Package “repDNA”

Name: repDNA

Type: Python Package

Version: 1.1.1

Date: 2014-11-04

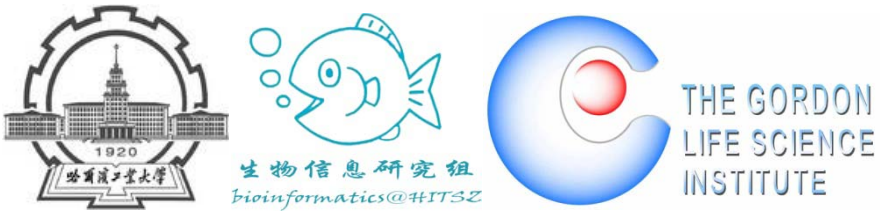
Description: a Python package to generate various modes of feature vectors for DNA sequences by incorporating user-defined physicochemical properties and sequence-order effects

Home-page: http://bioinformatics.hitsz.edu.cn/repDNA/

License: GPL

Download-URL: http://bioinformatics.hitsz.edu.cn/repDNA/download

Platform: MS Windows, Mac OS, Unix/Linux



Contents

1. Introduction

1.1 repDNA description

In the field of bioinformatics, more and more computational methods employed well-known machine learning techniques to construct their predictors, such as Support Vector Machines (SVMs), Random Forest (RF), Artificial Neural Networks (ANN), etc. These methods require fixed length feature vectors as inputs. A few web servers or programs for computing DNA features considering the sequence-order effects or structural information have been developed (Chen, et al., 2014). However, they are not comprehensive and can only be limited to a certain kind of features. Furthermore, most of them are not freely accessible, and the user-defined physicochemical properties cannot be used to compute the features. To facilitate the studies of DNA and nucleic acids, repDNA was proposed, which is a freely available Python package to generate various features of DNA sequences.

1.2 Various DNA representations in repDNA

The repDNA computes three feature groups composed of 15 different features in 3 categories, including:

Basic function

- Read DNA sequence data from FASTA files
- Normalization of physicochemical index

Nucleic acid composition

- Basic kmer
- Reverse compliment kmer
- Increment of diversity (ID)

Autocorrelation

- 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)
- Trinucleotide-based auto-cross covariance (TACC)

Pseudo nucleotide composition

- Pseudo dinucleotide composition (PseDNC)
- Pseudo k-tupler nucleotide composition (PseKNC)

- Parallel correlation pseudo dinucleotide composition (PC-PseDNC)
- Parallel correlation pseudo trinucleotide composition (PC-PseTNC)
- Series correlation pseudo dinucleotide composition (SC-PseDNC)
- Series correlation pseudo trinucleotide composition (SC-PseTNC)

1.3 The structure of repDNA package

There are four modules in repDNA package, including util, nac, ac and psenac. The structure of repDNA is shown in **Fig. 1**.

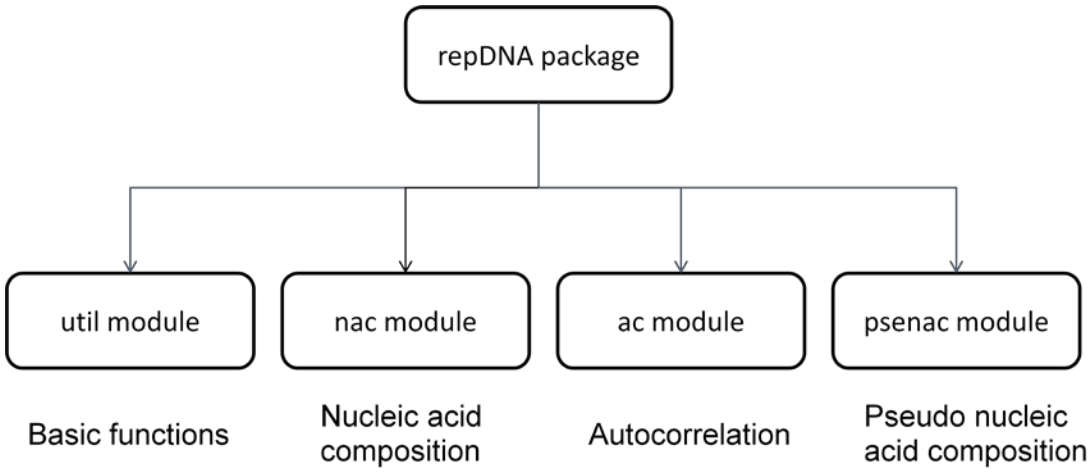


Fig. 1. The structure of repDNA Python package. util module implements some basic functions, such as reading DNA sequences, and normalizing the physicochemical indices provided by users. nac, ac and psenac modules are the implementations of the three feature groups: nucleic acid composition, autocorrelation and pseudo nucleotide composition, respectively.

1.4 Installing repDNA

1.4.1 Requirements

The repDNA package was tested to work under Python 2.7, 3.3 and 3.4 on Windows, Mac and Linux.

1.4.2 Install

If the user has installed the **pip** (a tool for installing and managing Python packages), just type:

```
pip install repDNA
```

For Windows:

To install repDNA on Windows OS, download the executable file, and run it to install this package.

The repDNA package can be also installed from the source code:

Download and unzip the repDNA package, go to the directory, and type:

```
python setup.py install
```

For Unix, Linux and Mac OS:

To install repDNA, download and unzip the repDNA package, go to the directory, and type:

```
sudo python setup.py install
```

2. Basic function

Before generating the DNA feature vectors by using the modules in the repDNA Python package, the DNA sequence data should be processed. For example, the DNA sequences should be read from files in FASTA format, and the physicochemical indices should be normalized.

2.1 Read sequence data from FASTA files

Function get_data in module util is used to process the original data.

Function

get_data(*input_data*[, *desc*])

This function returns DNA sequence data. It reads DNA sequences from an input file in FASTA format provided by the users. It will check whether the DNA sequences are legal or not, if illegal, the corresponding error messages will be shown on the screen. **Note** that all the nucleic acids in lowercases will be automatically converted into uppercases.

The parameters in get_data include:

- *input_data*: the input data, which should be a list or file type.
- *desc*: with this option, this function will return a Seq object, containing the description information, such as sequence name, sequence length, etc. Its default value is **False**.

Examples

Read the DNA data from a FASTA file named "example.fasta" containing one DNA sequence.

```
>>> from repDNA.util import get_data
>>> get_data(open('example.fasta'))
['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC']
>>> get_data(['GACTGAACTGCACTTTGGTTTCATATTATTTGctc'])
['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC']
>>> get_data(['GACTGAACTGCACTTTGGTTTCATATTATTTGctc1'])
Sorry, sequence 1 has illegal character 1.(The character must be A, C,
G or T)

>>> from repDNA.util import get_data
>>> seqs = get_data(open('example.fasta'), desc=True)
>>> for seq in seqs:
...     print seq
...
misc_ppid_3700  No:1   length:35
GACTGAACTGCACTTTGGTTTCATATTATTTGCTC
>>> seq = seqs[0]
>>> seq.name
'misc_ppid_3700'
>>> seq.seq
'GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'
>>> seq.no
1
```

2.2 Normalization of physicochemical index

Function normalize_index in module util is used to normalize the physicochemical index according to:

$$H_{\xi}(R_1R_2\cdots R_K)=\frac{H_{\xi}^0(R_1R_2\cdots R_K)-<H_{\xi}^0(R_1R_2\cdots R_K)>}{SD<H_{\xi}^0(R_1R_2\cdots R_K)>} \quad (1)$$

where the symbol $<\bullet>$ means taking the average of the quantity therein over the 4^K different combinations of A, C, G, and T for $R_1R_2\cdots R_K$, and SD is the corresponding standard deviation. $H_{\xi}^0(R_1R_2\cdots R_K)(\xi=1,2,\cdots,\Lambda)$ are the original physicochemical property values for the oligonucleotides, Λ is the number of physicochemical properties. For more information, please refer to (Chen, et al., 2014).

Function

normalize_index(*phyche_index*[, *is_convert_dict*])

The function returns the normalized physicochemical index.

The parameters in normalize_index include:

- *phyche_index*: the original physicochemical indices, whose type is a list. Each element in it is also a list, representing an individual physicochemical index. The property values for each index are sorted in the alphabet order (A, C, G, T, AA, AC... TT ...).
- *is_convert_dict*: with this option, this function will convert the normalized physicochemical index to a dict, the key is oligonucleotide (string), and its corresponding value is the list. The default value is **False**.

Examples

```
>>> from repDNA.util import normalize_index
>>> phyche_index = [
... [0.026, 0.036, 0.031, 0.033, 0.016, 0.026, 0.014, 0.031, 0.025,
0.025, 0.026, 0.036, 0.017, 0.025, 0.016, 0.026],
... [0.038, 0.038, 0.037, 0.036, 0.025, 0.042, 0.026, 0.037, 0.038,
0.036, 0.042, 0.038, 0.018, 0.038, 0.025, 0.038]]
>>> normalize_index(phyche_index)
[[0.06, 1.5, 0.78, 1.07, -1.38, 0.06, -1.66, 0.78, -0.08, -0.08, 0.06,
1.5, -1.23, -0.08, -1.38, 0.06], [0.5, 0.5, 0.36, 0.22, -1.36, 1.08,
-1.22, 0.36, 0.5, 0.22, 1.08, 0.5, -2.37, 0.5, -1.36, 0.5]]
```

3. Nucleic acid composition

The most straight forward approach to represent the DNA sequences is based on their nucleic acid composition. The kmer and its variants have been widely used for this aim. Module nac (nac is the abbreviation of nucleic acid composition) in repDNA allows users to generate various kinds of kmer-based feature vectors for given sequences or FASTA files by selecting different methods and parameters. This module aims at computing three types of nucleic acid composition, including basic kmer, reverse compliment kmer and increment of diversity. Let's introduce them one by one.

3.1 Basic 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.

In order to generate the basic kmer feature vector, it is required to import the Kmer class, construct a Kmer object and use the method make_kmer_vec.

Class

Kmer(*k*[, *normalize*, *upto*])

The Kmer class is the implementation of the basic kmer.

The attributes of Kmer class are:

- *k*: the *k* value of kmer, it should be an integer larger than 0.
- *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**.
- *upto*: with this option, this class function make_kmer_vec will 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**.

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.

Method

make_kmer_vec(*input_data*)

This method returns the basic kmer feature vector, which is in the class Kmer.

The parameter of make_kmer_vec method includes:

- *input_data*: the input data, which should be a list or file type.

Examples

```
>>> from repDNA.nac import Kmer
>>> kmer = Kmer(k=2)
>>> kmer.make_kmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
[[1, 3, 0, 3, 2, 0, 0, 4, 2, 2, 1, 1, 2, 2, 4, 7]]
>>> kmer = Kmer(k=2, upto=True)
>>> kmer.make_kmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
[[7, 7, 6, 15, 1, 3, 0, 3, 2, 0, 0, 4, 2, 2, 1, 1, 2, 2, 4, 7]]
>>> kmer = Kmer(k=2, normalize=True)
>>> kmer.make_kmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
[[0.029, 0.088, 0.0, 0.088, 0.059, 0.0, 0.0, 0.118, 0.059, 0.059,
0.029, 0.029, 0.059, 0.059, 0.118, 0.206]]
>>> kmer = Kmer(k=2, normalize=True, upto=True)
>>> kmer.make_kmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
```

```
[[0.2, 0.2, 0.171, 0.429, 0.029, 0.088, 0.0, 0.088, 0.059, 0.0, 0.0,
0.118, 0.059, 0.059, 0.029, 0.029, 0.059, 0.059, 0.118, 0.206]]
```

3.2 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)

In order to generate the reverse compliment kmer feature vector, we need to import the RevcKmer class, construct a RevcKmer object and use the method make_revckmer_vec.

Class

RevcKmer(*k*)

The RevcKmer class is the implementation of the reverse compliment kmer.

The attributes of RevcKmer class are:

- *k*: the *k* value of kmer, it should be an integer larger than 0.
- *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 vector represent the frequencies of kmers. The default value of this parameter is **False**.
- *upto*: with this option, method make_revckmer_vec will generate the kmers: 1mer, 2mer, …, kmer. The feature vector is the combination of all these kmers. The default value of this parameter is **False**.

Note: if the parameters *normalize* and *upto* are both **True**, 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.

Method

make_revckmer_vec(*input_data*)

This method returns the reverse compliment kmer feature vector and it is in the class RevcKmer.

The parameter of make_revckmer_vec method includes:

- *input_data*: the input data, which should be a list or file type.

Examples

```
>>> from repDNA.nac import RevcKmer
>>> revckmer = RevcKmer(k=2)
>>>
revckmer.make_revckmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
[[8, 4, 4, 3, 6, 1, 0, 4, 2, 2]]
>>> revckmer = RevcKmer(k=2, upto=True)
>>>
revckmer.make_revckmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
[[22, 13, 8, 4, 4, 3, 6, 1, 0, 4, 2, 2]]
>>> revckmer = RevcKmer(k=2, normalize=True)
>>>
revckmer.make_revckmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
[[0.235, 0.118, 0.118, 0.088, 0.176, 0.029, 0.0, 0.118, 0.059, 0.059]]
>>> revckmer = RevcKmer(k=2, normalize=True, upto=True)
>>>
revckmer.make_revckmer_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
[[0.629, 0.371, 0.235, 0.118, 0.118, 0.088, 0.176, 0.029, 0.0, 0.118,
0.059, 0.059]]
```

3.3 Increment of diversity

The increment of diversity has been successfully applied in the prediction of exon-intron splice sites for several model genomes (Zhang and Luo, 2003), transcription start site prediction, and studying the organization of nucleosomes around splice sites (Lv 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*:

$$\mathrm{ID} = \mathrm{Diversity}(X + S) - \mathrm{Diversity}(S) - \mathrm{Diversity}(X) \quad (2)$$

Given a sequence *X* with *r* feature variables (ID_1 to ID_r), we obtain an *r*-dimensional feature vector $\boldsymbol{R} = (\mathrm{ID}_1, \mathrm{ID}_2, \cdots, \mathrm{ID}_r)$. The feature vector $\boldsymbol{R}$ 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.

For more information of this approach, please refer to (Chen, et al., 2010) and (Liu, et al., 2012).

In order to generate the increment of diversity feature vector, we need to import the IDkmer class, construct an IDkmer object and use the method make_idkmer_vec.

Class

IDkmer([*k*, *upto*])

The IDkmer class is the implementation of the increment of diversity.

The attributes of IDkmer class are:

- *k*: the *k* value of kmer, it should be an integer larger than 0, the default value is 6.
- *upto*: with this option, this class function make_idkmer_vec will generate all the kmers: 1mer, 2mer, …, kmer. The feature vector is the combination of all these kmers. The default value of this parameter is **True**.

Method

make_idkmer_vec(*input_data*, *pos_source*, *neg_source*)

This method returns the increment of diversity feature vector and it is in the class IDKmer.

The parameters of make_idkmer_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *pos_source*: the positive source data, which should be a list or file type.
- *neg_source*: the negative source data, which should be a list or file type.

Examples

```
>>> from repDNA.nac import IDkmer
>>> idkmer = IDkmer()
>>> idkmer.make_idkmer_vec(open('example.fasta'), open('pos.fasta'),
open('neg.fasta'))
[[4.54, 29.19, -103.245, -92.99, -144.589, -145.351, -154.0, -154.0,
-153.58, -153.58, -147.207, -147.207]]
>>> idkmer = IDkmer(k=2)
>>> idkmer.make_idkmer_vec(open('example.fasta'), open('pos.fasta'),
open('neg.fasta'))
[[4.54, 29.19, -103.245, -92.99]]
>>> idkmer = IDkmer(k=2, upto=False)
```

```
>>> idkmer.make_idkmer_vec(open('example.fasta'), open('pos.fasta'),
open('neg.fasta'))
[[16.288, 62.27]]
```

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 repDNA 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 L nucleic acid residues; i.e.

$$\mathbf{D} = \mathrm{R}_1\mathrm{R}_2\mathrm{R}_3\mathrm{R}_4\mathrm{R}_5\mathrm{R}_6\mathrm{R}_7 \cdots \mathrm{R}_L \tag{3}$$

where R_1 represents the nucleic acid residue at the sequence position 1, 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:

$$\mathrm{DAC}(u, lag) = \sum_{i=1}^{L-lag-1} (P_u(\mathrm{R}_i\mathrm{R}_{i+1}) - \overline{P}_u)(P_u(\mathrm{R}_{i+lag}\mathrm{R}_{i+lag+1}) - \overline{P}_u) / (L - lag - 1) \tag{4}$$

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

$$\overline{P_u} = \sum_{j=1}^{L-1} P_u(\mathrm{R}_j\mathrm{R}_{j+1}) / (L - 1) \tag{5}$$

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 protein fold recognition (Dong, et al., 2009).

In order to generate the DAC feature vector, we need to import the DAC class, construct a DAC object and use the method make_dac_vec.

Class

DAC(*lag*)

The DAC class is the implementation of the DAC.

The attribute of DAC class is:

- *lag*: 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.

Method

make_dac_vec(*input_data*[, *phyche_index*, *all_property*, *extra_phyche_index*])

This method returns the DAC feature vector and it is in the class DAC.

The parameters of make_dac_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *phyche_index*: the physicochemical indices, it should be a list type and there are 38 different physicochemical indices (**Table 1**), which the users can choose.
- *all_property*: with this option, all the 38 physicochemical indices will be employed to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, 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. Note that if the user-defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.ac import DAC
>>> dac = DAC(2)
```

```
>>> dac.make_dac_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'])
[[-0.175, -0.185, -0.173, -0.004]]
>>> vec = dac.make_dac_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> print len(vec[0])
76
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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],
... [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]]
>>> from repDNA.util import normalize_index
>>> dac.make_dac_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
[[-0.175, -0.185, -0.5, -0.504, -0.173, -0.004, 0.009, 0.106]]
```

4.2 Dinucleotide-based cross covariance

Given a DNA sequence **D** (**Eq. 3**), 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:

$$\mathrm{DCC}(u_1,u_2,lag) = \sum_{i=1}^{L-lag-1} (P_{u_1}(R_iR_{i+1}) - \overline{P}_{u_1})(P_{u_2}(R_{i+lag}R_{i+lag+1}) - \overline{P}_{u_2}) / (L - lag - 1) \qquad (6)$$

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

$$\overline{P_u} = \sum_{j=1}^{L-1} P_u(\mathrm{R}_j\mathrm{R}_{j+1}) / (L-1) \qquad (7)$$

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 protein fold recognition (Dong, et al., 2009).

In order to generate the DCC feature vector, we need to import the DCC class, construct a DCC object, and use the method make_dcc_vec.

Class

DCC(*lag*)

The DCC class is the implementation of the DCC.

The attribute of DCC class is:

- *lag*: 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.

Method

make_dcc_vec(*input_data*[, *phyche_list*, *all_property*, *extra_phyche_index*]**)**

This method returns the DCC feature vector and it is in the class DCC.

The parameters of make_dcc_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *phyche_index*: the physicochemical indices, it should be a list and there are 38 different physicochemical indices (**Table 1**), which the users can choose.
- *all_property*: with this option, all the 38 physicochemical indices will be employed to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, 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. **Note** that if the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.ac import DCC
>>> dcc = DCC(2)
>>> dcc.make_dcc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'])
[[-0.141, -0.238, -0.064, -0.047]]
>>> vec = dcc.make_dcc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> print len(vec[0])
```

```
2812
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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],
... [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]]
>>> from repDNA.util import normalize_index
>>> dcc.make_dcc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
[[-0.141, -0.464, -0.535, -0.238, -0.521, -0.512, -0.18, -0.128, -
0.477, -0.097, -0.132, -0.403, -0.064, 0.027, 0.042, -0.047, 0.103,
0.142, -0.153, -0.216, -0.061, 0.041, -0.021, 0.162]]
```

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).

In order to generate the DACC feature vector, we need to import the DACC class, construct a DACC object and use the method make_dacc_vec.

Class

DACC(*lag*)

The DACC class is the implementation of the DACC.

The attribute of DACC class is:

- *lag*: 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 distance between two dinucleotides.

Method

make_dacc_vec(*input_data*[, *phyche_index*, *all_property*, *extra_phyche_index*]**)**

This method returns DACC feature vector and it is in the class DACC.

The parameters of make_dacc_vector method include:

- *input_data*: the input data, which should be a list or file type.

- *phyche_index*: the physicochemical indices, it should be a list and there are 38 different physicochemical indices (**Table 1**), which the users can choose.
- *all_property*: with this option, all the 38 physicochemical indices will be employed to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, 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. **Note** that if the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.ac import DACC
>>> dacc = DACC(2)
>>> dacc.make_dacc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'])
[[-0.175, -0.185, -0.173, -0.004, -0.141, -0.238, -0.064, -0.047]]
>>> vec = dacc.make_dacc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> print len(vec[0])
2888
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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],
... [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]]
>>> from repDNA.util import normalize_index
>>> dacc.make_dacc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
[[-0.175, -0.185, -0.5, -0.504, -0.173, -0.004, 0.009, 0.106, -0.141,
-0.464, -0.535, -0.238, -0.521, -0.512, -0.18, -0.128, -0.477, -0.097,
-0.132, -0.403, -0.064, 0.027, 0.042, -0.047, 0.103, 0.142, -0.153, -
0.216, -0.061, 0.041, -0.021, 0.162]]
```

4.4 Trinucleotide-based auto covariance

Given a DNA sequence **D** (**Eq. 3**), the TAC approach measures the correlation of the same physicochemical index between two trinucleotides separated by *lag* nucleic acids along the sequence, which can be calculated as:

$$\mathrm{TAC}(lag,u)=\sum_{i=1}^{L-lag-2}(P_u(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2})-\overline{P}_u)(P_u(\mathrm{R}_{i+lag}\mathrm{R}_{i+lag+1}\mathrm{R}_{i+lag+2})-\overline{P}_u)/(L-lag-2) \quad (8)$$

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

$$\overline{P}_u=\sum_{j=1}^{L-2}P_u(\mathrm{R}_j\mathrm{R}_{j+1}\mathrm{R}_{j+2})/(L-2) \quad (9)$$

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).
In order to generate the TAC feature vector, we need to import the TAC class, construct a TAC object and use the method make_tac_vec.

Class

TAC(*lag*)

The TAC class is the implementation of the TAC.

The attribute of TAC class is:

- *lag*: an integer larger than or equal to 0 and less than or equal to *L*-3 (*L* means the length of the shortest DNA sequence in the dataset). It represents the distance between two trinucleotides.

Method

make_tac_vec(*input_data*[, *phyche_index*, *all_property*, *extra_phyche_index*])

This method returns the TAC feature vector and it is in the class TAC.

The parameters of make_tac_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *phyche_index*: the physicochemical indices, it should be a list and there are 12 different physicochemical indices (**Table 2**), which the users can choose.
- *all_property*: with this option, all the 12 physicochemical indices will be employed to generate the feature vector. Its default value is **False**.

- *extra_phyche_index*: with this option, the users can use their own indices to generate the feature vector. It should be a dict, the key is trinucleotide (string), and its corresponding value is a list. Note that if the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.ac import TAC
>>> tac = TAC(2)
>>> tac.make_tac_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'])
[[-0.332, 0.493, 0.319, 0.012]]
>>> vec = tac.make_tac_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> print len(vec[0])
24
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [[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]]
>>> from repDNA.util import normalize_index
>>> tac.make_tac_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
[[-0.332, 0.493, 0.296, 0.319, 0.012, -0.308]]
```

4.5 Trinucleotide-based cross covariance

Given a DNA sequence **D** (**Eq. 3**), 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:

$$\mathrm{TCC}(u_1,u_2,\mathrm{lag}) = \sum_{i=1}^{L-lag-2} (P_{u_1}(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2}) - \overline{P}_{u_1})(P_{u_2}(\mathrm{R}_{i+lag}\mathrm{R}_{i+lag+1}\mathrm{R}_{i+lag+2}) - \overline{P}_{u_2}) / (L\text{-}lag\text{-}2) \quad (10)$$

where u_1, u_2 are two physicochemical indices, L is the length of the DNA sequence, $P_{u_1}(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2})$ ($P_{u_2}(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2})$) represents the numerical value of the

physicochemical index u_1 (u_2) for the trinucleotide $R_iR_{i+1}R_{i+2}$ at position i, $\overline{P}_{u_1}$ ($\overline{P}_{u_2}$) is the average value for physicochemical index value u_1 (u_2) along the whole sequence:

$$\overline{P_u} = \sum_{j=1}^{L-2} P_u(R_jR_{j+1}R_{j+2}) / (L-2) \quad (11)$$

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).

In order to generate the TCC feature vector, we need to import the TCC class, construct a TCC object and use the method make_tcc_vec.

Class

TCC(*lag*)

The DCC class is the implementation of the TCC.

The attribute of TCC class is:

- *lag*: an integer larger than or equal to 0 and less than or equal to L-3 (L means the length of the shortest sequence in the dataset). It represents the distance between two trinucleotides.

Method

make_tcc_vec(*input_data*[, *phyche_index*, *all_property*, *extra_phyche_index*])

This method returns the TCC feature vector and it is in the class TCC.

The parameters of make_tcc_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *Phyche_index*: the physicochemical indices, it should be a list and there are 12 different physicochemical indices (**Table 2**), which the users can choose.
- *all_property*: with this option, all the 12 physicochemical indices will be employed to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, the users can use their own indices to generate the feature vector. It should be a dict, the key is trinucleotide (string), and its corresponding value is a list. **Note** that if the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.ac import TCC
>>> tcc = TCC(2)
>>> tcc.make_tcc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'])
[[-0.299, -0.11, -0.24, 0.001]]
>>> vec = tcc.make_tcc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> len(vec[0])
264
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [[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]]
>>> from repDNA.util import normalize_index
>>> tcc.make_tcc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
[[-0.299, 0.356, -0.11, -0.155, -0.561, -0.307, -0.24, 0.199, 0.001,
0.003, 0.002, 0.257]]
```

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).

In order to generate the TACC feature vector, we need to import the TACC class, construct a TACC object and use the method make_tacc_vec.

Class

TACC(*lag*)

The TACC class is the implementation of the TACC.

The attribute of TACC class is:

- *lag*: an integer larger than or equal to 0 and less than or equal to *L*-3 (*L* means the length of the shortest DNA sequence in the dataset). It represents the distance between two trinucleotides.

Method

make_tacc_vec(*input_data*[, *phyche_index*, *all_property*, *extra_phyche_index*])

This method is in the class TACC, it returns the TACC feature vector.

The parameters of make_tacc_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *phyche_list*: the physicochemical indices, it should be a list and there are 12 different physicochemical indices (**Table 2**), which the users can choose.
- *all_property*: with this option, all the 12 physicochemical indices will be employed to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, the users can use their own indices to generate the feature vector. It should be a dict, the key is trinucleotide (string), and its corresponding value is a list. **Note** that if the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.ac import TACC
>>> tacc = TACC(2)
>>> tacc.make_tacc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'])
[[-0.332, 0.493, 0.319, 0.012, -0.299, -0.11, -0.24, 0.001]]
>>> vec = tacc.make_tacc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> print len(vec[0])
288
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [[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]]
>>> from repDNA.util import normalize_index
>>> tacc.make_tacc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', Nucleosome'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
[[-0.332, 0.493, 0.296, 0.319, 0.012, -0.308, -0.299, 0.356, -0.11, -
0.155, -0.561, -0.307, -0.24, 0.199, 0.001, 0.003, 0.002, 0.257]]
```

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** (**Eq. 3**), the feature vector of **D** is defined:

$$\mathbf{D} = [d_1 \; d_2 \; \cdots \; d_{16} \; d_{16+1} \; \cdots \; d_{16+\lambda}]^{\mathrm{T}} \tag{12}$$

where

$$d_k = \begin{cases} \dfrac{f_k}{\sum_{i=1}^{16} f_i + w\sum_{j=1}^{\lambda} \theta_j} (1 \le k \le 16) \\ \dfrac{w\theta_{k-16}}{\sum_{i=1}^{16} f_i + w\sum_{j=1}^{\lambda} \theta_j} (17 \le \mathrm{k} \le 16+\lambda) \end{cases} \tag{13}$$

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:

$$\begin{cases} \theta_1 = \dfrac{1}{L\text{-}2}\sum_{i=1}^{L-2}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+1}\mathrm{R}_{i+2}) \\ \theta_2 = \dfrac{1}{L\text{-}3}\sum_{i=1}^{L-3}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+2}\mathrm{R}_{i+3}) \\ \theta_3 = \dfrac{1}{L\text{-}4}\sum_{i=1}^{L-4}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+3}\mathrm{R}_{i+4}) \\ \cdots\cdots \\ \theta_\lambda = \dfrac{1}{L\text{-}1\text{-}\lambda}\sum_{i=1}^{L-1-\lambda}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+\lambda}\mathrm{R}_{i+\lambda+1}) \end{cases} \quad (\lambda < L) \tag{14}$$

where the correlation function is given by

$$\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_j\mathrm{R}_{j+1}) = \frac{1}{\mu}\sum_{u=1}^{\mu}[P_u(\mathrm{R}_i\mathrm{R}_{i+1}) - P_u(\mathrm{R}_j\mathrm{R}_{j+1})]^2 \tag{15}$$

where μ is the number of physicochemical indices, in this study, 6 indices reflecting the local DNA structural properties (**Table 3**) were employed to generate the PseDNC feature vector; $P_u(\mathrm{R}_i\mathrm{R}_{i+1})$ represents the numerical value of the *u*-th ($u = 1, 2, \cdots, \mu$) physicochemical index of the dinucleotide $\mathrm{R}_i\mathrm{R}_{i+1}$ at position *i* and $P_u(\mathrm{R}_j\mathrm{R}_{j+1})$ represents the corresponding value of the dinucleotide $\mathrm{R}_j\mathrm{R}_{j+1}$ at position *j*.

For more information about this approach, please refer to (Chen, et al., 2013).

In order to generate the PseDNC feature vector, we need to import the PseDNC class, construct a PseDNC object and use the method make_psednc_vec.

Class

PseDNC([*lamada*, *w*]**)**

The PseDNC class is the implementation of the PseDNC.

The attributes of PseDNC class are:

- *lamada*: 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.

Method

make_psednc_vec(*input_data*[, extra_phyche_index])

This method returns the PseDNC feature vector and it is in the class PseDNC.

The parameters of make_psednc_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *extra_phyche_index*: with this option, the user defined dinucleotide physiochemical indices can be used to generate the PseDNC feature vector. The type of this parameter should be a dict, in which the key is the dinucleotide (string), and its corresponding value is a list. **Note** that the PseDNC feature vector will be generated based on the original six physical structures and the user defined indices. If the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.psenac import PseDNC
>>> psednc = PseDNC()
>>> vec =
psednc.make_psednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
>>> vec
[[0.023, 0.069, 0.0, 0.069, 0.046, 0.0, 0.0, 0.092, 0.046, 0.046,
0.023, 0.023, 0.046, 0.046, 0.092, 0.16, 0.0823, 0.0705, 0.0694]]
>>> len(vec[0])
19
>>> psednc = PseDNC(lamada=2, w=0.1)
>>> vec =
psednc.make_psednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
>>> vec
[[0.021, 0.063, 0.0, 0.063, 0.042, 0.0, 0.0, 0.084, 0.042, 0.042,
0.021, 0.021, 0.042, 0.042, 0.084, 0.148, 0.152, 0.1301]]
>>> len(vec[0])
18
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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]]
>>> from repDNA.util import normalize_index
>>> vec =
psednc.make_psednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
>>> vec
[[0.021, 0.063, 0.0, 0.063, 0.042, 0.0, 0.0, 0.085, 0.042, 0.042,
```

```
0.021, 0.021, 0.042, 0.042, 0.085, 0.148, 0.1515, 0.1292]]
>>> len(vec[0])
18
```

5.2 Pseudo *k*-tupler composition

PseKNC improved the PseDNC approach by incorporating *k*-tuple nucleotide composition.

Given a DNA sequence **D** (**Eq. 3**), the feature vector of **D** is defined:

$$\mathbf{D} = [d_1\ d_2 \cdots d_{4^k}\ d_{4^k+1}\ \cdots\ d_{4^k+\lambda}]^{\mathrm{T}} \tag{16}$$

where

$$d_u = \begin{cases} \dfrac{f_u}{\sum_{i=1}^{4^k} f_i + w\sum_{j=1}^{\lambda}\theta_j} (1 \le u \le 4^k) \\ \dfrac{w\theta_{u-4^k}}{\sum_{i=1}^{4^k} f_i + w\sum_{j=1}^{\lambda}\theta_j} (4^k \le u \le 4^k + \lambda) \end{cases} \tag{17}$$

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

$$\theta_j = \frac{1}{L\text{-}j\text{-}1}\sum_{i=1}^{L-j-1}\Theta(\mathrm{R}_i\mathrm{R}_{i+1}, \mathrm{R}_{i+j}\mathrm{R}_{i+j+1})(j = 1, 2, \cdots, \lambda; \lambda < L) \tag{18}$$

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

$$\Theta(\mathrm{R}_i\mathrm{R}_{i+1}, \mathrm{R}_{i+j}\mathrm{R}_{i+j+1}) = \frac{1}{\mu}\sum_{v=1}^{\mu}[P_v(\mathrm{R}_i\mathrm{R}_{i+1}) - P_v(\mathrm{R}_{i+j}\mathrm{R}_{i+j+1})]^2 \tag{19}$$

where μ is the number of physicochemical indices, in this study, 6 indices reflecting the local DNA structural properties (**Table 3**) were employed to generate the PseKNC feature vector; $P_v(\mathrm{R}_i\mathrm{R}_{i+1})$ represents the numerical value of the v-th ($u = 1, 2, \cdots, \mu$) physicochemical indices for the dinucleotide $\mathrm{R}_i\mathrm{R}_{i+1}$ at position i and $P_v(\mathrm{R}_{i+j}\mathrm{R}_{i+j+1})$ represents the corresponding value for the dinucleotide $\mathrm{R}_{i+j}\mathrm{R}_{i+j+1}$ at position i+j.

For more information about this approach, please refer to (Guo, et al., 2014)

In order to generate the PseKNC feature vector, we need to import the PseKNC class, construct a PseKNC object and use the method make_pseknc_vec.

Class

PseKNC(*k*[, *lamada*, *w*])

The PseKNC class is the implementation of the PseKNC.

The attributes of PseKNC class are:

- *k*: an integer larger than 0 represents the k-tuple. Its default value is 3.
- *lamada*: 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), representing the highest counted rank (or tier) of the correlation along a DNA sequence. The default value is 1.
- *w*: the weight factor ranged from 0 to 1. Its default value is 0.05.

Method

make_pseknc_vec(*input_data*[, *extra_phyche_index*])

This method returns the PseKNC feature vector and it is in the class PseKNC.

The parameters of make_pseknc_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *extra_phyche_index*: with this option, the user defined dinucleotide physiochemical indices can be used to generate the PseKNC feature vector. The type of this parameter should be a dict, in which the key is the dinucleotide (string), and its corresponding value is a list. **Note** that the PseKNC feature vector will be generated based on the original six physical structures and the user defined indices. If the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.psenac import PseKNC
>>> pseknc = PseKNC()
>>> vec =
pseknc.make_pseknc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
>>> vec
[[0.0, 0.015, 0.0, 0.0, 0.0, 0.0, 0.0, 0.044, 0.0, 0.0, 0.0, 0.0,
0.015, 0.0, 0.0, 0.029, 0.0, 0.015, 0.0, 0.015, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.015, 0.029, 0.015, 0.015, 0.015, 0.0, 0.0,
0.015, 0.0, 0.0, 0.015, 0.0, 0.0, 0.0, 0.015, 0.0, 0.0, 0.0, 0.015,
0.0, 0.0, 0.0, 0.029, 0.015, 0.0, 0.0, 0.0, 0.015, 0.029, 0.015, 0.0,
0.015, 0.015, 0.029, 0.044, 0.5142]]
```

```
>>> len(vec[0])
65
>>> pseknc = PseKNC(k=2, lamada=1, w=0.05)
>>> vec =
pseknc.make_pseknc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'])
>>> vec
[[0.027, 0.08, 0.0, 0.08, 0.053, 0.0, 0.0, 0.106, 0.053, 0.053, 0.027,
0.027, 0.053, 0.053, 0.106, 0.186, 0.0957]]
>>> len(vec[0])
17
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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]]
>>> from repDNA.util import normalize_index
>>> vec =
pseknc.make_pseknc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
>>> vec
[[0.027, 0.08, 0.0, 0.08, 0.053, 0.0, 0.0, 0.106, 0.053, 0.053, 0.027,
0.027, 0.053, 0.053, 0.106, 0.186, 0.0953]]
>>> len(vec[0])
17
```

5.3 Parallel correlation pseudo dinucleotide composition

In PC-PseDNC approach, the users cannot only select the 38 built-in physiochemical indices (**Table 1**), but also can upload their own indices to generate the PC-PseDNC feature vector.

Given a DNA sequence **D** (**Eq. 3**), the PC-PseDNC feature vector of **D** is defined:

$$\mathbf{D} = [d_1 \ d_2 \ \cdots \ d_{16} \ d_{16+1} \ \cdots \ d_{16+\lambda}]^{\mathrm{T}} \tag{20}$$

where

$$d_k = \begin{cases} \dfrac{f_k}{\sum_{i=1}^{16} f_i + w\sum_{j=1}^{\lambda} \theta_j} (1 \le k \le 16) \\ \dfrac{w\theta_{k-16}}{\sum_{i=1}^{16} f_i + w\sum_{j=1}^{\lambda} \theta_j} (17 \le \mathrm{k} \le 16+\lambda) \end{cases} \tag{21}$$

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 dinucleotides along a DNA sequence, which is defined:

$$\begin{cases} \theta_1 = \dfrac{1}{L\text{-}2}\sum_{i=1}^{L-2}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+1}\mathrm{R}_{i+2}) \\ \theta_2 = \dfrac{1}{L\text{-}3}\sum_{i=1}^{L-3}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+2}\mathrm{R}_{i+3}) \\ \theta_3 = \dfrac{1}{L\text{-}4}\sum_{i=1}^{L-4}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+3}\mathrm{R}_{i+4}) \quad (\lambda < L) \\ \cdots\cdots \\ \theta_\lambda = \dfrac{1}{L\text{-}1\text{-}\lambda}\sum_{i=1}^{L-1-\lambda}\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_{i+\lambda}\mathrm{R}_{i+\lambda+1}) \end{cases} \tag{22}$$

where the correlation function is given by

$$\Theta(\mathrm{R}_i\mathrm{R}_{i+1},\mathrm{R}_j\mathrm{R}_{j+1}) = \frac{1}{\mu}\sum_{u=1}^{\mu}[P_u(\mathrm{R}_i\mathrm{R}_{i+1}) - P_u(\mathrm{R}_j\mathrm{R}_{j+1})]^2 \tag{23}$$

where μ is the number of physicochemical indices considered that are listed in the **Table 1**; $P_u(\mathrm{R}_i\mathrm{R}_{i+1})$ ($P_u(\mathrm{R}_j\mathrm{R}_{j+1})$) represents the numerical value of the *u*-th ($u = 1, 2, \cdots, \mu$) physicochemical index for the dinucleotide $\mathrm{R}_i\mathrm{R}_{i+1}$ ($\mathrm{R}_j\mathrm{R}_{j+1}$) at position *i* and *j*, respectively.

For more information of PC-PseDNC approach, you can refer to (Chen, et al., 2014).

In order to generate the PC-PseDNC feature vector, we need to import the PCPseDNC class, construct a PCPseDNC object and use the method make_pcpsednc_vec.

Class

PCPseDNC([*lamada*, *w*])

The PCPseDNC class is the implementation of the PC-PseDNC.

The attributes of PCPseDNC class are:

- *lamada*: 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), representing the highest counted rank (or tier) of the correlation along a DNA sequence. Its default value is 1.
- *w*: the weight factor ranged from 0 to 1, its default value is 0.05.

Method

make_pcpsednc_vec(*input_data*, *phyche_index*[, *all_property*, *extra_phyche_index*])

This method returns PC-PseDNC feature vector and it is in the class PCPseDNC.

The parameters of make_pcpsednc_vec method include:

- *input_data*: the input data, the input data, which should be a list or file type.
- *phyche_index*: The 38 built-in physicochemical indices (**Table 1**), which the users can choose. Its type should be a list.
- *all_property*: with this option, this method will use all the 38 physicochemical indices to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, the user defined dinucleotide physiochemical indices can be used to generate the PC-PseDNC feature vector. The type of this parameter should be a dict, in which the key is the dinucleotide (string), and its corresponding value type is a list. **Note** that the PC-PseDNC feature vector will be generated based on the user selected indices and the user defined indices. If the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.psenac import PCPseDNC
>>> pc_psednc = PCPseDNC()
>>> vec =
pc_psednc.make_pcpsednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'])
>>> vec
[[0.027, 0.08, 0.0, 0.08, 0.053, 0.0, 0.0, 0.106, 0.053, 0.053, 0.027,
0.027, 0.053, 0.053, 0.106, 0.186, 0.0948]]
>>> len(vec[0])
17
>>> pc_psednc = PCPseDNC(lamada=2, w=0.05)
>>> vec =
pc_psednc.make_pcpsednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> vec
[[0.025, 0.075, 0.0, 0.075, 0.05, 0.0, 0.0, 0.1, 0.05, 0.05, 0.025,
0.025, 0.05, 0.05, 0.1, 0.175, 0.072, 0.0757]]
>>> len(vec[0])
18
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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]]
```

```
>>> from repDNA.util import normalize_index
>>> vec =
pc_psednc.make_pcpsednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
>>> vec
[[0.025, 0.074, 0.0, 0.074, 0.049, 0.0, 0.0, 0.098, 0.049, 0.049,
0.025, 0.025, 0.049, 0.049, 0.098, 0.172, 0.0869, 0.0771]]
>>> len(vec[0])
18
```

5.4 Parallel correlation pseudo trinucleotide composition

In PC-PseTNC approach, 12 built-in trinucleotide physiochemical indices (**Table 2**) are incorporated to generate the representations of DNA sequences. Furthermore, the user defined indices can be also used to generate the feature vector.

Given a DNA sequence **D** (**Eq. 3**), the PC-PseTNC feature vector of **D** is defined:

$$\mathbf{D} = [d_1 \ d_2 \ \cdots \ d_{64} \ d_{64+1} \ \cdots \ d_{64+\lambda}]^{\mathrm{T}} \tag{24}$$

where

$$d_k = \begin{cases} \dfrac{f_k}{\sum_{i=1}^{64} f_i + w\sum_{j=1}^{\lambda}\theta_j} (1 \le k \le 64) \\ \dfrac{w\theta_{k-64}}{\sum_{i=1}^{64} f_i + w\sum_{j=1}^{\lambda}\theta_j} (65 \le k \le 64+\lambda) \end{cases} \tag{25}$$

where f_k（k=1,2,⋯,64) is the normalized occurrence frequency of trinucleotide 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 trinucleotide along a DNA sequence, which is defined:

$$\begin{cases} \theta_1 = \dfrac{1}{L\text{-}3}\sum_{i=1}^{L-3}\Theta(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2}, \mathrm{R}_{i+1}\mathrm{R}_{i+2}\mathrm{R}_{i+3}) \\ \theta_2 = \dfrac{1}{L\text{-}4}\sum_{i=1}^{L-4}\Theta(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2}, \mathrm{R}_{i+2}\mathrm{R}_{i+3}\mathrm{R}_{i+4}) \\ \theta_3 = \dfrac{1}{L\text{-}5}\sum_{i=1}^{L-5}\Theta(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2}, \mathrm{R}_{i+3}\mathrm{R}_{i+4}\mathrm{R}_{i+5}) \quad (\lambda < L) \\ \cdots\cdots \\ \theta_\lambda = \dfrac{1}{L\text{-}2\text{-}\lambda}\sum_{i=1}^{L-2-\lambda}\Theta(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2}, \mathrm{R}_{i+\lambda}\mathrm{R}_{i+\lambda+1}\mathrm{R}_{i+\lambda+2}) \end{cases} \tag{26}$$

where the correlation function is given by

$$\Theta(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2},\mathrm{R}_j\mathrm{R}_{j+1}\mathrm{R}_{j+2})=\frac{1}{\mu}\sum_{u=1}^{\mu}[P_u(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2})-P_u(\mathrm{R}_j\mathrm{R}_{j+1}\mathrm{R}_{j+2})]^2 \quad (27)$$

where μ is the number of physiochemical indices (**Table 2**); $P_u(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2})$ ($P_u(\mathrm{R}_j\mathrm{R}_{j+1}\mathrm{R}_{j+2})$) represents the numerical value of the u-th ($u=1,2,\cdots,\mu$) physiochemical index for the trinucleotide $\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2}$ ($\mathrm{R}_j\mathrm{R}_{j+1}\mathrm{R}_{j+2}$) at position i(j).

For more information of PC-PseTNC approach, you can refer to (Chen, et al., 2014) (Qiu, et al., 2014)

In order to generate the PC-PseTNC feature vector, we need to import the PCPseTNC class, construct a PCPseTNC object and use the method make_pcpsetnc_vec.

Class

PCPseTNC([*lamada*, *w*])

The PCPseTNC class is the implementation of the PC-PseTNC.

The attributes of PCPseTNC class are:

- *lamada*: an integer larger than or equal to 0 and less than or equal to *L*-3 (*L* means the length of the shortest sequence in the dataset), representing the highest counted rank (or tier) of the correlation along a DNA sequence. The default value is 1.
- *w*: the weight factor ranged from 0 to 1, its default value is 0.05.

Method

make_pcpsetnc_vec (*input_data*, *phyche_index*[, *all_property*, *extra_phyche_index*])

This method returns the PC-PseTNC feature vector and it is in the class PCPseTNC.

The parameters of make_pcpsetnc_vec method include:

- *input_data*: the input data, the input data, which should be a list or file type.
- *phyche_index*: the 12 built-in physicochemical indices (**Table 2**), which the users can choose. Its type should be a list.
- *all_property*: with this option, this method will use all 12 physicochemical indices to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, the user defined trinucleotide physiochemical indices can be used to generate the PC-PseTNC feature vector.

The type of this parameter should be a dict, in which the key is the trinucleotide (string), and its corresponding value type is a list. **Note** that the PC-PseTNC feature vector will be generated based on the user selected indices and the user defined indices. If the user defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.psenac import PCPseTNC
>>> pc_psetnc = PCPseTNC()
>>> vec =
pc_psetnc.make_pcpsetnc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'])
>>> vec
[[0.0, 0.027, 0.0, 0.0, 0.0, 0.0, 0.0, 0.08, 0.0, 0.0, 0.0, 0.0,
0.027, 0.0, 0.0, 0.053, 0.0, 0.027, 0.0, 0.027, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.027, 0.053, 0.027, 0.027, 0.027, 0.0, 0.0,
0.027, 0.0, 0.0, 0.027, 0.0, 0.0, 0.0, 0.027, 0.0, 0.0, 0.0, 0.027,
0.0, 0.0, 0.0, 0.053, 0.027, 0.0, 0.0, 0.0, 0.027, 0.053, 0.027, 0.0,
0.027, 0.027, 0.053, 0.08, 0.1229]]
>>> len(vec[0])
65
>>> pc_psetnc = PCPseTNC(lamada=2, w=0.05)
>>> vec =
pc_psetnc.make_pcpsetnc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> vec
[[0.0, 0.024, 0.0, 0.0, 0.0, 0.0, 0.0, 0.073, 0.0, 0.0, 0.0, 0.0,
0.024, 0.0, 0.0, 0.048, 0.0, 0.024, 0.0, 0.024, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.024, 0.048, 0.024, 0.024, 0.024, 0.0, 0.0,
0.024, 0.0, 0.0, 0.024, 0.0, 0.0, 0.0, 0.024, 0.0, 0.0, 0.0, 0.024,
0.0, 0.0, 0.0, 0.048, 0.024, 0.0, 0.0, 0.0, 0.024, 0.048, 0.024, 0.0,
0.024, 0.024, 0.048, 0.073, 0.0851, 0.1147]]
>>> len(vec[0])
66
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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]]
>>> from repDNA.util import normalize_index
>>> vec =
pc_psetnc.make_pcpsetnc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
>>> vec
[[0.0, 0.023, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07, 0.0, 0.0, 0.0, 0.0,
0.023, 0.0, 0.0, 0.046, 0.0, 0.023, 0.0, 0.023, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.023, 0.046, 0.023, 0.023, 0.023, 0.0, 0.0,
0.023, 0.0, 0.0, 0.023, 0.0, 0.0, 0.0, 0.023, 0.0, 0.0, 0.0, 0.023,
0.0, 0.0, 0.0, 0.046, 0.023, 0.0, 0.0, 0.0, 0.023, 0.046, 0.023, 0.0,
0.023, 0.023, 0.046, 0.07, 0.1102, 0.1229]]
>>> len(vec[0])
66
```

5.5 Series correlation pseudo dinucleotide composition

SC-PseDNC is a variant of PC-PseDNC. Given a DNA sequence **D** (**Eq. 3**), the SC-PseDNC feature vector of **D** is defined:

$$\mathbf{D} = [d_1 \; d_2 \; \cdots \; d_{16} \; d_{16+1} \; \cdots \; d_{16+\lambda} \; d_{16+\lambda+1} \; \cdots \; d_{16+\lambda\Lambda}]^{\mathrm{T}} \tag{28}$$

where

$$d_k = \begin{cases} \dfrac{f_k}{\sum_{i=1}^{16} f_i + w\sum_{j=1}^{\lambda}\theta_j} (1 \le k \le 16) \\ \dfrac{w\theta_{k-16}}{\sum_{i=1}^{16} f_i + w\sum_{j=1}^{\lambda\Lambda}\theta_j} (17 \le \mathrm{k} \le 16+\lambda\Lambda) \end{cases} \tag{29}$$

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; Λ is the number of physicochemical indices; θ_j (j = 1, 2, ⋯, λ) is called the j-tier correlation factor that reflects the sequence-order correlation between all the most contiguous dinucleotides along a DNA sequence, which is defined:

$$\begin{cases} \theta_1 = \dfrac{1}{L\text{-}3}\sum_{i=1}^{L-3} \mathrm{J}_{i,i+1}^{1} \\ \theta_2 = \dfrac{1}{L\text{-}3}\sum_{i=1}^{L-3} \mathrm{J}_{i,i+1}^{2} \\ \cdots\cdots \\ \theta_\Lambda = \dfrac{1}{L\text{-}3}\sum_{i=1}^{L-3} \mathrm{J}_{i,i+1}^{\Lambda} \qquad \lambda < (\mathrm{L}-2) \\ \cdots\cdots \\ \theta_{\lambda\Lambda-1} = \dfrac{1}{L\text{-}\lambda\text{-}2}\sum_{i=1}^{L-\lambda-2} \mathrm{J}_{i,i+\lambda}^{\Lambda-1} \\ \theta_{\lambda\Lambda} = \dfrac{1}{L\text{-}\lambda\text{-}2}\sum_{i=1}^{L-\lambda-2} \mathrm{J}_{i,i+\lambda}^{\Lambda} \end{cases} \tag{30}$$

The correlation function is given by

$$\begin{cases} \mathrm{J}_{i,i+m}^{\zeta} = P_u(\mathrm{R}_i\mathrm{R}_{i+1}) \cdot P_u(\mathrm{R}_{i+m}\mathrm{R}_{i+\mathrm{m}+1}) \\ \zeta = 1, 2, \cdots, \Lambda;\ m = 1, 2, \cdots, \lambda;\ i = 1, 2, \cdots, \mathrm{L}\text{-}\lambda\text{-}2 \end{cases} \tag{31}$$

where μ is the number of total physiochemical indices (**Table 1**); $P_u(\mathrm{R}_i\mathrm{R}_{i+1})$ ($P_u(\mathrm{R}_j\mathrm{R}_{j+1})$) represents the numerical value of the u-th ($u = 1,\ 2,\ \cdots,\ \mu$) physiochemical index for the dinucleotide $\mathrm{R}_i\mathrm{R}_{i+1}$ ($\mathrm{R}_j\mathrm{R}_{j+1}$) at position $i(j)$.

For more information of the SC-PseDNC, please refer to (Chen, et al., 2014).

In order to generate the SC-PseDNC feature vector, we need to import the SCPseDNC class, construct a SCPseDNC object and use the method make_scpsednc_vec.

Class

SCPseDNC([*lamada*, *w*])

The SCPseDNC class is the implementation of the SC-PseDNC.

The attributes of SCPseDNC class are:

- *lamada*: 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), representing the highest counted rank (or tier) of the correlation along a DNA sequence. The default value is 1.
- *w*: the weight factor ranged from 0 to 1, the default value is 0.05.

Method

make_scpsednc_vec(*input_data*[, *phyche_index*, *all_property*, *extra_phyche_index*])

This method returns the SC-PseDNC feature vector and it is in the class SCPseDNC.

The parameters of make_scpsednc_vec method include:

- *input_data*: the input data, it should be a list or file type.
- *phyche_index*: The 38 built-in physicochemical indices (**Table 1**), which the users can choose. Its type should be a list.
- *all_property*: with this option, this method will use all the 38 physicochemical indices to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, the user defined dinucleotide physiochemical indices can be used to generate the SC-PseDNC feature vector. The type of this parameter should be a dict, in which the key is the dinucleotide (string), and its corresponding value is a list. **Note** that the SC-PseDNC feature vector will be generated based on the built-in indices and the user-defined indices. If the user-defined physicochemical indices have not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.psenac import SCPseDNC
>>> sc_psednc = SCPseDNC()
>>> vec =
sc_psednc.make_scpsednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Twist', 'Tilt'])
>>> vec
[[0.03, 0.09, 0.0, 0.09, 0.06, 0.0, 0.0, 0.12, 0.06, 0.06, 0.03, 0.03,
0.06, 0.06, 0.12, 0.21, -0.0088, -0.0093]]
>>> len(vec[0])
18
>>> sc_psednc = SCPseDNC(lamada=2, w=0.05)
>>> vec =
sc_psednc.make_scpsednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> len(vec[0])
92
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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]]
>>> from repDNA.util import normalize_index
>>> vec =
sc_psednc.make_scpsednc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
```

```
phyche_index=['Twist', 'Tilt'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
>>> vec
[[0.03, 0.09, 0.0, 0.09, 0.06, 0.0, 0.0, 0.12, 0.06, 0.06, 0.03, 0.03,
0.06, 0.06, 0.12, 0.209, -0.0088, -0.0093, 0.0004, -0.0088, 0.0,
0.01]]
>>> len(vec[0])
22
```

5.6 Series correlation pseudo trinucleotide composition

SC-PseTNC is a variant of PC-PseTNC. Given a DNA sequence **D** (**Eq. 3**), the SC-PseTNC feature vector of **D** is defined:

$$\mathbf{D}=[d_1\ d_2\ \cdots\ d_{64}\ d_{64+1}\ \cdots\ d_{64+\lambda}\ d_{64+\lambda+1}\ \cdots\ d_{64+\lambda\Lambda}]^{\mathrm{T}} \tag{32}$$

where

$$d_k=\begin{cases}\dfrac{f_k}{\sum_{i=1}^{64}f_i+w\sum_{j=1}^{\lambda}\theta_j}(1\le k\le 64)\\ \dfrac{w\theta_{k-64}}{\sum_{i=1}^{64}f_i+w\sum_{j=1}^{\lambda\Lambda}\theta_j}(65\le k\le 64+\lambda\Lambda)\end{cases} \tag{33}$$

where f_k（k=1, 2, ⋯, 64) is the normalized occurrence frequency of trinucleotide 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; Λ is the number of physicochemical indices; θ_j (j=1, 2, ⋯, λ) is called the j-tier correlation factor that reflects the sequence-order correlation between all the most contiguous trinucleotides along a DNA sequence, which is defined:

$$\begin{cases}\theta_1=\dfrac{1}{L\text{-}4}\sum_{i=1}^{L-4}\mathrm{J}_{i,i+1}^{1}\\ \theta_2=\dfrac{1}{L\text{-}4}\sum_{i=1}^{L-4}\mathrm{J}_{i,i+1}^{2}\\ \cdots\cdots\\ \theta_\Lambda=\dfrac{1}{L\text{-}4}\sum_{i=1}^{L-4}\mathrm{J}_{i,i+1}^{\Lambda}\qquad \lambda<(\mathrm{L}-3)\\ \cdots\cdots\\ \theta_{\lambda\Lambda-1}=\dfrac{1}{L\text{-}\lambda\text{-}3}\sum_{i=1}^{L-\lambda-3}\mathrm{J}_{i,i+\lambda}^{\Lambda-1}\\ \theta_{\lambda\Lambda}=\dfrac{1}{L\text{-}\lambda\text{-}3}\sum_{i=1}^{L-\lambda-3}\mathrm{J}_{i,i+\lambda}^{\Lambda}\end{cases} \tag{34}$$

The correlation function is given by

$$\begin{cases}\mathrm{J}_{i,i+m}^{\zeta}=P_u(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2})\cdot P_u(\mathrm{R}_{i+m}\mathrm{R}_{i+m+1}\mathrm{R}_{i+m+2})\\ \zeta=1,2,\cdots,\Lambda;\ m=1,2,\cdots,\lambda;\ i=1,2,\cdots,\mathrm{L}\text{-}\lambda\text{-}3\end{cases} \tag{35}$$

where μ is the number of physiochemical indices (**Table 2**); $P_u(\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2})$ ($P_u(\mathrm{R}_j\mathrm{R}_{j+1}\mathrm{R}_{j+2})$) represents the numerical value of the *u*-th ($u = 1, 2, \cdots, \mu$) physiochemical index for the trinucleotide $\mathrm{R}_i\mathrm{R}_{i+1}\mathrm{R}_{i+2}$ ($\mathrm{R}_j\mathrm{R}_{j+1}\mathrm{R}_{j+2}$) at position *i*(*j*).

For more information of the SC-PseTNC approach, please refer to (Chen, et al., 2014)

In order to generate the SC-PseTNC feature vector, we need to import the SCPseTNC class, construct a SCPseTNC object and use the method make_scpsetnc_vec.

Class

SCPseTNC([*lamada*, *w*])

The SCPseTNC class is the implementation of the SC-PseTNC.

The attributes of SCPseTNC class are:

- *lamada*: an integer larger than or equal to 0 and less than or equal to *L*-3 (*L* means the length of the shortest DNA sequence in the dataset), representing the highest counted rank (or tier) of the correlation along a DNA sequence. The default value is 1.
- *w*: the weight factor ranged from 0 to 1, the default value is 0.05.

Method

make_scpsetnc_vec(*input_data*[, *phyche_index*, *all_property*, *extra_phyche_index*])

This method returns the SC-PseTNC feature vector and it is in the class SCPseTNC.

The parameters of make_scpsetnc_vec method include:

- *input_data*: the input data, which should be a list or file type.
- *phyche_index*: the 12 built-in physicochemical indices (**Table 2**), which the users can choose. Its type should be a list.
- *all_property*: with this option, this method will use all the 12 physicochemical indices to generate the feature vector. Its default value is **False**.
- *extra_phyche_index*: with this option, the user defined trinucleotide physiochemical indices can be used to generate the SC-PseTNC feature vector. The type of this parameter should be a dict, in which the key is the trinucleotide (string), and its corresponding value is a list. **Note** that the SC-PseTNC feature vector will be generated based on the user selected indices and the user-defined indices. If the user-defined physicochemical indices have

not been normalized, it should be normalized by using function normalize_index with the parameter *is_convert_dict*=**True** in util module.

Examples

```
>>> from repDNA.psenac import SCPseTNC
>>> sc_psetnc = SCPseTNC()
>>> vec =
sc_psetnc.make_scpsetnc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'])
>>> len(vec[0])
66
>>> sc_psetnc = SCPseTNC(lamada=2, w=0.05)
>>> vec =
sc_psetnc.make_scpsetnc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
all_property=True)
>>> len(vec[0])
88
# Use user-defined physicochemical indices to generate vector.
>>> phyche_index = [
... [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]]
>>> from repDNA.util import normalize_index
>>> vec =
sc_psetnc.make_scpsetnc_vec(['GACTGAACTGCACTTTGGTTTCATATTATTTGCTC'],
phyche_index=['Dnase I', 'Nucleosome'],
extra_phyche_index=normalize_index(phyche_index,is_convert_dict=True))
>>> len(vec[0])
70
```

Table 1. The names of the 38 physicochemical indices for dinucleotides.

Base stacking	Protein induced deformability	B-DNA twist
A-philicity	Propeller twist	Duplex stability: (freeenergy)
DNA denaturation	Bending stiffness	Protein 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	Rise	

Table 2. 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

Table 3. The names of the 6 physicochemical indices for dinucleotides.

Twist	Tilt	Roll
Shift	Slide	Rise

Reference

Chen, W., *et al.* (2013) iRSpot-PseDNC: identify recombination spots with pseudo dinucleotide composition, *Nucleic Acids Res*, **41**, e68.

Chen, W., *et al.* (2014) PseKNC: a flexible web server for generating pseudo K-tuple nucleotide composition, *Analytical biochemistry*, **456**, 53-60.

Chen, W., Luo, L. and Zhang, L. (2010) The organization of nucleosomes around splice sites, *Nucleic Acids Res*, **38**, 2788-2798.

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

Guo, S.H., *et al.* (2014) iNuc-PseKNC: a sequence-based predictor for predicting nucleosome positioning in genomes with pseudo k-tuple nucleotide composition, *Bioinformatics*, **30**, 1522-1529.

Gupta, S., *et al.* (2008) Predicting human nucleosome occupancy from primary sequence, *PLoS computational biology*, **4**, e1000134.

Lee, D., Karchin, R. and Beer, M.A. (2011) Discriminative prediction of mammalian enhancers from DNA sequence, *Genome research*, **21**, 2167-2180.

Liu, G., *et al.* (2012) Sequence-dependent prediction of recombination hotspots in Saccharomyces cerevisiae, *Journal of theoretical biology*, **293**, 49-54.

Lv, J. and Luo, L.F. (2008) Prediction for human transcription start site using diversity measure with quadratic discriminant, *Bioinformation*, **2**, 316-321.

Noble, W.S., *et al.* (2005) Predicting the in vivo signature of human gene regulatory sequences, *Bioinformatics*, **21 Suppl 1**, i338-343.

Qiu, W.R., Xiao, X. and Chou, K.C. (2014) iRSpot-TNCPseAAC: identify recombination spots with trinucleotide composition and pseudo amino acid components, *International journal of molecular sciences*, **15**, 1746-1766.

Zhang, L.R. and Luo, L.F. (2003) Splice site prediction with quadratic discriminant analysis using diversity measure, *Nucleic Acids Res*, **31**, 6214-6220.