Package: lncRna (via r-universe)

June 5, 2026

Title A Comprehensive Workflow for Long Non-coding RNA Identification and Functional Analysis

Version 1.1.0

Description Provides a complete workflow for the identification, analysis, and functional annotation of long non-coding RNAs (lncRNAs) from RNA-Seq data. The package includes functions for filtering transcripts from GTF files, evaluating the performance of multiple coding potential prediction tools (e.g., CPC2, PLEK, CPAT), and summarizing their agreement. It enables systematic performance analysis of individual tools, ``at least N'' tool consensus, and all possible tool combinations. Functional analysis is supported through the identification of potential cis- and trans-acting interactions with protein-coding genes, followed by enrichment analysis. Results can be visualized using a variety of plots, including radar plots, clock plots, and interactive Sankey diagrams.

License MIT + file LICENSE

URL https://github.com/prodakt/lncRna

BugReports https://github.com/prodakt/lncRna/issues

biocViews Software, GeneExpression, RNASeq, Transcription, Visualization, QualityControl, FunctionalGenomics, Classification, FunctionalPrediction

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Imports fmsb, ggplot2, grDevices, graphics, Hmisc, patchwork, plotly, Polychrome, tidyr, S4Vectors, scales, stats, stringr, GenomicRanges, utils

Suggests IRanges, methods, BiocStyle, gprofiler2, knitr, rmarkdown, rtracklayer, seqinr, testthat (>= 3.0.0), venn

Config/testthat/edition 3

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev

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

Date/Publication 2026-04-28 13:07:14 UTC

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

RemoteRef HEAD

RemoteSha ae94230cbb791cb2e13b18e9e8555f7a054f1ebc

Contents

`aggregateCodPot` *Aggregate Coding Potential Tool Results*

Description

Reads output files from various coding potential prediction tools and aggregates their results into a structured list. Non-coding predictions are encoded as 1, and coding predictions as 0.

Usage

```
aggregateCodPot(
  CPC2_outfile = NULL,
  PLEK_outfile = NULL,
  FEELnc_outfile = NULL,
  CPAT_outfile = NULL,
  CPAT_cutoff = 0.364,
```

```
  CNCI_outfile = NULL,
  LncFinder_outfile = NULL,
  lncRNA_Mdeep_outfile = NULL
)
```

Arguments

`CPC2_outfile`	Path to the CPC2 output file.
`PLEK_outfile`	Path to the PLEK output file.
`FEELnc_outfile`	Path to the FEELnc output file.
`CPAT_outfile`	Path to the CPAT output file.
`CPAT_cutoff`	Cutoff value for CPAT probability (default: 0.364).
`CNCI_outfile`	Path to the CNCI output file.
`LncFinder_outfile`	Path to the LncFinder output file.
`lncRNA_Mdeep_outfile`	Path to the lncRNA_Mdeep output file.

Value

A list containing `seqIDs` (a character vector of all unique sequence IDs found across all provided files) and `tools` (a list where each element is a binary vector of predictions for a given tool).

Examples

```
# --- 1. Create temporary output files for a reproducible example ---
cpc2File <- tempfile()
plekFile <- tempfile()

# CPC2 format: ID ... classification
write.table(
  data.frame(V1="ID1", V8="noncoding"),
  cpc2File, col.names=FALSE, row.names=FALSE, sep="\t"
)
write.table(
  data.frame(V1="ID2", V8="coding"),
  cpc2File, col.names=FALSE, row.names=FALSE, sep="\t", append=TRUE
)

# PLEK format: Classification Score >ID
write.table(
  data.frame(V1="Non-coding", V2="0.9", V3=">ID2"),
  plekFile, col.names=FALSE, row.names=FALSE, sep=" "
)

# --- 2. Run the function with the temporary files ---
codpotResults <- aggregateCodPot(CPC2_outfile = cpc2File, PLEK_outfile = plekFile)

print("Sequence IDs:")
```

```
print(codpotResults$seqIDs)
print("Tool predictions:")
print(codpotResults$tools)

# --- 3. Clean up the temporary files ---
unlink(c(cpc2File, plekFile))
```

`annotateInteractions` *Annotate Genomic Interactions with Functional Enrichment Results*

Description

Merges a table of genomic interactions (e.g., lncRNA-mRNA) with functional enrichment results (e.g., from g:Profiler), annotating each interaction with relevant functional terms.

Usage

```
annotateInteractions(
  gostResult,
  interactionTable,
  type,
  lncRnaCol = "lncRNAId",
  targetCol = "targetRNAId"
)
```

Arguments

`gostResult`	A `data.frame` from a g:Profiler analysis (`gost` function), containing at least "term_name" and "intersection" columns.
`interactionTable`	A `data.frame` of interactions. Must contain columns for lncRNA and target identifiers.
`type`	A character string specifying the interaction type to be added to the output (e.g., "cis", "trans", "LncTar").
`lncRnaCol`	An optional character string specifying the column name in `interactionTable` for lncRNA IDs. Defaults to `lncRNAId`.
`targetCol`	An optional character string specifying the column name in `interactionTable` for target RNA IDs. Defaults to `targetRNAId`.

Value

A `data.frame` where enrichment terms are merged with the interactions they are associated with.

Examples

```
# --- 1. Create mock input data ---
# a) Mock g:Profiler result
mockGostResult <- data.frame(
  term_name = c("response to stress", "cell activation"),
  intersection = c("TARGET_G1,TARGET_G2", "TARGET_G3"),
  source = c("GO:BP", "GO:BP"),
  stringsAsFactors = FALSE
)

# b) Mock interaction table with standard column names
mockInteractionTable1 <- data.frame(
  lncRNAId = c("LNC_G1", "LNC_G2", "LNC_G3"),
  targetRNAId = c("TARGET_G1", "TARGET_G2", "TARGET_G3")
)

# --- 2. Run with standard column names ---
annotated_trans <- annotateInteractions(
  gostResult = mockGostResult,
  interactionTable = mockInteractionTable1,
  type = "trans"
)
print(annotated_trans)

# --- 3. Run with custom column names ---
# a) Mock interaction table with custom column names
mockInteractionTable2 <- data.frame(
  Query = "LNC_G1",
  Target = "TARGET_G1"
)

annotated_lnctar <- annotateInteractions(
  gostResult = mockGostResult,
  interactionTable = mockInteractionTable2,
  type = "LncTar",
  lncRnaCol = "Query",
  targetCol = "Target"
)
print(annotated_lnctar)
```

`bestTool` *Evaluate Performance of Coding Potential Tools*

Description

Computes a comprehensive set of confusion matrix statistics (e.g., accuracy, sensitivity, specificity) for individual coding potential prediction tools. The function operates on summary data prepared by `summarizeTestSet`.

Usage

```
bestTool(summaryList, tools = NULL, digits = 4)
```

Arguments

summaryList	A list generated by summarizeTestSet(), containing $tools (a list of 0/1 prediction vectors) and $isNC (a numeric vector of true labels: 1=nc, 0=cds).
tools	An optional character vector specifying the names of the tools (from names(summaryList$tools)) to analyze. If NULL, the behavior depends on the session type: in an interactive session, the user will be prompted to select tools; in a non-interactive session, all available tools will be analyzed by default.
digits	The number of decimal places to round the final statistics to (default: 4).

Value

A data frame where rows correspond to performance metrics and columns correspond to the selected tools. Returns NULL if the input is invalid or no statistics can be calculated.

Examples

```
# --- 1. Create a mock object mimicking the output of prepareEvaluationSets ---
# This object contains filtered data ready for evaluation.
set.seed(123)
n_seq <- 100
exampleEvaluationSummary <- list(
  tools = list(
    CPC2 = sample(c(0, 1), n_seq, replace = TRUE),
    CPAT = sample(c(0, 1), n_seq, replace = TRUE),
    PLEK = sample(c(0, 1), n_seq, replace = TRUE)
  ),
  isNC = sample(c(0, 1), n_seq, replace = TRUE) # True labels
)

# --- 2. Run bestTool ---
# Example 1: Analyze a specific subset of tools
performance_stats <- bestTool(
  summaryList = exampleEvaluationSummary,
  tools = c("CPC2", "CPAT")
)
print(performance_stats)

# Example 2: Analyze all available tools (non-interactively)
all_tools_stats <- bestTool(summaryList = exampleEvaluationSummary)
print(all_tools_stats)

# --- 3. Interactive example (do not run in scripts) ---
if (interactive()) {
  # The following would prompt you to select tools from the console:
  # interactive_stats <- bestTool(summaryList = exampleSummaryList)
}
```

`bestToolAtleast` *Evaluate Performance of Agreement Thresholds*

Description

Computes confusion matrix statistics for each "at least n" agreement threshold generated by `evaluateToolsThresholds`.

Usage

```
bestToolAtleast(agreementSummary, digits = 4)
```

Arguments

`agreementSummary`	A list generated by `evaluateToolsThresholds()`, containing `$isNC` (true labels) and `$atLeastN` (a list of 0/1 prediction vectors named `atl1`, `atl2`, etc.).
`digits`	The number of decimal places to round the final statistics to (default: 4).

Value

A data frame where rows are performance metrics and columns correspond to the "at least n" (`atl1`, `atl2`, ...) thresholds. Returns `NULL` if the input is invalid or no statistics can be calculated.

Examples

```
# --- 1. Create a mock object mimicking the output of prepareEvaluationSets ---
# This object serves as input for evaluating thresholds.
set.seed(789)
n_seq <- 100
# First, create the summary object (as if from prepareEvaluationSets)
evaluationSummary <- list(
  seqIDs = paste0("S", 1:n_seq),
  isNC = sample(c(0, 1), n_seq, replace = TRUE),
  type = sample(c("nc", "cds"), n_seq, replace = TRUE),
  tools = list(
    ToolA = sample(c(0, 1), n_seq, replace = TRUE),
    ToolB = sample(c(0, 1), n_seq, replace = TRUE),
    ToolC = sample(c(0, 1), n_seq, replace = TRUE)
  )
)

# --- 2. Create the agreement summary using evaluateToolsThresholds ---
# This is the object that bestToolAtleast actually takes as input
agreementSummary <- evaluateToolsThresholds(summaryList = evaluationSummary)

# --- 3. Run bestToolAtleast on the result ---
if (!is.null(agreementSummary)) {
  performance_thresholds <- bestToolAtleast(
    agreementSummary = agreementSummary
```

```
  )
  print(performance_thresholds)
}
```

`bestToolCombination` *Evaluate Performance of Tool Combinations*

Description

Computes confusion matrix statistics for predictions generated by various tool combinations, using the output from `evaluateToolCombinations`.

Usage

```
bestToolCombination(combinationSummaryList, combinations = NULL, digits = 4)
```

Arguments

`combinationSummaryList`	A list generated by `evaluateToolCombinations()`, containing `$isNC` (true labels) and `$toolCombinations` (a list of prediction vectors for each combination).
`combinations`	An optional character vector specifying the names of the tool combinations (e.g., "ToolA+ToolB") to analyze. If `NULL`, behavior depends on the session: interactive mode prompts for selection, while non-interactive mode analyzes all available combinations.
`digits`	The number of decimal places to round final statistics to (default: 4).

Value

A data frame where rows are performance metrics and columns correspond to the analyzed tool combinations. Returns `NULL` on error or if no statistics can be calculated.

Examples

```
# --- 1. Create a mock object mimicking evaluateToolCombinations output ---
set.seed(202)
n_seq <- 80
exampleCombinationSummary <- list(
  isNC = sample(c(0, 1), n_seq, replace = TRUE),
  toolCombinations = list(
    `ToolA+ToolB` = sample(c(0, 1), n_seq, replace = TRUE, prob = c(0.7, 0.3)),
    `ToolA+ToolC` = sample(c(0, 1), n_seq, replace = TRUE, prob = c(0.6, 0.4)),
    `ToolB+ToolC` = sample(c(0, 1), n_seq, replace = TRUE, prob = c(0.5, 0.5))
  )
  # Other elements like seqIDs, type would also be present
)
```

```
# --- 2. Run the function ---
# Example 1: Analyze specific combinations
perf_specific <- bestToolCombination(
  combinationSummaryList = exampleCombinationSummary,
  combinations = c("ToolA+ToolB", "ToolB+ToolC")
)
print("Performance for specific combinations:")
print(perf_specific)

# Example 2: Analyze all combinations (non-interactively)
perf_all <- bestToolCombination(
  combinationSummaryList = exampleCombinationSummary
)
print("Performance for all combinations:")
print(perf_all)
```

`calculateCM` *Calculate and Filter Confusion Matrices for Tool Combinations*

Description

This function calculates confusion matrices for various tool combinations based on predictions from `evaluateToolCombinations`. It can also filter these matrices based on pre-calculated performance metrics and a given threshold.

Usage

```
calculateCM(
  combinationSummaryList,
  metricsData = NULL,
  printMetricThresholds = FALSE,
  threshold = 0.8,
  returnOnlyHigh = FALSE,
  metricToExtract = "Accuracy"
)
```

Arguments

`combinationSummaryList`
: A list generated by `evaluateToolCombinations()`, containing `$toolCombinations` (a list of 0/1 prediction vectors) and `$isNC` (the true class labels).

`metricsData`
: An optional data frame where rows are metric names and columns are tool combination names (typically the output of `bestToolCombination()`). If provided, `metricToExtract` is looked up here for threshold-based filtering and printing. Defaults to `NULL`.

`printMetricThresholds`
: Logical. If `TRUE`, prints the value of `metricToExtract` for each combination, indicating if it meets the `threshold`. Defaults to `FALSE`.

`threshold`
: A numeric value (0 to 1) used to evaluate `metricToExtract`. Defaults to 0.8.

`returnOnlyHigh`
: Logical. If `TRUE`, returns confusion matrices only for combinations meeting the `threshold`. Defaults to `FALSE`.

`metricToExtract`
: The performance metric to use for filtering and printing (e.g., "Accuracy", "Sensitivity"). Defaults to "Accuracy".

Value

A named list where each element represents a confusion matrix for a tool combination. Each element is itself a list containing:

`table`
: A 2x2 numeric matrix representing the confusion table.

`positive`
: A character string indicating the positive class level ("1").

`metrics`
: A named numeric vector of all calculated performance metrics.

The returned list is filtered if `returnOnlyHigh = TRUE`.

Examples

```
# --- 1. Create mock data mimicking the outputs of previous functions ---
set.seed(202)
n_seq <- 100
# a) Mock output from evaluateToolCombinations()
mockCombinationSummary <- list(
  isNC = sample(c(0, 1), n_seq, replace = TRUE),
  toolCombinations = list(
    `ToolA+ToolB` = sample(c(0,1), n_seq, replace=TRUE, prob=c(0.2,0.8)),
    `ToolA+ToolC` = sample(c(0,1), n_seq, replace=TRUE, prob=c(0.6,0.4)),
    `ToolB+ToolC` = sample(c(0,1), n_seq, replace=TRUE, prob=c(0.5,0.5))
  )
)
# b) Mock output from bestToolCombination()
mockMetricsData <- bestToolCombination(
  combinationSummaryList = mockCombinationSummary
)

# --- 2. Run calculateCM in different modes ---
# Example 1: Calculate all CMs without filtering
all_cms <- calculateCM(combinationSummaryList = mockCombinationSummary)
print(names(all_cms))
# Inspect the structure of a single element
str(all_cms[[1]])

# Example 2: Print and filter based on a Sensitivity threshold >= 0.5
filtered_cms <- calculateCM(
  combinationSummaryList = mockCombinationSummary,
  metricsData = mockMetricsData,
```

```
  printMetricThresholds = TRUE,
  returnOnlyHigh = TRUE,
  metricToExtract = "Sensitivity",
  threshold = 0.5
)
print("Filtered CMs (Sensitivity >= 0.5):")
print(names(filtered_cms))
```

`createTrainTestSets` *Split a Set of Sequence Names into Training and Test Sets*

Description

This function partitions a vector of sequence names into training and testing subsets based on a specified percentage. It uses random sampling, so for reproducible splits, set a seed with `set.seed()` before calling it. This single function replaces the previous, separate `test.train.cds` and `test.train.nc` functions.

Usage

```
createTrainTestSets(sequences, percentTrain = 0.6, prefix = "set")
```

Arguments

`sequences`	A character vector of sequence names, or an object with a `names` attribute (like the list returned by `seqinr::read.fasta`).
`percentTrain`	A numeric value between 0 and 1 indicating the proportion of sequences to allocate to the training set (default: 0.6).
`prefix`	A character string used as a prefix for the names of the elements in the returned list (default: "set"). For example, `prefix = "cds"` will result in list elements named "cds.train" and "cds.test".

Value

A list containing two character vectors. The names of the list elements are constructed using the `prefix` argument (e.g., `cds.train` and `cds.test`).

Examples

```
# --- Example 1: Splitting CDS sequences (replaces test.train.cds) ---
all_cds_names <- paste0("cds_seq_", 1:200)
set.seed(123) # for a reproducible split
cds_split <- createTrainTestSets(
  sequences = all_cds_names,
  percentTrain = 0.7,
  prefix = "cds"
)
```

```
names(cds_split)
length(cds_split$cds.train)
length(cds_split$cds.test)

# --- Example 2: Splitting non-coding sequences (replaces test.train.nc) ---
# Input can also be a list with names
nc_fasta_like <- as.list(paste0("nc_seq_", 1:100))
names(nc_fasta_like) <- paste0("nc_seq_", 1:100)
set.seed(456)
nc_split <- createTrainTestSets(
  sequences = nc_fasta_like,
  percentTrain = 0.5,
  prefix = "nc"
)
names(nc_split)
length(nc_split$nc.train)
length(nc_split$nc.test)
```

`evaluateToolCombinations` *Evaluate Performance of Tool Combinations*

Description

Identifies sequences predicted as non-coding by specific combinations (intersections) of tools. It generates prediction vectors for all combinations of 2 or more selected tools.

Usage

```
evaluateToolCombinations(summaryList, tools = NULL)
```

Arguments

`summaryList`	A list generated by `prepareEvaluationSets()`, containing `$seqIDs`, `$tools` (a list of 0/1 prediction vectors), `$isNC`, and `$type`.
`tools`	An optional character vector specifying which tools to use for creating combinations. If `NULL`, the behavior depends on the session: interactive mode will prompt for selection, while non-interactive mode will use all available tools.

Value

A list containing:

`seqIDs`	Original sequence identifiers from the input.
`isNC`	Original true labels (1=nc, 0=cds) from the input.
`type`	Original type annotation ('nc', 'cds') from the input.

`selectedToolsPredictions`
: A list of prediction vectors for the tools selected for this analysis.

`toolCombinations`
: A list where each element is named after a tool combination (e.g., "ToolA+ToolB") and contains a binary vector (0/1) indicating if a sequence was predicted as non-coding by ALL tools in that combination.

Examples

```
# --- 1. Create a mock object mimicking the output of prepareEvaluationSets ---
set.seed(101)
n_seq <- 50
evaluationSummary <- list(
  seqIDs = paste0("Seq", 1:n_seq),
  tools = list(
    ToolX = sample(c(0, 1), n_seq, replace = TRUE),
    ToolY = sample(c(0, 1), n_seq, replace = TRUE),
    ToolZ = sample(c(0, 1), n_seq, replace = TRUE)
  ),
  type = sample(c("nc", "cds"), n_seq, replace = TRUE),
  isNC = sample(c(0, 1), n_seq, replace = TRUE)
)

# --- 2. Run the function ---
# Example 1: Analyze combinations of specific tools
results_comb <- evaluateToolCombinations(
  summaryList = evaluationSummary,
  tools = c("ToolX", "ToolY", "ToolZ")
)

if (!is.null(results_comb)) {
  print("Names of generated combinations:")
  print(names(results_comb$toolCombinations))

  print("Head of 'ToolX+ToolY' combination results:")
  print(head(results_comb$toolCombinations[["ToolX+ToolY"]]))
}

# Example 2: Non-interactively analyze all tools
results_all_comb <- evaluateToolCombinations(summaryList = evaluationSummary)
```

---

evaluateToolsThresholds

Evaluate Agreement Thresholds Among Coding Potential Tools

Description

Calculates agreement statistics for a selection of coding potential tools. For each sequence, it determines if it was predicted as non-coding by at least 'n' of the selected tools, for all possible values of 'n'.

Usage

```
evaluateToolsThresholds(summaryList, tools = NULL)
```

Arguments

summaryList	A list generated by summarizeTestSet(), containing $seqIDs, $tools, $isNC, and $type.
tools	An optional character vector specifying which tools to include in the agreement analysis. If NULL, the behavior depends on the session: interactive mode will prompt for selection, while non-interactive mode will use all available tools.

Value

A list containing:

seqIDs	Original sequence identifiers from the input.
isNC	Original true labels (1=nc, 0=cds) from the input.
type	Original type annotation ('nc', 'cds') from the input.
selectedToolsPredictions	A list containing only the prediction vectors for the tools that were selected for analysis.
sumsSelectedTools	An integer vector with the count of selected tools that predicted "non-coding" (1) for each sequence.
atLeastN	A list where each element atl{n} is a binary vector indicating if a sequence was predicted as non-coding by at least n selected tools. This list contains all threshold sets.

Examples

```
# --- 1. Create Example Data (mimicking summarizeTestSet output) ---
set.seed(456)
n_seq <- 50
exampleSummaryList <- list(
  seqIDs = paste0("ID", 1:n_seq),
  tools = list(
    ToolA = sample(c(0, 1), n_seq, replace = TRUE),
    ToolB = sample(c(0, 1), n_seq, replace = TRUE),
    ToolC = sample(c(0, 1), n_seq, replace = TRUE)
  ),
  type = sample(c("nc", "cds"), n_seq, replace = TRUE),
  isNC = sample(c(0, 1), n_seq, replace = TRUE)
)

# --- 2. Run the function ---
results <- evaluateToolsThresholds(
  summaryList = exampleSummaryList,
  tools = c("ToolA", "ToolB")
)
```

```
if (!is.null(results)) {
  # Accessing the nested list of thresholds
  print("Accessing the 'at-least-2' threshold vector:")
  print(head(results$atLeastN$atl2))
}
```

`findCisInteractions` *Find Potential cis-Regulatory Interactions from FEELnc Output*

Description

This function reads and filters the output file from FEELnc (`.classes` file) to identify potential cis-interactions between lncRNAs and mRNAs based on user-specified criteria such as genomic distance and specific gene/transcript lists.

Usage

```
findCisInteractions(
  FEELncClassesFile,
  lncRnas = NULL,
  mRnas = NULL,
  filterIsBest = TRUE,
  lncRnaLevel = c("transcript", "gene"),
  mRnaLevel = c("gene", "transcript"),
  maxDist = 1e+05
)
```

Arguments

`FEELncClassesFile`	A character string specifying the path to the FEELnc `.classes` output file. This file is required.
`lncRnas`	An optional character vector of lncRNA IDs (gene or transcript) to filter the results. If `NULL`, no filtering by lncRNA ID is performed.
`mRnas`	An optional character vector of mRNA IDs (gene or transcript) to filter the results. If `NULL`, no filtering by mRNA ID is performed.
`filterIsBest`	Logical. If `TRUE` (default), only interactions marked as "best" (`isBest == 1`) in the FEELnc output are kept.
`lncRnaLevel`	A character string specifying the level for lncRNA filtering: `"transcript"` (default) or `"gene"`.
`mRnaLevel`	A character string specifying the level for mRNA filtering: `"gene"` (default) or `"transcript"`.
`maxDist`	A numeric value for the maximum distance (in base pairs) to consider for a cis-interaction (default: 100,000).

Value

A `data.frame` containing the filtered cis-interaction data with renamed columns (`lncRNAId`, `targetRNAId`).

Examples

```
# --- 1. Create a temporary FEELnc output file for a reproducible example ---
feelncFile <- tempfile()
mock_data <- data.frame(
  isBest = c(1, 1, 0, 1, 1),
  lncRNA_gene = c("LNC_G1", "LNC_G1", "LNC_G2", "LNC_G3", "LNC_G4"),
  lncRNA_transcript = c("LNC_T1", "LNC_T2", "LNC_T3", "LNC_T4", "LNC_T5"),
  partnerRNA_gene = c("TARGET_G1", "TARGET_G2", "TARGET_G1", "TARGET_G3", "TARGET_G4"),
  partnerRNA_transcript = c("T_T1", "T_T2", "T_T3", "T_T4", "T_T5"),
  distance = c(5000, 80000, 1000, 120000, 9000)
)
utils::write.table(mock_data, feelncFile, sep = "\t", row.names = FALSE, col.names = TRUE)

# --- 2. Define lncRNA and mRNA lists for filtering ---
lncRnaList <- c("LNC_T1", "LNC_T4") # Filter by transcript ID
mRnaList <- c("TARGET_G1")       # Filter by gene ID

# --- 3. Run the function ---
cis_interactions <- findCisInteractions(
  FEELncClassesFile = feelncFile,
  lncRnas = lncRnaList,
  mRnas = mRnaList,
  lncRnaLevel = "transcript",
  mRnaLevel = "gene",
  maxDist = 100000
)

print(cis_interactions)

# --- 4. Clean up the temporary file ---
unlink(feelncFile)
```

---

findTransInteractions *Estimates trans-interactions between lncRNAs and target RNAs based on expression correlation. This function requires the* `Hmisc` *and* `tidyr` *packages.*

Description

Estimates trans-interactions between lncRNAs and target RNAs based on expression correlation. This function requires the `Hmisc` and `tidyr` packages.

Usage

```
findTransInteractions(
  exprMatrix,
  corMethod = "pearson",
  rval = 0.7,
  pval = 0.05,
  lncRnaList = NULL,
  tarRnaList = NULL,
  fullCorMatrixFile = NULL
)
```

Arguments

exprMatrix	A numeric matrix or data.frame of expression values. Rownames should contain gene/transcript IDs and columns should be samples.
corMethod	Correlation method: "pearson" (default) or "spearman".
rval	The cutoff for the correlation coefficient (default: 0.7).
pval	The cutoff for the p-value (default: 0.05).
lncRnaList	A list of lncRNA gene/transcript IDs. Must be present in rownames(exprMatrix). If NULL, all rownames are considered.
tarRnaList	A list of target gene/transcript IDs. Must be present in rownames(exprMatrix). If NULL, all rownames are considered.
fullCorMatrixFile	An optional file path to save the full correlation matrix.

Value

A data.frame of significant trans-interactions with columns for lncRNA ID, target RNA ID, r-value, and p-value.

Examples

```
# --- 1. Create a mock expression matrix ---
set.seed(123)
lnc_genes <- paste0("LNC", 1:5)
target_genes <- paste0("TARGET", 1:20)
all_genes <- c(lnc_genes, target_genes)
mockExprMatrix <- matrix(rnorm(25 * 10), nrow = 25, ncol = 10,
                         dimnames = list(all_genes, paste0("Sample", 1:10)))
mockExprMatrix["LNC1", ] <- mockExprMatrix["TARGET1", ] * 2 + rnorm(10, 0, 0.2)

# --- 2. Run the function ---
trans_interactions <- findTransInteractions(
  exprMatrix = mockExprMatrix,
  lncRnaList = lnc_genes,
  tarRnaList = target_genes,
  rval = 0.9,
  pval = 0.05
```

```
)
print(trans_interactions)
```

`getBiotypes` *Extract Gene or Transcript Biotypes from a GRanges Object*

Description

This function extracts unique biotypes for genes or transcripts from a `GRanges` object, typically imported from a reference GTF/GFF file.

Usage

```
getBiotypes(refGtf, level = "transcript")
```

Arguments

`refGtf`	A `GRanges` object imported from a reference annotation file. It must contain metadata columns for biotypes (e.g., `gene_biotype`, `transcript_biotype`) and corresponding IDs (`gene_id`, `transcript_id`).
`level`	A character string specifying whether to extract information for `"gene"` or `"transcript"` features (default: `"transcript"`).

Value

A `DataFrame` (from the `S4Vectors` package) with two columns: the identifier (`gene_id` or `transcript_id`) and the corresponding biotype (`gene_biotype` or `transcript_biotype`).

Examples

```
# --- 1. Create a sample GRanges object mimicking a reference GTF ---
sampleRefGtf <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges = IRanges::IRanges(start = 1:6, width = 100),
  gene_id = c("G1", "G1", "G2", "G2", "G3", "G3"),
  gene_biotype = c("protein_coding", "protein_coding", "lncRNA",
                   "lncRNA", "pseudogene", "pseudogene"),
  transcript_id = c("T1.1", "T1.2", "T2.1", "T2.1", "T3.1", "T3.1"),
  transcript_biotype = c("protein_coding", "protein_coding_variant",
                         "lncRNA", "lncRNA", "pseudogene", "pseudogene")
)

# --- 2. Extract biotypes at the transcript level ---
transcript_biotypes <- getBiotypes(refGtf = sampleRefGtf, level = "transcript")
print(transcript_biotypes)

# --- 3. Extract biotypes at the gene level ---
gene_biotypes <- getBiotypes(refGtf = sampleRefGtf, level = "gene")
```

```
print(gene_biotypes)
```

---

getGtfStats *Extract Transcript Statistics from a GTF object*

Description

This function takes a GRanges object (imported from a GTF file) and calculates the number of exons and total transcript length for each transcript.

Usage

```
getGtfStats(gtfObject)
```

Arguments

gtfObject	A GRanges object, typically imported from a GTF file using `rtracklayer::import()`.

Value

A `data.frame` with columns: "transcript_id", "exons", and "trans_length".

Examples

```
# Create a sample GRanges object to mimic a GTF import
sample_gtf <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges = IRanges::IRanges(
    start = c(100, 300, 800, 950),
    end = c(200, 400, 900, 1050)
  ),
  strand = "+",
  type = "exon",
  transcript_id = c("T1", "T1", "T2", "T2"),
  exon_number = c("1", "2", "1", "20") # Example with exon_number > 9
)

# Calculate statistics
transcript_stats <- getGtfStats(sample_gtf)
print(transcript_stats)
```

plotClockMetrics *Create Clock Plots for Confusion Matrix Metrics*

Description

Generates circular bar plots (clock plots) to visualize performance metrics. Supports both single and multiple plot layouts for comparing methods.

Usage

```
plotClockMetrics(
  cmList,
  methods = NULL,
  metrics = c("Accuracy", "Sensitivity", "Specificity", "Precision", "Recall"),
  plotTitle = "Clock Plot of Metrics",
  colors = NULL,
  layout = c("single", "multiple"),
  ...
)
```

Arguments

cmList	A named list where each element represents a confusion matrix, typically the output from calculateCM.
methods	An optional character vector of method names to include. If NULL, interactive selection is triggered in an interactive session; otherwise, all methods are used.
metrics	A character vector of metrics to display. Defaults to c("Accuracy", "Sensitivity", "Specificity", "Precision", "Recall").
plotTitle	The title for the plot(s).
colors	An optional vector of colors for each method.
layout	The layout of the plots: "single" (default) for one plot, or "multiple" for a grid of plots.
...	Additional arguments (not currently used).

Value

A ggplot object (for layout = "single") or a patchwork object (for layout = "multiple"), which can be printed to display.

Examples

```
# --- 1. Create a mock cmList object (as from calculateCM) ---
set.seed(123)
mockCmList <- list(
  `MethodA` = list(metrics = c(Accuracy=0.9, Sensitivity=0.8, Specificity=0.95,
                               Precision=0.85, Recall=0.8)),
```

```
  `MethodB` = list(metrics = c(Accuracy=0.8, Sensitivity=0.9, Specificity=0.7,
                               Precision=0.75, Recall=0.9))
)

# --- 2. Run the plot function ---
# Example 1: Single plot comparing selected methods
plotClockMetrics(
  cmList = mockCmList,
  methods = c("MethodA", "MethodB"),
  plotTitle = "Comparison Clock Plot"
)

# Example 2: Multiple plots, one for each method
plotClockMetrics(cmList = mockCmList, layout = "multiple")
```

`plotRadarMetrics` *Create Radar Plots for Confusion Matrix Metrics*

Description

Generates radar plots to visualize and compare performance metrics for different methods or tool combinations. Supports a single plot for comparing multiple methods or a grid of plots for individual evaluation.

Usage

```
plotRadarMetrics(
  cmList,
  methods = NULL,
  metrics = c("Accuracy", "Sensitivity", "Specificity", "Precision", "Recall"),
  plotTitle = "Radar Plot of Metrics",
  colors = NULL,
  layout = c("single", "multiple"),
  displayArea = FALSE,
  displayFill = TRUE,
  saveData = FALSE,
  fileName = NULL,
  ...
)
```

Arguments

`cmList`	A named list where each element represents a confusion matrix, typically the output from `calculateCM`. Each element must be a list containing at least a named numeric vector called `$metrics`.
`methods`	An optional character vector of method names (from `names(cmList)`) to include in the plot. If `NULL`, behavior depends on the session: interactive mode prompts for selection, while non-interactive mode uses all available methods.

metrics	A character vector of metrics to visualize on the radar plot axes. Defaults to c("Accuracy", "Sensitivity", "Specificity", "Precision", "Recall").
plotTitle	A character string for the plot title.
colors	An optional character vector of colors for each method. If NULL, default colors are generated.
layout	Specifies the plot layout: "single" (default) for one plot comparing all methods, or "multiple" for a grid of individual plots.
displayArea	Logical. If TRUE, displays the calculated polygon area.
displayFill	Logical. If TRUE (default), fills radar chart polygons with a semi-transparent color.
saveData	Logical. If TRUE, saves the underlying data frame to a CSV file.
fileName	The name of the file for saving data (required if saveData is TRUE).
...	Additional arguments passed to fmsb::radarchart.

Value

This function is called for its side effect of generating plots and does not return a value (invisible(NULL)).

Examples

```
# --- 1. Create a mock cmList object (as from calculateCM) ---
set.seed(123)
mockCmList <- list(
  `MethodA` = list(metrics = c(Accuracy=0.9, Sensitivity=0.8, Specificity=0.95,
                               Precision=0.85, Recall=0.8)),
  `MethodB` = list(metrics = c(Accuracy=0.8, Sensitivity=0.9, Specificity=0.7,
                               Precision=0.75, Recall=0.9)),
  `MethodC` = list(metrics = c(Accuracy=0.85, Sensitivity=0.85, Specificity=0.85,
                               Precision=0.85, Recall=0.85))
)

# --- 2. Run the plot function ---
# To prevent plots from showing up during automated checks, we wrap in a device
temp_png <- tempfile(fileext = ".png")
png(temp_png)

# Example 1: Single plot comparing selected methods
plotRadarMetrics(
  cmList = mockCmList,
  methods = c("MethodA", "MethodC"),
  plotTitle = "Comparison Plot"
)

# Example 2: Multiple plots, one for each method
plotRadarMetrics(cmList = mockCmList, layout = "multiple")

dev.off()
unlink(temp_png)
```

plotSankeyInteractions *Plot Functional Interactions as a Sankey Diagram*

Description

Visualizes genomic interaction data (lncRNAId -> Target -> Functional Term) as an interactive Sankey diagram using Plotly. Replaces `plot_by_terms`, `plot_by_lnc`, etc., providing a unified interface for filtering and plotting.

Usage

```
plotSankeyInteractions(
  interactionData,
  groupBy = NULL,
  selectIds = NULL,
  showLabels = FALSE,
  highlightSelected = FALSE,
  color = NULL,
  title = NULL
)
```

Arguments

`interactionData`	A `data.frame` containing interaction data, typically created by `annotateInteractions()`. Must contain columns: `"lncRNAId"`, `"intersection"`, `"term_name"`, and `"type"`.
`groupBy`	A character string specifying the filtering criteria. Valid options are: `"lncRNAId"`, `"target"`, `"term"`, `"type"`. If `NULL`, interactive selection is triggered.
`selectIds`	Optional character vector of IDs/terms to filter or highlight. If `NULL` and interactive, prompts for selection.
`showLabels`	Logical. If `TRUE`, displays labels on nodes. Defaults to `FALSE`.
`highlightSelected`	Logical. If `TRUE`, plots all data but colors only the nodes/links associated with `selectIds`. If `FALSE` (default), plots only the data matching `selectIds`.
`color`	Optional character string. A single color for all nodes/links. If `NULL`, a random palette (Polychrome) is generated.
`title`	Optional character string for the plot title.

Value

A `plotly` object containing the Sankey diagram.

Examples

```
# --- 1. Create mock interaction data ---
mockData <- data.frame(
  lncRNAId = c(rep("LNC1", 3), rep("LNC2", 2)),
  intersection = c("T1", "T2", "T1", "T3", "T2"),
  term_name = c("Stress", "Growth", "Stress", "Immunity", "Growth"),
  type = c("cis", "cis", "trans", "trans", "cis"),
  stringsAsFactors = FALSE
)

# --- 2. Run in non-interactive mode ---

# Example 1: Filter by specific Term
fig1 <- plotSankeyInteractions(
  interactionData = mockData,
  groupBy = "term",
  selectIds = "Stress",
  title = "Stress Interactions"
)
# fig1

# Example 2: Highlight specific lncRNAId
fig2 <- plotSankeyInteractions(
  interactionData = mockData,
  groupBy = "lncRNAId",
  selectIds = "LNC1",
  highlightSelected = TRUE,
  title = "Highlighting LNC1"
)
# fig2
```

`plotVennCodPot` *Create Venn Diagram from Coding Potential Results*

Description

Generates a Venn diagram of noncoding predictions from multiple tools, based on the output of `aggregateCodPot`. This function requires the 'venn' package, which should be declared in the `Suggests` field of the DESCRIPTION file.

Usage

```
plotVennCodPot(
  codPot,
  selection = NULL,
  vennColors = grDevices::palette.colors(n = 9, palette = "Okabe-Ito")
)
```

Arguments

codPot	A list object generated by aggregateCodPot().
selection	An optional logical vector indicating which tools to include (TRUE for include, FALSE for exclude). If NULL, all available tools are used.
vennColors	A vector of colors for the Venn diagram segments.

Value

Invisibly returns the result of venn::venn. The primary effect is plotting the diagram to the active graphics device.

Examples

```
  # Example with all tools

  mockCodPot <- list(
  seqIDs = c("tx1", "tx2", "tx3", "tx4"),
  tools = list(
    CPC2 = c(0, 1, 1, 0),
    PLEK = c(0, 1, 0, 1)
  )
)
if (requireNamespace("venn", quietly = TRUE)) {
  plotVennCodPot(codPot = mockCodPot)
}
```

---

prepareEvaluationSets *Prepare Data Sets for Performance Evaluation*

Description

Filters coding potential results to include only sequences present in provided test sets. It annotates each sequence as non-coding (nc) or protein-coding (cds), creating a summary object ready for various evaluation functions (e.g., bestTool, bestToolAtleast).

Usage

```
prepareEvaluationSets(codPotList, ncTest, cdsTest)
```

Arguments

codPotList	A list object, typically from aggregateCodPot(). Must contain $seqIDs (character vector) and $tools (a list of numeric vectors).
ncTest	A character vector of sequence IDs known to be non-coding.
cdsTest	A character vector of sequence IDs known to be protein-coding.

Value

A list containing elements filtered to include only sequences from `ncTest` or `cdsTest`:

`seqIDs`	Filtered character vector of sequence IDs.
`tools`	Filtered list of tool prediction vectors.
`type`	Character vector with annotation ("nc" or "cds").
`isNC`	Integer vector: 1 if type is "nc", 0 if "cds".
`sums`	Integer vector with the sum of predictions across all tools.

Examples

```
# --- 1. Create mock data mimicking package outputs ---
# a) Output from aggregateCodPot()
mockCodPotList <- list(
  seqIDs = c("nc_seq1", "cds_seq1", "nc_seq2", "other_seq"),
  tools = list(
    CPC2 = c(1, 0, 1, 1),
    PLEK = c(1, 1, 0, 0)
  )
)

# b) Outputs from createTrainTestSets()
mockNcSets <- list(
  nc.train = c("nc_seq3", "nc_seq4"),
  nc.test = c("nc_seq1", "nc_seq2")
)
mockCdsSets <- list(
  cds.train = c("cds_seq2"),
  cds.test = c("cds_seq1")
)

# --- 2. Run the function to prepare the evaluation set ---
evaluationSummary <- prepareEvaluationSets(
  codPotList = mockCodPotList,
  ncTest = mockNcSets$nc.test,
  cdsTest = mockCdsSets$cds.test
)

# --- 3. Inspect the prepared data ---
# Note: "other_seq" was filtered out as it was not in the test sets.
print(evaluationSummary)
```

Index