rDNAse: R package for generating various numerical representation schemes of DNA sequences

Min-feng Zhu, Jie Dong, Dong-Sheng Cao

Package Version: 1.1-0
2016-07-12

COMPUTATIONAL BIOLOGY &
DRUG DESIGN GROUP
CENTRAL SOUTH UNIV., CHINA

Contents

Abstract

The rDNAse R package can generate various feature vectors for DNA sequences, this R package could: 1) Calculate three nucleic acid composition features describing the local sequence information by means of kmers (subsequences of DNA sequences); 2) Calculate six autocorrelation features describing the level of correlation between two oligonucleotides along a DNA sequence in terms of their specific physicochemical properties; 3) Calculate six pseudo nucleotide composition features, which can be used to represent a DNA sequence with a discrete model or vector yet still keep considerable sequence order information, particularly the global or long-range sequence order information, via the physicochemical properties of its constituent oligonucleotides. There are five modules in the **rDNAse** package, including basic tools, nucleic acid composition, autocorrelation, pseudo nucleotide composition and similarity calculation. The basic tools module contains several basic functions manipulating DNA data, including reading DNA data from files or list (a data structure in R), checking the validity and get DNA sequences from Genbank by GI ID(s), etc. The three modules nucleic acid composition, autocorrelation, pseudo nucleotide composition respond to the calculation of 11 different features from three feature categories. The nucleic acid composition aims at computing three types of nucleic acid composition, including basic kmer, reverse compliment kmer and increment of diversity; The autocorrelation aims at computing six types of autocorrelation, including dinucleotide-based auto covariance (DAC), dinucleotide-based cross covariance (DCC), dinucleotide-based auto-cross covariance (DACC), trinucleotide-based auto covariance (TAC), trinucleotide-based cross covariance (TCC), and trinucleotide- based auto-cross covariance (TACC); The pseudo nucleotide composition aims at computing two types of pseudo nucleic acid composition: pseudo dinucleotide composition (PseDNC), pseudo k-tuple nucleotide composition (PseKNC). The similarity calculation module contains similarity calculation by sequence alignment and GO semantic similarity measures.

Keywords: DNA sequence, Descriptor, Sequence alignment, Gene Ontology, Sequence features

1. The Full Workflow Using rDNAse

Here we use the DNaseI HSs in the human genome presented in the study of Noble *et al.* (2005) to demonstrate the full workflow when using rDNAse.

We select two classes of DNA as our benchmark dataset. 280 validated erythroid HS sequences were treated as the positive sample set, 737 validated erythroid HS sequences were treated as the negative sample set.

First, we load the **rDNAse** package, then read the DNA sequences stored in two separated FASTA files with `readFASTA()`:

```
require(rDNAse)

# load FASTA files
pos_hs = readFASTA(system.file('dnaseq/hs.fasta',
                               package = 'rDNAse'))
neg_hs = readFASTA(system.file('dnaseq/non-hs.fasta',
                               package = 'rDNAse'))
```

```
length(pos_hs)
```

```
## [1] 280
```

```
length(neg_hs)
```

```
## [1] 737
```

To assure that the DNA sequences only have the dour standard deoxyribonucleic acid types which is required for the descriptor computation, we use the `dnacheck()` function in **rDNAse** to do the deoxyribonucleic acid type sanity checking and remove the *non-standard* sequences:

```
pos_hs = pos_hs[(sapply(pso_hs, dnacheck))]
neg_hs = neg_hs[(sapply(neg_hs, dnacheck))]
```

```
length(pos_hs)
```

```
## [1] 280
```

```
length(neg_hs)
```

```
## [1] 737
```

For the remaining sequences, we calculate the kmer descriptor, i.e., the kmer descriptor (Noble *et al.* 2005) and make class labels for classification modeling.

```
# calculate kmer descriptors
x1 = t(sapply(pos_hs, kmer))
x2 = t(sapply(neg_hs, kmer))
x  = rbind(x1, x2)

# make class labels
labels = as.factor(c(rep(0, length(pos_hs)), rep(1, length(neg_hs))))
```

Next, we will split the data into a 75% training set and a 25% test set.

```
# split training and test set
set.seed(1001)
tr.idx = c(sample(1:nrow(x1), round(nrow(x1) * 0.75)),
           sample(nrow(x1) + 1:nrow(x2), round(nrow(x2) * 0.75)))
te.idx = setdiff(1:nrow(x), tr.idx)
x.tr   = x[tr.idx, ]
x.te   = x[te.idx, ]
y.tr   = labels[tr.idx]
y.te   = labels[te.idx]
```

We will train a random forest classification model on the training set with 5-fold cross-validation, using the **randomForest** package.

```
require(randomForest)
rf.fit = randomForest(x.tr, y.tr, cv.fold = 5)
print(rf.fit)
```

The training result is:

```
## Call:
##  randomForest(x = x.tr, y = y.tr, cv.fold = 5)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 13.89%
## Confusion matrix:
##     0   1 class.error
## 0 138  72  0.34285714
## 1  34 519  0.06148282
```

With the model trained on the training set, we predict on the test set and plot the ROC curve with the **pROC** package, as is shown in figure 1.

```
# predict on test set
rf.pred = predict(rf.fit, newdata = x.te, type = 'prob')[, 1]

# plot ROC curve
require(pROC)
plot.roc(y.te, rf.pred, col = '#0080ff', grid = TRUE, print.auc = TRUE)
```

The area under the ROC curve (AUC) is:

```
## Call:
## plot.roc.default(x = y.te, predictor = rf.pred, col = "#0080ff",
##                  grid = TRUE, print.auc = TRUE)
##
## Data: rf.pred in 70 controls (y.te 0) > 184 cases (y.te 1).
## Area under the curve: 0.7799
```

2. Package Overview

The **rDNAse** package is freely available from the Comprehensive R Archive Network (http://CRAN.R-project.org/package=rDNAse). This vignette corresponds to **rDNAse** version 1.1-0 and was typeset on 2016-07-12.

The **rDNAse** computes three feature groups composed of 11 different features, including:

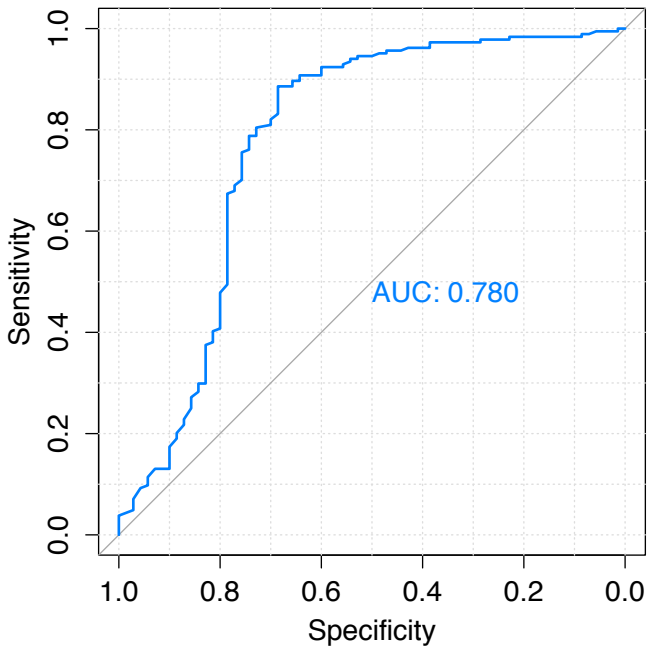

Figure 1: ROC curve achieved by the predictor based on kmers for predicting DNaseI HSs in the human genome

- Basic function
  - `readFASTA()` - Read DNA sequence data from FASTA files
- Nucleic acid composition
  - `kmer()` - Basic kmer and Reverse compliment kmer
  - `make_idkmer_vec()` - Increment of diversity (ID)
- Autocorrelation
  - `extrDAC()` - Dinucleotide-based auto covariance
  - `extrDCC()` - Dinucleotide-based cross covariance
  - `extrDACC()` - Dinucleotide-based auto-cross covariance
  - `extrTAC()` - Trinucleotide-based auto covariance
  - `extrTCC()` - Trinucleotide-based cross covariance
  - `extrDACC()` - Trinucleotide-based auto-cross covariance
- Pseudo nucleotide composition
  - `extrPseDNC()` - Pseudo dinucleotide composition
  - `extrPseKNC()` - Pseudo k-tupler nucleotide composition

The **rDNAse** package integrates the function of parallelized similarity score computation derived by local or global DNA sequence alignment between a list of DNA sequences, the sequence alignment computation is provided by **Biostrings**, the corresponding functions listed in the **rDNAse** package include:

- `twoSeqSim()` - Similarity calculation derived by sequence alignment between two DNA sequences

The **rDNAse** package also integrates the function of parallelized similarity score computation derived by Gene Ontology (GO) semantic similarity measures between a list of GO terms / Entrez Gene IDs, the GO similarity computation is provided by **GOSemSim**, the corresponding functions listed in the **rDNAse** package include:

- `twoGOSim()` - Similarity calculation derived by GO-terms semantic similarity measures between two GO terms / Entrez Gene IDs

In the next sections, we'll introduce the descriptors and function usage in this order.

3. Commonly Used Descriptors

Disclaimer. Users of the **rDNAse** package need to intelligently evaluate the underlying details of the descriptors provided, instead of using rDNAse with their data blindly, especially for the descriptor types with more flexibility. It would be wise for the users to use some negative and positive control comparisons where relevant to help guide interpretation of the results.

A DNA or deoxyribonucleic acid sequence with N deoxyribonucleic acid could be generally represented as $\{ R_1, R_2, \ldots, R_n \}$, where R_i represents the residue at the i-th position in the sequence. The labels i and j are used to index deoxyribonucleic acid position in a sequence, and r, s, t are used to represent the Deoxyribonucleic acid type .

A DNA sequence could be divided equally into segments and the methods, described as follows for the global sequence, could be applied to each segment.

3.1. kmer

Basic kmer is the simplest approach to represent the DNAs, in which the DNA sequences are represented as the occurrence frequencies of k neighboring nucleic acids. This approach has been successfully applied to human gene regulatory sequence prediction (Noble *et al.* 2005), enhancer identification (Lee *et al.* 2011), etc.

$$f(r,s) = \frac{N_{rs}}{N-1} \quad r,s = 1,2,\ldots,16.$$

where N_{rs} is the number of dipeptide represented by deoxyribonucleic acid r and type s. here we use `kmer()` to compute the descriptors:

```
require(rDNAse)

## Loading required package:  rDNAse

x = "AATTCATGCGTCCGGACTTCTGCCTCGAGCCGCCGTACACTGGGCCCTGCAAAGCTC"
kmer(x)

## AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
##  3  3  2  2  3  6  5  6  2  7  3  2  1  5  4  2
```

if reverse is TRUE, we can use `kmer()` to compute the reverse compliment kmer.The reverse compliment kmer is a variant of the basic kmer, in which the kmers are not expected to be strand-specific, so reverse complements are collapsed into a single feature. For example, if k=2, there are totally 16 basic kmers ('AA', 'AC', 'AG', 'AT', 'CA', 'CC', 'CG', 'CT', 'GA', 'GC', 'GG', 'GT', 'TA', 'TC', 'TG', 'TT'), but by removing the reverse compliment kmers, there are only 10 distinct kmers in the reverse compliment kmer approach ('AA', 'AC', 'AG', 'AT', 'CA', 'CC', 'CG', 'GA', 'GC', 'TA'). For more information of this approach, please refer to (Noble *et al.* 2005) (Gupta *et al.* 2008).

```
kmer(x, k = 2, reverse = TRUE)

## AA AC AG AT CA CC CG GA GC TA
##  5  5  8  2  7  9  5  7  7  1
```

3.2. Increment of diversity

The increment of diversity has been successfully applied in the prediction of exonintron splice sites for several model genomes Zhang and Luo (2003), transcription start site prediction, and studying the organization of nucleosomes around splice sites Lu and Luo (2008). In this method, the sequence features are converted into the increment of diversity (ID), defined by the relation of sequence X with standard source S:

$$ID = Diversity(X + S) - Diversity(S) - Diversity(X)$$

Given a sequence X with r feature variables (ID_1 to IDr), we obtain an r-dimensional feature vector $\mathbf{R} = (ID_1, ID_2, \ldots, ID_r)$. The feature vector $\mathbf{R}$ is designed by the following considerations. The kmers are responsible for the discrimination between positive samples and negative samples, and therefore they construct the diversity sources. Based on this, 2 kmer-based increments of diversities ID_1 (ID_2) between sequence `X` and the standard source in positive (negative) training set can be easily introduced as the feature vectors. 9 For more information of this approach, please refer to (Chen *et al.* 2010) and (Liu *et al.* 2012).

```
pos = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNAse'))
neg = readFASTA(system.file('dnaseq/non-hs.fasta', package = 'rDNAse'))
make_idkmer_vec(k = 6, x, pos, neg)
```

```
## [[1]]
##  [1]    1.510142    3.801528 -194.195500 -194.304161 -277.470938
##  [6] -277.745802 -304.936679 -304.734915 -303.579784 -303.579784
## [11] -296.422865 -296.422865
```

4. Autocorrelation

Autocorrelation, as one of the multivariate modeling tools, can transform the DNA sequences of different lengths into fixed-length vectors by measuring the correlation between any two properties. Autocorrelation results in two kinds of variables: autocorrelation `AC` between the same property, and cross-covariance `CC` between two different properties. Module **ac** (ac is the abbreviation of autocorrelation) in rDNAse allows the users to generate various kinds of autocorrelation feature vectors for given DNA sequences or FASTA files by selecting different methods and parameters. This module aims at computing six types of autocorrelation, including dinucleotide-based auto covariance `DAC`, dinucleotide-based cross covariance `DCC`, dinucleotide-based auto-cross covariance `DACC`, trinucleotide-based auto covariance `TAC`, trinucleotide-based cross covariance `TCC`, and trinucleotide-based auto-cross covariance `TACC`. Let's introduce them one by one.

4.1. Dinucleotide-based auto covariance

Suppose a DNA sequence **D** with $\boldsymbol{L}$ nucleic acid residues; i.e.

$$\mathbf{D} = \boldsymbol{R_1 R_2 R_3 R_4 R_5 R_6 R_7 \ldots R_L}$$

where $\boldsymbol{R_1}$ represents the nucleic acid residue at the sequence position 1, $\boldsymbol{R_2}$ the nucleic acid residue at position 2 and so forth. The `DAC` measures the correlation of the same physicochemical index between two dinucleotide separated by a distance of lag along the sequence, which can be calculated as:

$$\boldsymbol{DAC(u, lag)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{L-lag-1}} \frac{(\boldsymbol{P_u(R_i R_{i+1})} - \boldsymbol{\bar{P}_u})(\boldsymbol{P_u(R_{i+lag} R_{i+lag+1})} - \boldsymbol{\bar{P}_u})}{\boldsymbol{L - lag - 1}}$$

where $\boldsymbol{u}$ is a physicochemical index, $\boldsymbol{L}$ is the length of the DNA sequence, $\boldsymbol{P_u(R_i R_{i+1})}$ means the numerical value of the physicochemical index $\boldsymbol{u}$ for the dinucleotide $\boldsymbol{R_i R_{i+1}}$ at position $\boldsymbol{i}$, $\boldsymbol{\bar{P}_u}$ sequence: is the average value for physicochemical index $\boldsymbol{u}$ along the whole sequence:

$$\boldsymbol{\bar{P}_u} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{L-1}} \frac{\boldsymbol{P_u(R_j R_{j+1})}}{\boldsymbol{L - 1}}$$

In such a way, the length of `DAC` feature vector is N $*$ LAG, where N is the number of physicochemical indices and LAG is the maximum of lag (lag = 1, 2, ..., LAG). This `DAC` approach is similar as the approach used for DNA fold recognition (Dong *et al.* 2009).

```
extrDAC(x)

## Twist.lag1 Twist.lag2  Tilt.lag1  Tilt.lag2
##     -0.305      0.167     -0.181      0.157

phyche_index = data.frame(cust1 = c(2.26, 3.03, 2.03, 3.83, 1.78, 1.65, 2.00,
2.03, 1.93, 2.61, 1.65, 3.03, 1.20, 1.93, 1.78, 2.26), cust2 = c(7.65, 8.93,
7.08, 9.07, 6.38, 8.04, 6.23, 7.08, 8.56, 9.53, 8.04, 8.93, 6.23, 8.56, 6.38,
7.65))
custom props = t(phyche_index)
colnames(custom props) = make_kmer_index(2, alphabet = "ACGT")
extrDAC(x, normaliztion = TRUE, custom props = custom props)

## Twist.lag1 Twist.lag2  Tilt.lag1  Tilt.lag2 cust1.lag1 cust1.lag2
##     -0.305      0.167     -0.181      0.157     -0.195      0.082
## cust2.lag1 cust2.lag2
##     -0.521      0.157
```

4.2. Dinucleotide-based cross covariance

Given a DNA sequence D, the DCC approach measures the correlation of two different physicochemical indices between two dinucleotides separated by lag nucleic acids along the sequence, which can be calculated by:

$$DCC(u_1, u_2, lag) = \sum_{i=1}^{L-lag-1} \frac{(P_{u_1}(R_i R_{i+1}) - \bar{P_{u_1}})(P_{u_2}(R_{i+lag} R_{i+lag+1}) - \bar{P_{u_2}})}{L - lag - 1}$$

where u_1, u_2 are two different physicochemical indices, L is the length of the DNA sequence, $P_{u_1}(R_i R_{i+1})(P_{u_2}(R_i R_{i+1}))$ is the numerical value of the physicochemical index $u_1(u_2)$ for the dinucleotide at position i, $\bar{P_{u_1}}(\bar{P_{u_2}})$ is the average value for physicochemical index value u_1, u_2 along the whole sequence:

In such a way, the length of the DCC feature vector is N∗(N-1)∗LAG, where N is the number of physicochemical indices and LAG is the maximum of lag (lag=1, 2, ..., LAG).

This DCC approach is similar as the approach used for DNA fold recognition (Dong *et al.* 2009).

```
extrDCC(x)

## Twist.Tilt.lag.1 Twist.Tilt.lag.2 Tilt.Twist.lag.1 Tilt.Twist.lag.2
##           -0.281            0.144           -0.287            0.206

phyche_index = data.frame(cust1 = c(2.26, 3.03, 2.03, 3.83, 1.78, 1.65, 2.00,
2.03, 1.93, 2.61, 1.65, 3.03, 1.20, 1.93, 1.78, 2.26), cust2 = c(7.65, 8.93,
```

```
7.08, 9.07, 6.38, 8.04, 6.23, 7.08, 8.56, 9.53, 8.04, 8.93, 6.23, 8.56, 6.38,
7.65))
custompros = t(phyche_index)
colnames(custompros) = make_kmer_index(2, alphabet = "ACGT")
extrDCC(x, normaliztion = TRUE, custompros = custompros)

##  Twist.Tilt.lag.1  Twist.Tilt.lag.2  Tilt.Twist.lag.1  Tilt.Twist.lag.2
##            -0.281             0.144            -0.288             0.206
## Twist.cust1.lag.1 Twist.cust1.lag.2 cust1.Twist.lag.1 cust1.Twist.lag.2
##            -0.322             0.215            -0.123             0.080
## Twist.cust2.lag.1 Twist.cust2.lag.2 cust2.Twist.lag.1 cust2.Twist.lag.2
##            -0.434             0.261            -0.239             0.139
##  Tilt.cust1.lag.1  Tilt.cust1.lag.2  cust1.Tilt.lag.1  cust1.Tilt.lag.2
##            -0.323             0.227            -0.115             0.073
##  Tilt.cust2.lag.1  Tilt.cust2.lag.2  cust2.Tilt.lag.1  cust2.Tilt.lag.2
##            -0.426             0.276            -0.144             0.075
## cust1.cust2.lag.1 cust1.cust2.lag.2 cust2.cust1.lag.1 cust2.cust1.lag.2
##            -0.236             0.114            -0.405             0.237
```

4.3. Dinucleotide-based auto-cross covariance

DACC is a combination of DAC and DCC. Therefore, the length of the DACC feature vector is N∗N∗LAG, where N is the number of physicochemical indices and LAG is the maximum of lag (lag = 1, 2, . . ., LAG).

```
extrDACC(x)

##       Twist.lag1       Twist.lag2        Tilt.lag1        Tilt.lag2
##           -0.305            0.167           -0.181            0.157
## Twist.Tilt.lag.1 Twist.Tilt.lag.2 Tilt.Twist.lag.1 Tilt.Twist.lag.2
##           -0.281            0.144           -0.287            0.206

phyche_index = data.frame(cust1 = c(2.26, 3.03, 2.03, 3.83, 1.78, 1.65, 2.00,
2.03, 1.93, 2.61, 1.65, 3.03, 1.20, 1.93, 1.78, 2.26), cust2 = c(7.65, 8.93,
7.08, 9.07, 6.38, 8.04, 6.23, 7.08, 8.56, 9.53, 8.04, 8.93, 6.23, 8.56, 6.38,
7.65))
custompros = t(phyche_index)
colnames(custompros) = make_kmer_index(2, alphabet = "ACGT")
extrDACC(x, normaliztion = TRUE, custompros = custompros)

##       Twist.lag1       Twist.lag2        Tilt.lag1        Tilt.lag2
##           -0.305            0.167           -0.181            0.157
##       cust1.lag1       cust1.lag2       cust2.lag1       cust2.lag2
##           -0.195            0.082           -0.521            0.157
##  Twist.Tilt.lag.1  Twist.Tilt.lag.2  Tilt.Twist.lag.1  Tilt.Twist.lag.2
```

```
##            -0.281             0.144            -0.288             0.206
## Twist.cust1.lag.1 Twist.cust1.lag.2 cust1.Twist.lag.1 cust1.Twist.lag.2
##            -0.322             0.215            -0.123             0.080
## Twist.cust2.lag.1 Twist.cust2.lag.2 cust2.Twist.lag.1 cust2.Twist.lag.2
##            -0.434             0.261            -0.239             0.139
##  Tilt.cust1.lag.1  Tilt.cust1.lag.2  cust1.Tilt.lag.1  cust1.Tilt.lag.2
##            -0.323             0.227            -0.115             0.073
##  Tilt.cust2.lag.1  Tilt.cust2.lag.2  cust2.Tilt.lag.1  cust2.Tilt.lag.2
##            -0.426             0.276            -0.144             0.075
## cust1.cust2.lag.1 cust1.cust2.lag.2 cust2.cust1.lag.1 cust2.cust1.lag.2
##            -0.236             0.114            -0.405             0.237
```

4.4. Trinucleotide-based auto covariance

Given a DNA sequence $\boldsymbol{D}$, the TAC approach measures the correlation of the same physicochemical index between two trinucleotides separated by $\boldsymbol{lag}$ nucleic acids along the sequence, which can be calculated as:

$$\boldsymbol{TAC(u, lag) = \sum_{i=1}^{L-lag-2} \frac{(P_u(R_i R_{i+1} R_{i+2}) - \bar{P}_u)(P_u(R_{i+lag} R_{i+lag+1} R_{i+lag+2}) - \bar{P}_u)}{L - lag - 2}}$$

where $\boldsymbol{u}$ is a physicochemical index, $\boldsymbol{L}$ is the length of the DNA sequence, $\boldsymbol{P_u(R_i R_{i+1} R_{i+2})}$ represents the numerical value of the physicochemical index $\boldsymbol{u}$ for the trinucleotide $\boldsymbol{R_i R_{i+1} R_{i+2}}$ at position $\boldsymbol{i}$, $\boldsymbol{\bar{P}_u}$ is the average value for physicochemical index $\boldsymbol{u}$ value along the whole sequence:

$$\boldsymbol{\bar{P}_u = \sum_{j=1}^{L-2} \frac{P_u(R_j R_{j+1} R_{j+2})}{L - 2}}$$

In such a way, the length of TAC feature vector is N∗LAG, where N is the number of physicochemical indices and LAG is the maximum of lag (lag=1, 2, . . ., LAG).

```
extrTAC(x)

##    Dnase I.lag1    Dnase I.lag2 Nucleosome.lag1 Nucleosome.lag2
##           0.253          -0.037           0.482           0.009

phyche_index = data.frame(cust = c(7.176, 6.272, 4.736, 7.237, 3.810, 4.156,
4.156, 6.033, 3.410, 3.524, 4.445, 6.033, 1.613, 5.087, 2.169, 7.237, 3.581,
3.239, 1.668, 2.169, 6.813, 3.868, 5.440, 4.445, 3.810, 4.678, 5.440, 4.156,
2.673, 3.353, 1.668, 4.736, 4.214, 3.925, 3.353, 5.087, 2.842, 2.448, 4.678,
3.524, 3.581, 2.448, 3.868, 4.156, 3.467, 3.925, 3.239, 6.272, 2.955, 3.467,
2.673, 1.613, 1.447, 3.581, 3.810, 3.410, 1.447, 2.842, 6.813, 3.810, 2.955,
4.214, 3.581, 7.176))
```

```
custompropS = t(phyche_index)
colnames(custompropS) = make_kmer_index(3, alphabet = "ACGT")
extrTAC(x, normaliztion = TRUE, custompropS = custompropS)

##    Dnase I.lag1    Dnase I.lag2 Nucleosome.lag1 Nucleosome.lag2
##           0.249          -0.037           0.474           0.009
##       cust.lag1       cust.lag2
##           0.144          -0.035
```

4.5. Trinucleotide-based cross covariance

Given a DNA sequence $\boldsymbol{D}$, the TCC approach measures the correlation of two different physicochemical indices between two trinucleotides separated by lag nucleic acids along the sequence, which can be calculated by:

$$\boldsymbol{TCC(u_1, u_2, lag)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{L-lag-2}} \frac{(\boldsymbol{P_{u_1}(R_iR_{i+1}R_{i+2})} - \bar{\boldsymbol{P}}_{\boldsymbol{u_1}})(\boldsymbol{P_{u_2}(R_{i+lag}R_{i+lag+1}R_{i+lag+2})} - \bar{\boldsymbol{P}}_{\boldsymbol{u_2}})}{\boldsymbol{L - lag - 2}}$$

where $\boldsymbol{u_1}$, $\boldsymbol{u_2}$ are two physicochemical indices, $\boldsymbol{L}$ is the length of the DNA sequence,$\boldsymbol{P_{u_1}(R_iR_{i+1}R_{i+2})}$ $\boldsymbol{(P_{u_2}(R_iR_{i+1}R_{i+2}))}$ represents the numerical value of the physicochemical index $\boldsymbol{u_1(u_2)}$ for the trinucleotide $\boldsymbol{R_iR_{i+1}R_{i+2}}$ at position $\boldsymbol{i}$,

$$\bar{\boldsymbol{P}}_{\boldsymbol{u}} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{L-2}} \frac{\boldsymbol{P_u(R_jR_{j+1}R_{j+2})}}{\boldsymbol{L-2}}$$

In such a way, the length of TCC feature vector is N∗(N-1)∗LAG, where N is the number of physicochemical index and LAG is the maximum of lag (lag = 1, 2, ..., LAG).

```
extrTCC(x)

## Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2 Nucleosome.Dnase I.lag.1
##                   -0.375                   -0.042                   -0.294
## Nucleosome.Dnase I.lag.2
##                    0.095

phyche_index = data.frame(cust = c(7.176, 6.272, 4.736, 7.237, 3.810, 4.156,
4.156, 6.033, 3.410, 3.524, 4.445, 6.033, 1.613, 5.087, 2.169, 7.237, 3.581,
3.239, 1.668, 2.169, 6.813, 3.868, 5.440, 4.445, 3.810, 4.678, 5.440, 4.156,
2.673, 3.353, 1.668, 4.736, 4.214, 3.925, 3.353, 5.087, 2.842, 2.448, 4.678,
3.524, 3.581, 2.448, 3.868, 4.156, 3.467, 3.925, 3.239, 6.272, 2.955, 3.467,
2.673, 1.613, 1.447, 3.581, 3.810, 3.410, 1.447, 2.842, 6.813, 3.810, 2.955,
4.214, 3.581, 7.176))
custompropS = t(phyche_index)
colnames(custompropS) = make_kmer_index(3, alphabet = "ACGT")
extrTCC(x, normaliztion = TRUE, custompropS = custompropS)
```

```
## Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2 Nucleosome.Dnase I.lag.1
##                   -0.369                   -0.041                   -0.289
## Nucleosome.Dnase I.lag.2       Dnase I.cust.lag.1       Dnase I.cust.lag.2
##                    0.093                    0.145                   -0.097
##       cust.Dnase I.lag.1       cust.Dnase I.lag.2    Nucleosome.cust.lag.1
##                    0.207                    0.009                   -0.136
##    Nucleosome.cust.lag.2    cust.Nucleosome.lag.1    cust.Nucleosome.lag.2
##                    0.170                   -0.275                   -0.065
```

4.6. Trinucleotide-based auto-cross covariance

TACC is a combination of TAC and TCC. Therefore, the length of the TACC feature vector is N∗N∗LAG, where N is the number of physicochemical indices and LAG is the maximum of lag (lag = 1, 2, . . ., LAG).

```
extrTACC(x)
```

```
##             Dnase I.lag1             Dnase I.lag2          Nucleosome.lag1
##                    0.253                   -0.037                    0.482
##          Nucleosome.lag2 Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2
##                    0.009                   -0.375                   -0.042
## Nucleosome.Dnase I.lag.1 Nucleosome.Dnase I.lag.2
##                   -0.294                    0.095
```

```
phyche_index = data.frame(cust = c(7.176, 6.272, 4.736, 7.237, 3.810, 4.156,
4.156, 6.033, 3.410, 3.524, 4.445, 6.033, 1.613, 5.087, 2.169, 7.237, 3.581,
3.239, 1.668, 2.169, 6.813, 3.868, 5.440, 4.445, 3.810, 4.678, 5.440, 4.156,
2.673, 3.353, 1.668, 4.736, 4.214, 3.925, 3.353, 5.087, 2.842, 2.448, 4.678,
3.524, 3.581, 2.448, 3.868, 4.156, 3.467, 3.925, 3.239, 6.272, 2.955, 3.467,
2.673, 1.613, 1.447, 3.581, 3.810, 3.410, 1.447, 2.842, 6.813, 3.810, 2.955,
4.214, 3.581, 7.176))
custompros = t(phyche_index)
colnames(custompros) = make_kmer_index(3, alphabet = "ACGT")
extrTACC(x, normaliztion = TRUE, custompros = custompros)
```

```
##             Dnase I.lag1             Dnase I.lag2          Nucleosome.lag1
##                    0.249                   -0.037                    0.474
##          Nucleosome.lag2                cust.lag1                cust.lag2
##                    0.009                    0.144                   -0.035
## Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2 Nucleosome.Dnase I.lag.1
##                   -0.369                   -0.041                   -0.289
## Nucleosome.Dnase I.lag.2       Dnase I.cust.lag.1       Dnase I.cust.lag.2
##                    0.093                    0.145                   -0.097
##       cust.Dnase I.lag.1       cust.Dnase I.lag.2    Nucleosome.cust.lag.1
##                    0.207                    0.009                   -0.136
##    Nucleosome.cust.lag.2    cust.Nucleosome.lag.1    cust.Nucleosome.lag.2
##                    0.170                   -0.275                   -0.065
```

5. Pseudo nucleic acid composition

PseNAC is a kind of powerful approaches to represent the DNA sequences considering both DNA local sequence-order information and long range or global sequence-order effects. Module psenac (psenac is the abbreviation of pseudo nucleic acid composition) in repDNA allows users to generate various kinds of PseNAC- based feature vectors for given sequences or FASTA files by selecting different methods and parameters. This module aims at computing six types of pseudo nucleic acid composition: pseudo dinucleotide composition (PseDNC), pseudo k-tuple nucleotide composition (PseKNC), parallel correlation pseudo dinucleotide composition (PC-PseDNC), parallel correlation pseudo trinucleotide composition (PC-PseTNC), series correlation pseudo dinucleotide composition (SC-PseDNC), and series correlation pseudo trinucleotide composition (SC-PseTNC). Let's introduce them one by one.

5.1. Pseudo dinucleotide composition

PseDNC is an approach incorporating the contiguous local sequence-order information and the global sequence-order information into the feature vector of the DNA sequence.

Given a DNA sequence **D**, the feature vector of **D** is defined:

$$\mathbf{D} = [d_1 d_2 \dots d_{16} d_{16+1} \dots d_{16+\lambda}]^T$$

where

$$d_k = \begin{cases} \dfrac{f_k}{\sum_{i=1}^{16} f_i + w \sum_{j=1}^{\lambda} \theta_j} & 1 \le k \le 16 \\ \dfrac{w\theta_{k-16}}{\sum_{i=1}^{16} f_i + w \sum_{j=1}^{\lambda} \theta_j} & 17 \le k \le 16+\lambda \end{cases}$$

where f_k (k = 1, 2, ..., 16) is the normalized occurrence frequency of dinucleotide in the DNA sequence; the parameter λ is an integer, representing the highest counted rank(or tier) of the correlation along a DNA sequence; w is the weight factor ranged from 0 to 1; θ_j (j = 1, 2, ..., λ) is called the j-tier correlation factor that reflects the sequence order correlation between all the most contiguous dinucleotide along a DNA sequence, which is defined:

$$\begin{cases} \theta_1 = \dfrac{1}{L-2} \sum_{i=1}^{L-2} \Theta(R_i R_{i+1}, R_{i+1} R_{i+2}) \\ \theta_2 = \dfrac{1}{L-3} \sum_{i=1}^{L-3} \Theta(R_i R_{i+1}, R_{i+2} R_{i+3}) \\ \theta_3 = \dfrac{1}{L-4} \sum_{i=1}^{L-4} \Theta(R_i R_{i+1}, R_{i+3} R_{i+4}) \\ \dots \\ \theta_1 = \dfrac{1}{L-1-\lambda} \sum_{i=1}^{L-1-\lambda} \Theta(R_i R_{i+1}, R_{i+\lambda} R_{i+\lambda+1}) \end{cases} \quad \lambda < L$$

where the correlation function is given by

$$\Theta(R_iR_{i+1}, R_jR_{j+1}) = \frac{1}{u}\sum_{u=1}^{u}[P_u(R_iR_{i+1}) - P_u(R_jR_{j+1})]^2$$

where μ is the number of physicochemical indices, in this study, 6 indices reflecting the local DNA structural properties were employed to generate the PseDNCfeature vector;$P_u(R_iR_{i+1})$ represents the numerical value of the u-th (u = 1, 2, ..., μ) physicochemical index of the dinucleotide R_iR_{i+1} at position i and $P_u(R_jR_{j+1})$ represents the corresponding value of the dinucleotide R_jR_{j+1} at position j. For more information about this approach, please refer to (Chen *et al.* 2013).

```
extrPseDNC(x)

##       Xc1.AA       Xc1.AC       Xc1.AG       Xc1.AT       Xc1.CA
##        0.044        0.044        0.029        0.029        0.044
##       Xc1.CC       Xc1.CG       Xc1.CT       Xc1.GA       Xc1.GC
##        0.087        0.073        0.087        0.029        0.102
##       Xc1.GG       Xc1.GT       Xc1.TA       Xc1.TC       Xc1.TG
##        0.044        0.029        0.015        0.073        0.058
##       Xc1.TT Xc2.lambda.1 Xc2.lambda.2 Xc2.lambda.3
##        0.029        0.075        0.047        0.065

phyche_index = data.frame(cust1 = c(1.019, -0.918, 0.488, 0.567, 0.567,
-0.070, -0.579, 0.488, - 0.654, -2.455,-0.070, -0.918, 1.603, -0.654,
0.567, 1.019))
custompros = t(phyche_index)
colnames(custompros) = make_kmer_index(2, alphabet = "ACGT")
extrPseDNC(x, normalize = TRUE, custompros = custompros, lambda = 2,
w = 0.1)

##       Xc1.AA       Xc1.AC       Xc1.AG       Xc1.AT       Xc1.CA
##        0.038        0.038        0.025        0.025        0.038
##       Xc1.CC       Xc1.CG       Xc1.CT       Xc1.GA       Xc1.GC
##        0.076        0.063        0.076        0.025        0.088
##       Xc1.GG       Xc1.GT       Xc1.TA       Xc1.TC       Xc1.TG
##        0.038        0.025        0.013        0.063        0.051
##       Xc1.TT Xc2.lambda.1 Xc2.lambda.2
##        0.025        0.175        0.118
```

5.2. Pseudo k-tupler composition

PseKNC improved the PseDNC approach by incorporating k-tuple nucleotide composition. Given a DNA sequence **D**, the feature vector of **D** is defined:

$$\mathbf{D} = [d_1 d_2 \ldots d_{4^k} d_{4^k+1} \ldots d_{4^k+\lambda}]^T$$

$$d_u = \begin{cases} \dfrac{f_u}{\sum_{i=1}^{4^k} f_i + w\sum_{j=1}^{\lambda}\theta_j} & 1 \le u \le 4^k \\ \dfrac{w\theta_{u-4^k}}{\sum_{i=1}^{4^k} f_i + w\sum_{j=1}^{\lambda}\theta_j} & 4^k \le u \le 4^k + \lambda \end{cases}$$

where λ is the number of the total counted ranks (or tiers) of the correlations along a DNA sequence; f_u (u = 1, 2, ..., 4^k) is the frequency of oligonucleotide that is normalized to $\sum_{i=1}^{4^k} f_i = 1$; w is a weight factor; θ_j is given by:

$$\theta_j = \frac{1}{L-j-1}\sum_{i=1}^{L-j-1}\Theta(R_iR_{i_1}, R_{i+j}R_{i+j+1}) \qquad j = 1, 2, \ldots, \lambda; \lambda < L$$

which represents the j-tier structural correlation factor between all the j^{th} most contiguous dinucleotides. The correlation function $\Theta(R_iR_{i_1}, R_{i+j}R_{i+j+1})$ is defined by

$$\Theta(R_iR_{i+1}, R_{i+j}R_{i+j+1}) = \frac{1}{u}\sum_{v=1}^{u}[P_v(R_iR_{i+1}) - P_v(R_{i+j}R_{i+j+1})]^2$$

where μ is the number of physicochemical indices, in this study, 6 indices reflecting the local DNA structural properties were employed to generate the PseKNC feature vector; $P_v(R_iR_{i+1})$ represents the numerical value of the v-th (u = 1, 2, ..., μ) physicochemical indices for the dinucleotide R_iR_{i+1} at position i and $P_v(R_{i+j}R_{i+j+1})$ represents the corresponding value for the dinucleotide $R_{i+j}R_{i+j+1}$ atposition $i+j$. For more information about this approach, please refer to (Guo *et al.* 2014).

```
extrPseKNC(x)
```

```
##      Xc1.AAA      Xc1.AAC      Xc1.AAG      Xc1.AAT      Xc1.ACA
##        0.008        0.000        0.008        0.008        0.008
##      Xc1.ACC      Xc1.ACG      Xc1.ACT      Xc1.AGA      Xc1.AGC
##        0.000        0.000        0.017        0.000        0.017
##      Xc1.AGG      Xc1.AGT      Xc1.ATA      Xc1.ATC      Xc1.ATG
##        0.000        0.000        0.000        0.000        0.008
##      Xc1.ATT      Xc1.CAA      Xc1.CAC      Xc1.CAG      Xc1.CAT
##        0.008        0.008        0.008        0.000        0.008
##      Xc1.CCA      Xc1.CCC      Xc1.CCG      Xc1.CCT      Xc1.CGA
##        0.000        0.008        0.025        0.017        0.008
##      Xc1.CGC      Xc1.CGG      Xc1.CGT      Xc1.CTA      Xc1.CTC
##        0.008        0.008        0.017        0.000        0.017
##      Xc1.CTG      Xc1.CTT      Xc1.GAA      Xc1.GAC      Xc1.GAG
##        0.025        0.008        0.000        0.008        0.008
##      Xc1.GAT      Xc1.GCA      Xc1.GCC      Xc1.GCG      Xc1.GCT
##        0.000        0.008        0.034        0.008        0.008
##      Xc1.GGA      Xc1.GGC      Xc1.GGG      Xc1.GGT      Xc1.GTA
##        0.008        0.008        0.008        0.000        0.008
```

```
##      Xc1.GTC      Xc1.GTG      Xc1.GTT      Xc1.TAA      Xc1.TAC
##        0.008        0.000        0.000        0.000        0.008
##      Xc1.TAG      Xc1.TAT      Xc1.TCA      Xc1.TCC      Xc1.TCG
##        0.000        0.000        0.008        0.008        0.008
##      Xc1.TCT      Xc1.TGA      Xc1.TGC      Xc1.TGG      Xc1.TGT
##        0.008        0.000        0.025        0.008        0.000
##      Xc1.TTA      Xc1.TTC      Xc1.TTG      Xc1.TTT Xc2.lambda.1
##        0.000        0.017        0.000        0.000        0.533
```

```
phyche_index = data.frame(cust1 = c(1.019, -0.918, 0.488, 0.567, 0.567,
-0.070, -0.579, 0.488, - 0.654, -2.455,-0.070, -0.918, 1.603, -0.654,
0.567, 1.019))
custompropS = t(phyche_index)
colnames(custompropS) = make_kmer_index(2, alphabet = "ACGT")
extrPseKNC(x, normalize = TRUE, custompropS = custompropS, lambda = 1,
w = 0.05, k = 2)
```

```
##       Xc1.AA       Xc1.AC       Xc1.AG       Xc1.AT       Xc1.CA
##        0.048        0.048        0.032        0.032        0.048
##       Xc1.CC       Xc1.CG       Xc1.CT       Xc1.GA       Xc1.GC
##        0.096        0.080        0.096        0.032        0.112
##       Xc1.GG       Xc1.GT       Xc1.TA       Xc1.TC       Xc1.TG
##        0.048        0.032        0.016        0.080        0.064
##       Xc1.TT Xc2.lambda.1
##        0.032        0.104
```

6. Similarity Calculation by Sequence Alignment

Similarity computation derived by local or global DNA sequence alignment between a list of DNA sequences is great need in the DNA related research and applications. However, this sort of pairwise similarity computation often computationally intensive, especially when there exists many DNA sequences. Luckily, this process is also highly parallelizable, the **rDNAse** package integrates the function of parallelized similarity computation derived by local or global DNA sequence alignment between a list of DNA sequences.

The function `twoSeqSim()` calculates the alignment result between two DNA sequences, and the function `parSeqSim()` calculates the pairwise similarity calculation with a list of DNA sequences in parallel:

```
> s1 = readFASTA(system.file('dnaseq/P00750.fasta', package = 'rDNAse'))[[1]]
> s2 = readFASTA(system.file('dnaseq/P08218.fasta', package = 'rDNAse'))[[2]]
> s3 = readFASTA(system.file('dnaseq/P10323.fasta', package = 'rDNAse'))[[3]]
> s4 = readFASTA(system.file('dnaseq/P20160.fasta', package = 'rDNAse'))[[4]]
> s5 = readFASTA(system.file('dnaseq/Q9NZP8.fasta', package = 'rDNAse'))[[5]]
> plist   = list(s1, s2, s3, s4, s5)
```

```
> psimmat = parSeqSim(plist, cores = 4, type = 'local', submat = 'BLOSUM62')
> print(psimmat)
```

```
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.0000000 0.3330758 0.2763540 0.3044898 0.3408780
[2,] 0.3330758 1.0000000 0.3321031 0.3272556 0.3134858
[3,] 0.2763540 0.3321031 1.0000000 0.2962784 0.3000359
[4,] 0.3044898 0.3272556 0.2962784 1.0000000 0.2944940
[5,] 0.3408780 0.3134858 0.3000359 0.2944940 1.0000000
```

It should be noted that for a small number of DNAs, calculating their pairwise similarity scores derived by sequence alignment in parallel may not significantly reduce the overall computation time, since each of the task only requires a relatively small time to finish, thus, computational overheads may exist and affect the performance. In testing, we used about 1,000 DNA sequences on 64 CPU cores, and observed significant performance improvement comparing to the sequential computation.

Users should install the packages **foreach** and **doParallel** before using `parSeqSim()`, according to their operation system. The **rDNAse** package will automatically decide which backend to use.

7. Similarity Calculation by GO Semantic Similarity Measures

The **rDNAse** package also integrates the function of similarity score computation derived by Gene Ontology (GO) semantic similarity measures between a list of GO terms / Entrez Gene IDs.

The function `twoGOSim()` calculates the similarity derived by GO-terms semantic similarity measures between two GO terms / Entrez Gene IDs, and the function `parGOSim()` calculates the pairwise similarity with a list of GO terms / Entrez Gene IDs:

```
# 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)
```

```
      [,1]  [,2]  [,3]
[1,] 1.000 0.077 0.055
[2,] 0.077 1.000 0.220
[3,] 0.055 0.220 1.000
```

```
# by Entrez gene id
> genelist = list(c('150', '151', '152', '1814', '1815', '1816'))
```

```
> gsimmat2 = parGOSim(genelist, type = 'gene')
> print(gsimmat2)

       150   151   152  1814  1815  1816
150  0.689 0.335 0.487 0.133 0.169 0.160
151  0.335 0.605 0.441 0.171 0.198 0.274
152  0.487 0.441 0.591 0.151 0.178 0.198
1814 0.133 0.171 0.151 0.512 0.401 0.411
1815 0.169 0.198 0.178 0.401 0.619 0.481
1816 0.160 0.274 0.198 0.411 0.481 0.819
```

8. Miscellaneous Tools

In this section, we will briefly introduce some useful tools provided by the **rDNAse** package.

8.1. Retrieve DNA Sequences from UniProt

This function `getGenbank()` gets DNA sequences from Genbank by GI ID(s). The input `ID` is a character vector specifying the GI ID(s). The returned sequences are stored in a list:

```
> ids = c('2', '392893239', '357180087')
> dnas = getGenbank(ids)
> print(dnas)

$gi_2
[1] "AATTCATGCGTCCGGACTTCTGCCTCGAGCCGCCGTACACTGGGCCCTGCAAAGCTCGTATCATCCGTTACT
TCTACAATGCAAAGGCAGGCCTGTGTCAGACCTTCGTATACGGCGGTTGCCGTGCTAAGCGTAACAACTTCAAATCC
GCGGAAGACTGCGAACGTACTTGCGGTGGTCCTTAGTAAAGCTTG"

$gi_392893239
[1] "GCCTAAAGACGACCGCGACGCGGCCGCTCGCACTCATAGACTACGCTAGTGGTGAGATACGCAGAGAAAAAG
ACGAGAGAGTATTGAGAGAATGGAGACATCACTACATCTAACATAGGGTCGCCAGTCGTCACCGAATTATTGGATTC
AAATTTAGGTCCC"

$gi_357180087
[1] "TTAAAGGGGACGTGGCCATTGGCCAGCTCACAGGCCGTGATTCCCACACTGTAGATGTCAGACTTGGCATCG
TAACCCTGGAGATTCTGCTGGAGGACCTCTGGGCTGAGCCAGGGCAGAACCTTGATACTGTACTTGGGGAAGTCGTG
GACCACACGCTGCCGCTGCCCGTGGCTGATCATGCTGAGGTTGCTTCGTAAACCTGACAGGTAGACCTTCCCATCAG
CGGAGATCAGAACGTGGCTGGCTTTGACGCTCCTGTGCACGTAGCCCATGTGGTGGATGTAATCCAAGGCCTTGAGC
GCCCCCTGCAGGATGTAAGCGATGGCCAGCTCGCTCATGCCGTCCATGAAGTGCGTGCAGATGAGGTCCTTGGCCGA
GCCGTATGCCATGAACGACGTGACGACCCACAGCTCGTTGTCCGCGATGAAGGTGGCTCCGTAGGGCAGGATGTTGG
GGTGGCTGAAGAGTTTAGAGACATGGAGTTCCCCCTGTAAGAACGTCACCATCTCGTTGGAGCAAGCCTCGAGGTTA
ATCCTTCGTACCGTCACATACTCCCCTGTTGGTTTGTACCTTGCTAGATTCACTGTCATCAGGTCCTCGAATCCTTT
GCCTATAACGGTGAGCAGCTCATAACACCCTCCCTCTGGCAGGAAGCTACTCATGATCTCCGGCTTAGAGAAGGGTG
CTATCGACTCTGAGCTCGCCTCATTGGTAAGAAAAGACATGAGTTCCTAACGTGTAGACCTACTTCAATTTCAAAAA
TTTAAACCTGGTCCTGGAGTTAGCAAGCAGTCCCGGCCCAGTCTCAGCTCCGCCTCCTTGGGTTAAGGCCCG"
```

8.2. Read FASTA Format files

The `readFASTA()` function provides a convenient way to read DNA sequences stored in FASTA format files. See `?readFASTA` for details. The returned sequences are stored in a named list, whose components are named with the DNA sequences' names.

8.3. Sanity Check of the Deoxyribonucleic Acid Types

The `dnacheck()` function checks if the DNA sequence's deoxyribonucleic acids types are in the 4 default types, which returns a `TRUE` if all the deoxyribonucleic acids in the sequence belongs to the 4 default types:

```
x = readFASTA(system.file('dnaseq/hs.fasta', package = 'rDNAse'))[[1]]
# A real sequence
dnacheck(x)

## [1] TRUE

# An artificial sequence
dnacheck(paste(x, 'Z', sep = ''))

## [1] FALSE
```

9. Summary

The summary of the descriptors in the **rDNAse** package is listed in table 1.

Table 1: List of commonly used descriptors in **rDNAse**

Descriptor Group	Descriptor Name	Descriptor Dimension	Function Name
Nucleic acid composition	Basic kmer	16^1	`kmer()`
	Reverse compliment kmer	10^1	`kmer()`
	Increment of diversity	12^2	`make_idkmer_vec()`
Autocorrelation	Dinucleotide-based auto covariance	4^3	`extrDAC()`
	Dinucleotide-based cross covariance	4^3	`extrDCC()`
	Dinucleotide-based auto-cross covariance	8^3	`extrDACC()`
	Trinucleotide-based auto covariance	4^3	`extrTAC()`
	Trinucleotide-based cross covariance	4^3	`extrTCC()`
	Trinucleotide-based auto-cross covariance	8^3	`extrTACC()`
Pseudo nucleotide composition	Pseudo dinucleotide composition	19^4	`extrPseDNC()`
	Pseudo k-tupler nucleotide composition	65^5	`extrPseKNC()`
Similarity calculation	Sequence alignment		`parSeqSim()`
	GO semantic similarity measures		`parGOSim()`

The summary of the names of the 38 physicochemical indices for dinucleotides. in the **rDNAse** package is listed in table 2.

Table 2: The names of the 38 physicochemical indices for dinucleotides

Base stacking	DNA induced deformability	B-DNA twist
Propeller twist	Propeller twist	Duplex stability: (freeenergy)
DNA denaturation	Bending stiffness	DNA DNA twist
Aida_BA_transition	Breslauer_dG	Breslauer_dH
Electron_interaction	Hartman_trans_free_energy	Helix-Coil_transition
Lisser_BZ_transition	Polar_interaction	SantaLucia_dG
SantaLucia_dS	Sarai_flexibility	Stability
Sugimoto_dG	Sugimoto_dH	Sugimoto_dS
Duplex tability (disruptenergy)	Stabilising energy of Z-DNA	Breslauer_dS
Ivanov_BA_transition	SantaLucia_dH	Stacking_energy
Watson-Crick_interaction	Dinucleotide GC Content	Twist
Tilt	Roll	Shift
Slide	Rose	

The summary of the names of the 12 physicochemical indices for trinucleotides. in the **rDNAse** package is listed in table 3.

Table 3: The names of the 12 physicochemical indices for trinucleotides.

Bendability (DNAse)	Bendability (consensus)	Trinucleotide GC Content
Consensus_roll	Consensus-Rigid	Dnase I
MW-Daltons	MW-kg	Nucleosome
Nucleosome positioning	Dnase I-Rigid	Nucleosome-Rigid

The summary of the names of the 6 physicochemical indices for dinucleotides. in the **rDNAse** package is listed in table 4.

Table 4: The names of the 6 physicochemical indices for dinucleotides.

Twist	Tilt	Roll
Shift	Slide	Rise

In this manual, we discussed the functions of the **rDNAse** package, which is trying to offer a comprehensive and unique toolkit for DNA sequence descriptor calculation and similarity computation.

Acknowledgments

This work is financially supported by the National key basic research program (2015CB910700), the National Natural Science Foundation of China (Grants No. 81402853), grants from the

[1]The number depends on the choice of the k value of kmer. The default is `k` = 2.

[2]The number depends on the choice of the k value of kmer. The default is `k` = 6.

[3]The number depends on the maximum value of `lag`. By default `lag` = 2. And the number of the physicochemical indices, By default the length of `index` = 2.

[4]The number depends on the maximum value of `lambda`. By default `lambda` = 3.

[5]The number depends on the maximum value of `lambda`. By default `lambda` = 1.

Project of Innovation-driven Plan in Central South University, and the Postdoctoral Science Foundation of Central South University, the Chinese Postdoctoral Science Foundation (2014T70794, 2014M562142). The studies meet with the approval of the university's review board.

References

Chen W, Feng PM, Lin H, Chou KC (2013). "iRSpot-PseDNC: identify recombination spots with pseudo dinucleotide composition." *Nucleic acids research*, p. gks1450.

Chen W, Lei TY, Jin DC, Lin H, Chou KC (2014). "PseKNC: a flexible web server for generating pseudo K-tuple nucleotide composition." *Analytical biochemistry*, **456**, 53–60.

Chen W, Luo L, Zhang L (2010). "The organization of nucleosomes around splice sites." *Nucleic acids research*, **38**(9), 2788–2798.

Dong Q, Zhou S, Guan J (2009). "A new taxonomy-based protein fold recognition approach based on autocross-covariance transformation." *Bioinformatics*, **25**(20), 2655–2662.

Guo SH, Deng EZ, Xu LQ, Ding H, Lin H, Chen W, Chou KC (2014). "iNuc-PseKNC: a sequence-based predictor for predicting nucleosome positioning in genomes with pseudo k-tuple nucleotide composition." *Bioinformatics*, p. btu083.

Gupta S, Dennis J, Thurman RE, Kingston R, Stamatoyannopoulos JA, Noble WS (2008). "Predicting human nucleosome occupancy from primary sequence." *PLoS Comput Biol*, **4**(8), e1000134.

Lee D, Karchin R, Beer MA (2011). "Discriminative prediction of mammalian enhancers from DNA sequence." *Genome research*, **21**(12), 2167–2180.

Liu G, Liu J, Cui X, Cai L (2012). "Sequence-dependent prediction of recombination hotspots in Saccharomyces cerevisiae." *Journal of theoretical biology*, **293**, 49–54.

Lu J, Luo L (2008). "Prediction for human transcription start site using diversity measure with quadratic discriminant." *Bioinformation*, **2**(7), 316–321.

Noble WS, Kuehn S, Thurman R, Yu M, Stamatoyannopoulos J (2005). "Predicting the in vivo signature of human gene regulatory sequences." *Bioinformatics*, **21**(suppl 1), i338–i343.

Zhang L, Luo L (2003). "Splice site prediction with quadratic discriminant analysis using diversity measure." *Nucleic Acids Research*, **31**(21), 6214–6220.

Affiliation:

Min-feng Zhu
the third xiangya hospital
School of Pharmaceutical Sciences
Central South University
Changsha, Hunan, P. R. China
E-mail: wind2zhu@163.com

Jie Dong
School of Pharmaceutical Sciences
Central South University
Changsha, Hunan, P. R. China
E-mail: biomed@csu.edu.cn

Dong-Sheng Cao
School of Pharmaceutical Sciences
Central South University
Changsha, Hunan, P. R. China
E-mail: oriental-cds@163.com