score = input_score,
   meta_feature = NULL,
   method = "correlation",
   cmethod = "spearman"
)

# A customized function using ks-test function
customized_ks_rowscore <- function(FS, input_score, meta_feature=NULL, alternative="less"){

  # Check if meta_feature is provided
```

```
if(!is.null(meta_feature)){
  # Getting the position of the known meta features
  locs <- match(meta_feature, row.names(FS))

  # Taking the union across the known meta features
  if(length(locs) > 1) {
    meta_vector <- as.numeric(ifelse(colSums(FS[locs,]) == 0, 0, 1))
  }else{
    meta_vector <- as.numeric(FS[locs,])
  }

  # Remove the meta features from the binary feature matrix
  # and taking logical OR btw the remaining features with the meta vector
  FS <- base::sweep(FS[-locs, , drop=FALSE], 2, meta_vector, `|`)*1

  # Check if there are any features that are all 1s generated from
  # taking the union between the matrix
  # We cannot compute statistics for such features and thus they need
  # to be filtered out
  if(any(rowSums(FS) == ncol(FS))){
    warning("Features with all 1s generated from taking the matrix union ",
            "will be removed before progressing...\n")
    FS <- FS[rowSums(FS) != ncol(FS), , drop=FALSE]
  }
}

# KS is a ranked-based method
# So we need to sort input_score from highest to lowest values
input_score <- sort(input_score, decreasing=TRUE)

# Re-order the matrix based on the order of input_score
FS <- FS[, names(input_score), drop=FALSE]

# Compute the scores using the KS method
ks <- apply(FS, 1, function(r){
  x = input_score[which(r==1)];
  y = input_score[which(r==0)];
  res <- ks.test(x, y, alternative=alternative)
  return(c(res$statistic, res$p.value))
})

# Obtain score statistics
stat <- ks[1,]

# Obtain p-values and change values of 0 to the machine lowest value
# to avoid taking -log(0)
pval <- ks[2,]
pval[which(pval == 0)] <- .Machine$double.xmin

# Compute the -log(pval)
# Make sure scores has names that match the row names of FS object
scores <- -log(pval)
names(scores) <- rownames(FS)
```

```
  return(scores)

}

# Search for best features using a custom-defined function
custom_rowscore_result <- calc_rowscore(
  FS = mat,
  input_score = input_score,
  meta_feature = NULL,
  method = "custom",
  custom_function = customized_ks_rowscore,
  custom_parameters = NULL
)
```

---

candidate_search *Candidate Search*

Description

Performs heuristic search on a set of binary features to determine whether there are features whose union is more skewed (enriched at the extremes) than either features alone. This is the main functionality of the CaDrA package.

Usage

```
candidate_search(
  FS,
  input_score,
 method = c("ks_pval", "ks_score", "wilcox_pval", "wilcox_score", "revealer", "knnmi",
    "correlation", "custom"),
  method_alternative = c("less", "greater", "two.sided"),
  cmethod = c("spearman", "pearson"),
  custom_function = NULL,
  custom_parameters = NULL,
  weights = NULL,
  search_start = NULL,
  top_N = 1,
  search_method = c("both", "forward"),
  max_size = 7,
  best_score_only = FALSE,
  do_plot = FALSE,
  verbose = FALSE
)
```

Arguments

FS	a matrix of binary features or a SummarizedExperiment class object from SummarizedExperiment package where rows represent features of interest (e.g. genes, transcripts, exons, etc...) and columns represent the samples. The assay of FS contains binary (1/0) values indicating the presence/absence of omics features.
input_score	a vector of continuous scores representing a phenotypic readout of interest such as protein expression, pathway activity, etc. NOTE: input_score object must have names or labels that match the column names of FS object.
method	a character string specifies a scoring method that is used in the search. There are 7 options: ("ks_pval" or ks_score or "wilcox_pval" or wilcox_score or "revealer" (conditional mutual information from REVEALER) or "knnmi" (K-Nearest Neighbor Mutual Information Estimator from knnmi) or "correlation" (based on simple correlation - pearson or spearman) or "custom" (a user-defined scoring method)). Default is ks_pval.
method_alternative	a character string specifies an alternative hypothesis testing ("two.sided" or "greater" or "less"). Default is less for left-skewed significance testing. NOTE: This argument only applies to ks_pval and wilcox_pval method
cmethod	correlation method to use - spearman or pearson. Default is "spearman". NOTE: This argument only applies to correlation method only
custom_function	if method is "custom", specifies a user-defined function here. Default is NULL. NOTE: custom_function must take FS and input_score as its input arguments and its final result must return a vector of row-wise scores where its labels or names match the row names of FS object.
custom_parameters	if method is "custom", specifies a list of additional arguments (excluding FS and input_score) to be passed to the custom_function. For example: custom_parameters = list(alternative = "less"). Default is NULL.
weights	if method is ks_score or ks_pval, specifying a vector of weights will perform a weighted-KS testing. Default is NULL. NOTE: weights must have names or labels that match the labels of input_score.
search_start	a vector of character strings (separated by commas) specifies feature names in the FS object to start the search with. If search_start is provided, then top_N parameter will be ignored and vice versa. Default is NULL.
top_N	an integer specifies the number of features to start the search over. By default, it starts with the feature that has the highest best score (top_N = 1). NOTE: If top_N is provided, then search_start parameter will be ignored and vice versa. If top_N > 10, it may result in a longer search time.
search_method	a character string specifies an algorithm to filter out the best features ("forward" or "both"). Default is both (i.e. backward and forward).
max_size	an integer specifies a maximum size that a meta-feature can extend to do for a given search. Default is 7.

best_score_only
: a logical value indicates whether or not to return the best score corresponding to each top N searches only. Default is FALSE.

do_plot
: a logical value indicates whether or not to plot the overlapping features of the resulting meta-feature matrix.

  NOTE: plot can only be produced if the resulting meta-feature matrix contains more than 1 feature (e.g. length(search_start) > 1 or top_N > 1). Default is FALSE.

verbose
: a logical value indicates whether or not to print the diagnostic messages. Default is FALSE.

Details

NOTE: The legacy function topn_eval is equivalent to the recommended candidate_search function

Value

If best_score_only = TRUE, the heuristic search will return the best feature whose its union meta-feature matrix has the highest score among the top_N feature searches. If best_score_only = FALSE, a list of objects pertaining to top_N feature searches will be returned. For each top_N feature search, the candidate search will contain 7 objects: (1) its best meta-feature matrix (feature_set), (2) its observed input scores (input_score), (3) its corresponding best score pertaining to the union meta-feature matrix (score), (4) names of the best meta-features (best_features), (5) rank of the best meta-features in term of their best scores (best indices), (6) marginal scores of the best meta-features (marginal_best_scores), (7) cumulative scores of the best meta-features (cumulative_best_scores).

Examples

```
# Load pre-computed feature set
data(sim_FS)

# Load pre-computed input scores
data(sim_Scores)

# Define additional parameters and run the function
candidate_search_result <- candidate_search(
  FS = sim_FS, input_score = sim_Scores,
  method = "ks_pval", method_alternative = "less", weights = NULL,
  search_start = NULL, top_N = 3, search_method = "both",
  max_size = 7, best_score_only = FALSE
)
```

CCLE_MUT_SCNA *Genomic Data from CCLE MUT + SCNA*

Description

A SummarizedExperiment object consists of 17,724 genomic features across 82 samples.

Usage

```
data(CCLE_MUT_SCNA)
```

Format

An object of class `SummarizedExperiment` from `SummarizedExperiment` package containing a matrix of 17,724 rows (features) and 82 columns (samples). See `SummarizedExperiment` for more details.

Value

a SummarizedExperiment object

References

Kim, J., Botvinnik, O., Abudayyeh, O. et al. Characterizing genomic alterations in cancer by complementary functional associations. Nat Biotechnol 34, 539–546 (2016). https://doi.org/10.1038/nbt.3527

CTNBB1_reporter *Transcriptional Activity of Beta-Catenin in Cancers*

Description

A vector of continuous scores represents the activation of B-catenin across multiple cancer cell lines

Usage

```
data(CTNBB1_reporter)
```

Format

Consists of a vector of continuous scores of B-catenin activity across 82 cancer cell lines. The mutation and copy number associated with this sample cohorts can be found in `CCLE_MUT_SCNA` dataset.

Value

a vector of continuous scores

References

Kim, J., Botvinnik, O., Abudayyeh, O. et al. Characterizing genomic alterations in cancer by complementary functional associations. Nat Biotechnol 34, 539–546 (2016). https://doi.org/10.1038/nbt.3527

`generate_permutations` *Random permutation matrix generator*

Description

Produces a random permutation score matrix given a vector of sample-specific scores representing a phenotypic readout of interest such as protein expression, pathway activity, etc.

Usage

```
generate_permutations(input_score, n_perm)
```

Arguments

`input_score`	a vector of continuous scores of a molecular phenotype of interest such as protein expression, pathway activity, etc. NOTE: The `input_score` object must have names or labels to track samples by.
`n_perm`	a number of permutations to generate. This determines the number of rows in the permutation matrix.

Value

a matrix of values where each row contains scores of a single permuted `input_score`.

Examples

```
# Load pre-simulated scores
data(sim_Scores)

# Set seed for permutation
set.seed(123)

# Define number of permutations
n_perm = 1000

# Generate permuted scores
perm_matrix <- generate_permutations(
  input_score = sim_Scores,
  n_perm = n_perm
)
```

meta_plot *Candidate Drivers Search Plot*

Description

By utilizing the top N results obtained from candidate_search, we can find the best meta-feature among the top N searches using topn_best. meta_plot is then used to produce graphics including a tile plot for the top meta-features that associated with a molecular phenotype of interest (e.g. input_score), the KS enrichment plot of the meta-features, and lastly, a density diagram of the distribution of the observed input scores sorted from largest to smallest at the top.

Usage

```
meta_plot(topn_best_list, input_score_label = NULL, plot_title = NULL)
```

Arguments

topn_best_list	a list of objects returned from candidate_search corresponding to the search of top N features given by top_N value. The topn_best_list contains the best meta-feature matrix, its corresponding best score, its observed input scores, rank of the best features based on their scores, marginal best scores, and cumulative best scores.
input_score_label	a label that references to the input_score variable that was used to compute the top N best features. Default is NULL.
plot_title	a title to the plot. Default is NULL.

Value

3 plots stacked on top of each other: 1. a density diagram of observed input scores sorted from highest to lowest 2. a tile plot of the top features within the meta-feature set 3. a KS enrichment plot of the meta-feature set (this correspond to the logical OR of the features)

Examples

```
# Load pre-computed Top-N list generated for sim_FS dataset
data(topn_list)

# With the results obtained from top-N evaluation,
# We can find the combination of features that gives the best score in
# top N searches
topn_best_meta <- topn_best(topn_list = topn_list)

# Now we can plot this set of best meta-feature
meta_plot(topn_best_list = topn_best_meta)
```

perm_res *Pre-computed permutation results for simulated data (sim_FS)*

Description

The permutation result returned from CaDrA using pre-simulated dataset (FS = sim_FS), pre-simulated input scores (input_score = sim_Scores), top_N = 7, method = "ks_pval", alternative = "less", search_method = "both", max_size = 10, obs_best_score = NULL and n_perm = 1000 as inputs to the function.

Usage

```
data(perm_res)
```

Format

A list of objects returned from CaDrA function. The resulting object contains a list of key parameters that was used to run the permutation-based testing, a vector of permuted best scores for a given n_perm, an observed best score, and a permuted p-value.

To visualize the Empirical Null Distribution of the permuted best scores over n_perm iterations, just pass the resulting list to permutation_plot.

See permutation_plot for more details.

Value

a list of objects returned from CaDrA function

References

Kartha VK, Kern JG, Sebastiani P, Zhang L, Varelas X, Monti S (2017) CaDrA: A computational framework for performing candidate driver analyses using binary genomic features. (Frontiers in Genetics)

Examples

```
# Load the pre-computed permutation results for sim_FS
data(perm_res)

# Plot the Empirical Null Distribution of the permuted best scores
# against its observed best score
permutation_plot(perm_res = perm_res)
```

permutation_plot *Permutation Best Scores Plot*

Description

Plot the Empirical Null Distribution of Permutation Best Scores returned from `CaDrA` function

Usage

```
permutation_plot(perm_res)
```

Arguments

`perm_res`	a list of objects returned from `CaDrA` function. The returning object contains a list of key parameters that are used to run the permutation-based testing, a vector of permuted best scores for a given `n_perm`, an observed best score, and a computed permutation p-value.

Value

a density plot

Examples

```
# Load pre-computed permutation results
data(perm_res)

# Plot the permutation results
permutation_plot(perm_res)
```

prefilter_data *Pre-filter features*

Description

Pre-filter a dataset prior running `candidate_search` to avoid testing features that are too prevalent or too sparse across samples in the dataset

Usage

```
prefilter_data(FS, max_cutoff = 0.6, min_cutoff = 0.03, verbose = FALSE)
```

Arguments

FS	a matrix of binary features or a SummarizedExperiment class object from SummarizedExperiment package where rows represent features of interest (e.g. genes, transcripts, exons, etc...) and columns represent the samples. The assay of FS contains binary (1/0) values indicating the presence/absence of 'omics' features.
max_cutoff	a numeric value between 0 and 1 describing the absolute prevalence of a feature across all samples in the FS object which the feature will be filtered out. Default is 0.6 (feature that occur in 60 percent or more of the samples will be removed)
min_cutoff	a numeric value between 0 and 1 describing the absolute prevalence of a feature across all samples in the FS object which the feature will be filtered out. Default is 0.03 (feature that occur in 3 percent or less of the samples will be removed)
verbose	a logical value indicates whether or not to print the diagnostic messages. Default is FALSE.

Value

A SummarizedExperiment object with only the filtered-in features given the filtered thresholds

Examples

```
# Load pre-computed feature set
data(sim_FS)

# Filter out features having < 3% and > 60% prevalence across all samples
# by (default)
sim_FS_filt1 <- prefilter_data(FS = sim_FS)

# Change the min cut-off to 1% prevalence, instead of the default of 3%
sim_FS_filt2 <- prefilter_data(FS = sim_FS, min_cutoff  = 0.01)

# Change the max cut-off to 65% prevalence, instead of the default of 60%
sim_FS_filt3 <- prefilter_data(FS = sim_FS, max_cutoff = 0.65)
```

---

sim_FS *Simulated Genomic Data*

Description

A simulated SummarizedExperiment object that comprises of 1000 genomic features (rows) and 100 sample profiles (columns). Each row is represented by a vector of binary values (1/0) indicating the presence/absence of the feature in the samples. This simulated data includes 10 left-skewed (i.e. True Positive or TP) and 990 uniformly-distributed (i.e. True Null or TN) features.

Usage

```
data(sim_FS)
```

Format

An object of class `SummarizedExperiment` from `SummarizedExperiment` package containing an assay of 1000 rows (features) and 100 columns (samples). Each row is represented by a vector of binary values (1/0) indicating the presence/absence of the feature in the samples.

See `?SummarizedExperiment` for more details.

Value

a SummarizedExperiment object

References

Kartha VK, Kern JG, Sebastiani P, Zhang L, Varelas X, Monti S (2019) CaDrA: A computational framework for performing candidate driver analyses using binary genomic features. (Frontiers in Genetics)

`sim_Scores` *Simulated Input Scores*

Description

A simulated vector of continuous scores generated from `rnorm(n=ncol(sim_FS), mean=0, sd=1)` with `set.seed(123)` based on the number of samples in the simulated dataset (sim_FS)

Usage

```
data(sim_Scores)
```

Format

A vector of continuous scores randomly generated from `rnorm(n=ncol(sim_FS), mean=0, sd=1)` with `set.seed(123)` based on the number of samples in the simulated dataset (sim_FS)

Value

a vector of continuous scores

References

Kartha VK, Kern JG, Sebastiani P, Zhang L, Varelas X, Monti S (2019) CaDrA: A computational framework for performing candidate driver analyses using binary genomic features. (Frontiers in Genetics)

TAZYAP_BRCA_ACTIVITY *YAP/TAZ Activity in TCGA BRCA dataset*

Description

A vector of continuous scores represents oncogenic YAP/TAZ activity in human breast carcinomas

Usage

```
data(TAZYAP_BRCA_ACTIVITY)
```

Format

consists of a vector of continuous scores of YAP/TAZ activity across 951 profiles. The mutation and copy number associated with this sample cohorts can be found in `BRCA_GISTIC_MUT_SIG` dataset.

Value

a vector of continuous scores

References

Kartha VK, Kern JG, Sebastiani P, Zhang L, Varelas X, Monti S (2019) CaDrA: A computational framework for performing candidate driver analyses using binary genomic features. (Frontiers in Genetics)

topn_best *Top 'N' Best Meta-Features*

Description

Take the resulting list of meta-features returned from `candidate_search` over top N feature searches and fetch the meta-feature with the best score.

Usage

```
topn_best(topn_list)
```

Arguments

`topn_list`	A nested list of objects that are returned from `candidate_search` using the following parameters: `FS = sim_FS`, `input_score = sim_Scores`, `top_N = 7`, `method = "ks_pval"`, `alternative = "less"`, `search_method = "both"`, `max_size = 10`, and `best_score_only = FALSE`.

Value

A list of objects containing the best meta-feature matrix, its corresponding best score, its observed input scores, rank of best meta-features based on their scores, its marginal and cumulative best scores.

Examples

```
# Load pre-computed Top-N list generated for sim_FS dataset
data(topn_list)

# Get the best meta-features list
topn_best_meta <- topn_best(topn_list = topn_list)
```

---

topn_list *Top-N Results for Simulated Data (sim_FS)*

Description

A list of objects returned from `candidate_search` using simulated dataset `FS = sim_FS`, `input_score = sim_Scores`, `top_N = 7`, `method = "ks_pval"`, `alternative = "less"`, `search_method = "both"`, `max_size = 10`, and `best_score_only = FALSE`.

Usage

```
data(topn_list)
```

Format

A list of objects returned from `candidate_search` including a set of best meta-feature matrix, its corresponding best score, its observed input scores, rank of the best features based on their scores, marginal best scores, and cumulative best scores. pertaining to each top N feature searches.

See `candidate_search` for more information.

Details

NOTE: `max_size` is set to 10 as we would like to account for the presence of 10 left-skewed (i.e. true positive or TP) features in `sim_FS` dataset.

Value

a list of objects returned from `candidate_search` function

References

Kartha VK, Kern JG, Sebastiani P, Zhang L, Varelas X, Monti S (2019) CaDrA: A computational framework for performing candidate driver analyses using binary genomic features. (Frontiers in Genetics)

Examples

```
# Load pre-computed Top-N list generated for sim_FS and sim_Scores dataset
data(topn_list)

# Fetch the first meta-feature
topn_list[[1]]$feature_set

# Fetch the second meta-feature
topn_list[[2]]$feature_set

# Retrieve the meta-feature with the best score among top_N = 7 runs
topn_best_meta <- topn_best(topn_list = topn_list)

# Visualize the best meta-feature using meta_plot function
meta_plot(topn_best_list = topn_best_meta)

# Visualize overlap of meta-features across top_N = 7
# using topn_plot function
topn_plot(topn_list = topn_list)
```

---

topn_plot *Top 'N' Plot*

Description

Generate a heatmap representation of overlapping meta-features across top N feature searches using `candidate_search` function

Usage

```
topn_plot(topn_list)
```

Arguments

`topn_list`	a list of objects obtained from `candidate_search` using the following parameters: `FS = sim_FS`, `input_score = sim_Scores`, `top_N = 7`, `method = "ks_pval"`, `alternative = "less"`, `search_method = "both"`, `max_size = 10`, and `best_score_only = FALSE`.

Value

a heatmap of overlapping meta-features across top N feature searches

Examples

```
# Load pre-computed Top-N list generated for sim_FS dataset
data(topn_list)

# Get the overlapping top N plot
topn_plot(topn_list = topn_list)
```

Index