Package ‘rDNAse’

July 14, 2016

Type Package
Version 1.1-1
Date 2016-6-24
Title Generating Various Numerical Representation Schemes of DNA Sequences
Description Comprehensive toolkit for generating various numerical representation schemes of DNA sequence. The descriptors and similarity scores included are extensively used in bioinformatics and chemogenomics.
Author Min-feng Zhu <wind2zhu@163.com>, Jie Dong <biomed@csu.edu.cn>, Dong-sheng Cao <oriental-cds@163.com>
Maintainer Min-feng Zhu <wind2zhu@163.com>
License GPL (>= 2)
URL https://github.com/wind2zhu/rDNAse
BugReports https://github.com/wind2zhu/rDNAse/issues
LazyData yes
Suggests Biostrings, GOSemSim, foreach, doParallel, RCurl, org.Hs.eg.db
RoxygenNote 5.0.1
NeedsCompilation no
Repository CRAN
Date/Publication 2016-07-14 07:18:52

R topics documented:
dnacheck ................................................. 2
extrDAC .................................................. 3
extrDACC ............................................... 4
extrDCC .................................................. 5
extrPseDNC ............................................. 6
extrPseKNC ............................................. 8
dnacheck

Check if the DNA sequence are in the 4 default types

Description
Check if the DNA sequence are in the 4 default types

Usage
dnacheck(x)

Arguments
x A character vector, as the input DNA sequence.

Details
This function checks if the DNA sequence types are in the 4.

Value
Logical. TRUE if all of the DNA types of the sequence are within the 4 default types.
The result character vector

Author(s)
Min-feng Zhu <<wind2zhu@163.com>>

Examples
x = 'GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'
dnacheck(x) # TRUE
dnacheck(paste(x, 'Z', sep = '')) # FALSE
**The Dinucleotide-based Auto Covariance Descriptor**

**Description**

The Dinucleotide-based Auto Covariance Descriptor

**Usage**

```r
extrDAC(x, index = c("Twist", "Tilt"), nlag = 2, normalization = FALSE,
       customprops = NULL, allprop = FALSE)
```

**Arguments**

- `x`: the input data, which should be a list or file type.
- `index`: the physicochemical indices, it should be a list and there are 38 different physicochemical indices (Table 1), which the users can choose.
- `nlag`: an integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest DNA sequence in the dataset). It represents the distance between two dinucleotides.
- `normalization`: with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.
- `customprops`: the users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.
- `allprop`: all the 38 physicochemical indices will be employed to generate the feature vector. Its default value is False.

**Details**

This function calculates the dinucleotide-based auto covariance descriptor

**Value**

A vector

**Note**

if the user defined physicochemical indices have not been normalized, it should be normalized.

**Author(s)**

Min-feng Zhu <<wind2zhu@163.com>>
References


See Also

See extrDCC and extrDACC

Examples

```r
x = 'GACTGAACTGCACTTTGTTTCATATTATTTGCTC'
extrDAC(x)
```

---

**extrDAC**

*The Dinucleotide-based Auto-cross Covariance Descriptor*

**Description**

The Dinucleotide-based Auto-cross Covariance Descriptor

**Usage**

```r
extrDAC(x, index = c("Twist", "Tilt"), nlag = 2, normalization = FALSE, customprops = NULL, allprop = FALSE)
```

**Arguments**

- `x`: the input data, which should be a list or file type.
- `index`: the physicochemical indices, it should be a list and there are 38 different physicochemical indices (Table 1), which the users can choose.
- `nlag`: an integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest DNA sequence in the dataset). It represents the distance between two dinucleotides.
- `normalization`: with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.
- `customprops`: the users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.
- `allprop`: all the 38 physicochemical indices will be employed to generate the feature vector. Its default value is False.

**Details**

This function calculates the dinucleotide-based auto-cross covariance descriptor.
extrDCC

Value
A vector

Note
if the user defined physicochemical indices have not been normalized, it should be normalized.

Author(s)
Min-feng Zhu <<wind2zhu@163.com>>

References

See Also
See extrDAC and extrDCC

Examples
```r
x = 'GACTGAACGTGACCTTTGGTTTCATATTATGCT'
extrDAC(x)
```

---

extrDCC | *The Dinucleotide-based Cross Covariance Descriptor*

Description
The Dinucleotide-based Cross Covariance Descriptor

Usage
```r
extrDCC(x, index = c("Twist","Tilt"), nlag = 2, normalization = FALSE, 
customprops = NULL, allprop = FALSE)
```

Arguments
- **x**: the input data, which should be a list or file type.
- **index**: the physicochemical indices, it should be a list and there are 38 different physicochemical indices (Table 1), which the users can choose.
- **nlag**: an integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest DNA sequence in the dataset). It represents the distance between two dinucleotides.
normaliztion with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.

customprops the users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.

allprop all the 38 physicochemical indices will be employed to generate the feature vector. Its default value is False.

Details
This function calculates the dinucleotide-based cross covariance descriptor

Value
A vector

Note
if the user defined physicochemical indices have not been normalized, it should be normalized.

Author(s)
Min-feng Zhu  

References

See Also
See extrDAC and extrDACC

Examples
```r
x = 'GACTGAACTGACTTTGGTCTATATTGGCTC'
extrDCC(x)
```

extrPseDNC The Pseudo Dinucleotide Composition Descriptor

Description
The Pseudo Dinucleotide Composition Descriptor

Usage
extrPseDNC(x, lambda = 3, w = 0.05, normalize = FALSE, customprops = NULL)
**Arguments**

- **x**
  - The input data, which should be a list or file type.
- **lambda**
  - An integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest sequence in the dataset). It represents the highest counted rank (or tier) of the correlation along a DNA sequence. Its default value is 3.
- **w**
  - The weight factor ranged from 0 to 1. Its default value is 0.05.
- **normalize**
  - With this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.
- **customprops**
  - The users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.

**Details**

This function calculates the pseudo dinucleotide composition Descriptor.

**Value**

A vector

**Note**

If the user-defined physicochemical indices have not been normalized, it should be normalized.

**Author(s)**

Min-feng Zhu <<wind2zhu@163.com>>

**References**


**See Also**

See extrPseKNC

**Examples**

```python
x = 'GACTGAACTGACTTTGGTTTCATATTATTTGCTC'
extrPseDNC(x)
```
The Pseudo K-tupler Composition Descriptor

**Description**

The Pseudo K-tupler Composition Descriptor

**Usage**

```python
extrPseKNC(x, lambda = 1, k = 3, normalize = False, w = 0.5,
             customprops = NULL)
```

**Arguments**

- **x**
  - the input data, which should be a list or file type.
- **lambda**
  - an integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest sequence in the dataset). It represents the highest counted rank (or tier) of the correlation along a DNA sequence. Its default value is 3.
- **k**
  - an integer larger than 0 represents the k-tuple. Its default value is 3.
- **normalize**
  - with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.
- **w**
  - the weight factor ranged from 0 to 1. Its default value is 0.05.
- **customprops**
  - the users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.

**Details**

This function calculates the pseudo k-tupler composition Descriptor

**Value**

A vector

**Note**

if the user defined physicochemical indices have not been normalized, it should be normalized.

**Author(s)**

Min-feng Zhu <wind2zhu@163.com>

**References**

The Trinucleotide-based Auto Covariance Descriptor

Usage

extrTAC(x, index = c("Dnase I", "Nucleosome"), nlag = 2,
        normalization = FALSE, customprops = NULL, allprop = FALSE)

Arguments

x the input data, which should be a list or file type.
index the physicochemical indices, it should be a list and there are 12 different physicochemical indices (Table 2), which the users can choose.
nlag an integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest DNA sequence in the dataset). It represents the distance between two dinucleotides.

normalization with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.
customprops the users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.
allprop all the 12 physicochemical indices will be employed to generate the feature vector. Its default value is False.

Details

This function calculates the trinucleotide-based auto covariance Descriptor.

Value

A vector

Note

if the user defined physicochemical indices have not been normalized, it should be normalized.
Author(s)

Min-feng Zhu <wind2zhu@163.com>

See Also

See extrTCC and extrTACC

Examples

```python
x = extrTACC('GACTGAACCTGGTTTTCATATTATTTGCTC')
```

Description

The Trinucleotide-based Auto-cross Covariance Descriptor

Usage

```python
extrTACC(x, index = c("Dnase I", "Nucleosome"), nlag = 2,
    normalization = FALSE, customprops = NULL, allprop = FALSE)
```

Arguments

- `x`: the input data, which should be a list or file type.
- `index`: the physicochemical indices, it should be a list and there are 12 different physicochemical indices (Table 2), which the users can choose.
- `nlag`: an integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest DNA sequence in the dataset). It represents the distance between two dinucleotides.
- `normalization`: with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.
- `customprops`: the users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.
- `allprop`: all the 12 physicochemical indices will be employed to generate the feature vector. Its default value is False.

Details

This function calculates the trinucleotide-based auto-cross covariance descriptor

Value

A vector
Note
if the user defined physicochemical indices have not been normalized, it should be normalized.

Author(s)
Min-feng Zhu <<wind2zhu@163.com>>

See Also
See extrTAC and extrTCC

Examples
```
x = 'GACTGAACTGCACCTTGGTTCTATATTATTTGCTC'
extrTACC(x)
```

### extrTCC

**The Trinucleotide-based Cross Covariance Descriptor**

#### Description

The Trinucleotide-based Cross Covariance Descriptor

#### Usage

```r
extrTCC(x, index = c("Dnase I", "Nucleosome"), nlag = 2,
customprops = NULL, normalization = FALSE)
```

#### Arguments

- **x**
  - the input data, which should be a list or file type.
- **index**
  - the physicochemical indices, it should be a list and there are 12 different physicochemical indices (Table 2), which the users can choose.
- **nlag**
  - an integer larger than or equal to 0 and less than or equal to L-2 (L means the length of the shortest DNA sequence in the dataset). It represents the distance between two dinucleotides.
- **customprops**
  - the users can use their own indices to generate the feature vector. It should be a dict, the key is dinucleotide (string), and its corresponding value is a list type.
- **normalization**
  - with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.

#### Details

This function calculates the trinucleotide-based cross covariance Descriptor
getGenbank

Value
A vector

Note
if the user defined physicochemical indices have not been normalized, it should be normalized.

Author(s)
Min-feng Zhu <<wind2zhu@163.com>>

See Also
See extrTAC and extrTACC

Examples
```r
x <- 'GACTGAACTGCACCTTTGATATTATTTTGCCTC'
eextrTCC(x)
```

getGenbank  Get DNA/RNA Sequences from Genbank by GI ID

Description
Get DNA/RNA Sequences from Genbank by GI ID

Usage
```r
getGenbank(id)
```

Arguments

id  A character vector, as the GI ID(s).

Details
This function get DNA/RNA sequences from Genbank by GI ID(s).

Value
A list, each component contains one of the DNA/RNA sequences.

Author(s)
Min-feng Zhu <<wind2zhu@163.com>>
The Basic Kmer Descriptor

Usage

\[ \text{kmer}(x, \ k = 2, \ \text{upto} = \text{FALSE}, \ \text{normalize} = \text{FALSE}, \ \text{reverse} = \text{FALSE}) \]

Arguments

- **x**: the input data, which should be a list or file type.
- **k**: the k value of kmer, it should be an integer larger than 0.
- **upto**: generate all the kmers: 1mer, 2mer, ..., kmer. The output feature vector is the combination of all these kmers. The default value of this parameter is False.
- **normalize**: with this option, the final feature vector will be normalized based on the total occurrences of all kmers. Therefore, the elements in the feature vectors represent the frequencies of kmers. The default value of this parameter is False.
- **reverse**: make reverse complements into a single feature. The default value of this parameter is False. if reverse is True, this method returns the reverse compliment kmer feature vector.

Details

This function calculates the basic kmer descriptor

Value

A vector
Note

if the parameters normalize and upto are both True, and then the feature vector is the combination of all these normalized kmers, e.g. the combination of normalized 1-kmer and normalized 2-kmer when k=2, normalize=True, upto=True.

Author(s)

Min-feng Zhu <<wind2zhu@163.com>>

References


See Also

See make_kmer_index

Examples

```r
x = 'GACTGAACTGCACTTTGTTTCATATTATTTGCTC'
kmer(x)
```

```r
make_idkmer_vec(k = 6, x, pos, neg, upto = TRUE)
```

Arguments

- **k** the k value of kmer, it should be an integer larger than 0, the default value is 6.
- **x** the input data, which should be a list or file type.
- **pos** the positive source data, which should be a or type.
- **neg** the negative source data, which should be or type.
- **upto** generate all the kmers: 1mer, 2mer, ..., kmer. The output feature vector is the combination of all these kmers. The default value of this parameter is True.
**make_kmer_index**

**Details**

This function calculates the The Basic Kmer Descriptor

**Value**

if upto is True, A length k * 2 named vector, k is the k value of kmer; if upto is False, A length 2 named vector

**Author(s)**

Min-feng Zhu <<wind2zhu@163.com>>

**References**


**See Also**

See [kmer](#)

**Examples**

```r
pos = readFasta(system.file('dnaseq/pos.fasta', package = 'rDNAse'))
neg = readFasta(system.file('dnaseq/neg.fasta', package = 'rDNAse'))
x = 'GACTGAACCTGACCTTTGATCATATTATGCTC'
made_idkmer_vec(k = 6, x, pos, neg)
```

---

**Description**

Calculate The Basic Kmer Feature Vector

**Usage**

```r
make_kmer_index(k, alphabet = "ACGT")
```

**Arguments**

- `k`: the k value of kmer, it should be an integer larger than 0.
- `alphabet`: the

**Details**

This function calculate the basic kmer feature vector.
parGOSim

Value

The result character vector

Author(s)

Min-feng Zhu <wind2zhu@163.com>

See Also

See kmer

Examples

make_kmer_index(2, alphabet = "ACGT")

Description

DNA Sequence Similarity Calculation based on Gene Ontology (GO) Similarity

Usage

parGOSim(golist, type = c("go", "gene"), ont = "MF", organism = "human", measure = "Resnik", combine = "BMA")

Arguments

golist A character vector, each component contains a character vector of GO terms or one Entrez Gene ID.
type Input type of golist, 'go' for GO Terms, 'gene' for gene ID.
ont Default is 'MF', could be one of 'MF', 'BP', or 'CC' subontologies.
organism Default is 'human', could be one of 'anopheles', 'arabidopsis', 'bovine', 'canine', 'chicken', 'chimp', 'coelicolor', 'ecolik12', 'ecsakai', 'fly', 'human', 'malaria', 'mouse', 'pig', 'rat', 'rhesus', 'worm', 'xenopus', 'yeast' or 'zebrafish'.
measure Default is 'Resnik', could be one of 'Resnik', 'Lin', 'Rel', 'Jiang' or 'Wang'.
combine Default is 'BMA', could be one of 'max', 'average', 'rcmax' or 'BMA' for combining semantic similarity scores of multiple GO terms associated with DNA.

Details

This function calculates DNA sequence similarity based on Gene Ontology (GO) similarity.
Value

A n x n similarity matrix.

Author(s)

Min-feng Zhu <wind2zhu@163.com>

See Also

See `twoGOSim` for calculating the GO semantic similarity between two groups of GO terms or two Entrez gene IDs. See `parSeqSim` for paralleled DNA similarity calculation based on Smith-Waterman local alignment.

Examples

```r
## Not run:
# Be careful when testing this since it involves GO similarity computation
# and might produce unpredictable results in some environments

require(GOSemSim)
require(org.Hs.eg.db)

# by GO Terms
go1 = c('GO:0005215', 'GO:0005488', 'GO:0005515', 'GO:0005625', 'GO:0005802', 'GO:0005905') # AP4B1
go2 = c('GO:0005515', 'GO:0005634', 'GO:0005681', 'GO:0008380', 'GO:0031202') # BCAS2
go3 = c('GO:0003735', 'GO:0005622', 'GO:0005840', 'GO:0006412') # PDE4DIP
glist = list(go1, go2, go3)
gsimmat1 = parGOSim(glist, type = 'go', ont = 'CC')
print(gsimmat1)

# by Entrez gene id
genelist = list(c('150', '151', '152', '1814', '1815', '1816'))
gsimmat2 = parGOSim(genelist, type = 'gene')
print(gsimmat2)
## End(Not run)
```

### Description

Parallellized DNA/RNA Sequence Similarity Calculation based on Sequence Alignment

### Usage

```r
parSeqSim(dnalist, cores = 2, type = "local", submat = "BLOSUM62")
```
**Arguments**

- **dnalist**: A length \( n \) list containing \( n \) DNA/RNA sequences, each component of the list is a character string, storing one DNA/RNA sequence. Unknown sequences should be represented as 'G'.
- **cores**: Integer. The number of CPU cores to use for parallel execution, default is 2. Users could use the `detectCores()` function in the `parallel` package to see how many cores they could use.
- **type**: Type of alignment, default is 'local', could be 'global' or 'local', where 'global' represents Needleman-Wunsch global alignment; 'local' represents Smith-Waterman local alignment.
- **submat**: Substitution matrix, default is 'BLOSUM62', could be one of 'BLOSUM45', 'BLOSUM50', 'BLOSUM62', 'BLOSUM80', 'BLOSUM100', 'PAM30', 'PAM40', 'PAM70', 'PAM120', 'PAM250'.

**Details**

This function implemented the parallellized version for calculating DNA/RNA sequence similarity based on sequence alignment.

**Value**

A \( n \times n \) similarity matrix.

**Author(s)**

Min-feng Zhu <wind2zhu@163.com>

**See Also**

See `twoSeqSim` for DNA/RNA sequence alignment for two DNA/RNA sequences. See `parGOSim` for DNA/RNA similarity calculation based on Gene Ontology (GO) semantic similarity.

**Examples**

```r
# Be careful when testing this since it involves parallelisation
# and might produce unpredictable results in some environments

require(Biostrings)
require(foreach)
require(doParallel)

s1 = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNA'))[[1]]
s2 = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNA'))[[2]]
s3 = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNA'))[[3]]
s4 = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNA'))[[4]]
s5 = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNA'))[[5]]
plist = list(s1, s2, s3, s4, s5)
psimmat = parSeqSim(plist, cores = 2, type = 'local', submat = 'BLOSUM62')
print(psimmat)
```
Description

Read DNA/RNA Sequences in FASTA Format

Usage

readFASTA(file, legacy.mode = TRUE, seqonly = FALSE)

Arguments

file The name of the file which the sequences in fasta format are to be read from. If it does not contain an absolute or relative path, the file name is relative to the current working directory, getwd. The default here is to read the example.fasta file which is present in the protseq directory of the protR package.

legacy.mode If set to TRUE, lines starting with a semicolon ';' are ignored. Default value is TRUE.

seqonly If set to TRUE, only sequences as returned without attempt to modify them or to get their names and annotations (execution time is divided approximately by a factor 3). Default value is FALSE.

Details

This function reads DNA/RNA sequences in FASTA format.

Value

The result character vector

Note

Note

Author(s)

Min-feng Zhu <<wind2zhu0163.com>>

References


Examples

x = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNAse'))
### Description

The Reverse chars

### Usage

```r
test <- revchars(x)
```

### Arguments

- **x**: the input data, which should be a string.

### Details

This function calculates Reverse chars

### Value

A vector

### Note

If the user-defined physicochemical indices have not been normalized, it should be normalized.

### Author(s)

Min-feng Zhu <<wind2zhu@163.com>>

### Examples

```r
x <- 'GACTGAAGACTGCAACCTTGTCTTTCATATTATTTGCTC'
test <- revchars(x)
```
twoGOSim  DNA Similarity Calculation based on Gene Ontology (GO) Similarity

Description
DNA Similarity Calculation based on Gene Ontology (GO) Similarity

Usage
twoGOSim(id1, id2, type = c("go", "gene"), ont = "MF", organism = "human", measure = "Resnik", combine = "BMA")

Arguments
id1  A character vector. length > 1: each element is a GO term; length = 1: the Entrez Gene ID.
id2  A character vector. length > 1: each element is a GO term; length = 1: the Entrez Gene ID.
type Input type of id1 and id2, 'go' for GO Terms, 'gene' for gene ID.
ont  Default is 'MF', could be one of 'MF', 'BP', or 'CC' subontologies.
organism Default is 'human', could be one of 'anopheles', 'arabidopsis', 'bovine', 'canine', 'chicken', 'chimp', 'colicilcolor', 'ecolik12', 'ecskakai', 'fly', 'human', 'malaria', 'mouse', 'pig', 'rat', 'rhesus', 'worm', 'xenopus', 'yeast' or 'zebrafish'.
measure Default is 'Resnik', could be one of 'Resnik', 'Lin', 'Rel', 'Jiang' or 'Wang'.
combine Default is 'BMA', could be one of 'max', 'average', 'rcmax' or 'BMA' for combining semantic similarity scores of multiple GO terms associated with DNA.

Details
This function calculates the Gene Ontology (GO) similarity between two groups of GO terms or two Entrez gene IDs.

Value
A $n \times n$ matrix.

Author(s)
Min-feng Zhu <wind2zhu@163.com>
See Also

See parGOSim for DNA similarity calculation based on Gene Ontology (GO) semantic similarity. See parSeqSim for paralleled DNA similarity calculation based on Smith-Waterman local alignment.

Examples

```r
## Not run:
# Be careful when testing this since it involves GO similarity computation
# and might produce unpredictable results in some environments

require(GOSemSim)
require(org.Hs.eg.db)

# by GO terms
go1 = c("GO:0004022", "GO:0004024", "GO:0004023")
go2 = c("GO:0009055", "GO:0020379")
gsim1 = twoGOSim(go1, go2, type = 'go', ont = 'MF', measure = 'Wang')
print(gsim1)

# by Entrez gene id
gene1 = '241'
gene2 = '251'
gsim2 = twoGOSim(gene1, gene2, type = 'gene', ont = 'BP', measure = 'Lin')
print(gsim2)
## End(Not run)
```

---

twoSeqSim DNA/RNA Sequence Alignment for Two DNA/RNA Sequences

**Description**

DNA/RNA Sequence Alignment for Two DNA/RNA Sequences

**Usage**

twoSeqSim(seq1, seq2, type = "local", submat = "BLOSUM62")

**Arguments**

- seq1: A character string, containing one DNA/RNA sequence.
- seq2: A character string, containing another DNA/RNA sequence.
- type: Type of alignment, default is 'local', could be 'global' or 'local', where 'global' represents Needleman-Wunsch global alignment; 'local' represents Smith-Waterman local alignment.
- submat: Substitution matrix, default is 'BLOSUM62', could be one of 'BLOSUM45', 'BLOSUM50', 'BLOSUM62', 'BLOSUM80', 'BLOSUM100', 'PAM30', 'PAM40', 'PAM70', 'PAM120', 'PAM250'.

Details

This function implements the sequence alignment between two DNA/RNA sequences.

Value

An Biostrings object containing the scores and other alignment information.

Author(s)

Min-feng Zhu <<wind2zhu@163.com>>

See Also

See parSeqSim for paralleled pairwise DNA/RNA similarity calculation based on sequence alignment. See twoGOsim for calculating the GO semantic similarity between two groups of GO terms or two Entrez gene IDs.

Examples

# Be careful when testing this since it involves sequence alignment
# and might produce unpredictable results in some environments

require(Biostrings)

s1 = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNA'))[[1]]
s2 = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNA'))[[2]]
seqalign = twoSeqSim(s1, s2)
summary(seqalign)
print(seqalign@score)
# Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Functions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DACC</strong></td>
<td>extrDACC, extrDAC</td>
<td>4, 3</td>
</tr>
<tr>
<td><strong>DAC</strong></td>
<td>extrDAC, extrDACC</td>
<td>3, 4</td>
</tr>
<tr>
<td><strong>DCC</strong></td>
<td>extrDCC, extrDAC</td>
<td>5, 4</td>
</tr>
<tr>
<td><strong>FASTA</strong></td>
<td>readFASTA, extrDAC, extrDACC</td>
<td>19, 4</td>
</tr>
<tr>
<td><strong>GO</strong></td>
<td>parGOsim, twoGOsim</td>
<td>16, 21</td>
</tr>
<tr>
<td><strong>Genbank</strong></td>
<td>getGenbank, extrDAC, extrDACC</td>
<td>12, 4</td>
</tr>
<tr>
<td><strong>Ontology</strong></td>
<td>parGOsim, twoGOsim</td>
<td>16, 21</td>
</tr>
<tr>
<td><strong>PseDNC</strong></td>
<td>extrPseDNC, extrDAC, extrDACC</td>
<td>6, 4</td>
</tr>
<tr>
<td><strong>PseKNC</strong></td>
<td>extrPseKNC, extrDAC, extrDACC</td>
<td>8, 4</td>
</tr>
<tr>
<td><strong>TACC</strong></td>
<td>extrTACC, extrDAC</td>
<td>9, 10</td>
</tr>
<tr>
<td><strong>TAC</strong></td>
<td>extrTAC, extrDAC</td>
<td>9, 10</td>
</tr>
<tr>
<td><strong>TCC</strong></td>
<td>extrTCC, extrDAC</td>
<td>11, 10</td>
</tr>
<tr>
<td><strong>alignment</strong></td>
<td>parSeqSim, twoSeqSim</td>
<td>17, 22</td>
</tr>
<tr>
<td><strong>check</strong></td>
<td>dnacheck, extrDAC, extrDACC</td>
<td>2, 4</td>
</tr>
<tr>
<td><strong>diversity</strong></td>
<td>make_idkmer_vec</td>
<td>14, 14</td>
</tr>
<tr>
<td><strong>extract</strong></td>
<td>extrDAC, extrDACC, extrDCC, extrPseDNC, extrPseKNC</td>
<td>3, 4, 5, 6, 4, 5, 6, 5, 6, 8</td>
</tr>
<tr>
<td><strong>increment</strong></td>
<td>make_idkmer_vec</td>
<td>14, 14</td>
</tr>
<tr>
<td><strong>index</strong></td>
<td>make_kmer_index, extrDAC, extrDACC</td>
<td>15, 15</td>
</tr>
<tr>
<td><strong>kmer</strong></td>
<td>kmer, make_kmer_index, extrDAC, extrDACC</td>
<td>13, 15</td>
</tr>
<tr>
<td><strong>of</strong></td>
<td>make_idkmer_vec</td>
<td>14, 14</td>
</tr>
<tr>
<td><strong>parallel</strong></td>
<td>parSeqSim, twoSeqSim</td>
<td>17, 22</td>
</tr>
<tr>
<td><strong>read</strong></td>
<td>readFASTA, extrDAC, extrDACC</td>
<td>19, 4</td>
</tr>
<tr>
<td><strong>reverse_chars</strong></td>
<td>revchars, extrDAC, extrDACC</td>
<td>20, 4</td>
</tr>
<tr>
<td><strong>similarity</strong></td>
<td>parGOsim, twoGOsim, extrDACC</td>
<td>16, 21</td>
</tr>
<tr>
<td><strong>the</strong></td>
<td>make_idkmer_vec</td>
<td>14, 14</td>
</tr>
<tr>
<td><strong>train</strong></td>
<td>make_idkmer_vec</td>
<td>14, 14</td>
</tr>
<tr>
<td><strong>TAC</strong></td>
<td>extrTAC, extrDACC</td>
<td>9, 11, 12</td>
</tr>
<tr>
<td><strong>TACC</strong></td>
<td>extrTACC, extrDAC</td>
<td>10, 11, 12</td>
</tr>
<tr>
<td><strong>TCC</strong></td>
<td>extrTCC, extrDAC</td>
<td>10, 11, 11</td>
</tr>
</tbody>
</table>
INDEX

getGenbank, 12
getwd, 19

IncDiv(make_idkmer_vec), 14

kmer, 13, 15, 16

make_idkmer_vec, 14
make_kmer_index, 14, 15

parGOSim, 16, 18, 22
parSeqSim, 17, 17, 22, 23

readFASTA, 13, 19
revchars, 20

twoGOSim, 17, 21, 23
twoSeqSim, 22