Package 'transite'

April 15, 2020

Title RNA-binding protein motif analysis

Version 1.4.0

Maintainer Konstantin Krismer <krismer@mit.edu>

Description transite is a computational method that allows comprehensive analysis of the regulatory role of RNA-binding proteins in various cellular processes by leveraging preexisting gene expression data and current knowledge of binding preferences of RNA-binding proteins.

License MIT + file LICENSE

URL https://transite.mit.edu

Depends R (>= 3.5)

Imports BiocGenerics (>= 0.26.0), Biostrings (>= 2.48.0), dplyr (>= 0.7.6), GenomicRanges (>= 1.32.6), ggplot2 (>= 3.0.0), ggseqlogo (>= 0.1), gridExtra (>= 2.3), methods, parallel, Rcpp (>= 0.12.18), scales (>= 1.0.0), stats, TFMPvalue (>= 0.0.8), utils

Suggests knitr (>= 1.20), rmarkdown (>= 1.10), roxygen2 (>= 6.1.0)

LinkingTo Rcpp (>= 0.12.18)

VignetteBuilder knitr

biocViews GeneExpression, Transcription, DifferentialExpression, Microarray, mRNAMicroarray, Genetics, GeneSetEnrichment

Encoding UTF-8

RoxygenNote 6.1.1

SystemRequirements C++11

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

git_branch RELEASE_3_10

git_last_commit c8ddca0

git_last_commit_date 2019-11-04

Date/Publication 2020-04-14

Author Konstantin Krismer [aut, cre, cph] (<https://orcid.org/0000-0001-8994-3416>), Anna Gattinger [aut] (<https://orcid.org/0000-0001-7094-9279>), Michael Yaffe [ths, cph] (<https://orcid.org/0000-0002-9547-3251>), Ian Cannell [ths] (<https://orcid.org/0000-0001-5832-9210>)

R topics documented:

`calculateKmerEnrichment` k-*mer Enrichment between Foreground and Background Sets*

Description

Calls `computeKmerEnrichment` to compute *k*-mer enrichment values for multiple foregrounds. Calculates enrichment for foreground sets in parallel.

Usage

```
calculateKmerEnrichment(foreground.sets, background.set, k,
  permutation = FALSE, chisq.p.value.threshold = 0.05,
  p.adjust.method = "BH", n.cores = 4)
```

Arguments

`foreground.sets`	list of foreground sets; a foreground set is a character vector of DNA or RNA sequences (not both) and a strict subset of the `background.set`
`background.set`	character vector of DNA or RNA sequences that constitute the background set
`k`	length of *k*-mer, either `6` for hexamers or `7` for heptamers
`permutation`	if `TRUE`, only the enrichment value is returned (efficiency mode used for permutation testing)
`chisq.p.value.threshold`	threshold below which Fisher's exact test is used instead of Pearson's chi-squared test
`p.adjust.method`	see `p.adjust`
`n.cores`	number of computing cores to use

Value

A list with two entries:

(1) dfs: a list of data frames with results from `computeKmerEnrichment` for each of the foreground sets (2) kmers: a character vector of all k-mers

See Also

Other *k*-mer functions: `checkKmers`, `computeKmerEnrichment`, `drawVolcanoPlot`, `empiricalEnrichmentMeanCDF`, `generateKmers`, `generatePermutedEnrichments`, `homopolymerCorrection`, `permTestGeometricMean`, `runKmerSPMA`, `runKmerTSMA`

Examples

```
# define simple sequence sets for foreground and background
foreground.set1 <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
```

```
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA"
)
foreground.set2 <- c("UUAUUUA", "AUCCUUUACA", "UUUUUUU", "UUUCAUCAUU")
foreground.sets <- list(foreground.set1, foreground.set2)
background.set <- c(foreground.set1, foreground.set2,
                    "CCACACAC", "CUCAUUGGAG", "ACUUUGGGACA", "CAGGUCAGCA")

# single-threaded
kmer.enrichment.values.st <- calculateKmerEnrichment(foreground.sets,
  background.set, 6, n.cores = 1)
## Not run:
# multi-threaded
kmer.enrichment.values.mt <- calculateKmerEnrichment(foreground.sets,
  background.set, 6)
## End(Not run)
```

`calculateKmerScores` k-*mer Score Calculation*

Description

C++ implementation of *k*-mer score calculation

Usage

```
calculateKmerScores(kmers, pwm)
```

Arguments

`kmers`	list of *k*-mers
`pwm`	position weight matrix

Value

list of PWM scores for the specified *k*-mers

Examples

```
motif <- getMotifById("M178_0.6")[[1]]
kmers <- c("AAAAAA", "CAAAAA", "GAAAAA")
calculateKmerScores(kmers, as.matrix(motifMatrix(motif)))
```

`calculateLocalConsistency` *Local Consistency Score*

Description

C++ implementation of Local Consistency Score algorithm.

Usage

```
calculateLocalConsistency(x, numPermutations, minPermutations, e)
```

Arguments

`x`	numeric vector that contains values for shuffling
`numPermutations`	maximum number of permutations performed in Monte Carlo test for consistency score
`minPermutations`	minimum number of permutations performed in Monte Carlo test for consistency score
`e`	stop criterion for consistency score Monte Carlo test: aborting permutation process after observing `e` random consistency values with more extreme values than the actual consistency value

Value

list with `score`, `p.value`, and `n` components, where `score` is the raw local consistency score (usually not used), `p.value` is the associated p-value for that score, obtained by Monte Carlo testing, and `n` is the number of permutations performed in the Monte Carlo test (the higher, the more significant)

Examples

```
poor.enrichment.spectrum <- c(0.1, 0.5, 0.6, 0.4,
  0.7, 0.6, 1.2, 1.1, 1.8, 1.6)
local.consistency <- calculateLocalConsistency(poor.enrichment.spectrum,
  1000000, 1000, 5)

enrichment.spectrum <- c(0.1, 0.3, 0.6, 0.7, 0.8,
  0.9, 1.2, 1.4, 1.6, 1.4)
local.consistency <- calculateLocalConsistency(enrichment.spectrum,
  1000000, 1000, 5)
```

`calculateMotifEnrichment` *Binding Site Enrichment Value Calculation*

Description

This function is used to calculate binding site enrichment / depletion scores between predefined foreground and background sequence sets. Significance levels of enrichment values are obtained by Monte Carlo tests.

Usage

```
calculateMotifEnrichment(foreground.scores.df, background.scores.df,
  background.total.sites, background.absolute.hits,
  n.transcripts.foreground, max.fg.permutations = 1e+06,
  min.fg.permutations = 1000, e = 5, p.adjust.method = "BH")
```

Arguments

`foreground.scores.df`	result of `scoreTranscripts` on foreground sequence set (foreground sequence sets must be a subset of the background sequence set)
`background.scores.df`	result of `scoreTranscripts` on background sequence set
`background.total.sites`	number of potential binding sites per sequence (returned by `scoreTranscripts`)
`background.absolute.hits`	number of putative binding sites per sequence (returned by `scoreTranscripts`)
`n.transcripts.foreground`	number of sequences in the foreground set
`max.fg.permutations`	maximum number of foreground permutations performed in Monte Carlo test for enrichment score
`min.fg.permutations`	minimum number of foreground permutations performed in Monte Carlo test for enrichment score
`e`	integer-valued stop criterion for enrichment score Monte Carlo test: aborting permutation process after observing e random enrichment values with more extreme values than the actual enrichment value
`p.adjust.method`	adjustment of p-values from Monte Carlo tests to avoid alpha error accumulation, see `p.adjust`

Value

A data frame with the following columns:

`motif.id`	the motif identifier that is used in the original motif library
`motif.rbps`	the gene symbol of the RNA-binding protein(s)
`enrichment`	binding site enrichment between foreground and background sequences

p.value	unadjusted p-value from Monte Carlo test
p.value.n	number of Monte Carlo test permutations
adj.p.value	adjusted p-value from Monte Carlo test (usually FDR)

See Also

Other matrix functions: runMatrixSPMA, runMatrixTSMA, scoreTranscriptsSingleMotif, scoreTranscripts

Examples

```
foreground.seqs <- c("CAGUCAAGACUCC", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AGAU", "CCAGUAA")
background.seqs <- c(foreground.seqs, "CAACAGCCUUAAUU", "CUUUGGGGAAU",
                     "UCAUUUUAUUAAA", "AUCAAAUUA", "GACACUUAAAGAUCCU",
                     "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
                     "AUAGAC", "AGUUC")
foreground.scores <- scoreTranscripts(foreground.seqs, cache = FALSE)
background.scores <- scoreTranscripts(background.seqs, cache = FALSE)
enrichments.df <- calculateMotifEnrichment(foreground.scores$df,
  background.scores$df,
  background.scores$total.sites, background.scores$absolute.hits,
  length(foreground.seqs),
  max.fg.permutations = 1000
)
```

---

calculateTranscriptMC *Motif Enrichment calculation*

Description

C++ implementation of Motif Enrichment calculation

Usage

```
calculateTranscriptMC(absoluteHits, totalSites, relHitsForeground, n,
  maxPermutations, minPermutations, e)
```

Arguments

absoluteHits	number of putative binding sites per sequence (returned by scoreTranscripts)
totalSites	number of potential binding sites per sequence (returned by scoreTranscripts)
relHitsForeground	relative number of hits in foreground set
n	number of sequences in the foreground set
maxPermutations	maximum number of foreground permutations performed in Monte Carlo test for enrichment score
minPermutations	minimum number of foreground permutations performed in Monte Carlo test for enrichment score
e	stop criterion for enrichment score Monte Carlo test: aborting permutation process after observing e random enrichment values with more extreme values than the actual enrichment value

Value

list with p-value and number of iterations of Monte Carlo sampling for foreground enrichment

Examples

```
foreground.seqs <- c("CAGUCAAGACUCC", "AAUUGGUUGUGGGGCUUCCCUGUACAU",
                     "AGAU", "CCAGUAA", "UGUGGGG")
background.seqs <- c(foreground.seqs, "CAACAGCCUUAAUU", "CUUUGGGGAAU",
                     "UCAUUUUAUUAAA", "AUCAAAUUA", "GACACUUAAAGAUCCU",
                     "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
                     "AUAGAC", "AGUUC")
motif.db <- getMotifById("M178_0.6")
fg <- scoreTranscripts(foreground.seqs, cache = FALSE,
  motifs = motif.db)
bg <- scoreTranscripts(background.seqs, cache = FALSE,
  motifs = motif.db)

mc.result <- calculateTranscriptMC(unlist(bg$absolute.hits),
 unlist(bg$total.sites),
 fg$df$absolute.hits / fg$df$total.sites,
 length(foreground.seqs), 1000, 500, 5)
```

---

checkKmers *Check Validity of Set of* k*-mers*

Description

Checks if the provided set of *k*-mers is valid. A valid set of *k*-mers is (1) non-empty, (2) contains either only hexamers or only heptamers, and (3) contains only characters from the RNA alphabet (A, C, G, U)

Usage

```
checkKmers(kmers)
```

Arguments

kmers	set of *k*-mers

Value

TRUE if set of *k*-mers is valid

See Also

Other *k*-mer functions: calculateKmerEnrichment, computeKmerEnrichment, drawVolcanoPlot, empiricalEnrichmentMeanCDF, generateKmers, generatePermutedEnrichments, homopolymerCorrection, permTestGeometricMean, runKmerSPMA, runKmerTSMA

Examples

```
# valid set
checkKmers(c("ACGCUC", "AAACCC", "UUUACA"))

# invalid set (contains hexamers and heptamers)
checkKmers(c("ACGCUC", "AAACCC", "UUUACAA"))
```

`computeKmerEnrichment` k-*mer Enrichment between Foreground and Background Sets*

Description

Compares foreground sequence set to background sequence set and computes enrichment values for each possible *k*-mer.

Usage

```
computeKmerEnrichment(foreground.kmers, background.kmers,
  permutation = FALSE, chisq.p.value.threshold = 0.05,
  p.adjust.method = "BH")
```

Arguments

`foreground.kmers`	*k*-mer counts of the foreground set (generated by `generateKmers`)
`background.kmers`	*k*-mer counts of the background set (generated by `generateKmers`)
`permutation`	if `TRUE`, only the enrichment value is returned (efficiency mode used for permutation testing)
`chisq.p.value.threshold`	threshold below which Fisher's exact test is used instead of Pearson's chi-squared test
`p.adjust.method`	see `p.adjust`

Details

Usually uses Pearson's chi-squared test, but recalculates p-values with Fisher's exact test for Pearson's chi-squared test p-values <= `chisq.p.value.threshold`. The reason this is done is computational efficiency. Fisher's exact tests are computationally demanding and are only performed in situations, where exact p-values are preferred, e.g., if expected < 5 or significant p-values.

Value

enrichment of *k*-mers in specified foreground sequences. A data frame with the following columns is returned:

`foreground.count`	foreground counts for each *k*-mer
`background.count`	background counts for each *k*-mer
`enrichment`	*k*-mer enrichment
`p.value`	p-value of *k*-mer enrichment (either from Fisher's exact test or Pearson's chi-squared test)
`adj.p.value`	multiple testing corrected p-value

See Also

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, drawVolcanoPlot, empiricalEnrichmentMeanCDF, generateKmers, generatePermutedEnrichments, homopolymerCorrection, permTestGeometricMean, runKmerSPMA, runKmerTSMA

Examples

```
# define simple sequence sets for foreground and background
foreground.set <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA"
)
background.set <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA",
  "UUAUUUA", "AUCCUUUACA", "UUUUUUU", "UUUCAUCAUU",
  "CCACACAC", "CUCAUUGGAG", "ACUUUGGGACA", "CAGGUCAGCA"
)
foreground.kmers <- generateKmers(foreground.set, 6)
background.kmers <- generateKmers(background.set, 6)


kmer.enrichment.values <- computeKmerEnrichment(foreground.kmers,
  background.kmers)
```

`computeMotifScore` *Motif Score Algorithm*

Description

C++ implementation of motif score algorithm.

Usage

```
computeMotifScore(kmers)
```

Arguments

`kmers`	list of *k*-mers

Value

data frame with columns `score`, `top.kmer`, and `top.kmer.enrichment`

createKmerMotif *Creates Transite motif object from character vector of* k*-mers*

Description

Takes a position weight matrix (PWM) and meta info and returns an object of class RBPMotif.

Usage

```
createKmerMotif(id, rbps, kmers, type, species, src)
```

Arguments

id	motif id (character vector of length 1)
rbps	character vector of names of RNA-binding proteins associated with this motif
kmers	character vector of *k*-mers that are associated with the motif, set of *k*-mers is valid if (1) all *k*-mers must have the same length, (2) only hexamers or heptamers allowed, (3) allowed characters are A, C, G, U
type	type of motif (e.g., 'HITS-CLIP', 'EMSA', 'SELEX', etc.)
species	species where motif was discovered (e.g., 'Homo sapiens')
src	source of motif (e.g., 'RBPDB v1.3.1')

Value

object of class RBPMotif

Examples

```
custom.motif <- createKmerMotif(
  "custom.motif", "RBP1",
  c("AAAAAAA", "CAAAAAA"), "HITS-CLIP",
  "Homo sapiens", "user"
)
```

createMatrixMotif *Creates Transite motif object from position weight matrix*

Description

Takes a position weight matrix (PWM) and meta info and returns an object of class RBPMotif.

Usage

```
createMatrixMotif(id, rbps, matrix, type, species, src)
```

Arguments

id	motif id (character vector of length 1)
rbps	character vector of names of RNA-binding proteins associated with this motif
matrix	data frame with four columns (A, C, G, U) and 6 - 15 rows (positions), where cell (i, j) contains weight of nucleotide j on position i
type	type of motif (e.g., 'HITS-CLIP', 'EMSA', 'SELEX', etc.)
species	species where motif was discovered (e.g., 'Homo sapiens')
src	source of motif (e.g., 'RBPDB v1.3.1')

Value

object of class RBPMotif

Examples

```
custom.motif <- createMatrixMotif(
  "custom.motif", "RBP1",
  transite:::toy.motif.matrix, "HITS-CLIP",
  "Homo sapiens", "user"
)
```

---

drawVolcanoPlot k-*mer Enrichment Volcano Plot*

Description

Uses a volcano plot to visualize *k*-mer enrichment. X-axis is $\log_2$ enrichment value, y-axis is $\log_1 0$ significance, i.e., multiple testing corrected p-value from Fisher's exact test or Pearson's chi-squared test.

Usage

```
drawVolcanoPlot(kmers, motif.kmers, motif.rbps,
  significance.threshold = 0.01, show.legend = TRUE)
```

Arguments

kmers	data frame with the following columns: kmer, adj.p.value, enrichment
motif.kmers	set of *k*-mers that are associated with a certain motif, will be highlighted in volcano plot
motif.rbps	name of RNA-binding proteins associated with highlighted *k*-mers (character vector of length 1)
significance.threshold	p-value threshold for significance, e.g., 0.05 or 0.01
show.legend	whether or not a legend should be shown

Value

volcano plot

See Also

Other TSMA functions: runKmerTSMA, runMatrixTSMA

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, computeKmerEnrichment, empiricalEnrichmentMe generateKmers, generatePermutedEnrichments, homopolymerCorrection, permTestGeometricMean, runKmerSPMA, runKmerTSMA

Examples

```
motif <- getMotifById("951_12324455")
drawVolcanoPlot(transite:::kmers.enrichment, motifHexamers(motif[[1]]),
  motifRbps(motif[[1]]))

## Not run:
foreground.set <- c("UGUGGG", "GUGGGG", "GUGUGG", "UGUGGU")
background.set <- unique(c(foreground.set, c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA",
  "CCACACAC", "CUCAUUGGAG", "ACUUUCCCACA", "CAGGUCAGCA",
  "CCACACCAG", "CCACACAUCAGU", "CACACACUCC", "CAGCCCCCCACAGGCA"
)))

motif <- getMotifById("M178_0.6")
results <- runKmerTSMA(list(foreground.set), background.set,
                       motifs = motif)
drawVolcanoPlot(results[[1]]$motif.kmers.dfs[[1]],
    motifHexamers(motif[[1]]), "test RBP")
## End(Not run)
```

`empiricalEnrichmentMeanCDF` *Significance of Observed Mean*

Description

`empiricalEnrichmentMeanCDF` returns an estimate of the significance of the observed mean, given a vector of means based on random permutations of the data.

Usage

```
empiricalEnrichmentMeanCDF(random.means, actual.mean,
  alternative = c("two.sided", "less", "greater"), conf.level = 0.95)
```

Arguments

`random.means`	numeric vector of means based on random permutations of the data (empirical null distribution)
`actual.mean`	observed mean
`alternative`	side of the test, one of the following: `"two.sided"`, `"less"`, `"greater"`
`conf.level`	confidence level for the returned confidence interval.

Value

A list with the following components:

p.value.estimate	the estimated p-value of the observed mean
conf.int	the confidence interval around that estimate

See Also

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, computeKmerEnrichment, drawVolcanoPlot, generateKmers, generatePermutedEnrichments, homopolymerCorrection, permTestGeometricMean, runKmerSPMA, runKmerTSMA

Examples

```
test.sd <- 1.0
test.null.distribution <- rnorm(n = 10000, mean = 1.0, sd = test.sd)

empiricalEnrichmentMeanCDF(test.null.distribution, test.sd * 2, "greater")
```

---

ge *Toy Gene Expression Data Set*

Description

This object contains a toy data set based on gene expression measurements and 3'-UTR sequences of 1000 genes. It comprises three data frames with RefSeq identifiers, log fold change values, and 3'-UTR sequences of genes, which are either upregulated or downregulated after some hypothetical treatment, as well as all measured genes. The actual values are not important. This data set merely serves as an example input for various functions.

Usage

```
ge
```

Format

A list with the following components:

foreground1.df	data frame that contains down-regulated genes after treatment
foreground2.df	data frame that contains up-regulated genes after treatment
background.df	data frame that contains all genes measured

---

generateIUPACByKmers *Generates IUPAC code for a character vector of* k*-mers*

Description

Generates a compact logo of a motif based on IUPAC codes given by a character vector of *k*-mers

Usage

```
generateIUPACByKmers(kmers, code = NULL)
```

Arguments

kmers	character vector of *k*-mers
code	if IUPAC code table has already been initialized by initIUPAClookupTable, it can be specified here

Details

IUPAC RNA nucleotide code:

A	Adenine
C	Cytosine
G	Guanine
U	Uracil
R	A or G
Y	C or U
S	G or C
W	A or U
K	G or U
M	A or C
B	C or G or U
D	A or G or U
H	A or C or U
V	A or C or G
N	any base

Value

the IUPAC string of the binding site

References

http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

See Also

Other motif functions: generateIUPACByMatrix, generateKmersFromIUPAC, getMotifById, getMotifByRBP, getMotifs, getPPM, initIUPAClookupTable, motifsMetaInfo, setMotifs

Examples

```
generateIUPACByKmers(c("AACCAA", "AACCGG", "CACCGA"))
```

`generateIUPACByMatrix` *Generates IUPAC code for motif matrix*

Description

Generates a compact logo of a motif based on IUPAC codes given by a position weight matrix

Usage

```
generateIUPACByMatrix(matrix, threshold = 0.215, code = NULL)
```

Arguments

`matrix`	the position probability matrix of an RNA-binding protein
`threshold`	the threshold probability (nucleotides with lower probabilities are ignored)
`code`	if IUPAC code table has already been initialized by `initIUPAClookupTable`, it can be specified here

Details

IUPAC RNA nucleotide code:

A	Adenine
C	Cytosine
G	Guanine
U	Uracil
R	A or G
Y	C or U
S	G or C
W	A or U
K	G or U
M	A or C
B	C or G or U
D	A or G or U
H	A or C or U
V	A or C or G
N	any base

Value

the IUPAC string of the binding site

References

http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

See Also

Other motif functions: `generateIUPACByKmers`, `generateKmersFromIUPAC`, `getMotifById`, `getMotifByRBP`, `getMotifs`, `getPPM`, `initIUPAClookupTable`, `motifsMetaInfo`, `setMotifs`

Examples

```
generateIUPACByMatrix(motifMatrix(getMotifById("M178_0.6")[[1]]))
```

---

generateKmers k-*mer Counts for Sequence Set*

Description

Counts occurrences of *k*-mers of length k in the given set of sequences. Corrects for homopolymeric stretches.

Usage

```
generateKmers(sequences, k)
```

Arguments

sequences	character vector of DNA or RNA sequences
k	length of *k*-mer, either 6 for hexamers or 7 for heptamers

Value

Returns a named numeric vector, where the elements are *k*-mer counts and the names are DNA *k*-mers.

Warning

generateKmers always returns DNA *k*-mers, even if sequences contains RNA sequences. RNA sequences are internally converted to DNA sequences. It is not allowed to mix DNA and RNA sequences.

See Also

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, computeKmerEnrichment, drawVolcanoPlot, empiricalEnrichmentMeanCDF, generatePermutedEnrichments, homopolymerCorrection, permTestGeometricMe runKmerSPMA, runKmerTSMA

Examples

```
# count hexamers in set of RNA sequences
rna.sequences <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA",
  "UUAUUUA", "AUCCUUUACA", "UUUUUUU", "UUUCAUCAUU",
  "CCACACAC", "CUCAUUGGAG", "ACUUUGGGACA", "CAGGUCAGCA"
)
hexamer.counts <- generateKmers(rna.sequences, 6)
```

```
# count heptamers in set of DNA sequences
dna.sequences <- c(
  "CAACAGCCTTAATT", "CAGTCAAGACTCC", "CTTTGGGGAAT",
  "TCATTTTATTAAA", "AATTGGTGTCTGGATACTTCCCTGTACAT",
  "ATCAAATTA", "AGAT", "GACACTTAAAGATCCT",
  "TAGCATTAACTTAATG", "ATGGA", "GAAGAGTGCTCA",
  "ATAGAC", "AGTTC", "CCAGTAA",
  "TTATTTA", "ATCCTTTACA", "TTTTTTT", "TTTCATCATT",
  "CCACACAC", "CTCATTGGAG", "ACTTTGGGACA", "CAGGTCAGCA"
)
hexamer.counts <- generateKmers(dna.sequences, 7)
```

`generateKmersFromIUPAC` *Generates all* k*-mers for IUPAC string*

Description

Generates all possible *k*-mers for a given IUPAC string.

Usage

```
generateKmersFromIUPAC(iupac, k)
```

Arguments

`iupac`	IUPAC string
`k`	length of *k*-mer, `6` (hexamers) or `7` (heptamers)

Details

IUPAC RNA nucleotide code:

`A`	Adenine
`C`	Cytosine
`G`	Guanine
`U`	Uracil
`R`	A or G
`Y`	C or U
`S`	G or C
`W`	A or U
`K`	G or U
`M`	A or C
`B`	C or G or U
`D`	A or G or U
`H`	A or C or U
`V`	A or C or G
`N`	any base

Value

list of *k*-mers

References

http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, getMotifById, getMotifByRBP, getMotifs, getPPM, initIUPAClookupTable, motifsMetaInfo, setMotifs

Examples

```
generateKmersFromIUPAC(motifIUPAC(getMotifById("M178_0.6")[[1]]), k = 6)
```

---

generatePermutedEnrichments

Generate Random Permutations of the Enrichment Data

Description

Calculates *k*-mer enrichment values for randomly sampled (without replacement) foreground sets.

Usage

```
generatePermutedEnrichments(n.transcripts.foreground, background.set, k,
  n.permutations = 1000, n.cores = 4)
```

Arguments

n.transcripts.foreground	number of transcripts in the original foreground set
background.set	character vector of DNA or RNA sequences that constitute the background set
k	length of *k*-mer, either 6 for hexamers or 7 for heptamers
n.permutations	number of permutations to perform
n.cores	number of computing cores to use

Value

The result of calculateKmerEnrichment for the random foreground sets.

See Also

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, computeKmerEnrichment, drawVolcanoPlot, empiricalEnrichmentMeanCDF, generateKmers, homopolymerCorrection, permTestGeometricMean, runKmerSPMA, runKmerTSMA

geometricMean *Geometric Mean*

Description

Calculates the geometric mean of the specified values.

Usage

```
geometricMean(x, na.rm = TRUE)
```

Arguments

x	numeric vector of values for which the geometric mean will be computed
na.rm	logical. Should missing values (including NaN) be removed?

Value

Geometric mean of x or 1 if length of x is 0

Examples

```
geometricMean(c(0.123, 0.441, 0.83))
```

getMotifById *Retrieve motif objects by id*

Description

Retrieves one or more motif objects identified by motif id.

Usage

```
getMotifById(id)
```

Arguments

id	character vector of motif identifiers

Value

A list of objects of class RBPMotif

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, generateKmersFromIUPAC, getMotifByRBP, getMotifs, getPPM, initIUPAClookupTable, motifsMetaInfo, setMotifs

Examples

```
getMotifById("M178_0.6")

getMotifById(c("M178_0.6", "M188_0.6"))
```

getMotifByRBP *Retrieve motif objects by gene symbol*

Description

Retrieves one or more motif objects identified by gene symbol.

Usage

```
getMotifByRBP(rbp)
```

Arguments

rbp	character vector of gene symbols of RNA-binding proteins

Value

A list of objects of class RBPMotif

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, generateKmersFromIUPAC, getMotifById, getMotifs, getPPM, initIUPAClookupTable, motifsMetaInfo, setMotifs

Examples

```
getMotifByRBP("ELAVL1")

getMotifByRBP(c("ELAVL1", "ELAVL2"))
```

getMotifs *Retrieve list of all motifs*

Description

Retrieves all Transite motifs

Usage

```
getMotifs()
```

Value

A list of objects of class Motif

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, generateKmersFromIUPAC, getMotifById, getMotifByRBP, getPPM, initIUPAClookupTable, motifsMetaInfo, setMotifs

Examples

```
transite.motifs <- getMotifs()
```

getPPM *Get Position Probability Matrix (PPM) from motif object*

Description

Return the position probability matrix of the specified motif.

Usage

```
getPPM(motif)
```

Arguments

motif	object of class RBPMotif

Value

The position probability matrix of the specified motif

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, generateKmersFromIUPAC, getMotifById, getMotifByRBP, getMotifs, initIUPAClookupTable, motifsMetaInfo, setMotifs

Examples

```
getPPM(getMotifById("M178_0.6")[[1]])
```

homopolymerCorrection *Correction for Homopolymeric Stretches*

Description

Counts all non-overlapping instances of *k*-mers in a given set of sequences.

Usage

```
homopolymerCorrection(sequences, k, kmers, is.rna = FALSE)
```

Arguments

sequences	character vector of DNA or RNA sequences
k	length of *k*-mer, either 6 for hexamers or 7 for heptamers
kmers	column sums of return value of Biostrings::oligonucleotideFrequency(sequences)
is.rna	if sequences are RNA sequences, this flag needs to be set

Value

Returns a named numeric vector, where the elements are *k*-mer counts and the names are *k*-mers.

See Also

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, computeKmerEnrichment, drawVolcanoPlot, empiricalEnrichmentMeanCDF, generateKmers, generatePermutedEnrichments, permTestGeometricMean, runKmerSPMA, runKmerTSMA

`initIUPAClookupTable` *Initializes the IUPAC lookup table*

Description

Initializes a hash table that serves as a IUPAC lookup table for the generateIUPACByMatrix function.

Usage

```
initIUPAClookupTable()
```

Details

IUPAC RNA nucleotide code:

A	Adenine
C	Cytosine
G	Guanine
U	Uracil
R	A or G
Y	C or U
S	G or C
W	A or U
K	G or U
M	A or C
B	C or G or U
D	A or G or U
H	A or C or U
V	A or C or G
N	any base

Value

an environment, the IUPAC lookup hash table

References

http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, generateKmersFromIUPAC, getMotifById, getMotifByRBP, getMotifs, getPPM, motifsMetaInfo, setMotifs

Examples

```
generateIUPACByMatrix(motifMatrix(getMotifById("M178_0.6")[[1]]),
  code = initIUPAClookupTable())
```

`kmers.enrichment` *Example* k-*mer Enrichment Data*

Description

This data frame with *k*-mer enrichment data (as produced by `runKmerTSMA`) is used in a code example for k-mer volcano plot function `drawVolcanoPlot`.

Usage

```
kmers.enrichment
```

Format

A data frame with the following columns:

`kmer`	contains all hexamers (AAAAAA to UUUUUU)
`foreground.count`	absolute *k*-mer frequency in foreground set
`background.count`	absolute *k*-mer frequency in background set
`enrichment`	enrichment of *k*-mer in foreground relative to background
`p.value`	associated p-value of enrichment
`adj.p.value`	multiple testing corrected p-value

`lookupKmerScores` k-*mer Score Lookup Table Access Function*

Description

C++ implementation of *k*-mer score hash table lookup.

Usage

```
lookupKmerScores(kmers, kmerScores)
```

Arguments

`kmers`	list of *k*-mers
`kmerScores`	position weight matrix

Value

numeric vector of *k*-mer scores

motifs *Transite Motif Database*

Description

The Transite motif database contains sequence motifs and associated *k*-mers of more than 100 different RNA-binding proteins, obtained from publicly available motif databases.

Usage

```
motifs
```

Format

A list of lists with the following components:

id	motif id
rbps	gene symbols of RNA-binding proteins associated with motif
matrix	data frame of sequence motif (position weight matrix)
hexamers	all motif-associated hexamers
heptamers	all motif-associated heptamers
length	length of motif in nucleotides
iupac	IUPAC string of sequence motif
type	type of motif, e.g., RNAcompete
species	usually human
src	source of motif, e.g., RNA Zoo

References

http://cisbp-rna.ccbr.utoronto.ca/

http://rbpdb.ccbr.utoronto.ca/

motifsMetaInfo *Displays motif meta information.*

Description

Generates a data frame with meta information about all Transite motifs.

Usage

```
motifsMetaInfo()
```

Value

A data frame containing meta information for all Transite motifs, with the following columns:

- id
- rbps

- length
- iupac
- type
- species
- src

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, generateKmersFromIUPAC, getMotifById, getMotifByRBP, getMotifs, getPPM, initIUPAClookupTable, setMotifs

Examples

```
motifsMetaInfo()
```

---

pCombine *P-value aggregation*

Description

pCombine is used to combine the p-values of independent significance tests.

Usage

```
pCombine(p, method = c("fisher", "SL", "MG", "tippett"), w = NULL)
```

Arguments

p	vector of p-values
method	one of the following: Fisher (1932) ('fisher'), Stouffer (1949), Liptak (1958) ('SL'), Mudholkar and George (1979) ('MG'), and Tippett (1931) ('tippett')
w	weights, only used in combination with Stouffer-Liptak. If is.null(w) then weights are set in an unbiased way

Details

The problem can be specified as follows: Given a vector of n p-values $p_1, ..., p_n$, find p_c, the combined p-value of the n significance tests. Most of the methods introduced here combine the p-values in order to obtain a test statistic, which follows a known probability distribution. The general procedure can be stated as:

$$T(h, C) = \sum_{i=1}^{n} h(p_i) * C$$

The function T, which returns the test statistic t, takes two arguments. h is a function defined on the interval $[0, 1]$ that transforms the individual p-values, and C is a correction term.

Fisher's method (1932), also known as the inverse chi-square method is probably the most widely used method for combining p-values. Fisher used the fact that if p_i is uniformly distributed (which p-values are under the null hypothesis), then $-2 \log p_i$ follows a chi-square distribution with two degrees of freedom. Therefore, if p-values are transformed as follows,

$$h(p) = -2 \log p,$$

and the correction term C is neutral, i.e., equals 1, the following statement can be made about the sampling distribution of the test statistic T_f under the null hypothesis: t_f is distributed as chi-square with $2n$ degrees of freedom, where n is the number of p-values.

Stouffer's method, or the inverse normal method, uses a p-value transformation function h that leads to a test statistic that follows the standard normal distribution by transforming each p-value to its corresponding normal score. The correction term scales the sum of the normal scores by the root of the number of p-values.

$$h(p) = \Phi^{-1}(1-p)$$

$$C = \frac{1}{\sqrt{n}}$$

Under the null hypothesis, t_s is distributed as standard normal. Φ^{-1} is the inverse of the cumulative standard normal distribution function.

An extension of Stouffer's method with weighted p-values is called Liptak's method.

The logit method by Mudholkar and George uses the following transformation:

$$h(p) = -\ln(p/(1-p))$$

When the sum of the transformed p-values is corrected in the following way:

$$C = \sqrt{\frac{3(5n+4)}{\pi^2 n(5n+2)}},$$

the test statistic t_m is approximately t-distributed with $5n+4$ degrees of freedom.

In Tippett's method the smallest p-value is used as the test statistic t_t and the combined significance is calculated as follows:

$$Pr(t_t) = 1 - (1-t_t)^n$$

Value

A list with the following components:

`statistic`	the test statistic
`p.value`	the corresponding p-value
`method`	the method used
`statistic.name`	the name of the test statistic

Examples

```
pCombine(c(0.01, 0.05, 0.5))

pCombine(c(0.01, 0.05, 0.5), method = "tippett")
```

---

permTestGeometricMean *Permutation Test Based Significance of Observed Mean*

Description

`permTestGeometricMean` returns an estimate of the significance of the observed mean, given a set of random permutations of the data.

Usage

```
permTestGeometricMean(actual.mean, motif.kmers, random.permutations,
  alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
  produce.plot = TRUE)
```

Arguments

`actual.mean`	observed mean
`motif.kmers`	set of *k*-mers that were used to compute the `actual.mean`
`random.permutations`	a set of random permutations of the original data, used to generate an empirical null distribution.
`alternative`	side of the test, one of the following: `"two.sided"`, `"less"`, `"greater"`
`conf.level`	confidence level for the returned confidence interval.
`produce.plot`	if distribution plot should be part of the returned list

Value

A list with the following components:

`p.value.estimate`	the estimated p-value of the observed mean
`conf.int`	the confidence interval around that estimate
`plot`	plot of the empirical distribution of geometric means of the enrichment values

See Also

Other *k*-mer functions: `calculateKmerEnrichment`, `checkKmers`, `computeKmerEnrichment`, `drawVolcanoPlot`, `empiricalEnrichmentMeanCDF`, `generateKmers`, `generatePermutedEnrichments`, `homopolymerCorrection`, `runKmerSPMA`, `runKmerTSMA`

`RBPMotif-class` *An S4 class to represent a RBPMotif*

Description

An S4 class to represent a RBPMotif

Getter Method motifId

Getter Method motifRbps

Getter Method motifMatrix

Getter Method motifHexamers

Getter Method motifHeptamers

Getter Method motifLength

Getter Method motifIUPAC

Getter Method motifType

Getter Method motifSpecies

Getter Method motifSource

Usage

```
motifId(object)

## S4 method for signature 'RBPMotif'
motifId(object)

motifRbps(object)

## S4 method for signature 'RBPMotif'
motifRbps(object)

motifMatrix(object)

## S4 method for signature 'RBPMotif'
motifMatrix(object)

motifHexamers(object)

## S4 method for signature 'RBPMotif'
motifHexamers(object)

motifHeptamers(object)

## S4 method for signature 'RBPMotif'
motifHeptamers(object)

motifLength(object)

## S4 method for signature 'RBPMotif'
motifLength(object)

motifIUPAC(object)

## S4 method for signature 'RBPMotif'
motifIUPAC(object)

motifType(object)

## S4 method for signature 'RBPMotif'
motifType(object)

motifSpecies(object)

## S4 method for signature 'RBPMotif'
motifSpecies(object)

motifSource(object)

## S4 method for signature 'RBPMotif'
motifSource(object)

## S4 method for signature 'RBPMotif'
```

```
show(object)

## S4 method for signature 'RBPMotif,ANY'
plot(x)
```

Arguments

object	RBPMotif object
x	RBPMotif object

Value

Object of type RBPMotif

Slots

`id` motif id (character vector of length 1)

`rbps` character vector of names of RNA-binding proteins associated with this motif

`matrix` data frame with four columns (A, C, G, U) and 6 - 15 rows (positions), where cell (i, j) contains weight of nucleotide j on position i

`hexamers` character vector of hexamers associated with this motif

`heptamers` character vector of heptamers associated with this motif

`length` length of the motif (i.e., `nrow(matrix)`)

`iupac` IUPAC code for motif matrix (see `generateIUPACByMatrix`)

`type` type of motif (e.g., `'HITS-CLIP'`, `'EMSA'`, `'SELEX'`, etc.)

`species` species where motif was discovered (e.g., `'Homo sapiens'`)

`src` source of motif (e.g., `'RBPDB v1.3.1'`)

Examples

```
kmers <- c("AAAAAAA", "CAAAAAA")
iupac <- generateIUPACByKmers(kmers,
  code = initIUPAClookupTable())
hexamers <- generateKmersFromIUPAC(iupac, 6)
heptamers <- generateKmersFromIUPAC(iupac, 7)
new("RBPMotif", id = "custom.motif", rbps = "RBP1",
  matrix = NULL, hexamers = hexamers, heptamers = heptamers, length = 7L,
  iupac = iupac, type = "HITS-CLIP", species = "Homo sapiens", src = "user"
)
```

---

runKmerSPMA k-*mer-based Spectrum Motif Analysis*

Description

SPMA helps to illuminate the relationship between RBP binding evidence and the transcript sorting criterion, e.g., fold change between treatment and control samples.

Usage

```
runKmerSPMA(background.set, motifs = NULL, k = 6, n.bins = 40,
  max.model.degree = 1, max.cs.permutations = 1e+07,
  min.cs.permutations = 5000, fg.permutations = 5000,
  p.adjust.method = "BH", p.combining.method = "fisher", n.cores = 1)
```

Arguments

`background.set`	character vector of ranked sequences, either DNA (only containing upper case characters A, C, G, T) or RNA (A, C, G, U). The sequences in `background.set` must be ranked (i.e., sorted). Commonly used sorting criteria are measures of differential expression, such as fold change or signal-to-noise ratio (e.g., between treatment and control samples in gene expression profiling experiments).
`motifs`	a list of motifs that is used to score the specified sequences. If `is.null(motifs)` then all Transite motifs are used.
`k`	length of *k*-mer, either `6` for hexamers or `7` for heptamers
`n.bins`	specifies the number of bins in which the sequences will be divided, valid values are between 7 and 100
`max.model.degree`	maximum degree of polynomial
`max.cs.permutations`	maximum number of permutations performed in Monte Carlo test for consistency score
`min.cs.permutations`	minimum number of permutations performed in Monte Carlo test for consistency score
`fg.permutations`	numer of foreground permutations
`p.adjust.method`	see `p.adjust`
`p.combining.method`	one of the following: Fisher (1932) (`"fisher"`), Stouffer (1949), Liptak (1958) (`"SL"`), Mudholkar and George (1979) (`"MG"`), and Tippett (1931) (`"tippett"`) (see `pCombine`)
`n.cores`	number of computing cores to use

Details

In order to investigate how motif targets are distributed across a spectrum of transcripts (e.g., all transcripts of a platform, ordered by fold change), Spectrum Motif Analysis visualizes the gradient of RBP binding evidence across all transcripts.

The *k*-mer-based approach differs from the matrix-based approach by how the sequences are scored. Here, sequences are broken into *k*-mers, i.e., oligonucleotide sequences of *k* bases. And only statistically significantly enriched or depleted *k*-mers are then used to calculate a score for each RNA-binding protein, which quantifies its target overrepresentation.

Value

A list with the following components:

foreground.scores	the result of runKmerTSMA for the binned data
spectrum.info.df	a data frame with the SPMA results
spectrum.plots	a list of spectrum plots, as generated by scoreSpectrum
classifier.scores	a list of classifier scores, as returned by spectrumClassifier

See Also

Other SPMA functions: runMatrixSPMA, scoreSpectrum, spectrumClassifier, subdivideData

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, computeKmerEnrichment, drawVolcanoPlot, empiricalEnrichmentMeanCDF, generateKmers, generatePermutedEnrichments, homopolymerCorrection, permTestGeometricMean, runKmerTSMA

Examples

```
# example data set
background.df <- transite:::ge$background
# sort sequences by signal-to-noise ratio
background.df <- dplyr::arrange(background.df, value)
# character vector of named and ranked (by signal-to-noise ratio) sequences
background.set <- gsub("T", "U", background.df$seq)
names(background.set) <- paste0(background.df$refseq, "|",
  background.df$seq.type)

results <- runKmerSPMA(background.set,
                       motifs = getMotifById("M178_0.6"),
                       n.bins = 20,
                       fg.permutations = 10)

## Not run:
results <- runKmerSPMA(background.set)
## End(Not run)
```

---

runKmerTSMA k-*mer-based Transcript Set Motif Analysis*

Description

Calculates the enrichment of putative binding sites in foreground sets versus a background set using *k*-mers to identify putative binding sites

Usage

```
runKmerTSMA(foreground.sets, background.set, motifs = NULL, k = 6,
  fg.permutations = 5000, kmer.significance.threshold = 0.01,
  produce.plot = TRUE, p.adjust.method = "BH",
  p.combining.method = "fisher", n.cores = 1)
```

Arguments

foreground.sets	list of foreground sets; a foreground set is a character vector of DNA or RNA sequences (not both) and a strict subset of the background.set
background.set	character vector of DNA or RNA sequences that constitute the background set
motifs	a list of motifs that is used to score the specified sequences. If is.null(motifs) then all Transite motifs are used.
k	length of *k*-mer, either 6 for hexamers or 7 for heptamers
fg.permutations	numer of foreground permutations
kmer.significance.threshold	p-value threshold for significance, e.g., 0.05 or 0.01 (used for volcano plots)
produce.plot	if TRUE volcano plots and distribution plots are created
p.adjust.method	see p.adjust
p.combining.method	one of the following: Fisher (1932) ("fisher"), Stouffer (1949), Liptak (1958) ("SL"), Mudholkar and George (1979) ("MG"), and Tippett (1931) ("tippett") (see pCombine)
n.cores	number of computing cores to use

Details

Motif transcript set analysis can be used to identify RNA binding proteins, whose targets are significantly overrepresented or underrepresented in certain sets of transcripts.

The aim of Transcript Set Motif Analysis (TSMA) is to identify the overrepresentation and underrepresentation of potential RBP targets (binding sites) in a set (or sets) of sequences, i.e., the foreground set, relative to the entire population of sequences. The latter is called background set, which can be composed of all sequences of the genes of a microarray platform or all sequences of an organism or any other meaningful superset of the foreground sets.

The *k*-mer-based approach breaks the sequences of foreground and background sets into *k*-mers and calculates the enrichment on a *k*-mer level. In this case, motifs are not represented as position weight matrices, but as lists of *k*-mers.

Statistically significantly enriched or depleted *k*-mers are then used to calculate a score for each RNA-binding protein, which quantifies its target overrepresentation.

Value

A list of lists (one for each transcript set) with the following components:

enrichment.df	the result of computeKmerEnrichment
motif.df	
motif.kmers.dfs	
volcano.plots	volcano plots for each motif (see drawVolcanoPlot)
perm.test.plots	plots of the empirical distribution of *k*-mer enrichment values for each motif
enriched.kmers.combined.p.values	
depleted.kmers.combined.p.values	

See Also

Other TSMA functions: drawVolcanoPlot, runMatrixTSMA

Other *k*-mer functions: calculateKmerEnrichment, checkKmers, computeKmerEnrichment, drawVolcanoPlot, empiricalEnrichmentMeanCDF, generateKmers, generatePermutedEnrichments, homopolymerCorrection, permTestGeometricMean, runKmerSPMA

Examples

```
# define simple sequence sets for foreground and background
foreground.set1 <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA"
)
foreground.set2 <- c("UUAUUUA", "AUCCUUUACA", "UUUUUUU", "UUUCAUCAUU")
foreground.sets <- list(foreground.set1, foreground.set2)
background.set <- unique(c(foreground.set1, foreground.set2, c(
  "CCACACAC", "CUCAUUGGAG", "ACUUUGGGACA", "CAGGUCAGCA",
  "CCACACCGG", "GUCAUCAGU", "GUCAGUCC", "CAGGUCAGGGGCA"
)))

# run k-mer based TSMA with all Transite motifs (recommended):
# results <- runKmerTSMA(foreground.sets, background.set)

# run TSMA with one motif:
motif.db <- getMotifById("M178_0.6")
results <- runKmerTSMA(foreground.sets, background.set, motifs = motif.db)
## Not run:
# define example sequence sets for foreground and background
foreground.set1 <- gsub("T", "U", transite:::ge$foreground1$seq)
foreground.set2 <- gsub("T", "U", transite:::ge$foreground2$seq)
foreground.sets <- list(foreground.set1, foreground.set2)
background.set <- gsub("T", "U", transite:::ge$background$seq)

# run TSMA with all Transite motifs
results <- runKmerTSMA(foreground.sets, background.set)

# run TSMA with a subset of Transite motifs
results <- runKmerTSMA(foreground.sets, background.set,
  motifs = getMotifByRBP("ELAVL1"))

# run TSMA with user-defined motif
toy.motif <- createKmerMotif(
  "toy.motif", "example RBP",
  c("AACCGG", "AAAACG", "AACACG"), "example type", "example species", "user"
)
results <- runMatrixTSMA(foreground.sets, background.set,
  motifs = list(toy.motif))

## End(Not run)
```

runMatrixSPMA *Matrix-based Spectrum Motif Analysis*

Description

SPMA helps to illuminate the relationship between RBP binding evidence and the transcript sorting criterion, e.g., fold change between treatment and control samples.

Usage

```
runMatrixSPMA(background.set, motifs = NULL, n.bins = 40,
  max.model.degree = 1, max.cs.permutations = 1e+07,
  min.cs.permutations = 5000, max.hits = 5,
  threshold.method = "p.value", threshold.value = 0.25^6,
  max.fg.permutations = 1e+06, min.fg.permutations = 1000, e = 5,
  p.adjust.method = "BH", n.cores = 1, cache = paste0(tempdir(),
  "/sc/"))
```

Arguments

background.set	named character vector of ranked sequences (only containing upper case characters A, C, G, T), where the names are RefSeq identifiers and sequence type qualifiers ("3UTR", "5UTR" or "mRNA"), separated by "|", e.g. "NM_010356|3UTR". Names are only used to cache results. The sequences in background.set must be ranked (i.e., sorted). Commonly used sorting criteria are measures of differential expression, such as fold change or signal-to-noise ratio (e.g., between treatment and control samples in gene expression profiling experiments).
motifs	a list of motifs that is used to score the specified sequences. If is.null(motifs) then all Transite motifs are used.
n.bins	specifies the number of bins in which the sequences will be divided, valid values are between 7 and 100
max.model.degree	maximum degree of polynomial
max.cs.permutations	maximum number of permutations performed in Monte Carlo test for consistency score
min.cs.permutations	minimum number of permutations performed in Monte Carlo test for consistency score
max.hits	maximum number of putative binding sites per mRNA that are counted
threshold.method	either "p.value" (default) or "relative". If threshold.method equals "p.value", the default threshold.value is 0.25^6, which is lowest p-value that can be achieved by hexamer motifs, the shortest supported motifs. If threshold.method equals "relative", the default threshold.value is 0.9, which is 90% of the maximum PWM score.
threshold.value	semantics of the threshold.value depend on threshold.method (default is 0.25^6)

`max.fg.permutations`
: maximum number of foreground permutations performed in Monte Carlo test for enrichment score

`min.fg.permutations`
: minimum number of foreground permutations performed in Monte Carlo test for enrichment score

`e`
: integer-valued stop criterion for enrichment score Monte Carlo test: aborting permutation process after observing `e` random enrichment values with more extreme values than the actual enrichment value

`p.adjust.method`
: adjustment of p-values from Monte Carlo tests to avoid alpha error accumulation, see `p.adjust`

`n.cores`
: the number of cores that are used

`cache`
: either logical or path to a directory where scores are cached. The scores of each motif are stored in a separate file that contains a hash table with RefSeq identifiers and sequence type qualifiers as keys and the number of putative binding sites as values. If `cache` is `FALSE`, scores will not be cached.

Details

In order to investigate how motif targets are distributed across a spectrum of transcripts (e.g., all transcripts of a platform, ordered by fold change), Spectrum Motif Analysis visualizes the gradient of RBP binding evidence across all transcripts.

The matrix-based approach skips the *k*-merization step of the *k*-mer-based approach and instead scores the transcript sequence as a whole with a position specific scoring matrix.

For each sequence in foreground and background sets and each sequence motif, the scoring algorithm evaluates the score for each sequence position. Positions with a relative score greater than a certain threshold are considered hits, i.e., putative binding sites.

By scoring all sequences in foreground and background sets, a hit count for each motif and each set is obtained, which is used to calculate enrichment values and associated p-values in the same way in which motif-compatible hexamer enrichment values are calculated in the *k*-mer-based approach. P-values are adjusted with one of the available adjustment methods.

An advantage of the matrix-based approach is the possibility of detecting clusters of binding sites. This can be done by counting regions with many hits using positional hit information or by simply applying a hit count threshold per sequence, e.g., only sequences with more than some number of hits are considered. Homotypic clusters of RBP binding sites may play a similar role as clusters of transcription factors.

Value

A list with the following components:

`foreground.scores`
: the result of `scoreTranscripts` for the foreground sets (the bins)

`background.scores`
: the result of `scoreTranscripts` for the background set

`enrichment.dfs`
: a list of data frames, returned by `calculateMotifEnrichment`

`spectrum.info.df`
: a data frame with the SPMA results

`spectrum.plots`
: a list of spectrum plots, as generated by `scoreSpectrum`

`classifier.scores`
: a list of classifier scores, as returned by `spectrumClassifier`

See Also

Other SPMA functions: runKmerSPMA, scoreSpectrum, spectrumClassifier, subdivideData

Other matrix functions: calculateMotifEnrichment, runMatrixTSMA, scoreTranscriptsSingleMotif, scoreTranscripts

Examples

```
# example data set
background.df <- transite:::ge$background
# sort sequences by signal-to-noise ratio
background.df <- dplyr::arrange(background.df, value)
# character vector of named and ranked (by signal-to-noise ratio) sequences
background.set <- gsub("T", "U", background.df$seq)
names(background.set) <- paste0(background.df$refseq, "|",
  background.df$seq.type)

results <- runMatrixSPMA(background.set,
                         motifs = getMotifById("M178_0.6"),
                         n.bins = 20,
                         max.fg.permutations = 10000)

## Not run:
results <- runMatrixSPMA(background.set)
## End(Not run)
```

---

runMatrixTSMA *Matrix-based Transcript Set Motif Analysis*

Description

Calculates motif enrichment in foreground sets versus a background set using position weight matrices to identify putative binding sites

Usage

```
runMatrixTSMA(foreground.sets, background.set, motifs = NULL,
  max.hits = 5, threshold.method = "p.value",
  threshold.value = 0.25^6, max.fg.permutations = 1e+06,
  min.fg.permutations = 1000, e = 5, p.adjust.method = "BH",
  n.cores = 1, cache = paste0(tempdir(), "/sc/"))
```

Arguments

`foreground.sets` a list of named character vectors of foreground sequences (only containing upper case characters A, C, G, T), where the names are RefSeq identifiers and sequence type qualifiers ("3UTR", "5UTR", "mRNA"), e.g. "NM_010356|3UTR". Names are only used to cache results.

`background.set` a named character vector of background sequences (naming follows same rules as foreground set sequences)

`motifs` a list of motifs that is used to score the specified sequences. If `is.null(motifs)` then all Transite motifs are used.

`max.hits` maximum number of putative binding sites per mRNA that are counted

`threshold.method`
either `"p.value"` (default) or `"relative"`. If `threshold.method` equals `"p.value"`, the default `threshold.value` is `0.25^6`, which is lowest p-value that can be achieved by hexamer motifs, the shortest supported motifs. If `threshold.method` equals `"relative"`, the default `threshold.value` is `0.9`, which is 90% of the maximum PWM score.

`threshold.value`
semantics of the `threshold.value` depend on `threshold.method` (default is 0.25^6)

`max.fg.permutations`
maximum number of foreground permutations performed in Monte Carlo test for enrichment score

`min.fg.permutations`
minimum number of foreground permutations performed in Monte Carlo test for enrichment score

`e` integer-valued stop criterion for enrichment score Monte Carlo test: aborting permutation process after observing `e` random enrichment values with more extreme values than the actual enrichment value

`p.adjust.method`
adjustment of p-values from Monte Carlo tests to avoid alpha error accumulation, see `p.adjust`

`n.cores` the number of cores that are used

`cache` either logical or path to a directory where scores are cached. The scores of each motif are stored in a separate file that contains a hash table with RefSeq identifiers and sequence type qualifiers as keys and the number of putative binding sites as values. If `cache` is `FALSE`, scores will not be cached.

Details

Motif transcript set analysis can be used to identify RNA binding proteins, whose targets are significantly overrepresented or underrepresented in certain sets of transcripts.

The aim of Transcript Set Motif Analysis (TSMA) is to identify the overrepresentation and underrepresentation of potential RBP targets (binding sites) in a set (or sets) of sequences, i.e., the foreground set, relative to the entire population of sequences. The latter is called background set, which can be composed of all sequences of the genes of a microarray platform or all sequences of an organism or any other meaningful superset of the foreground sets.

The matrix-based approach skips the *k*-merization step of the *k*-mer-based approach and instead scores the transcript sequence as a whole with a position specific scoring matrix.

For each sequence in foreground and background sets and each sequence motif, the scoring algorithm evaluates the score for each sequence position. Positions with a relative score greater than a certain threshold are considered hits, i.e., putative binding sites.

By scoring all sequences in foreground and background sets, a hit count for each motif and each set is obtained, which is used to calculate enrichment values and associated p-values in the same way in which motif-compatible hexamer enrichment values are calculated in the k -mer-based approach. P-values are adjusted with one of the available adjustment methods.

An advantage of the matrix-based approach is the possibility of detecting clusters of binding sites. This can be done by counting regions with many hits using positional hit information or by simply

applying a hit count threshold per sequence, e.g., only sequences with more than some number of hits are considered. Homotypic clusters of RBP binding sites may play a similar role as clusters of transcription factors.

Value

A list with the following components:

foreground.scores	the result of scoreTranscripts for the foreground sets
background.scores	the result of scoreTranscripts for the background set
enrichment.dfs	a list of data frames, returned by calculateMotifEnrichment

See Also

Other TSMA functions: drawVolcanoPlot, runKmerTSMA

Other matrix functions: calculateMotifEnrichment, runMatrixSPMA, scoreTranscriptsSingleMotif, scoreTranscripts

Examples

```
# define simple sequence sets for foreground and background
foreground.set1 <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA"
)
names(foreground.set1) <- c(
  "NM_1_DUMMY|3UTR", "NM_2_DUMMY|3UTR", "NM_3_DUMMY|3UTR",
  "NM_4_DUMMY|3UTR", "NM_5_DUMMY|3UTR", "NM_6_DUMMY|3UTR",
  "NM_7_DUMMY|3UTR",
  "NM_8_DUMMY|3UTR", "NM_9_DUMMY|3UTR", "NM_10_DUMMY|3UTR",
  "NM_11_DUMMY|3UTR",
  "NM_12_DUMMY|3UTR", "NM_13_DUMMY|3UTR", "NM_14_DUMMY|3UTR"
)

foreground.set2 <- c("UUAUUUA", "AUCCUUUACA", "UUUUUUU", "UUUCAUCAUU")
names(foreground.set2) <- c(
  "NM_15_DUMMY|3UTR", "NM_16_DUMMY|3UTR", "NM_17_DUMMY|3UTR",
  "NM_18_DUMMY|3UTR"
)

foreground.sets <- list(foreground.set1, foreground.set2)

background.set <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA",
  "UUAUUUA", "AUCCUUUACA", "UUUUUUU", "UUUCAUCAUU",
  "CCACACAC", "CUCAUUGGAG", "ACUUUGGGACA", "CAGGUCAGCA"
)
names(background.set) <- c(
  "NM_1_DUMMY|3UTR", "NM_2_DUMMY|3UTR", "NM_3_DUMMY|3UTR",
```

```
  "NM_4_DUMMY|3UTR", "NM_5_DUMMY|3UTR", "NM_6_DUMMY|3UTR",
  "NM_7_DUMMY|3UTR",
  "NM_8_DUMMY|3UTR", "NM_9_DUMMY|3UTR", "NM_10_DUMMY|3UTR",
  "NM_11_DUMMY|3UTR",
  "NM_12_DUMMY|3UTR", "NM_13_DUMMY|3UTR", "NM_14_DUMMY|3UTR",
  "NM_15_DUMMY|3UTR",
  "NM_16_DUMMY|3UTR", "NM_17_DUMMY|3UTR", "NM_18_DUMMY|3UTR",
  "NM_19_DUMMY|3UTR",
  "NM_20_DUMMY|3UTR", "NM_21_DUMMY|3UTR", "NM_22_DUMMY|3UTR"
)

# run cached version of TSMA with all Transite motifs (recommended):
# results <- runMatrixTSMA(foreground.sets, background.set)

# run uncached version with one motif:
motif.db <- getMotifById("M178_0.6")
results <- runMatrixTSMA(foreground.sets, background.set, motifs = motif.db,
cache = FALSE)

## Not run:
# define example sequence sets for foreground and background
foreground1.df <- transite:::ge$foreground1
foreground.set1 <- gsub("T", "U", foreground1.df$seq)
names(foreground.set1) <- paste0(foreground1.df$refseq, "|",
  foreground1.df$seq.type)

foreground2.df <- transite:::ge$foreground2
foreground.set2 <- gsub("T", "U", foreground2.df$seq)
names(foreground.set2) <- paste0(foreground2.df$refseq, "|",
  foreground2.df$seq.type)

foreground.sets <- list(foreground.set1, foreground.set2)

background.df <- transite:::ge$background
background.set <- gsub("T", "U", background.df$seq)
names(background.set) <- paste0(background.df$refseq, "|",
  background.df$seq.type)

# run cached version of TSMA with all Transite motifs (recommended)
results <- runMatrixTSMA(foreground.sets, background.set)

# run uncached version of TSMA with all Transite motifs
results <- runMatrixTSMA(foreground.sets, background.set, cache = FALSE)

# run TSMA with a subset of Transite motifs
results <- runMatrixTSMA(foreground.sets, background.set,
  motifs = getMotifByRBP("ELAVL1"))

# run TSMA with user-defined motif
toy.motif <- createMatrixMotif(
  "toy.motif", "example RBP", toy.motif.matrix,
  "example type", "example species", "user"
)
results <- runMatrixTSMA(foreground.sets, background.set,
  motifs = list(toy.motif))

## End(Not run)
```

scoreSequences *Score Sequences with PWM*

Description

C++ implementation of PWM scoring algorithm

Usage

```
scoreSequences(sequences, pwm)
```

Arguments

sequences	list of sequences
pwm	position weight matrix

Value

list of PWM scores for each sequence

Examples

```
motif <- getMotifById("M178_0.6")[[1]]
sequences <- c("CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
               "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
               "AUCAAAUUA", "UGUGGGG", "GACACUUAAAGAUCCU",
               "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA", "AUAGAC",
               "AGUUC", "CCAGUAA")
seq.char.vectors <- lapply(sequences, function(seq) {
  unlist(strsplit(seq, ""))
})
scoreSequences(seq.char.vectors, as.matrix(motifMatrix(motif)))
```

scoreSpectrum *Calculates spectrum scores and creates spectrum plots*

Description

Spectrum scores are a means to evaluate if a spectrum has a meaningful (i.e., biologically relevant) or a random pattern.

Usage

```
scoreSpectrum(x, p.value = array(1, length(x)),
  x.label = "log enrichment", midpoint = 0, max.model.degree = 3,
  max.cs.permutations = 1e+07, min.cs.permutations = 5000, e = 5)
```

Arguments

x	vector of values (e.g., enrichment values, normalized RBP scores) per bin
p.value	vector of p-values (e.g., significance of enrichment values) per bin
x.label	label of values (e.g., "enrichment value")
midpoint	for enrichment values the midpoint should be 1, for log enrichment values 0)
max.model.degree	maximum degree of polynomial
max.cs.permutations	maximum number of permutations performed in Monte Carlo test for consistency score
min.cs.permutations	minimum number of permutations performed in Monte Carlo test for consistency score
e	integer-valued stop criterion for consistency score Monte Carlo test: aborting permutation process after observing e random consistency values with more extreme values than the actual consistency value

Details

One way to quantify the meaningfulness of a spectrum is to calculate the deviance between the linear interpolation of the scores of two adjoining bins and the score of the middle bin, for each position in the spectrum. The lower the score, the more consistent the trend in the spectrum plot. Formally, the local consistency score x_c is defined as

$$x_c = \frac{1}{n}\sum_{i=1}^{n-2} |\frac{s_i + s_{i+2}}{2} - s_{i+1}|.$$

In order to obtain an estimate of the significance of a particular score x_c', Monte Carlo sampling is performed by randomly permuting the coordinates of the scores vector s and recomputing x_c. The probability estimate $\hat{p}$ is given by the lower tail version of the cumulative distribution function

$$\hat{Pr}(T(x)) = \frac{\sum_{i=1}^{n} 1(T(y_i) \leq T(x)) + 1}{n+1},$$

where 1 is the indicator function, n is the sample size, i.e., the number of performed permutations, and T equals x_c in the above equation.

An alternative approach to assess the consistency of a spectrum plot is via polynomial regression. In a first step, polynomial regression models of various degrees are fitted to the data, i.e., the dependent variable s (vector of scores), and orthogonal polynomials of the independent variable b (vector of bin numbers). Secondly, the model that reflects best the true nature of the data is selected by means of the F-test. And lastly, the adjusted R^2 and the sum of squared residuals are calculated to indicate how well the model fits the data. These statistics are used as scores to rank the spectrum plots. In general, the polynomial regression equation is

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_m x_i^m + \epsilon_i,$$

where m is the degree of the polynomial (usually $m \leq 5$), and ϵ_i is the error term. The dependent variable y is the vector of scores s and x to x^m are the orthogonal polynomials of the vector of bin numbers b. Orthogonal polynomials are used in order to reduce the correlation between the different powers of b and therefore avoid multicollinearity in the model. This is important, because

correlated predictors lead to unstable coefficients, i.e., the coefficients of a polynomial regression model of degree m can be greatly different from a model of degree $m + 1$.

The orthogonal polynomials of vector b are obtained by centering (subtracting the mean), QR decomposition, and subsequent normalization. Given the dependent variable y and the orthogonal polynomials of b x to x^m, the model coefficients β are chosen in a way to minimize the deviance between the actual and the predicted values characterized by

$$M(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_m x^m$$

$$M = argmin_M(\sum_{i=1}^{n} L(y_i, M(x_i))),$$

where L(actual value, predicted value) denotes the loss function.

Ordinary least squares is used as estimation method for the model coefficients β. The loss function of ordinary least squares is the sum of squared residuals (SSR) and is defined as follows $\text{SSR}(y, \hat{y}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$, where y are the observed data and $\hat{y}$ the model predictions.

Thus the ordinary least squares estimate of the coefficients $\hat{\beta}$ (including the intercept $\hat{\beta}_0$) of the model M is defined by

$$\hat{\beta} = argmin_\beta(\sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{m} \beta_j x_i^j)^2).$$

After polynomial models of various degrees have been fitted to the data, the F-test is used to select the model that best fits the data. Since the SSR monotonically decreases with increasing model degree (model complexity), the relative decrease of the SSR between the simpler model and the more complex model must outweigh the increase in model complexity between the two models. The F-test gives the probability that a relative decrease of the SSR between the simpler and the more complex model given their respective degrees of freedom is due to chance. A low p-value indicates that the additional degrees of freedom of the more complex model lead to a better fit of the data than would be expected after a mere increase of degrees of freedom.

The F-statistic is calculated as follows

$$F = \frac{(SSR_1 - SSR_2)/(p_2 - p_1)}{SSR_2/(n - p_2)},$$

where SSR_i is the sum of squared residuals and p_i is the number of parameters of model i. The number of data points, i.e., bins, is denoted as n. F is distributed according to the F-distribution with $df_1 = p_2 - p_1$ and $df_2 = n - p_2$.

Value

A list object of class `SpectrumScore` with the following components:

`adj.r.squared`	adjusted R^2 of polynomial model
`degree`	maximum degree of polynomial
`residuals`	residuals of polynomial model
`slope`	coefficient of the linear term of the polynomial model (spectrum "direction")
`f.statistic`	statistic of the F-test
`f.statistic.p.value`	p-value of F-test
`consistency.score`	normalized sum of deviance between the linear interpolation of the scores of two adjoin
`consistency.score.p.value`	obtained by Monte Carlo sampling (randomly permuting the coordinates of the scores v
`consistency.score.n`	number of permutations
`plot`	

See Also

Other SPMA functions: runKmerSPMA, runMatrixSPMA, spectrumClassifier, subdivideData

Examples

```
# random spectrum
scoreSpectrum(runif(n = 40, min = -1, max = 1), max.model.degree = 1)

# non-random linear spectrum
signal <- seq(-1, 0.99, 2 / 40)
noise <- rnorm(n = 40, mean = 0, sd = 0.5)
scoreSpectrum(signal + noise, max.model.degree = 1,
  max.cs.permutations = 100000)

# non-random quadratic spectrum
signal <- seq(-1, 0.99, 2 / 40)^2 - 0.5
noise <- rnorm(n = 40, mean = 0, sd = 0.2)
scoreSpectrum(signal + noise, max.model.degree = 2,
  max.cs.permutations = 100000)
```

---

scoreTranscripts *Scores transcripts with position weight matrices*

Description

This function is used to count the binding sites in a set of sequences for all or a subset of RNA-binding protein sequence motifs and returns the result in a data frame, which is subsequently used by calculateMotifEnrichment to obtain binding site enrichment scores.

Usage

```
scoreTranscripts(sequences, motifs = NULL, max.hits = 5,
  threshold.method = "p.value", threshold.value = 0.25^6,
  n.cores = 1, cache = paste0(tempdir(), "/sc/"))
```

Arguments

sequences	character vector of named sequences (only containing upper case characters A, C, G, T), where the names are RefSeq identifiers and sequence type qualifiers ("3UTR", "5UTR", "mRNA"), e.g. "NM_010356|3UTR"
motifs	a list of motifs that is used to score the specified sequences. If is.null(motifs) then all Transite motifs are used.
max.hits	maximum number of putative binding sites per mRNA that are counted
threshold.method	either "p.value" (default) or "relative". If threshold.method equals "p.value", the default threshold.value is 0.25^6, which is lowest p-value that can be achieved by hexamer motifs, the shortest supported motifs. If threshold.method equals "relative", the default threshold.value is 0.9, which is 90% of the maximum PWM score.
threshold.value	semantics of the threshold.value depend on threshold.method (default is 0.25^6)

n.cores	the number of cores that are used
cache	either logical or path to a directory where scores are cached. The scores of each motif are stored in a separate file that contains a hash table with RefSeq identifiers and sequence type qualifiers as keys and the number of putative binding sites as values. If cache is FALSE, scores will not be cached.

Value

A list with three entries:

(1) df: a data frame with the following columns:

motif.id	the motif identifier that is used in the original motif library
motif.rbps	the gene symbol of the RNA-binding protein(s)
absolute.hits	the absolute frequency of putative binding sites per motif in all transcripts
relative.hits	the relative, i.e., absolute divided by total, frequency of binding sites per motif in all transcripts
total.sites	the total number of potential binding sites
one.hit, two.hits, ...	number of transcripts with one, two, three, ... putative binding sites

(2) total.sites: a numeric vector with the total number of potential binding sites per transcript

(3) absolute.hits: a numeric vector with the absolute (not relative) number of putative binding sites per transcript

See Also

Other matrix functions: calculateMotifEnrichment, runMatrixSPMA, runMatrixTSMA, scoreTranscriptsSingleMo

Examples

```
foreground.set <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU",
  "UCAUUUUAUUAAA", "AAUUGGUGUCUGGAUACUUCCCUGUACAU",
  "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA",
  "AUAGAC", "AGUUC", "CCAGUAA"
)
# names are used as keys in the hash table (cached version only)
# ideally sequence identifiers (e.g., RefSeq ids) and region labels
# (e.g., 3UTR for 3'-UTR)
names(foreground.set) <- c(
  "NM_1_DUMMY|3UTR", "NM_2_DUMMY|3UTR", "NM_3_DUMMY|3UTR",
  "NM_4_DUMMY|3UTR", "NM_5_DUMMY|3UTR", "NM_6_DUMMY|3UTR",
  "NM_7_DUMMY|3UTR", "NM_8_DUMMY|3UTR", "NM_9_DUMMY|3UTR",
  "NM_10_DUMMY|3UTR", "NM_11_DUMMY|3UTR", "NM_12_DUMMY|3UTR",
  "NM_13_DUMMY|3UTR", "NM_14_DUMMY|3UTR"
)

# specific motifs, uncached
motifs <- getMotifByRBP("ELAVL1")
scores <- scoreTranscripts(foreground.set, motifs = motifs, cache = FALSE)
## Not run:
# all Transite motifs, cached (writes scores to disk)
scores <- scoreTranscripts(foreground.set)

# all Transite motifs, uncached
```

```
scores <- scoreTranscripts(foreground.set, cache = FALSE)

foreground.df <- transite:::ge$foreground1
foreground.set <- foreground.df$seq
names(foreground.set) <- paste0(foreground.df$refseq, "|",
   foreground.df$seq.type)
scores <- scoreTranscripts(foreground.set)

## End(Not run)
```

`scoreTranscriptsSingleMotif` *Scores transadsadscripts with position weight matrices*

Description

This function is used to count the putative binding sites (i.e., motifs) in a set of sequences for the specified RNA-binding protein sequence motifs and returns the result in a data frame, which is aggregated by `scoreTranscripts` and subsequently used by `calculateMotifEnrichment` to obtain binding site enrichment scores.

Usage

```
scoreTranscriptsSingleMotif(motif, sequences, max.hits = 5,
  threshold.method = "p.value", threshold.value = 0.25^6,
  cache.path = paste0(tempdir(), "/sc/"))
```

Arguments

`motif`	a Transite motif that is used to score the specified sequences
`sequences`	character vector of named sequences (only containing upper case characters A, C, G, T), where the names are RefSeq identifiers and sequence type qualifiers (`"3UTR"`, `"5UTR"`, `"mRNA"`), e.g. `"NM_010356|3UTR"`
`max.hits`	maximum number of putative binding sites per mRNA that are counted
`threshold.method`	either `"p.value"` (default) or `"relative"`. If `threshold.method` equals `"p.value"`, the default `threshold.value` is `0.25^6`, which is lowest p-value that can be achieved by hexamer motifs, the shortest supported motifs. If `threshold.method` equals `"relative"`, the default `threshold.value` is `0.9`, which is 90% of the maximum PWM score.
`threshold.value`	semantics of the `threshold.value` depend on `threshold.method` (default is 0.25^6)
`cache.path`	the path to a directory where scores are cached. The scores of each motif are stored in a separate file that contains a hash table with RefSeq identifiers and sequence type qualifiers as keys and the number of binding sites as values. If is.null(cache.path), scores will not be cached.

Value

A list with the following items:

motif.id	the motif identifier of the specified motif
motif.rbps	the gene symbol of the RNA-binding protein(s)
absolute.hits	the absolute frequency of binding sites per motif in all transcripts
relative.hits	the relative, i.e., absolute divided by total, frequency of binding sites per motif in all transcripts
total.sites	the total number of potential binding sites
one.hit, two.hits, ...	number of transcripts with one, two, three, ... binding sites

See Also

Other matrix functions: calculateMotifEnrichment, runMatrixSPMA, runMatrixTSMA, scoreTranscripts

---

setMotifs *Set Transite motif database*

Description

Globally sets Transite motif database, use with care.

Usage

```
setMotifs(value)
```

Arguments

value	list of Motif objects

Value

void

See Also

Other motif functions: generateIUPACByKmers, generateIUPACByMatrix, generateKmersFromIUPAC, getMotifById, getMotifByRBP, getMotifs, getPPM, initIUPAClookupTable, motifsMetaInfo

Examples

```
custom.motif <- createKmerMotif(
  "custom.motif", "RBP1",
  c("AAAAAAA", "CAAAAAA"), "HITS-CLIP",
  "Homo sapiens", "user"
)
setMotifs(list(custom.motif))
```

`spectrumClassifier` *Simple spectrum classifier based on empirical thresholds*

Description

Spectra can be classified based on the aggregate spectrum classifier score. If `sum(score) == 3` spectrum considered non-random, random otherwise.

Usage

```
spectrumClassifier(adj.r.squared, degree, slope, consistency.score.n,
  n.significant, n.bins)
```

Arguments

`adj.r.squared`	adjusted R^2 of polynomial model, returned by scoreSpectrum
`degree`	degree of polynomial, returned by scoreSpectrum
`slope`	coefficient of the linear term of the polynomial model (spectrum "direction"), returned by scoreSpectrum
`consistency.score.n`	number of performed permutations before early stopping, returned by scoreSpectrum
`n.significant`	number of bins with statistically significant enrichment
`n.bins`	number of bins

Value

a three-dimensional binary vector with the following components:

coordinate 1	`adj.r.squared >= 0.4`
coordinate 2	`consistency.score.n > 1000000`
coordinate 3	`n.significant >= floor(n.bins / 10)`

See Also

Other SPMA functions: `runKmerSPMA`, `runMatrixSPMA`, `scoreSpectrum`, `subdivideData`

Examples

```
n.bins <- 40

# random spectrum
random.sp <- scoreSpectrum(runif(n = n.bins, min = -1, max = 1),
  max.model.degree = 1)
score <- spectrumClassifier(
  spectrumAdjRSquared(random.sp), spectrumDegree(random.sp),
  spectrumSlope(random.sp), spectrumConsistencyScoreN(random.sp), 0, n.bins
)
sum(score)

# non-random linear spectrum with strong noise component
```

```
signal <- seq(-1, 0.99, 2 / 40)
noise <- rnorm(n = 40, mean = 0, sd = 0.5)
linear.sp <- scoreSpectrum(signal + noise, max.model.degree = 1,
  max.cs.permutations = 100000)
score <- spectrumClassifier(
  spectrumAdjRSquared(linear.sp), spectrumDegree(linear.sp),
  spectrumSlope(linear.sp), spectrumConsistencyScoreN(linear.sp), 10, n.bins
)
sum(score)
## Not run:
# non-random linear spectrum with weak noise component
signal <- seq(-1, 0.99, 2 / 40)
noise <- rnorm(n = 40, mean = 0, sd = 0.2)
linear.sp <- scoreSpectrum(signal + noise, max.model.degree = 1,
  max.cs.permutations = 100000)
score <- spectrumClassifier(
  spectrumAdjRSquared(linear.sp), spectrumDegree(linear.sp),
  spectrumSlope(linear.sp), spectrumConsistencyScoreN(linear.sp), 10, n.bins
)
sum(score)

## End(Not run)

# non-random quadratic spectrum with strong noise component
signal <- seq(-1, 0.99, 2 / 40)^2 - 0.5
noise <- rnorm(n = 40, mean = 0, sd = 0.2)
quadratic.sp <- scoreSpectrum(signal + noise, max.model.degree = 2,
  max.cs.permutations = 100000)
score <- spectrumClassifier(
  spectrumAdjRSquared(quadratic.sp), spectrumDegree(quadratic.sp),
  spectrumSlope(quadratic.sp), spectrumConsistencyScoreN(quadratic.sp), 10, n.bins
)
sum(score)
## Not run:
# non-random quadratic spectrum with weak noise component
signal <- seq(-1, 0.99, 2 / 40)^2 - 0.5
noise <- rnorm(n = 40, mean = 0, sd = 0.1)
quadratic.sp <- scoreSpectrum(signal + noise, max.model.degree = 2)
score <- spectrumClassifier(
  spectrumAdjRSquared(quadratic.sp), spectrumDegree(quadratic.sp),
  spectrumSlope(quadratic.sp), spectrumConsistencyScoreN(quadratic.sp), 10, n.bins
)
sum(score)

## End(Not run)
```

`SpectrumScore-class` *An S4 class to represent a scored spectrum*

Description

An S4 class to represent a scored spectrum

Getter Method spectrumAdjRSquared

Getter Method spectrumDegree

Getter Method spectrumResiduals

Getter Method spectrumSlope

Getter Method spectrumFStatistic

Getter Method spectrumFStatisticPValue

Getter Method spectrumConsistencyScore

Getter Method spectrumConsistencyScorePValue

Getter Method spectrumConsistencyScoreN

Usage

```
spectrumAdjRSquared(object)

## S4 method for signature 'SpectrumScore'
spectrumAdjRSquared(object)

spectrumDegree(object)

## S4 method for signature 'SpectrumScore'
spectrumDegree(object)

spectrumResiduals(object)

## S4 method for signature 'SpectrumScore'
spectrumResiduals(object)

spectrumSlope(object)

## S4 method for signature 'SpectrumScore'
spectrumSlope(object)

spectrumFStatistic(object)

## S4 method for signature 'SpectrumScore'
spectrumFStatistic(object)

spectrumFStatisticPValue(object)

## S4 method for signature 'SpectrumScore'
spectrumFStatisticPValue(object)

spectrumConsistencyScore(object)

## S4 method for signature 'SpectrumScore'
spectrumConsistencyScore(object)

spectrumConsistencyScorePValue(object)

## S4 method for signature 'SpectrumScore'
spectrumConsistencyScorePValue(object)

spectrumConsistencyScoreN(object)
```

```
## S4 method for signature 'SpectrumScore'
spectrumConsistencyScoreN(object)

## S4 method for signature 'SpectrumScore'
show(object)

## S4 method for signature 'SpectrumScore,ANY'
plot(x)
```

Arguments

object	SpectrumScore object
x	SpectrumScore object

Value

Object of type SpectrumScore

Slots

`adj.r.squared` adjusted R^2 of polynomial model

`degree` degree of polynomial (integer between 0 and 5)

`residuals` residuals of the polynomial model

`slope` coefficient of the linear term of the polynomial model (spectrum "direction")

`f.statistic` F statistic from the F test used to determine the degree of the polynomial model

`f.statistic.p.value` p-value associated with the F statistic

`consistency.score` raw local consistency score of the spectrum

`consistency.score.p.value` p-value associated with the local consistency score

`consistency.score.n` number of permutations performed to calculate p-value of local consistency score (permutations performed before early stopping criterion reached)

`plot` spectrum plot

Examples

```
new("SpectrumScore", adj.r.squared = 0,
    degree = 0L,
    residuals = 0,
    slope = 0,
    f.statistic = 0,
    f.statistic.p.value = 1,
    consistency.score = 1,
    consistency.score.p.value = 1,
    consistency.score.n = 1000L,
    plot = NULL
)
```

subdivideData *Subdivides Sequences into* n *Bins*

Description

Preprocessing function for SPMA, divides transcript sequences into *n* bins.

Usage

```
subdivideData(background.set, n.bins = 40)
```

Arguments

background.set	character vector of named sequences (names are usually RefSeq identifiers and sequence region labels, e.g., "NM_1_DUMMY|3UTR"). It is important that the sequences are already sorted by fold change, signal-to-noise ratio or any other meaningful measure.
n.bins	specifies the number of bins in which the sequences will be divided, valid values are between 7 and 100

Value

An array of n.bins length, containing the binned sequences

See Also

Other SPMA functions: runKmerSPMA, runMatrixSPMA, scoreSpectrum, spectrumClassifier

Examples

```
# toy example
toy.background.set <- c(
  "CAACAGCCUUAAUU", "CAGUCAAGACUCC", "CUUUGGGGAAU", "UCAUUUUAUUAAA",
  "AAUUGGUGUCUGGAUACUUCCCUGUACAU", "AUCAAAUUA", "AGAU", "GACACUUAAAGAUCCU",
  "UAGCAUUAACUUAAUG", "AUGGA", "GAAGAGUGCUCA", "AUAGAC", "AGUUC", "CCAGUAA"
)
# names are used as keys in the hash table (cached version only)
# ideally sequence identifiers (e.g., RefSeq ids) and
# sequence region labels (e.g., 3UTR for 3'-UTR)
names(toy.background.set) <- c(
  "NM_1_DUMMY|3UTR", "NM_2_DUMMY|3UTR", "NM_3_DUMMY|3UTR",
  "NM_4_DUMMY|3UTR", "NM_5_DUMMY|3UTR", "NM_6_DUMMY|3UTR",
  "NM_7_DUMMY|3UTR",
  "NM_8_DUMMY|3UTR", "NM_9_DUMMY|3UTR", "NM_10_DUMMY|3UTR",
  "NM_11_DUMMY|3UTR",
  "NM_12_DUMMY|3UTR", "NM_13_DUMMY|3UTR", "NM_14_DUMMY|3UTR"
)

foreground.sets <- subdivideData(toy.background.set, n.bins = 7)

# example data set
background.df <- transite:::ge$background
# sort sequences by signal-to-noise ratio
```

```
background.df <- dplyr::arrange(background.df, value)
# character vector of named sequences
background.set <- background.df$seq
names(background.set) <- paste0(background.df$refseq, "|",
  background.df$seq.type)

foreground.sets <- subdivideData(background.set)
```

`toy.motif.matrix` *Toy Motif Matrix*

Description

This toy motif matrix is used in code examples for various functions.

Usage

```
toy.motif.matrix
```

Format

A data frame with four columns (A, C, G, U) and seven rows (position 1 - 7)

`transite` *transite*

Description

transite is a computational method that allows comprehensive analysis of the regulatory role of RNA-binding proteins in various cellular processes by leveraging preexisting gene expression data and current knowledge of binding preferences of

Author(s)

Konstantin Krismer

Index