# *iLearn:* an integrated platform and meta-learner for feature engineering and machine learning analysis and modeling of DNA, RNA and protein sequence data

Zhen Chen[1,†], Pei Zhao[2,†], Fuyi Li[3], Tatiana T. Marquez-Lago[4,5], André Leier[4,5], Jerico Revote[3], David R. Powell[3], Tatsuya Akutsu[6], Geoffrey I. Webb[7], A. Ian Smith[3], Roger J. Daly[3], Kuo-Chen Chou[8,9], Jiangning Song[3,7,10,*]

[1]School of Basic Medical Science, Qingdao University, 38 Dengzhou Road, Qingdao, 266021, Shandong, China, [2]State Key Laboratory of Cotton Biology, Institute of Cotton Research of Chinese Academy of Agri-cultural Sciences (CAAS), Anyang, 455000, China, [3]Biomedicine Discovery Institute and Department of Biochemistry and Molecular Biology, Monash University, Melbourne, VIC 3800, Australia, [4]Department of Genetics, School of Medicine, University of Alabama at Birmingham, USA, [5]Department of Cell, Developmental and Integrative Biology, School of Medicine, University of Alabama at Birmingham, AL, USA, [6]Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto 611-0011, Japan, [7]Monash Centre for Data Science, Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia, [8]Gordon Life Science Institute, Boston, MA 02478, USA, [9]Key Laboratory for Neuro-Information of Ministry of Education, School of Life Science and Technology, Center for Informational Biology, University of Electronic Science and Technology of China, Chengdu, 610054, China, [10]ARC Centre of Excellence in Advanced Molecular Imaging, Monash University, Melbourne, VIC 3800, Australia

## Supplementary Material

Package Version: 1.0

# Content

1. Installation
2. Full Workflow of *iLearn*
3. Software Package Overview
4. Input Format of *iLearn*
5. Commonly Used Feature Descriptors for Nucleotide sequences
6. Commonly Used Feature Descriptors for Protein or Peptide sequences
7. Feature Analysis Using *iLearn*
8. Predictor Construction Using *iLearn*
9. Performance Evaluation Strategy of *iLearn*
10. Online Web Server
11. Summary
12. Acknowledgements
13. References

## Brief introduction

*iLearn* is a comprehensive Python-based toolkit, integrating feature extraction/calculation, feature analysis (clustering, feature selection, normalization and dimension reduction), predictor construction, best descriptor/model selection, ensemble learning and performance evaluation for DNA, RNA and protein sequences. *iLearn* is capable of calculating and extracting a wide spectrum of 18 major sequence encoding schemes that encompass 53 different types of feature descriptors for protein sequences, and also can be used to extract 6 major encoding schemes which encompass 26 and 18 different types of feature descriptors for DNA and RNA sequences. Developed from *iFeature* (1), *iLearn* also integrates six kinds of frequently-used feature clustering algorithms, five feature selection algorithms, and three dimensionality reduction algorithms. Four output feature formats are supported by *iLearn*, which can be directly used and processed in other tools. Furthermore, five commonly used machine learning algorithms are provided, including SVM (Support Vector Machine), RF (Random Forest), ANN (Artificial Neutral Network), KNN (*K*-Nearest Neighbours) and LR (Logistic Regression). In order to facilitate users' interpretability of outcomes, the clustering and dimensionality reduction results generated by *iLearn* can be further visualized in form of scatter diagrams, while the cross-validation result can be visualized in the form of ROC and PRC curves. This makes *iLearn* a unique and powerful tool that greatly facilitates feature generation, analysis, training and benchmarking of machine-learning models and predictions.

## 1. Installation

*iLearn* is an open-source Python-based toolkit, which operates depending on the Python environment (Python Version 3.0 or above) and can be run on multi-OS systems (such as Windows, Mac and Linux operating systems). Before running *iLearn*, user should make sure all the following packages are installed in their Python environment: sys, os, shutil, scipy, argparse, collections, platform, math, re, numpy (1.13.1), sklearn (0.19.1), matplotlib (2.1.0), and pandas (0.20.1). For convenience, we strongly recommended users to install the Anaconda Python 3.0 version (or above) in your local computer. The latter can be freely downloaded from https://www.anaconda.com/download/.

## 2. Full Workflow of *iLearn*

Here, we provide step-by-step user instruction illustrating the full workflow of the *iLearn* toolkit by running the example provided in the "examples" directory. *iLearn* includes sixteen main programs, which can be divided into four groups (Table 1).

Table 1. The sixteen main programs in *iLearn* package.

| Groups | Programs | Function |
|---|---|---|
| Group 1 | iLearn-protein-basic.py | Extracting 37 different types of feature descriptors for proteins sequences. |

| | | |
|---|---|---|
| | iLearn-protein-PseKRAAC.py | Extracting the 16 types of pseudo K-tuple reduced amino acid composition (PseKRAAC) feature descriptors for protein sequences. |
| | iLearn-nucleotide-basic.py | Extracting 14 different types of feature descriptors for nucleotide sequences. |
| | iLearn-nucleotide-acc.py | Extracting 6 different types of autocorrelation descriptors for nucleotide sequences. |
| | iLearn-nucleotide-Pse.py | Extracting 6 different types of pseudo-*k*-tuple composition descriptors for nucleotide sequences. |
| Group 2 | iLearn-clustering.py | Running the feature or sample clustering algorithms. |
| | iLearn-feature-normalization.py | Running the feature normalization algorithms |
| | iLearn-feature-selectior.py | Running the feature selection algorithms. |
| | iLearn-dimension-reduction.py | Running the dimension reduction algorithms. |
| Group 3 | iLearn-ML-SVM.py | Running the SVM algorithm. |
| | iLearn-ML-RF.py | Running the RF algorithm. |
| | iLearn-ML-MLP.py | Running the ANN algorithm. |
| | iLearn-ML-LR.py | Running the LR algorithm. |
| | iLearn-ML-KNN.py | Running the KNN algorithm. |
| Group 4 | iLearn-descriptor-estimater.py | Estimating the prediction ability for the specified descriptors |
| | iLearn-auto-pipline.py | Running the *iLearn* pipeline. |

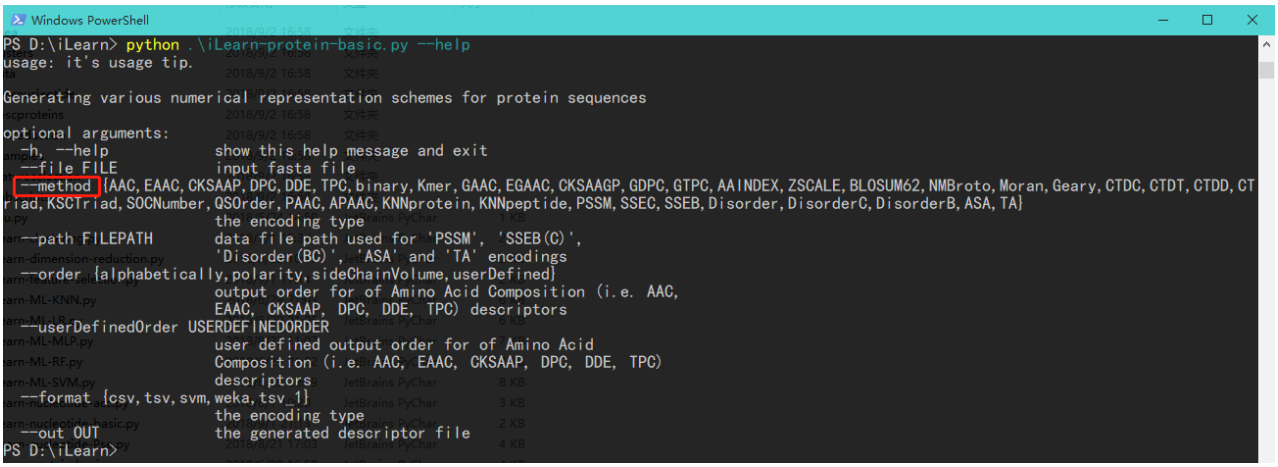
## 3. Software Package Overview

*iLearn* can generate a wide spectrum of 18 feature encoding schemes, encompassing a total of 53 different types of feature descriptors derived from protein or peptide amino acid sequences, and 42 different types of feature descriptors for nucleotide sequences. The 18 major encoding scheme groups for protein sequences and peptides can be found in (1) and the 6 encoding scheme groups for nucleotide sequences included in *iLearn* are summarized in **Table 2** of the main manuscript. Moreover, *iLearn* also integrates a variety of commonly used feature clustering, normalization, selection, dimensionality reduction and predictor construction algorithms, which greatly facilitates feature generation, importance analysis, model training and performance evaluation experiments. We describe the detailed functions of *iLearn* below.

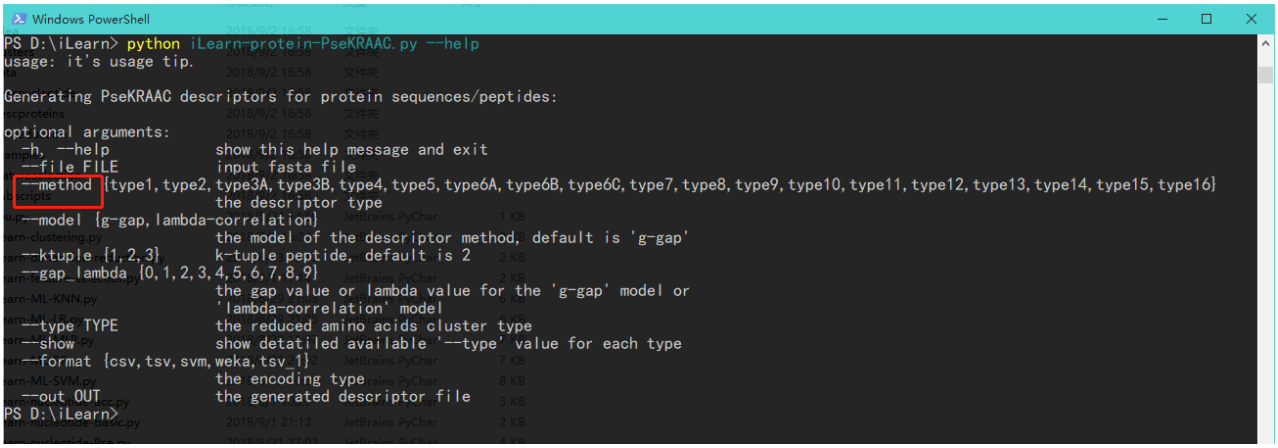
**Feature descriptor extraction:**

Generally, each type of feature descriptor can be calculated using the main programs "iLearn-protein-basic.py", "iLearn-protein-PseKRAAC.py", "iLearn-nucleotide-basic.py", "iLearn-nucleotide-acc.py" and "iLearn-nucleotide-Pse.py" implemented in the *iLearn* toolkit. Users are advised to specify the descriptor type by using the parameter `'--method'`.

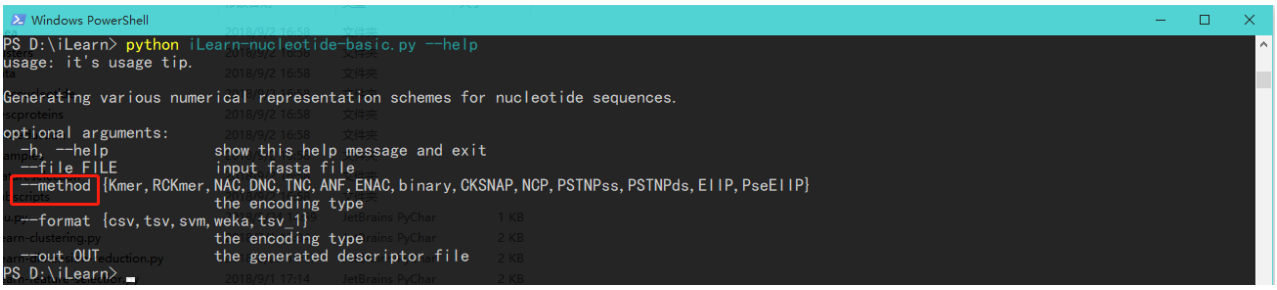
```
tcsh% python iLearn-protein-basic.py --help
```



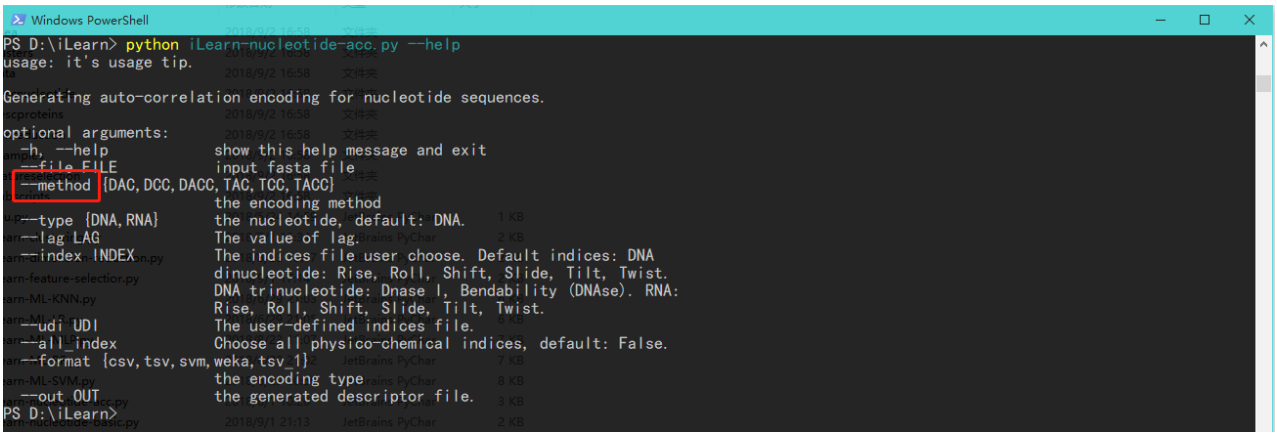

tcsh% python iLearn-protein-PseKRAAC.py --help



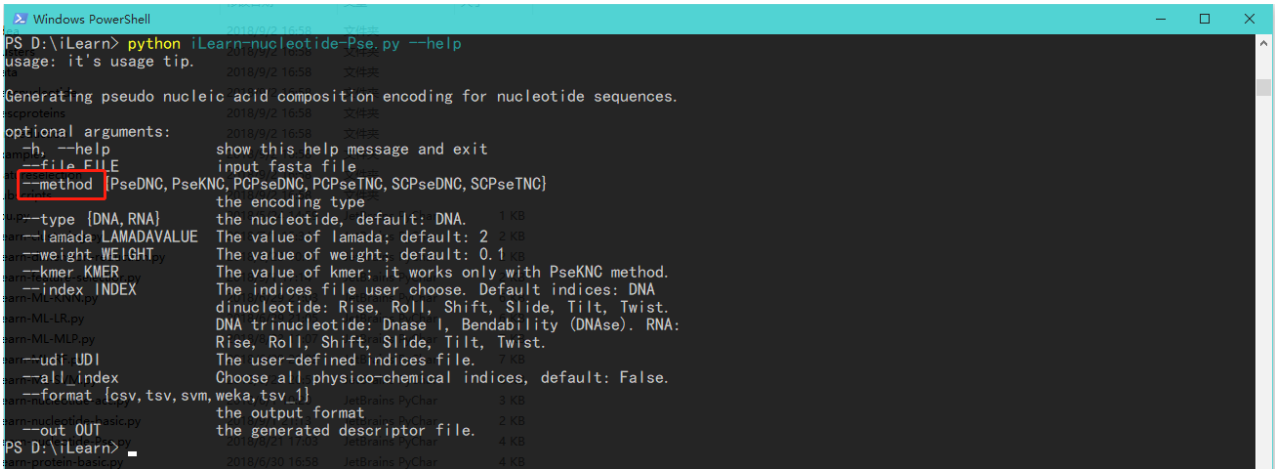

tcsh% python iLearn-nucleotide-basic.py --help




tcsh% python iLearn-nucleotide-acc.py --help




```
tcsh% python iLearn-nucleotide-Pse.py --help
```




**Feature clustering:**

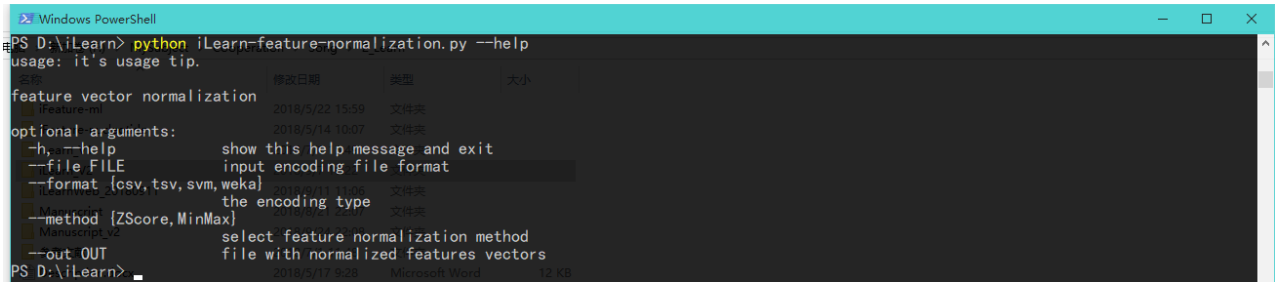
Use the following command to show the help information for all feature clustering algorithms in the *iLearn* package:

```
tcsh% python iLearn-clustering.py --help
```

**Feature normalization:**

Use the following command to show the help information for all feature normalization algorithms in the *iLearn* package:

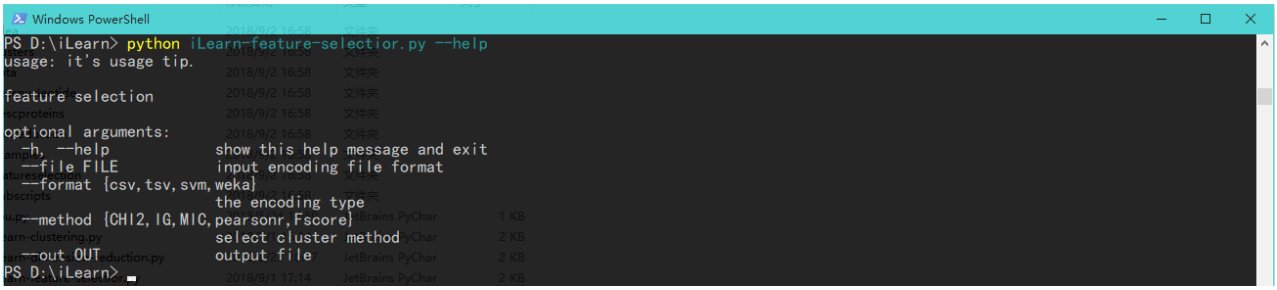
```
tcsh% python iLearn-feature-normalization.py --help
```




**Feature selection:**

Use the following command to show the help information for the feature selection algorithms implemented in the *iLearn* package:

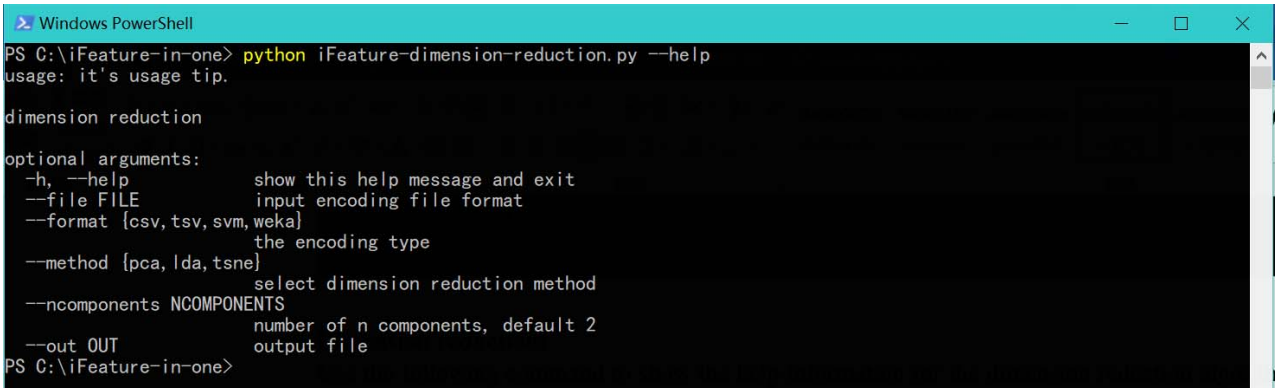
```
tcsh% python iLearn-feature-selectior.py --help
```




**Dimension reduction:**

Use the following command to show the help information for the dimension reduction algorithms implemented in the *iLearn* package:

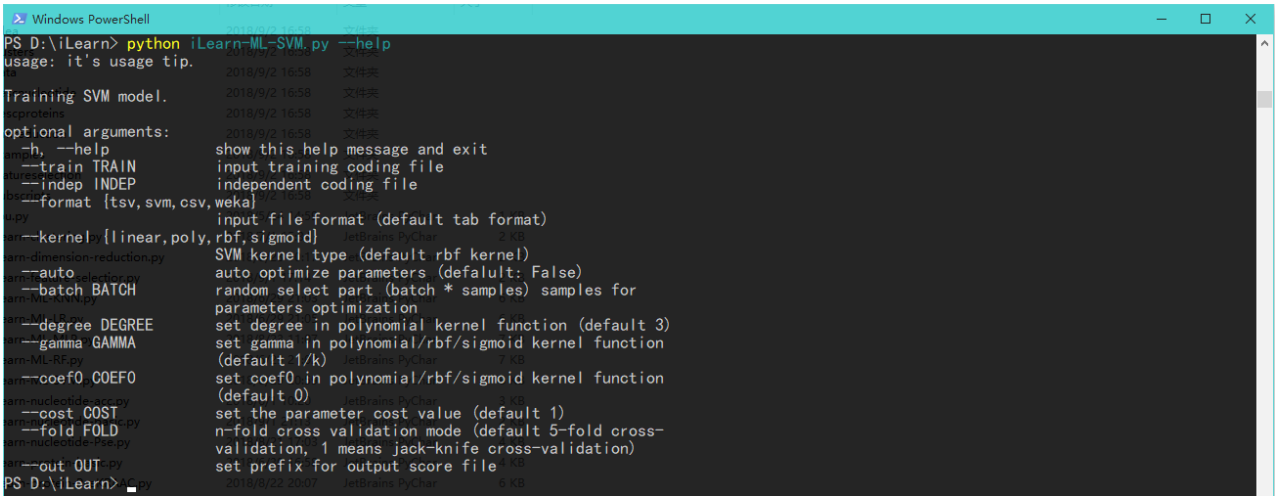
```
tcsh% python iLearn-dimension-reduction.py --help
```



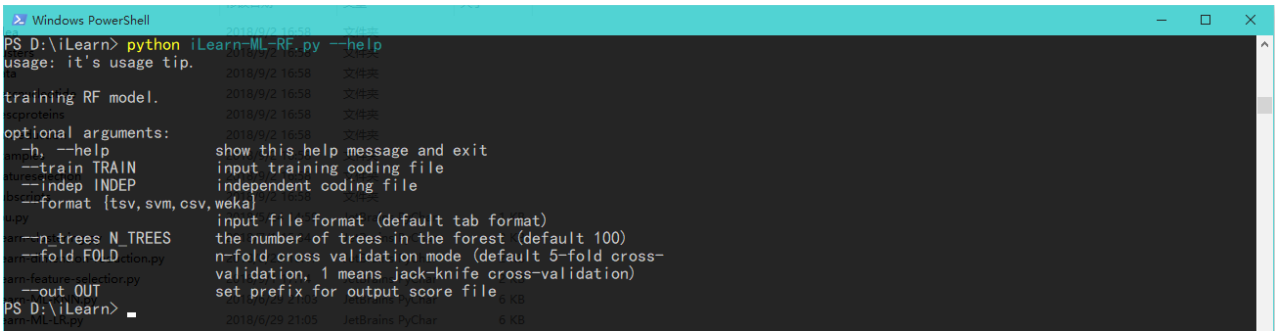

**Descriptor construction algorithms:**

Use the following command to show the help information for machine learning algorithms implemented in the *iLearn* package:

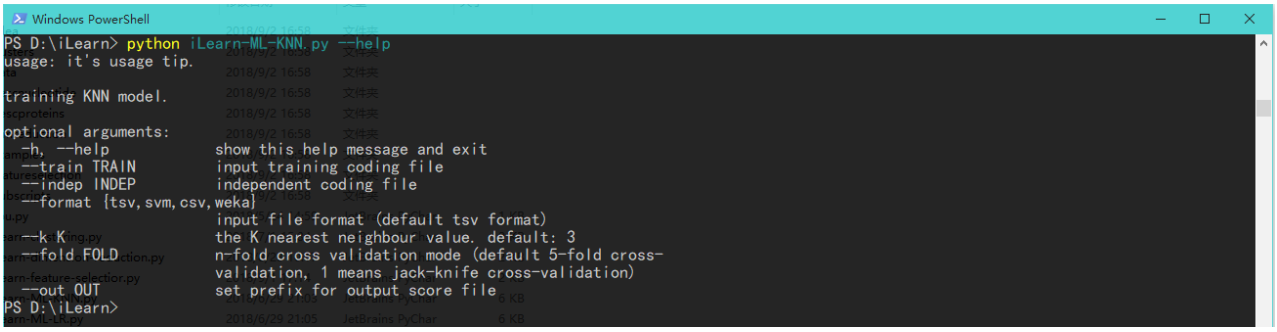
```
tcsh% python iLearn-ML-SVM.py --help
```



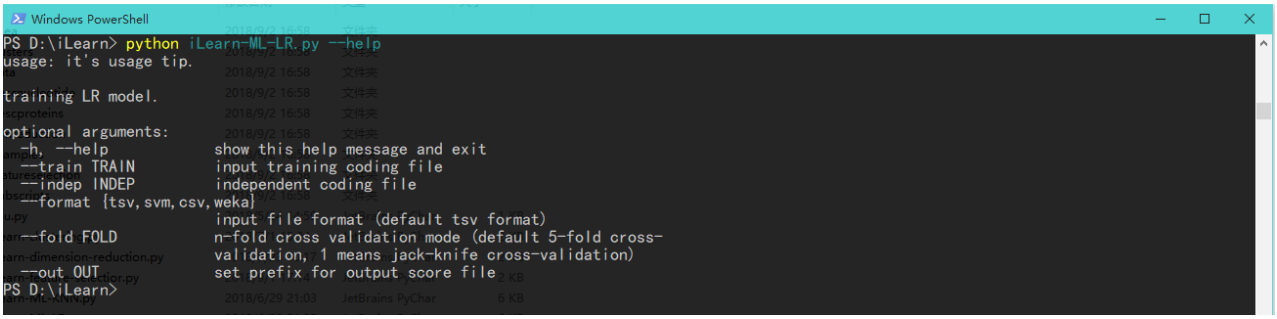

tcsh% python iLearn-ML-RF.py --help



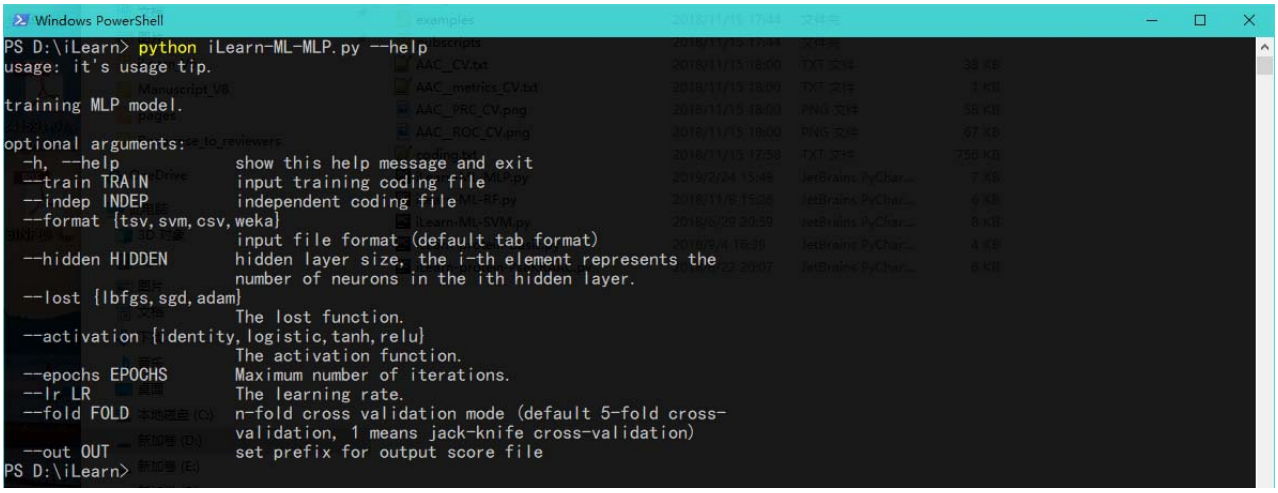

tcsh% python iLearn-ML-KNN.py --help




tcsh% python iLearn-ML-LR.py --help




tcsh% python iLearn-ML-MLP.py --help




**Descriptor evaluater:**

Use the following command to show the help information for predictors prediction ability evaluation implemented in the *iLearn* package:

**Ensemble learning:**

Use the following command to show the help information for ensemble learning implemented in the *iLearn* package:

## 4. The Input format of *iLearn*

The input for *iLearn* is a set of DNA, RNA or protein sequences in a special FASTA format. The FASTA header consists of three parts: part 1, part 2 and part 3, which are separated by the symbol '|' (Figure 2 in the main manuscript). Part 1 is the sequence name. Part 2 is the sample category information, which can be filled with any integer. For instance, users may use 1 to indicate the positive samples and -1 or 0 to represent the negative samples for a binary classification task, or use 0, 1, 2, … to represent the different class in multiclass classification tasks. Part 3 indicates the role of the sample, where e.g. "training" would indicate that the corresponding sequence would be used as the training set for K-fold validation test, and "testing" that the sequence would be used as the independent set for independent testing.

## 5. Commonly Used Feature Descriptors for nucleotide sequences

Let us assume that a nucleotide sequence with $L$ amino acid residues can be generally represented as $\{R_1, R_2, \ldots, R_L\}$, where $R_i$ represents the base at the *i-th* position in the sequence. The following commonly used feature descriptors can be calculated and extracted using *iLearn*.

### 5.1 Kmer

For kmer descriptor, the DNA or RNA sequences are represented as the occurrence frequencies of $k$ neighboring nucleic acids, which has been successfully applied to human gene regulatory sequence prediction (2) and enhancer identification (3). The Kmer ($k$=3) descriptor can be calculated as:

$$f(t) = \frac{N(t)}{N}, \quad t \in \{AAA, AAC, AAG, \ldots, TTT\}$$

where $N(t)$ is the number of kmer type $t$, while $N$ is the length of a nucleotide sequence.

Use the following command to extract the Kmer feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method Kmer --format svm
```

The parameters of iLearn-nucleotide-basic.py are:

- *file*: the input sequence file with FASTA format
- *method*: the descriptor type
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

Advanced usage:

```
tcsh% python descnucleotide/Kmer.py --file examples/DNA_training.txt
--kmer 3 --upto --normalize --format csv
```

The parameters of Kmer.py are:

- *kmer*: the value of kmer, it should be an integer larger than 0, default is 2
- *upto*: with this parameter the program will generate all the kmers: 1mer, 2mer, …, kmer
- *normalize*: with this parameter the final feature vector will be normalized based on the total occurrences of all kmers
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported

### 5.2 Reverse Compliment Kmer (RCKmer)

The reverse compliment kmer (2,4) is a variant of kmer descriptor, in which the kmers are not expected to be strand-specific. For instance, for a DNA sequence, there are 16 types of 2-mers (i.e. 'AA', 'AC', 'AG', 'AT', 'CA', 'CC', 'CG', 'CT', 'GA', 'GC', 'GG', 'GT', 'TA', 'TC', 'TG', 'TT'), 'TT' is reverse compliment with 'AA'. After 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').

Use the following command to extract the RCKmer feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method RCKmer --format svm
```

Advanced usage:

```
tcsh% python descnucleotide/RCKmer.py --file
examples/DNA_training.txt --kmer 2 --upto --normalize --format csv
```

The parameters of RCKmer.py are:

- *kmer*: the value of kmer, it should be an integer larger than 0, default is 2
- *upto*: with this parameter the program will generate all the kmers: 1mer, 2mer, …, kmer
- *normalize*: with this parameter the final feature vector will be normalized based on the total occurrences of all kmers
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported

### 5.3 Nucleic Acid Composition (NAC)

The Nucleic Acid Composition (NAC) encoding calculates the frequency of each nucleic acid type in a nucleotide sequence. The frequencies of all 4 natural nucleic acids (i.e. "ACGT or U") can be calculated as:

$$f(t) = \frac{N(t)}{N}, \quad t \in \{\text{A, C, G, T(U)}\}$$

where $N(t)$ is the number of nucleic acid type $t$, while $N$ is the length of a nucleotide sequence.

Use the following command to extract the NAC feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method NAC
```

### 5.4 Di-Nucleotide Composition (DNC)

The Di-Nucleotide Composition gives 16 descriptors. It is defined as:

$$\mathrm{D(r,s)} = \frac{N_{rs}}{N-1}, \ r,s \in \{A, C, G, T(U)\}$$

where $N_{rs}$ is the number of di-nucleotide represented by nucleic acid types $r$ and $s$.

Use the following command to extract the DNC feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method DNC
```

### 5.5 Tri-Nucleotide Composition (TNC)

The Tri-Nucleotide Composition gives 64 descriptors. It is defined as:

$$\mathrm{D(r,s,t)} = \frac{N_{rst}}{N-2}, \ r,s,t \in \{A, C, G, T(U)\}$$

where $N_{rst}$ is the number of tri-nucleotide represented by nucleic acid types $r$, $s$ and $t$.

Use the following command to extract the TNC feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method TNC
```

### 5.6 Enhanced Nucleic Acid Composition (ENAC)

The Enhanced Nucleic Acid Composition (ENAC) calculates the NAC based on the sequence window of fixed length (the default value is 5) that continuously slides from the 5' to 3' terminus of each nucleotide sequence and can be usually applied to encode the nucleotide sequence with an equal length. For more information of this approach, please refer to (1).

Use the following command to extract the ENAC feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method ENAC
```

Advanced usage:

```
tcsh% python descnucleotide/ENAC.py --file examples/DNA_training.txt
--slwindow 10
```

The parameters of RCKmer.py are:

- *slwindow*: the sliding window of ENAC descriptor, it should be an integer larger than 0, default is 5

### 5.7 binary

In the Binary encoding, each amino acid is represented by a 4-dimensional binary vector, e.g. A is encoded by (1000), C is encoded by (0100), G is encoded by (0010) and T(U) is encoded by (0001), respectively. This encoding scheme is often used to encode nucleotide sequence with an equal length.

Use the following command to extract the binary feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method binary
```

### 5.8 Composition of *k*-spaced Nucleic Acid Pairs (CKSNAP)

The CKSNAP feature encoding calculates the frequency of nucleic acid pairs separated by any $k$ nucleic acid ($k$ = 0, 1, 2, … , 5. The default maximum value of $k$ is 5) . Taking $k$ = 0 as an example, there are 16 0-spaced nucleic acid pairs (i.e. 'AA', 'AC', 'AG', 'AT', 'CA', 'CC', 'CG', 'CT', 'GA', 'GC', 'GG', 'GT', 'TA', 'TC', 'TG', 'TT'). Then, a feature vector can be defined as:

$$(\frac{N_{AA}}{N_{total}}, \frac{N_{Ac}}{N_{total}}, \frac{N_{AG}}{N_{total}}, \dots, \frac{N_{TT}}{N_{total}}, )_{16}$$

The value of each descriptor denotes the composition of the corresponding nucleic acid pair in the nucleotide sequence. For instance, if the nucleic acid pair AA appears $m$ times in the nucleotide sequence, the composition of the nucleic acid pair AA is equal to $m$ divided by the total number of *0*-spaced nucleic acid pairs ($N_{total}$) in the nucleotide sequence. For $k$ = 0, 1, 2, 3, 4 and 5, the value of $N_{total}$ is $P - 1$, $P - 2$, $P - 3$, $P - 4$, $P - 5$ and $P - 6$ for a nucleotide sequence of length $P$, respectively.

Use the following command to extract the CKSNAP feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method CKSNAP
```

Advanced usage:

```
tcsh% python descnucleotide/CKSNAP.py --file
examples/DNA_training.txt --gap 3
```

The parameters of CKSNAP.py are:

- *gap*: the *k*-space value for CKSNAP descriptor, it should be an integer larger than 0, default is 5

### 5.9 Nucleotide Chemical Property (NCP)

There are four different kinds of nucleotides in RNA, i.e., adenine (A), guanine (G), cytosine (C) and uracil (U). Each nucleotide has different chemical structure and chemical binding. The four kinds of nucleotides can be classified into three different groups in terms of these chemical properties (Table 1).

Table 1. Chemical structure of each nucleotide (5).

| Chemical property | Class | Nucleotides |
|---|---|---|
| Ring Structure | Purine | A, G |
| | Pyrimidine | C, U |
| Functional Group | Amino | A, C |
| | Keto | G, U |
| Hydrogen Bond | Strong | C, G |
| | Weak | A, U |

Based on chemical properties, A can be represented by coordinates (1, 1, 1), C can be represented by coordinates (0, 1, 0), G can be represented by coordinates (1, 0, 0), U can be represented by coordinates (0, 0, 1).

Use the following command to extract the NCP feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method NCP
```

### 5.10 Accumulated Nucleotide Frequency (ANF)

The Accumulated Nucleotide Frequency (ANF) encoding (5) include the nucleotide frequency information and the distribution of each nucleotide in the RNA sequence, the density $d_i$ of any nucleotide $s_i$ at position $i$ in RNA sequence by the following formula:

$$d_i = \frac{1}{|s_i|}\sum_{j=1}^{l} f(s_i),\ \ f(q) = \begin{cases} 1 & if\ s_i = q \\ 0 & other\ case \end{cases}$$

where $l$ is the sequence length, $|S_i|$ is the length of the $i$-th prefix string $\{s_1, s_2, \ldots, s_i\}$ in the

sequence, $q \in$ {A, C, G or U}. Suppose an example sequence "UCGUUCAUGG". The density of 'U' is 1 (1/1), 0.5 (2/4), 0.6 (3/5), 0.5 (4/8) at positions 1, 4, 5, and 8, respectively. The density of 'C' is 0.5 (1/2), 0.33 (2/6) at positions 2 and 6, respectively. The density of 'G' is 0.33 (1/3), 0.22 (2/9), 0.3 (3/10) at positions 3, 9, and 10, respectively. The density of 'A' is 0.14 (1/7) at position 7.

By integrating both the nucleotide chemical property and accumulated nucleotide information, the sample sequence "UCGUUCAUGG" can be represented by {(0, 0, 1, 1), (0, 1, 0, 0.5), (1, 0, 0, 0.33), (0, 0, 1, 0.5), (0, 0, 1, 0.6), (0, 1, 0, 0.33), (1, 1, 1, 0.14), (0, 0, 1, 0.5), (1, 0, 0, 0.22), (1, 0, 0, 0.3)}. By doing so, not only the chemical property was considered, but also the long-range sequence order information was incorporated. Therefore, the samples in the benchmark dataset were encoded in terms of both nucleotide chemical property and nucleotide densities.

Use the following command to extract the ANF feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method ANF
```

### 5.11.1 Position-specific trinucleotide propensity based on single-strand (PSTNPss)

The Position-specific trinucleotide propensity based on single-strand (PSTNPss) (6,7) using a statistical strategy based on single-stranded characteristics of DNA or RNA. There are $4^3 = 64$ trinucleotides: AAA, AAC, AAG, ..., TTT(UUU). So, for an $L$ bp sample, its details of the trinucleotides position specificity can be expressed by the following 64 × ($L$-2) matrix:

$$\mathrm{Z} = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,L-2} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,L-2} \\ \vdots & \vdots & \cdots & \vdots \\ z_{64,1} & z_{64,2} & \cdots & z_{64,L-2} \end{bmatrix}$$

where

$$z_{i,j} = F^{+}(3mer_i|j) - F^{-}(3mer_i|j), i = 1,2,\ldots,64; j = 1,2,\ldots L-2$$

$F^{+}(3mer_i|j)$ and $F^{-}(3mer_i|j)$ denote the frequency of the $i$-th trinucleotide ($3mer_i$) at the $j$-th position appear in the positive ($S^{+}$) and negative ($S^{-}$) data sets, respectively. In the formula, $3mer_1$ equals AAA,$3mer_2$ equals AAC, …, $3mer_{64}$ equals TTT.

Therefore, the sample can be expressed as:

$$\mathrm{S} = \left[\emptyset_1, \emptyset_2, \ldots, \emptyset_{L-2}\right]^T$$

where T is the operator of transpose and $\phi_u$ was defined as follows:

$$\emptyset_u = \begin{cases} z_{1,u}, & when\ N_u N_{u+1} N_{u+2} = AAA \\ z_{2,u}, & when\ N_u N_{u+1} N_{u+2} = AAG \\ & \vdots \\ z_{64,u}, & when\ N_u N_{u+1} N_{u+2} = TTT \end{cases}$$

Use the following command to extract the PSTNPss feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method PSTNPss
```

### 5.11.2 Position-specific trinucleotide propensity based on double-strand (PSTNPss)

Feature Position-specific trinucleotide propensity based on double-strand (PSTNPss) (6,7) using a statistical strategy based on double-stranded characteristics of DNA according to complementary base pairing, so they have more evident statistical features. At this point, we deem A and T as identical, the same to C and G. Thus, for every sample, it can be converted into a sequence contained A and T only.

Use the following command to extract the PSTNPds feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method PSTNPds
```

### 5.12.1 Electron-ion interaction pseudopotentials of trinucleotide (EIIP)

Nair (8) came up with electron-ion interaction pseudopotentials (EIIP) value of nucleotides A, G, C, T (A: 0.1260, C: 0.1340, G: 0.0806, T:0.1335). The EIIP directly use the EIIP value represent the nucleotide in the DNA sequence. Therefore, the dimension of the EIIP descriptor is the length of the DNA sequence.

Use the following command to extract the EIIP feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method EIIP
```

### 5.12.2 Electron-ion interaction pseudopotentials of trinucleotide (PseEIIP)

In these encoding, let $EIIP_A$, $EIIP_T$, $EIIP_G$, and $EIIP_C$ denote the EIIP values of nucleotides A, T, G and C, respectively. Then, the mean EIIP value of trinucleotides in each sample to construct feature vector, which can be formulated as:

$$V = [EIIP_{AAA} \cdot f_{AAA}, EIIP_{AAC} \cdot f_{AAC}, \ldots, EIIP_{TTT} \cdot f_{TTT}]$$

Where $f_{xyz}$ is the normalized frequency of the $i$-th trinucleotide, $EIIP_{xyz} = EIIP_x + EIIP_y + EIIP_z$ expresses the $EIIP$ value of one trinucleotide and X, Y, Z∈ [A, C, G, T]. Obviously, the dimension of vector $V$ is 64.

Use the following command to extract the PseEIIP feature descriptors:

```
tcsh% python iLearn-nucleotide-basic.py --file
examples/DNA_training.txt --method PseEIIP
```

## 5.13 Autocorrelation

The Autocorrelation encoding (9) can transform the nucleotide sequences of different lengths into fixed-length vectors by measuring the correlation between any two properties. Autocorrelation encoding can generate two kinds of variables (i.e. The autocorrelation (AC) between the same property, and the cross-covariance (CC) between two different properties). There are six types of autocorrelation encodings, 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). Users can run 'iLearn-nucleotide-acc.py' to get these six types of encodings.

The parameters of iLearn-nucleotide-acc.py are:

- *file*: the input sequence file with FASTA format
- *method*: the descriptor type (select from DAC, DCC, DACC, TAC, TCC, TACC)
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name
- *type*: the nucleotide type (DNA or RNA), default is DNA
- *lag*: an integer larger than or equal to 0 and less than or equal to *L*-2
- *index:* the physicochemical indices selection file, there are 148 physicochemical dinucleotides indices and 12 trinucleotides indices for DNA sequence (**Table 2 & Table 3**) and 22 physicochemical dinucleotides indices for RNA sequence (**Table 4**). If this parameter is not specified, the default physicochemical indices will be used. The default DNA dinucleotide indices are: Rise, Roll, Shift, Slide, Tilt, Twist, the default DNA trinucleotide indices are: 'Dnase I', 'Bendability (DNAse)'. And the default RNA dinucleotide indices are Rise (RNA), Roll (RNA), Shift (RNA), Slide (RNA), Tilt(RNA), Twist(RNA). The file should be written as follows:

```
Rise
Roll
Shift
Slide
Tilt
Twist
…
```

- *udi*: with this option, the users can use their own indices to generate the feature vector.
- *all_index*: with this option, all the physicochemical indices will be used to generate the feature vector. Its default value is False.

**Table 2. The names of the 148 physicochemical dinucleotides indices for DNA.**

| | | | | |
|---|---|---|---|---|
| Base stacking | Protein induced deformability | B-DNA twist | Propeller twist | Duplex stability:(freeenergy) |
| Duplex tability(disruptenergy) | Protein DNA twist | Stabilising energy of Z-DNA | Aida_BA_transition | Breslauer_dS |
| Electron_interaction | Hartman_trans_free_energy | Lisser_BZ_transition | Polar_interaction | SantaLucia_dG |
| Sarai_flexibility | Stability | Stacking_energy | Sugimoto_dS | Watson-Crick_interaction |
| Twist | Shift | Slide | Rise | Twist stiffness |
| Tilt stiffness | Shift_rise | Twist_shift | Enthalpy1 | Twist_twist |
| Shift2 | Tilt3 | Tilt1 | Slide (DNA-protein complex)1 | Tilt_shift |
| Twist_tilt | Roll_rise | Stacking energy | Stacking energy1 | Propeller Twist |
| Roll11 | Rise (DNA-protein complex) | Roll2 | Roll3 | Roll1 |
| Slide_slide | Enthalpy | Shift_shift | Flexibility_slide | Minor Groove Distance |
| Rise (DNA-protein complex)1 | Roll (DNA-protein complex)1 | Entropy | Cytosine content | Major Groove Distance |
| Twist (DNA-protein complex) | Purine (AG) content | Tilt_slide | Major Groove Width | Major Groove Depth |
| Free energy6 | Free energy7 | Free energy4 | Free energy3 | Free energy1 |
| Twist_roll | Flexibility_shift | Shift (DNA-protein complex)1 | Thymine content | Tip |

| | | | | |
|---|---|---|---|---|
| Keto (GT) content | Roll stiffness | Entropy1 | Roll_slide | Slide (DNA-protein complex) |
| Twist2 | Twist5 | Twist4 | Tilt (DNA-protein complex)1 | Twist_slide |
| Minor Groove Depth | Persistance Length | Rise3 | Shift stiffness | Slide3 |
| Slide2 | Slide1 | Rise1 | Rise stiffness | Mobility to bend towards minor groove |
| Dinucleotide GC Content | A-philicity | Wedge | DNA denaturation | Bending stiffness |
| Free energy5 | Breslauer_dG | Breslauer_dH | Shift (DNA-protein complex) | Helix-Coil_transition |
| Ivanov_BA_transition | Slide_rise | SantaLucia_dH | SantaLucia_dS | Minor Groove Width |
| Sugimoto_dG | Sugimoto_dH | Twist1 | Tilt | Roll |
| Twist7 | Clash Strength | Roll_roll | Roll (DNA-protein complex) | Adenine content |
| Direction | Probability contacting nucleosome core | Roll_shift | Shift_slide | Shift1 |
| Tilt4 | Tilt2 | Free energy8 | Twist (DNA-protein complex)1 | Tilt_rise |
| Free energy2 | Stacking energy2 | Stacking energy3 | Rise_rise | Tilt_tilt |
| Roll4 | Tilt_roll | Minor Groove Size | GC content | Inclination |
| Slide stiffness | Melting Temperature1 | Twist3 | Tilt (DNA-protein complex) | Guanine content |
| Twist6 | Major Groove Size | Twist_rise | Rise2 | Melting Temperature |
| Free energy | Mobility to bend towards major groove | Bend | | |

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

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

**Table 4. The names of the 22 physicochemical dinucleotides indices for RNA.**

| | | | | |
|---|---|---|---|---|
| Shift (RNA) | Hydrophilicity (RNA) | Hydrophilicity (RNA) | GC content | Purine (AG) content |
| Keto (GT) content | Adenine content | Guanine content | Cytosine content | Thymine content |
| Slide (RNA) | Rise (RNA) | Tilt (RNA) | Roll (RNA) | Twist (RNA) |
| Stacking energy (RNA) | Enthalpy (RNA) | Entropy (RNA) | Free energy (RNA) | Free energy (RNA) |
| Enthalpy (RNA) | Entropy (RNA) | | | |

### 5.13.1 Dinucleotide-based Auto Covariance (DAC)

The Dinucleotide-based Auto Covariance (DAC) encoding (9) measures the correlation of the same physicochemical index between two dinucleotide separated by a distance of *lag* along the sequence. The DAC can be calculated as:

$$DAC(u, lag) = \sum_{i=1}^{L-lag-1} ((P_u(R_iR_{i+1}) - \overline{P_u})(P_u(R_{i+lag}R_{i+lag+1}) - \overline{P_u})/(L - lag - 1))$$

where $u$ is a physicochemical index, $L$ is the length of the nucleotide sequence, $P_u(R_iR_{i+1})$ is the numerical value of the physicochemical index $u$ for the dinucleotide $R_iR_{i+1}$ at position $i$, $\overline{P_u}$ is the average value for physicochemical index $u$ along the whole sequence:

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

The dimension of the DAC feature vector is $N \times$ LAG, where $N$ is the number of physicochemical indices and LAG is the maximum of *lag* (*lag* = 1, 2, …, LAG).

Use the following command to extract the DAC feature descriptors:

```
tcsh% python iLearn-nucleotide-acc.py --file
examples/DNA_training.txt --method DAC --type DNA --lag 5
```

### 5.13.2 Dinucleotide-based Cross Covariance (DCC)

The Dinucleotide-based Cross Covariance (DCC) encoding (9) measures the correlation of two different physicochemical indices between two dinucleotides separated by *lag* nucleic acids along the sequence. The DCC encoding is calculated as:

$$\mathrm{DCC}(u_1, u_2, lag) = \sum_{i=1}^{L-lag-1} \left(P_{u_1}(R_i R_{i+1}) - \bar{P}_{u_1}\right)\left(P_{u_2}(R_{i+lag} R_{i+lag+1}) - \bar{P}_{u_2}\right)/(L - lag - 1)$$

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

$$\bar{P}_{u_a} = \sum_{j=1}^{L-1} P_{u_a}(R_j R_{j+1}) / (L-1)$$

The dimension of the DCC feature vector is $N \times (N\text{-}1) \times$ LAG, where $N$ is the number of physicochemical indices and LAG is the maximum of *lag* (*lag* = 1, 2, …, LAG).

Use the following command to extract the DCC feature descriptors:

```
tcsh% python iLearn-nucleotide-acc.py --file
examples/DNA_training.txt --method DCC --type DNA --lag 5
```

### 5.13.3 Dinucleotide-based Auto-Cross Covariance (DACC)

The Dinucleotide-based Auto-Cross Covariance (DACC) encoding (9) is a combination of DAC and DCC encoding. Thus, the dimension of the DACC encoding is $N \times N \times$ LAG, where $N$ is the number of physicochemical indices and LAG is the maximum of the *lag* (*lag* = 1, 2, …, LAG).

Use the following command to extract the DACC feature descriptors:

```
tcsh% python iLearn-nucleotide-acc.py --file
examples/DNA_training.txt --method DACC --type DNA --lag 5
```

### 5.13.4 Trinucleotide-based Auto Covariance (TAC)

The Trinucleotide-based Auto Covariance (TAC) encoding measures the correlation of the same physicochemical index between trinucleotides separated by lag nucleic acids along the sequence,

and can be calculated as:

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

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

$$\bar{P}_u = \sum_{j=1}^{L-2} P_u(R_iR_{i+1}R_{i+2}) /(L-2)$$

The dimension of the TAC feature vector is $N \times$ LAG, where $N$ is the number of physicochemical indices and LAG is the maximum of *lag* (*lag* = 1, 2, …, LAG).

Use the following command to extract the TAC feature descriptors:

```
tcsh% python iLearn-nucleotide-acc.py --file
examples/DNA_training.txt --method TAC --type DNA --lag 5
```

### 5.13.5 Trinucleotide-based Cross Covariance (TCC)

The Trinucleotide-based Cross Covariance (TCC) encoding measures the correlation of two different physicochemical indices between two trinucleotides separated by *lag* nucleic acids along the sequence. The TCC encoding can be calculated as:

$$\mathrm{DCC}(u_1,u_2,lag) = \sum_{i=1}^{L-lag-2} \left(P_{u_1}(R_iR_{i+1}R_{i+2}) - \bar{P}_{u_1}\right)\left(P_{u_2}(R_{i+lag}R_{i+lag+1}R_{i+lag+2}) - \bar{P}_{u_2}\right)/(L - lag - 2)$$

where $u_1$ and $u_2$ are different physicochemical indices, $L$ is the length of the nucleotide sequence, $R_iR_{i+1}R_{i+2}$ is the numerical value of the physicochemical index $u_a$ for the dinucleotide $R_iR_{i+1}R_{i+2}$ at position $i$, $\bar{P}_{u_a}$ is the average value for physicochemical index $u_a$ along the whole sequence:

$$\bar{P}_{u_a} = \sum_{j=1}^{L-1} P_{u_a}(R_jR_{j+1}R_{j+2}) /(L-2)$$

The dimension of the DCC feature vector is $N \times (N\text{-}1) \times$ LAG, where $N$ is the number of physicochemical indices and LAG is the maximum of *lag* (*lag* = 1, 2, …, LAG).

Use the following command to extract the DCC feature descriptors:

```
tcsh% python iLearn-nucleotide-acc.py --file
examples/DNA_training.txt --method TCC --type DNA --lag 5
```

### 5.13.6 Trinucleotide-based Auto-Cross Covariance (TACC)

Like DAC encoding, the Trinucleotide-based Auto-Cross Covariance (TACC) encoding (9) is a combination of TAC and TACC encoding. Thus, the dimension of the TACC encoding is $N \times N \times$

LAG, where *N* is the number of physicochemical indices and LAG is the maximum of the *lag* (*lag* = 1, 2, …, LAG).

Use the following command to extract the TACC feature descriptors:

```
tcsh% python iLearn-nucleotide-acc.py --file
examples/DNA_training.txt --method TACC --type DNA --lag 5
```

### 5.14 Pseudo Nucleic Acid Composition (PseNAC)

The Pseudo Nucleic Acid Composition (PseNAC) encodings consider both the local sequence-order information and long-range sequence-order effects (9). Six types of PseNAC encodings including 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) can be calculated by the 'iLearn-nucleotide-Pse.py' in iLearn package.

The parameters of iLearn-nucleotide-pse.py are:

- *file*: the input sequence file with FASTA format
- *method*: the descriptor type (select from PseDNC, PseKNC, PCPseDNC, PCPseTNC, SCPseDNC, SCPseTNC)
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name
- *type*: the nucleotide type (DNA or RNA), default is DNA
- *lamada*: an integer larger than or equal to 0 and less than or equal to *L*-2
- *weight*: the weight factor ranged from 0 to 1. Its default value is 0.05
- *kmer*: the value of kmer, it works only with PseKNC
- index: the physicochemical indices selection file, please refer to Table 2-4 for the names of the physicochemical dinucleotides or trinucleotides indices for DNA and RNA.

#### 5.14.1 Pseudo Dinucleotide Composition (PseDNC)

The Pseudo Dinucleotide Composition (PseDNC) encoding (10) incorporate contiguous local sequence-order information and the global sequence-order information into the feature vector of the nucleotide sequence. The PseDNC encoding is defined:

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

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 nucleotide sequence, λ represent the highest counted rank (or tie) of the correlation along the nucleotide sequence, *w* is the weight factor ranged from 0 to 1, and $\theta_j$ ($j$ =1,2, …, λ) is the $j$-tier correlation factor and is defined:

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

where the correlation function is defined:

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

where $\mu$ is the number of physicochemical indices. Six indices (i.e. 'Rise', 'Roll', 'Shift', 'Slide', 'Tilt', 'Twist') in Table 2 and six indices in Table 4 (i.e. 'Rise (RNA)', 'Roll (RNA)', 'Shift (RNA)', 'Slide (RNA)', 'Tilt (RNA)', 'Twist (RNA)') were set as the default indices for DNA and RNA sequences, separately. $P_u(R_iR_{i+1})$ is the numerical value of the $u$-th ($u$=1, 2, …, $\mu$) physicochemical index of the dinucleotide $R_iR_{i+1}$ at position $i$ and $P_u\left(R_jR_{j+1}\right)$ represents the corresponding value of the dinucleotide $R_jR_{j+1}$ at position $j$.

Use the following command to extract the PseDNC feature descriptors:

```
tcsh% python iLearn-nucleotide-Pse.py --file
examples/DNA_training.txt --method PseDNC --type DNA --lamada 2
--weight 0.1
```

### 5.14.2 Pseudo *k*-tupler Composition (PseKNC)

The Pseudo *k*-tupler Composition (PseKNC) encoding (11) incorporate the *k*-tuple nucleotide composition, which can be defined as:

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

where

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

where λ is the number of the total counted ranks (or tiers) of the correlations along a nucleotide 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 the factor, and $\theta_j$ is defined:

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

The correlation function $\Theta\left(R_iR_{i+1}, R_{i+j}R_{i+j+1}\right)$ is defined as:

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

where $\mu$ is the number of physicochemical indices. Six indices (i.e. 'Rise', 'Roll', 'Shift', 'Slide', 'Tilt', 'Twist') in Table 2 and six indices in Table 4 (i.e. 'Rise (RNA)', 'Roll (RNA)', 'Shift (RNA)', 'Slide (RNA)', 'Tilt (RNA)', 'Twist (RNA)') were set as the default indices for DNA and RNA sequences, separately. $P_v(R_iR_{i+1})$ is the numerical value of the $v$-th ($v$=1, 2, …, $\mu$) physicochemical index of the dinucleotide $R_iR_{i+1}$ at position $i$ and $P_v\left(R_{j+j}R_{i+j+1}\right)$ represents the corresponding value of the dinucleotide $R_{i+j}R_{i+j+1}$ at position $i$+$j$.

Use the following command to extract the PseKNC feature descriptors:

```
tcsh% python iLearn-nucleotide-Pse.py --file
examples/DNA_training.txt --method PseKNC --type DNA --lamada 2
--weight 0.1 --kmer 3
```

### 5.14.3 Parallel Correlation Pseudo Dinucleotide Composition (PCPseDNC)

The Parallel Correlation Pseudo Dinucleotide Composition (PCPseDNC) encoding has the same definition with the PseDNC, the different is PCPseDNC encoding used 38 default physiochemical indices instead of the six physiochemical indices in PseDNC encoding for DNA. The 38 physiochemical indices are listed in Table 5.

Use the following command to extract the PCPseDNC feature descriptors:

```
tcsh% python iLearn-nucleotide-Pse.py --file
examples/DNA_training.txt --method PCPseDNC --type DNA --lamada 2
--weight 0.1
```

**Table 5. The names of the 38 physicochemical dinucleotides indices for RNA.**

| | | | | |
|---|---|---|---|---|
| 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 | | |

### 5.14.4 Parallel Correlation Pseudo Trinucleotide Composition (PCPseTNC)

The Parallel Correlation Pseudo Trinucleotide Composition (PCPseTNC) encoding (12,13) is

defined as:

$$D = [d_1, d_2, \ldots, d_{64}, d_{64+1}, \ldots, d_{64+\lambda}]^T$$

where

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

where $f_k$ (k=1, 2, …, 64) is the normalized occurrence frequency of trinucleotide in the DNA sequence, λ represent the highest counted rank (or tie) of the correlation along the DNA sequence, $w$ is the weight factor ranged from 0 to 1, and $\theta_j$ ($j$ =1,2, …, λ) is the $j$-tier correlation factor and is defined:

$$\begin{cases} \theta_1 = \frac{1}{L-3}\sum_{i=1}^{L-3}\Theta(R_iR_{i+1}R_{i+2}, R_{i+1}R_{i+2}R_{i+3}) \\ \theta_2 = \frac{1}{L-4}\sum_{i=1}^{L-4}\Theta(R_iR_{i+1}R_{i+2}, R_{i+2}R_{i+3}R_{i+4}) \\ \theta_3 = \frac{1}{L-5}\sum_{i=1}^{L-5}\Theta(R_iR_{i+1}R_{i+2}, R_{i+3}R_{i+4}R_{i+5}) \qquad (\lambda < L) \\ \cdots \\ \theta_\lambda = \frac{1}{L-2-\lambda}\sum_{i=1}^{L-2-\lambda}\Theta(R_iR_{i+1}R_{i+2}, R_{i+\lambda}R_{i+\lambda+1}R_{i+\lambda+2}) \end{cases}$$

where the correlation function is defined:

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

where $\mu$ is the number of physicochemical indices. Two indices (i.e. 'Dnase I', 'Bendability (DNAse)' in Table 3 was set as the default indices for DNA sequences. $P_u(R_iR_{i+1}R_{i+2})$ is the numerical value of the $u$-th ($u$=1, 2, …, $\mu$) physicochemical index of the dinucleotide $R_iR_{i+1}R_{i+2}$ at position $i$ and $P_u\left(R_jR_{j+1}R_{j+2}\right)$ represents the corresponding value of the dinucleotide $R_jR_{j+1}R_{j+2}$ at position $j$.

Use the following command to extract the PCPseTNC feature descriptors:

```
tcsh% python iLearn-nucleotide-Pse.py --file
examples/DNA_training.txt --method PCPseTNC --type DNA --lamada 2
--weight 0.1
```

### 5.14.5 Series Correlation Pseudo Dinucleotide Composition (SCPseDNC)

The Series Correlation Pseudo Dinucleotide Composition (SCPseDNC) encoding (12) is defined as:

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

where

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

where $f_k$ (k=1, 2, …, 16) is the normalized occurrence frequency of dinucleotide in the nucleotide sequence, λ represent the highest counted rank (or tie) of the correlation along the nucleotide sequence, $w$ is the weight factor ranged from 0 to 1, Λ is the number of physicochemical indices and $\theta_j$ ($j$ =1,2, …, λ) is the $j$-tier correlation factor and is defined:

$$\begin{cases} \theta_1 = \frac{1}{L-3}\sum_{i=1}^{L-3} J_{i,i+1}^{1} \\ \theta_2 = \frac{1}{L-3}\sum_{i=1}^{L-3} J_{i,i+1}^{2} \\ \cdots \\ \theta_\Lambda = \frac{1}{L-3}\sum_{i=1}^{L-3} J_{i,i+1}^{\Lambda} \\ \cdots \\ \theta_{\lambda\Lambda} = \frac{1}{L-\lambda-2}\sum_{i=1}^{L-\lambda-2} J_{i,i+\lambda}^{\Lambda-1} \\ \theta_{\lambda\Lambda} = \frac{1}{L-\lambda-2}\sum_{i=1}^{L-\lambda-2} J_{i,i+\lambda}^{\Lambda} \end{cases} \quad (\lambda < L-2)$$

where the correlation function is defined:

$$\begin{cases} J_{i,i+m}^{\zeta} = P_u(R_iR_{i+1})P_u(R_{i+m}R_{i+m+1}) \\ \zeta = 1,2,\dots,\Lambda; m = 1,2,\dots,\lambda; i = 1,2,\dots,L-\lambda-2 \end{cases}$$

where $\mu$ is the number of physicochemical indices. Six indices (i.e. 'Rise', 'Roll', 'Shift', 'Slide', 'Tilt', 'Twist') in Table 2 and six indices in Table 4 (i.e. 'Rise (RNA)', 'Roll (RNA)', 'Shift (RNA)', 'Slide (RNA)', 'Tilt (RNA)', 'Twist (RNA)') were set as the default indices for DNA and RNA sequences, separately. $P_u(R_iR_{i+1})$ is the numerical value of the $u$-th ($u$=1, 2, …, $\mu$) 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 $j$.

Use the following command to extract the SCPseDNC feature descriptors:

```
tcsh% python iLearn-nucleotide-Pse.py --file
examples/DNA_training.txt --method SCPseDNC --type DNA --lamada 2
--weight 0.1
```

### 5.14.6 Series Correlation Pseudo Trinucleotide Composition (SCPseTNC)

The Series Correlation Pseudo Trinucleotide Composition (SCPseTNC) encoding (12) is defined as:

$$D = [d_1, d_2, \dots, d_{64}, d_{64+1}, \dots, d_{64+\lambda}, d_{64+\lambda+1}, d_{64+\lambda+1}, \dots, d_{64+\lambda\Lambda}]^T$$

where

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

where $f_k$ ($k$=1, 2, …, 64) is the normalized occurrence frequency of trinucleotide in the DNA sequence, λ represent the highest counted rank (or tie) of the correlation along the DNA sequence, $w$ is the weight factor ranged from 0 to 1, Λ is the number of physicochemical indices and $\theta_j$ ($j$ =1,2, …, λ) is the $j$-tier correlation factor and is defined:

$$\begin{cases} \theta_1 = \frac{1}{L-4}\sum_{i=1}^{L-4} J_{i,i+1}^1 \\ \theta_2 = \frac{1}{L-4}\sum_{i=1}^{L-4} J_{i,i+1}^2 \\ \cdots \\ \theta_\Lambda = \frac{1}{L-4}\sum_{i=1}^{L-4} J_{i,i+1}^\Lambda \quad (\lambda < L-3) \\ \cdots \\ \theta_{\lambda\Lambda} = \frac{1}{L-\lambda-3}\sum_{i=1}^{L-\lambda-3} J_{i,i+\lambda}^{\Lambda-1} \\ \theta_{\lambda\Lambda} = \frac{1}{L-\lambda-3}\sum_{i=1}^{L-\lambda-3} J_{i,i+\lambda}^{\Lambda} \end{cases}$$

where the correlation function is defined:

$$\begin{cases} J_{i,i+m}^{\zeta} = P_u(R_iR_{i+1})P_u(R_{i+m}R_{i+m+1}R_{i+m+2}) \\ \zeta = 1,2,\ldots,\Lambda; m = 1,2,\ldots,\lambda; i = 1,2,\ldots,L-\lambda-3 \end{cases}$$

where $\mu$ is the number of physicochemical indices. Two indices (i.e. 'Dnase I', 'Bendability (DNAse)' in Table 3 was set as the default indices for DNA sequences. $P_u(R_iR_{i+1}R_{i+2})$ is the numerical value of the $u$-th ($u$=1, 2, …, $\mu$) physicochemical index of the dinucleotide $R_iR_{i+1}R_{i+2}$ at position $i$ and $P_u(R_jR_{j+1}R_{j+2})$ represents the corresponding value of the dinucleotide $R_jR_{j+1}R_{j+2}$ at position $j$.

Use the following command to extract the SCPseTNC feature descriptors:

```
tcsh% python iLearn-nucleotide-Pse.py --file
examples/DNA_training.txt --method SCPseTNC --type DNA --lamada 2
--weight 0.1
```

## 6. Commonly Used Feature Descriptors for protein sequences

There are two main programs in *iLearn* package that are used to generate descriptors for protein/peptide sequences (i.e. iLearn-protein-basic.py and iLearn-protein-PseKRAAC.py). The description for these 53 types of protein descriptors have been introduced in our previously published *iFeature* (1) package. For more information, please refer to (1). Here we only briefly introduce the usage for each of the protein descriptor.

37 encoding schemes can be generated by iLearn-protein-basic.py, The parameters of

iLearn-protein-basic.py are:

- *file*: the input sequence file with FASTA format
- *method*: the descriptor type (select from Table 6)
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name
- *path*: data file path used for 'PSSM', 'SSEB(C)', 'Disorder(B/C)', 'ASA' and 'TA' encodings

### 6.1 Amino Acid Composition (AAC)

The Amino Acid Composition (AAC) encoding (14) calculates the frequency of each amino acid type in a protein or peptide sequence.

Use the following command to extract the AAC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method AAC
```

### 6.2 Enhanced Amino Acid Composition (EAAC)

The Enhanced Amino Acid Composition (EAAC) feature calculates the AAC based on the sequence window of fixed length (the default value is 5) that continuously slides from the N- to C-terminus of each peptide and can be usually applied to encode the peptides with an equal length.

Use the following command to extract the EAAC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method EAAC
```

Advanced users can adjust the size of the sliding window by running the 'EAAC.py' in the directory of 'descproteins'. The parameters of 'EAAC.py' are:

- *file*: the input sequence file with FASTA format
- *slwindow*: the size of sliding window, default is 5
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the sliding window to 3:

```
tcsh% python descproteins/EAAC.py --file
examples/peptide_sequences.txt --slwindow 3
```

### 6.3 Composition of k-spaced Amino Acid Pairs (CKSAAP)

The CKSAAP feature encoding calculates the frequency of amino acid pairs separated by any $k$ residues ($k$ = 0, 1, 2, … , 5. The default maximum value of $k$ is 5) (15-18).

Use the following command to extract the CKSAAP feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method CKSAAP
```

Advanced users can adjust the size of the sliding window by running the 'CKSAAP.py' in the directory of 'descproteins'. The parameters of 'CKSAAP.py' are:

- *file*: the input sequence file with FASTA format
- *gap*: the *k*-space value, default is 5
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the gap value to 3:

```
tcsh% python descproteins/CKSAAP.py --file
examples/protein_sequences.txt --gap 3
```

**6.4 Tri-Peptide Composition (TPC)**

The Tripeptide Composition (TPC) (14) gives 8000 descriptors.

Use the following command to extract the CKSAAP feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method TPC
```

**6.5 Grouped Amino Acid Composition (GAAC)**

In the GAAC encoding, the 20 amino acid types are further categorized into five classes according to their physicochemical properties, e.g. hydrophobicity, charge and molecular size (19). The five classes include the aliphatic group (*g1*: GAVLMI), aromatic group (*g2*: FYW), positive charge group (*g3*: KRH), negative charged group (*g4*: DE) and uncharged group (*g5*: STCPNQ).

Use the following command to extract the CKSAAP feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method GAAC
```

**6.6 Enhanced GAAC (EGAAC)**

The Enhanced GAAC (EGAAC) is also for the first time proposed in this work. It calculates GAAC in windows of fixed length (default is 5) continuously sliding from the N- to C-terminal of each peptide and is usually applied to peptides with an equal length.

Use the following command to extract the EGAAC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method EGAAC
```

Use the following command to extract the EGAAC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method EGAAC
```

Advanced users can adjust the size of the sliding window by running the 'EGAAC.py' in the directory of 'descproteins'. The parameters of 'EGAAC.py' are:

- *file*: the input sequence file with FASTA format

- *slwindow*: the size of sliding window, default is 5
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the sliding window to 3:

```
tcsh% python descproteins/EGAAC.py --file
examples/peptide_sequences.txt --slwindow 3
```

### 6.7 Composition of k-Spaced Amino Acid Group Pairs (CKSAAGP)

The Composition of *k*-Spaced Amino Acid Group Pairs (CKSAAGP) is a variation of the CKSAAP descriptor, which is our own proposal. It calculates the frequency of amino acid group pairs separated by any *k* residues (the default maximum value of *k* is set as 5).

Use the following command to extract the CKSAAGP feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method CKSAAGP
```

Advanced users can adjust the size of the sliding window by running the 'CKSAAGP.py' in the directory of 'descproteins'. The parameters of 'CKSAAGP.py' are:

- *file*: the input sequence file with FASTA format
- *gap*: the *k*-space value, default is 5
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the gap value to 3:

```
tcsh% python descproteins/CKSAAGP.py --file
examples/protein_sequences.txt --gap 3
```

### 6.8 Grouped Tri-Peptide Composition (GTPC)

The Grouped Tri-Peptide Composition encoding is also a variation of TPC descriptor, which generates 125 descriptors.

Use the following command to extract the GTPC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method GTPC
```

### 6.9 Binary (binary)

In the Binary encoding (20,21), each amino acid is represented by a 20-dimensional binary vector, e.g. A is encoded by (10000000000000000000), C is encoded by (01000000000000000000), …, Y is encoded by (00000000000000000001), respectively. This encoding scheme is often used to encode peptides with an equal length.

Use the following command to extract the binary feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
```

```
examples/protein_sequences.txt --method binary
```

### 6.10 Moran correlation (Moran)

The autocorrelation descriptors are defined based on the distribution of amino acid properties along the sequence (22-24). The amino acid properties used here are different types of amino acids index, which is retrieved from the AAindex Database (25) available at http://www.genome.jp/dbget/aaindex.html. The eight indices 'CIDH920105', 'BHAR880101', 'CHAM820101', 'CHAM820102', 'CHOC760101', 'BIGC670101', 'CHAM810101', 'DAYM780201' are used (26) as default.

Use the following command to extract the Moran feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method Moran
```

Advanced users can select the property and adjust the maximum value of the *nlag* (default is 30) by running the 'Moran.py' in the directory of 'descproteins'. The parameters of 'Moran.py' are:

- *file*: the input sequence file with FASTA format
- *props*: input the property names, the names were separated by the symbol ':'
- *nlag*: set the value of *nlag*, default is 30
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, select two property and set the *nlag* value is 15:

```
tcsh% python descproteins/Moran.py --file
examples/protein_sequences.txt --props CIDH920105:BHAR880101 --nlag
15
```

### 6.11 Geary correlation (Geary)

The Geary autocorrelation descriptors (24) is also a type of autocorrelation descriptor.

Use the following command to extract the Geary feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method Geary
```

Advanced users can select the property and adjust the maximum value of the *nlag* (default is 30) by running the 'Moran.py' in the directory of 'descproteins'. The parameters of 'Moran.py' are:

- *file*: the input sequence file with FASTA format
- *props*: input the property names, the names were separated by the symbol ':'
- *nlag*: set the value of *nlag*, default is 30
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, select two property and set the *nlag* value is 15:

```
tcsh% python descproteins/Geary.py --file
```

```
examples/protein_sequences.txt --props CIDH920105:BHAR880101 --nlag
15
```

### 6.12 Normalized Moreau-Broto Autocorrelation (NMBroto)

The Moreau-Broto autocorrelation descriptors (23) is also a type of autocorrelation descriptors.

Use the following command to extract the NMBroto feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method NMBroto
```

Advanced users can select the property and adjust the maximum value of the *nlag* (default is 30) by running the 'Moran.py' in the directory of 'descproteins'. The parameters of 'NMBroto.py' are:

- *file*: the input sequence file with FASTA format
- *props*: input the property names, the names were separated by the symbol ':'
- *nlag*: set the value of *nlag*, default is 30
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, select two property and set the *nlag* value is 15:

```
tcsh% python descproteins/Geary.py --file
examples/protein_sequences.txt --props CIDH920105:BHAR880101 --nlag
15
```

### 6.13 Composition/Transition/Distribution (CTD)

The Composition, Transition and Distribution (CTD) features represent the amino acid distribution patterns of a specific structural or physicochemical property in a protein or peptide sequence (27-31). Seven types of physicochemical properties have been previously used for computing these features. These include hydrophobicity, normalized Van der Waals Volume, polarity, polarizability, charge, secondary structures and solvent accessibility. These descriptors are calculated according to the following procedures: (i) The sequence of amino acids is transformed into a sequence of certain structural or physicochemical properties of residues; (ii) Twenty amino acids are divided into three groups for each of the seven different physicochemical attributes based on the main clusters of the amino acid indices of Tomii and Kanehisa (32). The groups of amino acids are listed in **Table 6**.

**Table 6.** Amino acid physicochemical attributes and the division of the amino acids into three groups according to each attribute.

| Attribute | Division | | |
|---|---|---|---|
| Hydrophobicity | Polar: RKEDQN | Neutral: GASTPHY | Hydrophobicity: CLVIMFW |
| Normalized van der Waals volume | Volume range: 0-2.78 GASTPD | Volume range: 2.95-94.0 NVEQIL | Volume range: 4.03-8.08 MHKFRYW |
| Polarity | Polarity value: | Polarity value: 8.0-9.2 | Polarity value: |

| | | | |
|---|---|---|---|
| | 4.9-6.2<br>LIFWCMVY | PATGS | 10.4-13.0<br>HQRKNED |
| Polarizability | Polarizability value: 0-1.08<br>GASDT | Polarizability value: 0.128-120.186<br>GPNVEQIL | Polarizability value: 0.219-0.409<br>KMHFRYW |
| Charge | Positive: KR | Neutral: ANCQGHILMFPSTWYV | Negative: DE |
| Secondary structure | Helix: EALMQKRH | Strand: VIYCWFT | Coil: GNPSD |
| Solvent accessibility | Buried: ALFCGIVW | Exposed: PKQEND | Intermediate: MPSTHY |

### 6.13.1 CTDC

Use the following command to extract the CTDC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method CTDC
```

### 6.13.2 CTDT

The Transition descriptor T also consists of three values (27,28): A transition from the polar group to the neutral group is the percentage frequency with which a polar residue is followed by a neutral residue or a neutral residue by a polar residue.

Use the following command to extract the CTDT feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method CTDT
```

### 6.13.2 CTDD

The Distribution descriptor consists of five values for each of the three groups (polar, neutral and hydrophobic) (27,28), namely the corresponding fraction of the entire sequence, where the first residue of a given group is located, and where 25, 50, 75 and 100% of occurrences are contained.

Use the following command to extract the CTDD feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method CTDD
```

## 6.14 Conjoint Triad (CTriad)

The Conjoint Triad descriptor (CTriad) considers the properties of one amino acid and its vicinal amino acids by regarding any three continuous amino acids as a single unit (33).

Use the following command to extract the CTriad feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method CTriad
```

## 6.15 *k*-Spaced Conjoint Triad (KSCTriad)

The *k*-Spaced Conjoint Triad (KSCTriad) descriptor is based on the Conjoint CTriad descriptor, which not only calculates the numbers of three continuous amino acid units, but also considers the continuous amino acid units that are separated by any *k* residues (The default maximum value of *k* is set to 5).

Use the following command to extract the KSCTriad feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method KSCTriad
```

Advanced users can adjust the value of *k* to <*k*> by running the 'KSCTriad.py' in the directory of 'descproteins'. The parameters of KSCTriad.py' are:

- *file*: the input sequence file with FASTA format
- *gap*: the *k*-space value, default is 5
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the gap value to 3:

```
tcsh% python descproteins/KSCTriad.py --file
examples/protein_sequences.txt --gap 3
```

**6.16 Sequence-Order-Coupling Number (SOCNumber)**

Use the following command to extract the SOCNumber feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method SOCNumber
```

Advanced users can adjust the value of *lag* (ingeger, default is 30) by running the 'SOCNumber.py' in the directory of 'descproteins'. The parameters of 'SOCNumber.py' are:

- *file*: the input sequence file with FASTA format
- *lag*: the lag value for SOCNumber, default is 30
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the *lag* value to 10:

```
tcsh% python descproteins/SOCNumber.py --file
examples/protein_sequences.txt --lag 10
```

**6.17 Quasi-sequence-order (QSOrder)**

Use the following command to extract the QSOrder feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method QSOrder
```

Advanced users can adjust the value of *lag* (ingeger, default is 30) by running the 'SOCNumber.py' in the directory of 'descproteins'. The parameters of 'SOCNumber.py' are:

- *file*: the input sequence file with FASTA format

- *lag*: the lag value for QSOrder, default is 30
- *weight*: the weight factor, ranged from 0-1, default is 0.1
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the *lag* value to 10 and set the weight factor is 0.2:

```
tcsh% python descproteins/QSOrder.py --file
examples/protein_sequences.txt --lag 10 --weight 0.2
```

**6.18 Pseudo-Amino Acid Composition (PAAC)**

This group of descriptors has been proposed in (34,35).

Use the following command to extract the PAAC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method PAAC
```

Advanced users can adjust the value of $\lambda$ (ingeger, default is 30) and weight factor (default is 0.05) by running the 'PAAC.py' in the directory of 'descproteins'. The parameters of 'PAAC.py' are:

- *file*: the input sequence file with FASTA format
- *lamada*: the lamada value for PAAC.py, default is 30
- *weiht*: the weight factor for PAAC.py, default is 0.05
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the $\lambda$ value to 10 and set the weight factor is 0.1:

```
tcsh% python descproteins/PAAC.py --file
examples/protein_sequences.txt --lamada 10 --weight 0.1
```

**6.19 Amphiphilic Pseudo-Amino Acid Composition (APAAC)**

Amphiphilic Pseudo-Amino Acid Composition (APAAC) was proposed in (34,35).

Use the following command to extract the APAAC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/protein_sequences.txt --method APAAC
```

Advanced users can adjust the value of $\lambda$ (ingeger, default is 30) by running the 'APAAC.py' in the directory of 'descproteins'. The parameters of 'APAAC.py' are:

- *file*: the input sequence file with FASTA format
- *lamada*: the lamada value for PAAC.py, default is 30
- *weiht*: the weight factor for PAAC.py, default is 0.05

- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, adjust the $\lambda$ value to 10 and set the weight factor is 0.1:

```
tcsh% python descproteins/APAAC.py --file
examples/protein_sequences.txt --lamada 10 --weight 0.1
```

### 6.20 *K*-Nearest Neighbor for peptides (KNNpeptide)

The *K*-Nearest Neighbor for peptides (KNNpeptide) descriptor (36) requires an extra training file and a label file. The training file is used to calculate the top *K*-Nearest Neighbor peptides by calculating the similarity score of two peptide sequences.

Use the following command to extract the KNNpeptide feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide_sequences.txt --method KNNpeptide
```

### 6.21 *K*-Nearest Neighbor for proteins (KNNprotein)

The *K*-Nearest Neighbor for Proteins (KNNProtein) descriptor is similar to the KNNpeptide descriptor. The only difference between these two descriptors is the way similarity is calculated. In KNNprotein the similarity score of two protein sequences is obtained by applying the Needleman-Wunsch algorithm (37).

Use the following command to extract the KNNprotein feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide_sequences.txt --method KNNprotein
```

### 6.22 PSSM profile (PSSM)

This feature descriptor (38,39) is extracted from the Position-Specific Scoring Matrix (PSSM) profile. The PSSM profile can be obtained by running PSI-BLAST (40) against the uniref 50 database. The PSSM descriptor is usually applied to encode the peptides with equal length. Each amino acid in the peptide is represented by a 20-dimensional vector.

Use the following command to extract the PSSM feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method PSSM --path
examples/predictedProteinProperty
```

### 6.23 AAindex (AAINDEX)

Physicochemical properties of amino acids are the most intuitive features for representing biochemical reactions and have been extensively applied in bioinformatics research. The amino acid indices (AAindex) database (25) collects many published indices representing physicochemical properties of amino acids. For each physicochemical property, there is a set of 20 numerical values for all amino acids. Currently, 544 physicochemical properties can be retrieved

from the AAindex database. After removing physicochemical properties with value 'NA' for any of the amino acids, 531 physicochemical properties were left. In contrast to the residue-based encoding methods of amino acid identity and evolutionary information, a vector of 531 mean values is used to represent a sample for various window sizes. The AAINDEX descriptor (41) can be applied to encode peptides of equal length.

Use the following command to extract the AAINDEX feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method AAINDEX
```

Advanced users can select the properties by running the 'AAINDEX.py' in the directory of 'descproteins'. The parameters of 'AAINDEX.py' are:

- *file*: the input sequence file with FASTA format
- *props*: input the property names, the names were separated by the symbol ':'
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name

For example, select two property:

```
tcsh% python descproteins/AAINDEX.py --file
examples/peptide_sequences.txt --props CIDH920105:BHAR880101
```

**6.24 BLOSUM62 (BLOSUM62)**

In this descriptor, the BLOSUM62 matrix is employed to represent the protein primary sequence information as the basic feature set. A matrix comprising of $m \times n$ elements is used to represent each residue in a training dataset, where $n$ denotes the peptide length and $m = 20$, which elements comprise 20 amino acids. Each row in the BLOSUM62 matrix is adopted to encode one of 20 amino acids. The BLOSUM62 descriptor (42) can be applied to encode peptides of equal length.

Use the following command to extract the BLOSUM62 feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide_sequences.txt --method BLOSUM62
```

**6.25 Secondary Structure Elements Content (SSEC)**

Protein secondary structure was first predicted by the PSIPRED V4.0 software (43).

Use the following command to extract the SSEC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method SSEC --path
examples/predictedProteinProperty
```

**6.26 Secondary Structure Elements Binary (SSEB)**

In the Secondary Structure Elements Binary (SSEB) descriptor, each residue in a peptide is represented by a 3-dimensional vector, i.e. Helix (001), Strand (010), Coil (100). The SSEB descriptor can be applied to encode peptides of equal length.

Use the following command to extract the SSEB feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method SSEB --path
examples/predictedProteinProperty
```

**6.27 Disorder (Disorder)**

Protein disorder information was first predicted by the VSL2 software (44,45). The predicted probability value is taken as the feature. The Disorder descriptor (38,46) can be applied to encode peptides of equal length.

Use the following command to extract the Disorder feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method Disorder --path
examples/predictedProteinProperty
```

**6.28 Disorder Content (DisorderC)**

Use the following command to extract the DisorderC feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method DisorderC --path
examples/predictedProteinProperty
```

**6.29 Disorder Binary (DisorderB)**

For the Disorder Binary (DisorderB) descriptor, each residue in a peptide sequence is represented by a 2-dimensional vector, namely an order residue by (10) and a disorder residue by (01). The DisorderB descriptor can be applied to encode peptides of equal length.

Use the following command to extract the DisorderB feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method DisorderB --path
examples/predictedProteinProperty
```

**6.30 Accessible Solvent accessibility (ASA)**

The protein Accessible Solvent Accessibility information was first predicted by the SPINE-X software (47-49). The predicted ASA value is used as input feature. The ASA descriptor can be applied to encode peptides with an equal length.

Use the following command to extract the ASA feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method ASA --path
examples/predictedProteinProperty
```

**6.31 Torsion angle (TA)**

The protein Torsion Angle information was also introduced first by the SPINE-X software (47-49). The predicted "phi" and "psi" values are used as features. The TA descriptor can be applied to encode peptides of equal length.

Use the following command to extract the TA feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide1_sequences.txt --method TA --path
examples/predictedProteinProperty
```

**6.32 Z-Scale (ZSCALE)**

For this descriptor, each amino acid is characterized by five physicochemical descriptor variables (cf. Table 2), which were developed by Sandberg et al. in 1998 (50). The ZSCALE descriptor (51) can be applied to encode peptides of equal length.

Use the following command to extract the ZSCALE feature descriptors:

```
tcsh% python iLearn-protein-basic.py --file
examples/peptide_sequences.txt --method ZSCALE
```

**6.32 pseudo K-tuple reduced amino acids composition (PseKRAAC)**

16 PseKRAAC encoding schemes can be generated by iLearn-protein-PseKRAAC.py, The parameters of iLearn-protein-PseKRAAC.py are:

- *file*: the input sequence file with FASTA format
- *method*: the descriptor type (select from type1, type2, type3A, type3B, type4, type5, type6A, type6B, type6C, type7, type8, type9, type10, type11, type12, type13, type14, type15, type16)
- *model*: feature types for protein sequence analysis, two alternative modes (g-gap and lambda-correlation) are available, with the 'g-gap' model as the default
- *ktuple*: *K*-tuple value, three K-tuple values (i.e. 1, 2 and 3) are available, default is 2
- *gap_lambda*: gap value for the 'g-gap' model or lambda value for the 'lambda-correlation' model, 10 values are available (i.e. 0, 1, 2, …, 9)
- *type*: the reduced amino acids cluster type
- *format*: the output format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *out*: the output file name
- *show*: show detailed available "--type" value for each type

Users can run the following command to view the available values for each descriptor type:

```
tcsh% python iLearn-protein-PseKRAAC.py --show
```

Use the following command to extract the PseKRAAC feature descriptors:

```
tcsh% python iLearn-protein-PseKRAAC.py --file
examples/protein_sequences.txt --method type1 --model
lambda-correlation --ktuple 2 --gap_lambda 2 --type 5
```

# 7. Feature Analysis Using *iLearn*

*iLearn* integrates several commonly used and useful clustering, feature selection, feature normalization, and dimensionality reduction algorithms. The clustering algorithms can be implemented by running the 'iLearn-clustering.py'. The parameters of iLearn-clustering.py are:

- *file*: the encoding file, and a '**tsv_1**' file format is required
- *method*: cluster algorithm (select from kmeans, hcluster, apc, meanshift and dbscan)
- *sof*: cluster for sample or feature, default is sample
- *ktuple*: *K*-tuple value, three K-tuple values (i.e. 1, 2 and 3) are available, default is 2
- *nclusters*: specify the cluster number for kmeans cluster algorithm, default is 3
- *type*: the reduced amino acids cluster type
- *out*: the output file name

## 7.1 *K*-Means clustering (kmeans)

The *K*-Means algorithm clusters data by trying to separate samples in *n* groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (52,53). This algorithm requires the number of clusters to be specified. It scales well to large numbers of samples and has been used across a broad range of application areas.

Use the following command to perform the *K*-Means clustering:

```
tcsh% python iLearn-clustering.py --file examples/code_for_cluster.txt --method kmeans --sof sample --nclusters 2
```

## 7.2 Gaussian Mixture clustering (gmm)

The Gaussian mixture model (GMM) attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset.

Use the following command to perform the *gmm* clustering:

```
tcsh% python iLearn-clustering.py --file examples/code_for_cluster.txt --method gmm --sof sample --nclusters 2
```

## 7.3 Hierarchical clustering (hcluster)

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively (52-54). This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.

Use the following command to perform the hcluster clustering:

```
tcsh% python iLearn-clustering.py --file
examples/code_for_cluster.txt --method hcluster
```

### 7.4 Affinity Propagation clustering (apc)

Affinity Propagation creates clusters by sending messages between pairs of samples until convergence (55). A dataset is then described using a small number of exemplars, which are identified as the most representative of samples. The messages sent between pairs represent the suitability for one sample to be the exemplar of the other, which is updated in response to the values from other pairs. This updating happens iteratively until convergence has been achieved, at which point the final exemplars are chosen, and hence the final clustering is given.

Use the following command to perform the apc clustering:

```
tcsh% python iLearn-clustering.py --file
examples/code_for_cluster.txt --method apc
```

### 7.5 Mean Shift clustering (meanshift)

MeanShift clustering aims to discover blobs in a smooth density of samples (56). It is a centroid based algorithm, which works by updating candidates for centroids that are the mean of the points within a given region. These candidates are then filtered in a post-processing stage to eliminate near-duplicates and to form the final set of centroids.

Use the following command to perform the meanshift clustering:

```
tcsh% python iLearn-clustering.py --file
examples/code_for_cluster.txt --method meanshift
```

### 7.6 DBSCAN clustering (dbscan)

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density (57).

Use the following command to perform the dbscan clustering:

```
tcsh% python iLearn-clustering.py --file
examples/code_for_cluster.txt --method dbscan
```

The feature normalization algorithms can be implemented by running the 'iLearn-feature-normalization.py'. The parameters of iLearn-feature-normalization.py are:

- *file*: the encoding file
- *method*: feature normalization algorithm (select from ZScore and MinMax)
- *format*: the input file format (select from csv, tsv, svm, weka)
- *out*: the output file name

### 7.7 ZScore (ZScore)

Use the following command to perform the ZScore feature normalization:

```
tcsh% python iLearn-feature-normalization.py --file examples/
DNA_code_training.txt --method ZScore --format svm
```

### 7.8 MinMax (MinMax)

Use the following command to perform the MinMax feature normalization:

```
tcsh% python iLearn-feature-normalization.py --file examples/
DNA_code_training.txt --method MinMax --format svm
```

The feature selection algorithms can be implemented by running the 'iLearn-feature-selectior.py'. The parameters of iLearn-feature-selectior.py are:

- *file*: the encoding file
- *method*: feature selection algorithm (select from CHI2, IG, MIC, pearsonr)
- *format*: the input file format (select from csv, tsv, svm, weka)
- *out*: the output file name

### 7.9 Chi-Square feature selection (CHI2)

Use the following command to perform the chi2 feature selection algorithm:

```
tcsh% python iLearn-feature-selectior.py --file
examples/DNA_code_testing.txt --method CHI2 --format svm
```

### 7.10 Information Gain feature selection (IG)

Information gain (IG) measures the amount of information in bits with respect to the class prediction, if the only information available is the presence of a feature and the corresponding class distribution (15,17).

Use the following command to perform the IG feature selection algorithm:

```
tcsh% python iLearn-feature-selectior.py --file
examples/DNA_code_testing.txt --method IG --format svm
```

### 7.11 F-Score (Fscore)

The F-score value of the *i*-th feature is defined in (58):

Use the following command to perform the F-score feature selection algorithm:

```
tcsh% python iLearn-feature-selectior.py --file
examples/DNA_code_testing.txt --method Fscore --format svm
```

### 7.12 Mutual Information feature selection (MIC)

Use the following command to perform the MI clustering:

```
tcsh% python iLearn-feature-selectior.py --file
examples/DNA_code_testing.txt --method MIC --format svm
```

### 7.13 Pearson Correlation coefficient feature selection (pearsonr)

Use the following command to perform the MI clustering:

```
tcsh% python iLearn-feature-selectior.py --file
examples/DNA_code_testing.txt --method pearsonr --format svm
```

The dimension reduction algorithms can be implemented by running the 'iLearn-dimension-reduction.py'. The parameters of iLearn-dimension-reduction.py are:

- *file*: the encoding file
- *method*: feature selection algorithm (select from CHI2, IG, MIC, pearsonr)
- *format*: the input file format (select from csv, tsv, svm, weka)
- *ncomponents*: the number of components, default is 3
- *out*: the output file name

### 7.14 Principal Component Analysis (pca)

PCA (59) is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance.

Use the following command to perform the pca clustering:

```
tcsh% python iLearn-dimension-reduction.py --file
examples/DNA_code_testing.txt --method pca --format svm
```

### 7.15 Latent Dirichlet Allocation (lda)

Latent Dirichlet Allocation (60) is a generative probabilistic model for collections of discrete dataset such as text corpora.

Use the following command to perform the lda clustering:

```
tcsh% python iLearn-dimension-reduction.py --file
examples/DNA_code_testing.txt --method lda --format svm
```

### 7.16 t-Distributed Stochastic Neighbor Embedding (tsne)

Use the following command to perform the tsne clustering:

```
tcsh% python iLearn-dimension-reduction.py --file
examples/DNA_code_testing.txt --method tsne --format svm
```

## 8 Predictor Construction Using *iLearn*

In *iLearn*, five commonly used machine learning algorithms were provided, include SVM, RF, ANN, KNN and LR.

### 8.1 Support Vector Machine (SVM)

SVM aims to accurately classify samples by generating optimal hyperplanes based on the feature dimensionality of the training data (61,62). The resulting mapping formula generated by SVM is usually not interpretable, but invariably yields to satisfactory classification/prediction performance. Therefore, SVM is usually the 'first choice' adopted in many bioinformatics studies (20,21). A variety of kernels have been developed for SVM, for different classification scenarios, including

gaussian radial basis function (RBF), linear/polynomial/sigmoid kernel, etc.

The parameters of iLearn-ML-SVM.py are:

- *train*: the training coding file, which is used to build a model
- *indep*: the independent coding file, which is used as the independent dataset
- *format*: the input code file format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *kernel*: kernel functions (select from linear, poly, rbf and sigmoid, default is rbf)
- *auto*: auto optimize parameters (default is False)
- *batch:* random select part (batch * samples) samples for parameters optimization
- *degree:* set degree in polynomial kernel function (default is 3)
- *gamma:* set gamma in polynomial/rbf/sigmoid kernel function (default is 0)
- *coef0:* set coef0 in polynomial/rbf/sigmoid kernel function (default is 0)
- *cost:* set the parameter cost value (default 1)
- *fold:* set *K*-fold cross-validation mode (default is 5-fold cross-validation)
- *out:* set prefix for output score file

Use the following command to perform the SVM algorighm:

```
tcsh% python iLearn-ML-SVM.py --train examples/DNA_code_training.txt
--indep examples/DNA_code_testing.txt --format svm --batch 0.5 --auto
--out SVM
```

**8.2 Random Forest (RF)**

Random forest (RF) (63) is another well-established and widely employed algorithm, which has been applied for many bioinformatics applications (64-67). RF is essentially an ensemble of a number of decision trees, $T$=$\{T_1(X), T_2(X), \ldots, T_N(X)\}$ built on $N$ random subsets of the training data, and the average prediction performance is usually reported in order to avoid over-fitting (63). The obvious advantage of RF is its interpretability, as every decision tree consists of a number of '*if…then…*' rules, which are fairly straightforward to explain. Such rules can potentially provide biologists with insights and knowledge discovery that would otherwise remain buried in the data. When applying RF, one should bear in mind that the number of decision trees is an important parameter and should be tested exhaustively based on the specific application or biological question, for optimal prediction performance.

The parameters of iLearn-ML-RF.py are:

- *train*: the training coding file, which is used to build a model
- *indep*: the independent coding file, which is used as the independent dataset
- *format*: the input code file format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *n_trees*: the number of trees in the forest (default is 100)
- *fold:* set *K*-fold cross-validation mode (default is 5-fold cross-validation)
- *out:* set prefix for output score file

Use the following command to perform the RF algorighm:

```
tcsh% python iLearn-ML-RF.py --train examples/DNA_code_training.txt
--indep examples/DNA_code_testing.txt --format svm --out RF
```

### 8.3 Artificial Neural Network

An Artificial Neural Network (ANN) usually contains multiple nodes as input and multiple layers to connect these input nodes, mimicking neurons and their functions/connectivity in human brains (68).

The parameters of iLearn-ML-MLP.py are:

- *train*: the training coding file, which is used to build a model
- *indep*: the independent coding file, which is used as the independent dataset
- *format*: the input code file format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *hidden*: set the hidden layer and size in each layer
- *lost*: set the lost function, choose from 'lbfgs', 'sgd' or 'adam'. Default is 'lbfgs'
- *activation*: activation function, choose from 'identity', 'logistic', 'tanh', 'relu'. Default is 'relu'
- *epochs*: set the maximum number of iterations. Default is 200
- *lr*: learning rate. Default is 0.0001
- *fold:* set *K*-fold cross-validation mode (default is 5-fold cross-validation)
- *out:* set prefix for output score file

Use the following command to perform the ANN algorighm:

```
tcsh% python iLearn-ML-MLP.py --train examples/DNA_code_training.txt
--indep examples/DNA_code_testing.txt --format svm --out ANN --hidden
32:32
```

### 8.4 *K*-Nearest Neighbours algorithm (KNN)

*K*-Nearest Neighbours (KNN) algorithm is another commonly employed unsupervised algorithm that clusters samples by calculating their similarities/distances (38). Given the training dataset $D = \{v_1, v_2, \dots, v_n\}$ and a testing sample $x$, KNN (38) calculates the distances between $x$ and all the instances in $D$. As a result, the query sample will be assigned to the same class as its nearest neighbor (shortest distance) in the training dataset.

The parameters of iLearn-ML-MLP.py are:

- *train*: the training coding file, which is used to build a model
- *indep*: the independent coding file, which is used as the independent dataset
- *format*: the input code file format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *k*: set the *K* nearest neighbor value (default is 3)
- *fold:* set *K*-fold cross-validation mode (default is 5-fold cross-validation)
- *out:* set prefix for output score file

Use the following command to perform the ANN algorighm:

```
tcsh% python iLearn-ML-KNN.py --train examples/DNA_code_training.txt
--indep examples/DNA_code_testing.txt --format svm --out KNN --k 5
```

**8.5 Logistic Regression (LR)**

LR can be used to build a classification model for many prediction tasks (69,70), which can be represented as (71):

$$h(x) = b + w_1x_1 + \cdots + w_nx_n,$$

where $x_i$ are the input features, $w_i$ the weight parameters, and $b$ is the bias value. Given an unlabeled input $x$, the likelihood of $x$ with the class label (a given PTM type) can be defined as:

$$P\big(h(x)\big) = \frac{1}{1+e^{-h(x)}}$$

The parameters of iLearn-ML-LR.py are:

- *train*: the training coding file, which is used to build a model
- *indep*: the independent coding file, which is used as the independent dataset
- *format*: the input code file format, four types of format (i.e. csv, tsv, svm and weka) are supported
- *fold:* set *K*-fold cross-validation mode (default is 5-fold cross-validation)
- *out:* set prefix for output score file

Use the following command to perform the LR algorighm:

```
tcsh% python iLearn-ML-LR.py --train examples/DNA_code_training.txt
--indep examples/DNA_code_testing.txt --format svm --out LR
```

Generally, SVM is suitable for dealing with binary classification tasks and is able to handle high dimensional data. According to our experience, it is hard to improve the performance of an SVM model by using the feature selection method (21,72,73), though the latter can significantly improve the performance for RF and KNN models (38,64). For the multi-class classification task, RF and ANN are better choices. ANNs are an alternative to LR, the statistical technique with which they share most similarities. ANNs offer a number of advantages, such as requiring less formal statistical training, ability to implicitly detect complex nonlinear relationships between dependent and independent variables, ability to detect all possible interactions between predictor variables, and availability of multiple training algorithms. However, the disadvantage of ANN and SVM is their “black box” nature (74). Lastly, the performance of an ANN model is dependent on the sample size of the training data, while RF is usually the most robust algorithm.

**8.6 Descriptor estimater**

For a prediction task, the *iLearn* package can select out the descriptor with the best performance by using the ‘iLearn-descriptor-estimater.py’.

Use the following command to run the *iLearn* descriptor estimater:

```
tcsh% python iLearn-descriptor-estimater.py
```

The parameters of iLearn-descriptor-estimater.py are:

- *config*: specify the configure file.

### 8.7 *iLearn* pipeline

All the individual functionalities in *iLearn* can be implemented as a pipeline by using the 'iLearn-auto-pipeline.py' script.

Use the following command to run the *iLearn* pipeline:

```
tcsh% python iLearn-auto-pipline.py
```

The parameters of iLearn-auto-pipeline.py are:

- *config*: specify the configure file.

The following is an example of the configure file:

```
## Input file information
Sequence_File=examples/example.txt
Sequence_Type=DNA
## Descriptor method
Method=DNC;TNC;Kmer
## parameters for nucleotide, protein and peptide descriptors
# Kmer, RCKmer & PseKNC
Kmer_Size=3
# ENAC, EAAC, EGAAC,
Sliding_Window=5
# CKSNAP, CKSAAP, CKSAAGP, KSCTriad
K_Space=5
# Auto-Correlation, NMBroto, Geary, Moran, SOCNumber, QSOrder
Lag_Value=2
# Auto-Correlation, Pseudo nucleic acid composition, QSOrder, PAAC, APAAC
Weight_Value=0.1
# Auto-Correlation & Pseudo nucleic acid composition, PAAC, APAAC
Lamada_Value=2
# Auto-Correlation & Pseudo nucleic acid composition
Di-DNA-Phychem=Rise;Roll;Shift;Slide;Tilt;Twist
Tri-DNA-Phychem=Dnase I;Bendability (DNAse)
Di-DNA-Phychem-default6=Rise;Roll;Shift;Slide;Tilt;Twist
Di-RNA-Phychem=Rise (RNA);Roll (RNA);Shift (RNA);Slide (RNA);Tilt (RNA);Twist (RNA)
All_Property=False
# AAindex properties, NMBroto, Geary, Moran
AAindex=ANDN920101;ARGP820101;ARGP820102;ARGP820103;BEGF750101;BEGF750102;BEGF750103;BHAR880101
# PseKRAAC
PseKRAAC_Model=g-gap
Ktuple=2
GapLamada=2
RAACCluster1=2
RAACCluster2=2
RAACCluster3A=2
RAACCluster3B=2
RAACCluster4=5
RAACCluster5=2
RAACCluster6A=5
RAACCluster6B=5
RAACCluster6C=5
RAACCluster7=2
RAACCluster8=2
RAACCluster9=2
RAACCluster10=2
RAACCluster11=2
RAACCluster12=2
RAACCluster13=4
RAACCluster14=2
RAACCluster15=2
RAACCluster16=2
## output format
Output_Format=svm
## Clustering
Clustering_Algorithm=
Kmean_Cluster_Number=2
Clustering_Type=sample
## Feature Normalization
Feature_Normalization_Algorithm=
## Feature selection
Feature_Selection_Algorithm=
Selected_Feature_Number=100
## Dimension reduction
Dimension_Reduction_Algorithm=
Dimension_Reduction_Number=3
## Model construction
ML=RF;SVM
# Parameters for SVM
Kernel=rbf
Cost=1.0
Gamma=
Auto_Opterimize=False
# RF
Tree_Number=100
# KNN
K_Nearest_Neighbour=3
# ANN
Hidden_Layer_Size=32;32
## K-fold Cross-Validation
Validation=5
## Ensemble learning
Ensemble=YES
```

## 9. Performance Evaluation Strategy of *iLearn*

iLearn support both the binary classification task and multiclass classification task. For binary classification problem, seven frequently-used measures are supported by iLearn, including Sensitivity (*Sn*), Specificity (*Sp*), Accuracy (*Acc*), Matthew correlation coefficient (*MCC*), *Recall*, *Precision*, F1-score, the area under ROC curve (AUROC) and the area under the PRC curve (AUPRC). *Sn*, *Sp*, *ACC*, *MCC*, Rec*all*, *Precision* and F1-score are defined as:

$$Sn = Recall = 1 - \frac{N_-^+}{N^+}$$

$$Sp = 1 - \frac{N_+^-}{N^-}$$

$$Acc = 1 - \frac{N_-^+ + N_+^-}{N^+ + N^-}$$

$$MCC = \frac{1 - (\frac{N_-^+ + N_+^-}{N^+ + N^-})}{\sqrt{(1 + \frac{N_+^- - N_-^+}{N^+})(1 + \frac{N_-^+ - N_+^-}{N^-})}}$$

$$Precision = 1 - \frac{N_+^-}{N^+ + N_+^-}$$

$$F1 - score = 1 - \frac{N_+^- + N_-^+}{2 \times N^+ + N_+^- + N_-^+}$$

where $N^+$, $N_+^-$, $N^-$ and $N_-^+$ represent the numbers of true positives, false positives, true negatives and false negatives respectively. The AUROC value is calculated based on the Receiver-Operating-Characteristic (ROC) curve, and takes values between 0 and 1, while the AUPRC value is calculated based on the Precision-Recall curve, where, the higher the AUROC and AUPRC value, the better the prediction performance.

For multiclass classification problems, the *Acc* is used to evaluate the performance, which is defined as (75,76):

$$Acc = 1 - \frac{N_-^+(i) + N_+^-(i)}{N^+(i) + N^-(i)}$$

where $N^+(i)$, $N_+^-$(i), $N^-(i)$ and $N_-^+(i)$ represent the numbers of samples in the *i*-th class, the total number of the samples in the *i*-th class but predicted as one of the other classes, the total number of the samples not in the *i*-th class and the total number of the samples not in the *i*-th class but predicted as the *i*-th class, respectively.

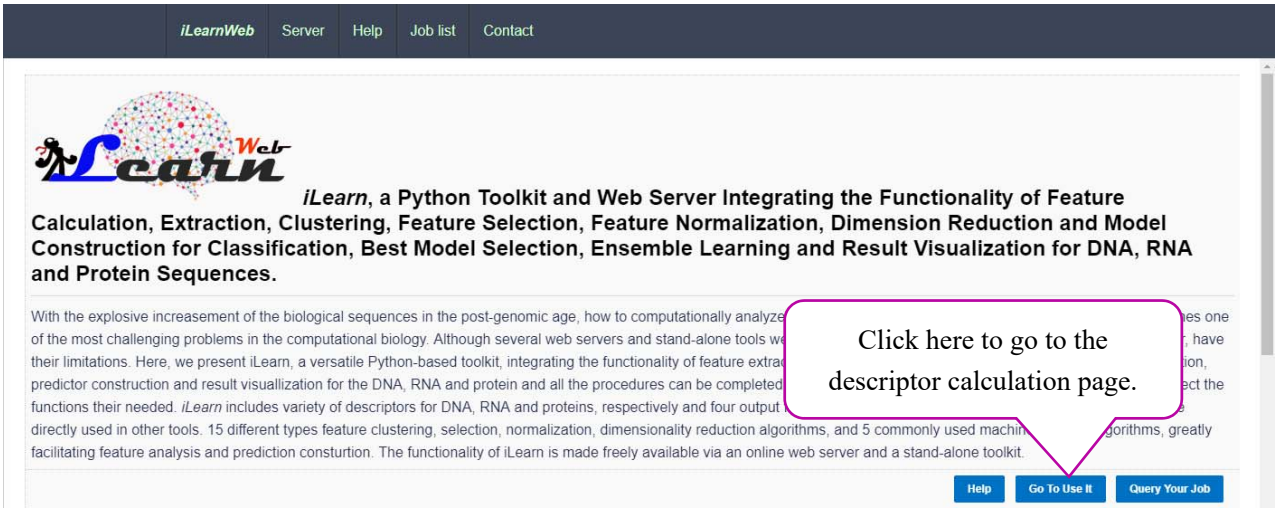
## 10. Online Web Server

Moreover, for users that are not familiar with computer programming using Python we also implemented an online web server of *iLearn*, which is publicly available at http://ilearn.erc.monash.edu/. It is configured for the extensible cloud computing facility supported

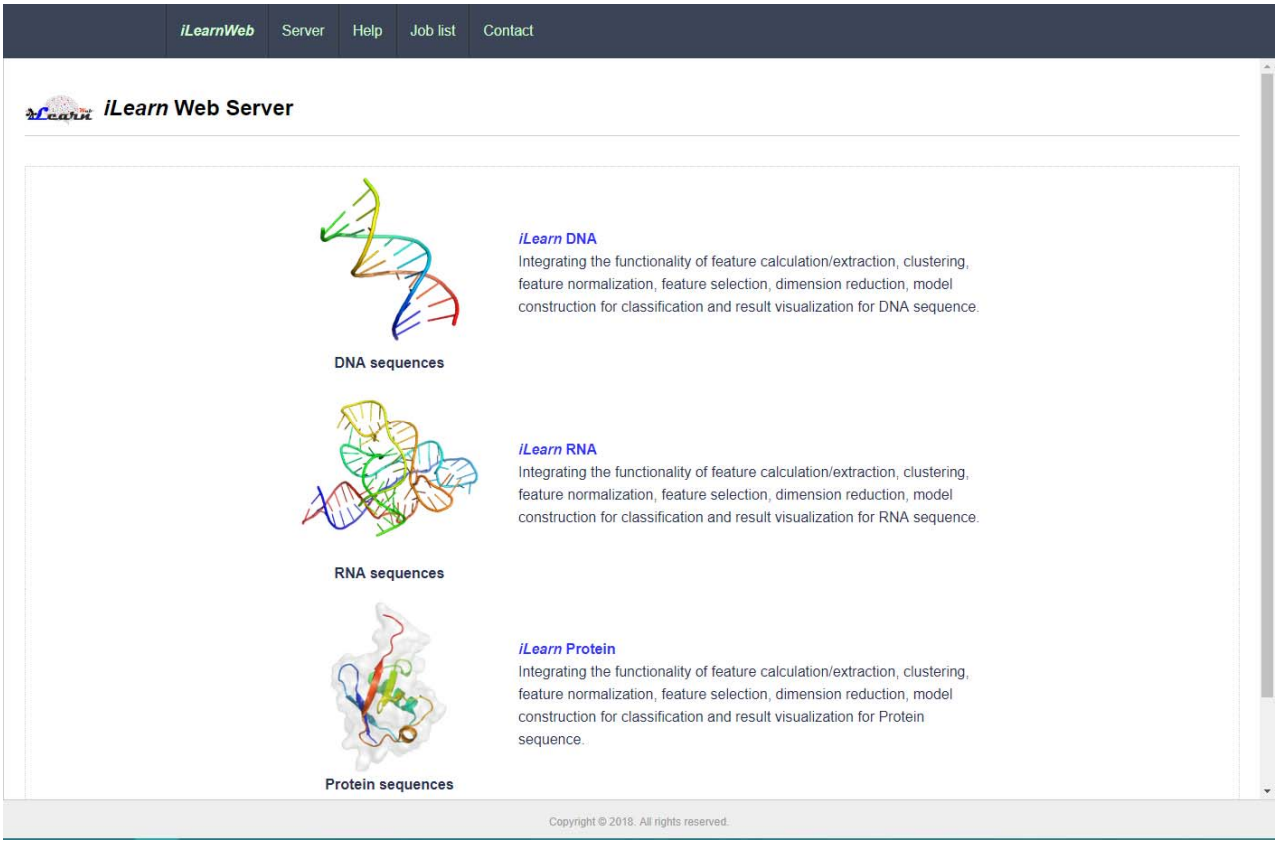
by the e-Research Centre at Monash University, equipped with 16 cores, 64 GB memory and a 2 TB hard disk. This configuration can be easily upgraded in line with increasing user demands in the future.

The *iLearn* web server is a user-friendly online platform for implementing the function integrated in the *iLearn* package. Take the sub web server for DNA analysis as an example: 26 descriptors can be selected to transform the DNA sequences into feature vectors. For example, when "Kmer" is selected, the parameter of "Kmer size" will be displayed on the page. Then, users can decide which feature analysis procedure (e.g. clustering, feature selection, dimensionality reduction and feature normalization) is included in the analysis process. When the SVM algorithm is selected to construct the predictor, one should also specify the kernel (the default is RBF kernel). Users not only can set the values of 'gamma' and 'cost' for the RBF kernel, but also use the function of automatic parameter optimization by clicking the "Auto optimize parameters" button. K-fold cross-validation is supported by the iLearn web server. Users can select the commonly used 5-fold and 10-fold cross-validation, or input a K value. The step-by-step of usage instructions is as follows:

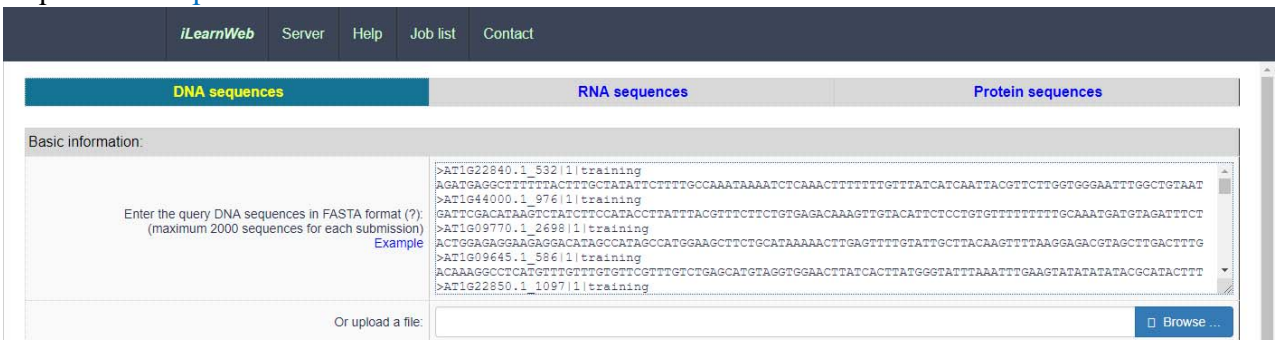
Input "http://ilearn.erc.monash.edu" on your browser, and click the "Go To Use It" button. Then, you will see the descriptor calculation page.




**Step 1**. Select the biological sequence type. For example, we select the DNA sequence.

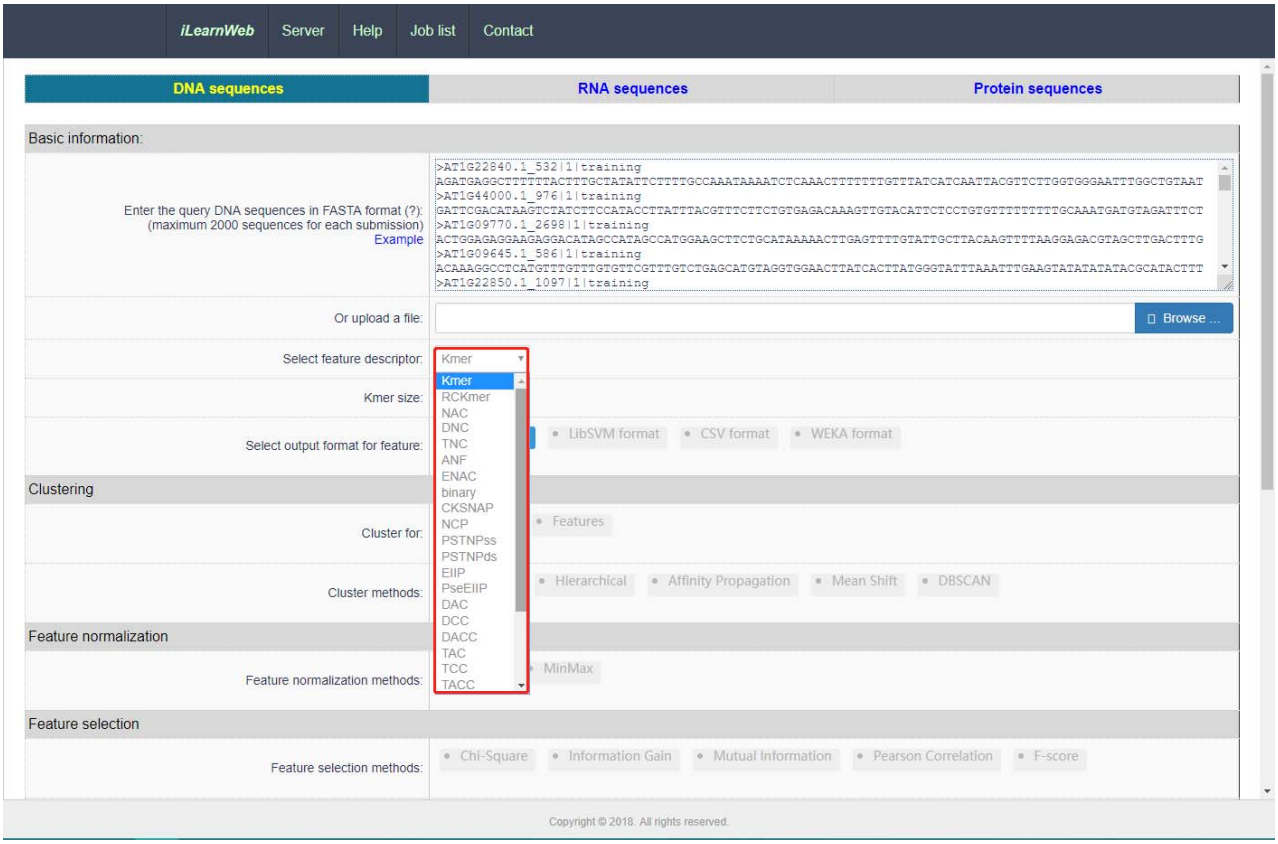



**Step 2**. Input your fasta sequences in the designated text area or upload a file that includes the sequences in 'special FASTA' format format.

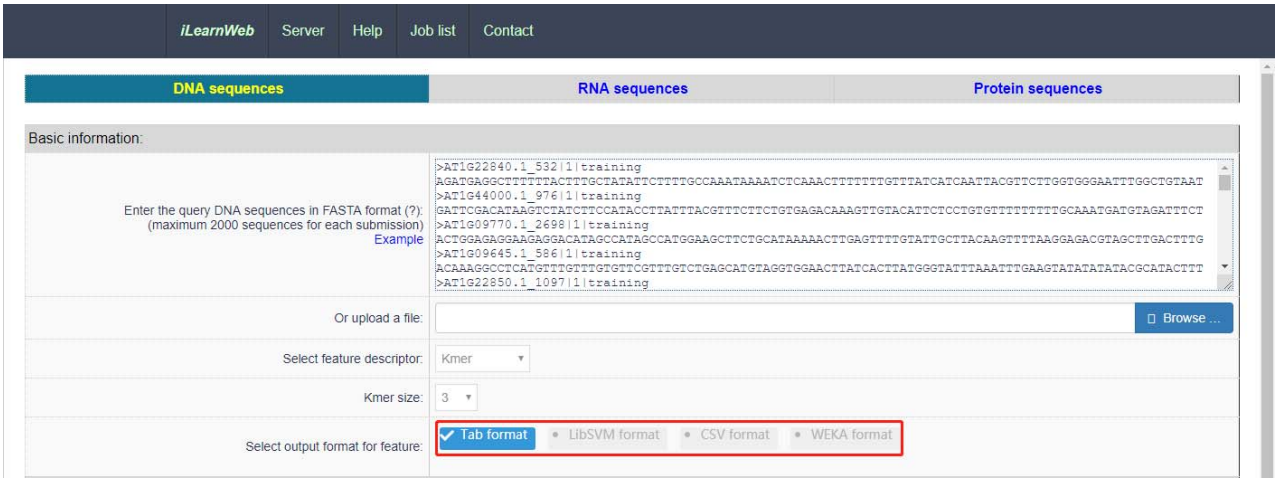



Note: Paste your protein (or peptide) sequences in the 'TEXTAREA' or upload a file that includes the sequences. The biological sequences must be in a **specified 'FASTA' format**. *iLearn* was designed to accept at most 2000 sequences at once.

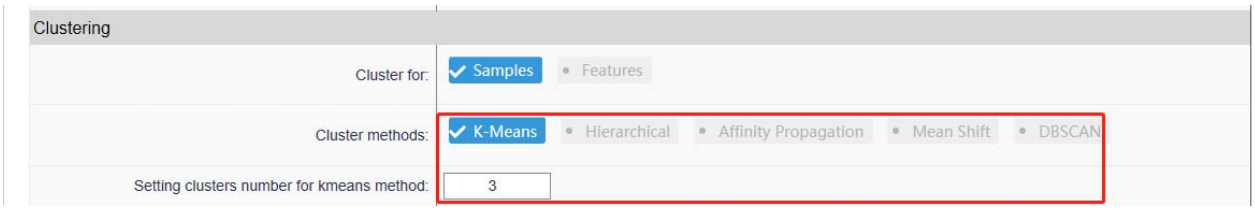
**Step 3**. Select descriptor



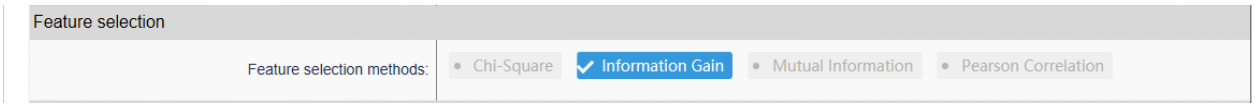

**Step 4**. Select output format of the descriptor. Four output format are supported by iLearnWeb.




**Step 5**. Select clustering algorithm and set the parameter for the selected clustering algorithm.

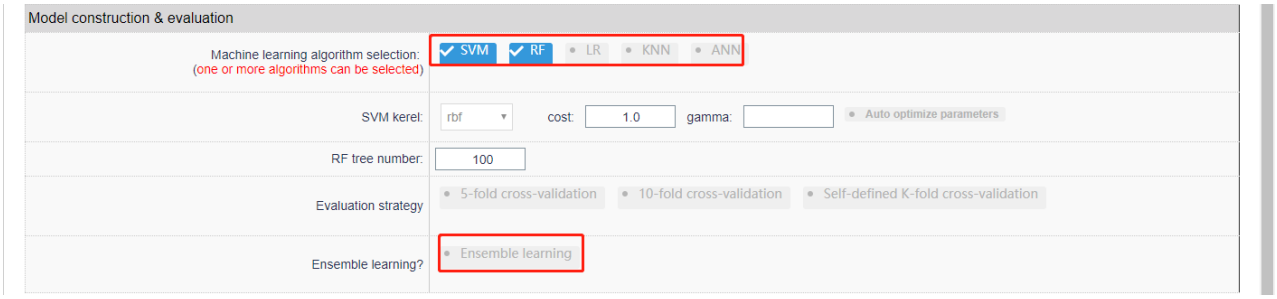



**Step 6**. Select feature selection algorithm.



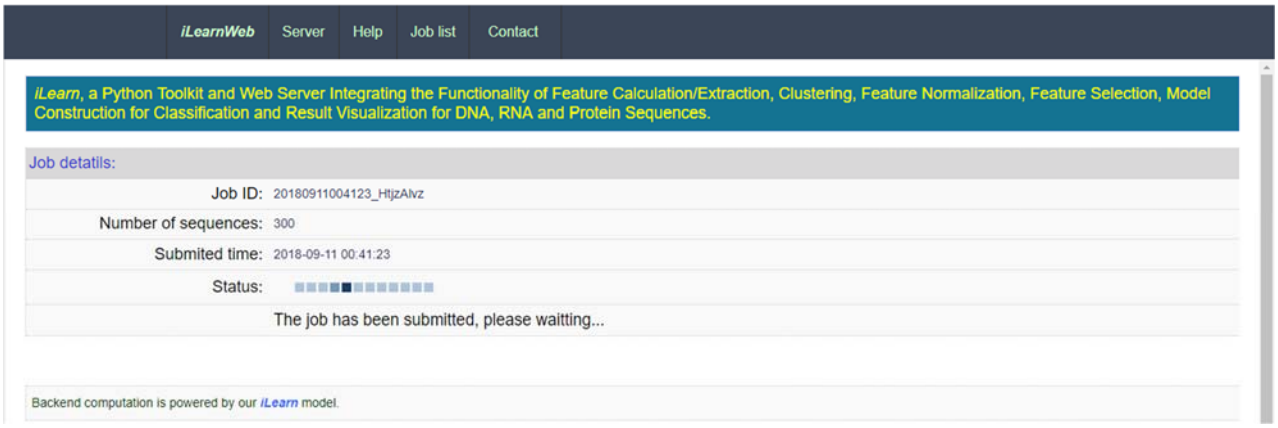
**Step 7**. Dimension reduction algorithm.

**Step 8**. Select machine learning algorithm and set the parameters for the machine learning algorithm. If more than one machine learning algorithms are selected, iLearnWeb will build a model for each of the algorithm and identify the model with the best predictive performance. If the "Ensemble learning" button is selected, the iLearnWeb will calculate and report the performance for all possible combinations of the selected algorithms through a logistic regression method and return the machine learning algorithm combination that achieves the best performance.




At last, click 'Submit' to calculate the descriptors and run the selected clustering, feature selection and machine learning algorithms.

**Step 9.** Waiting for your result.



*iFeature-in-oneWeb* Job Result Detatils:

Job ID: 20180701234038_DkZGqVUZ

Number of training sequences: 4200

Number of testing sequences: 836

Descriptor method: binary

Training codings:
View all with format

| # | label | BINARY.F1 | BINARY.F2 | BINARY.F3 | BINARY.F4 | BINARY.F5 | BINARY.F6 | BINARY.F7 | BINARY.F8 | BINARY.F9 | BINARY.F10 | BINARY.F11 | BINARY.F12 | BINARY.F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AT1G22840.1_532 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| AT1G44000.1_976 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| AT1G09770.1_2698 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| AT1G09645.1_586 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

Testing codings:
View all with format

| # | label | BINARY.F1 | BINARY.F2 | BINARY.F3 | BINARY.F4 | BINARY.F5 | BINARY.F6 | BINARY.F7 | BINARY.F8 | BINARY.F9 | BINARY.F10 | BINARY.F11 | BINARY.F12 | BINARY.F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AT1G09780.1_1772 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| AT1G31812.1_443 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| AT1G77810.1_1523 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| AT1G14690.1_2242 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

Feature selection method: CHI2

The top ranked features:
View all ranked features

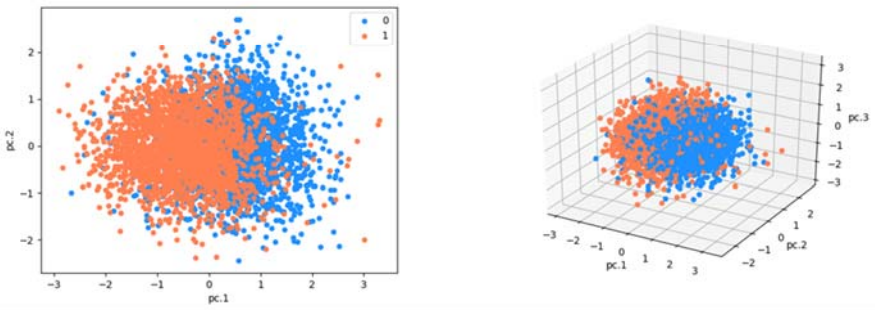
| feature | CHI-value |
|---|---|
| f.344 | 106.410 |
| f.404 | 91.494 |
| f.400 | 88.810 |
| f.388 | 88.083 |
| f.248 | 87.309 |

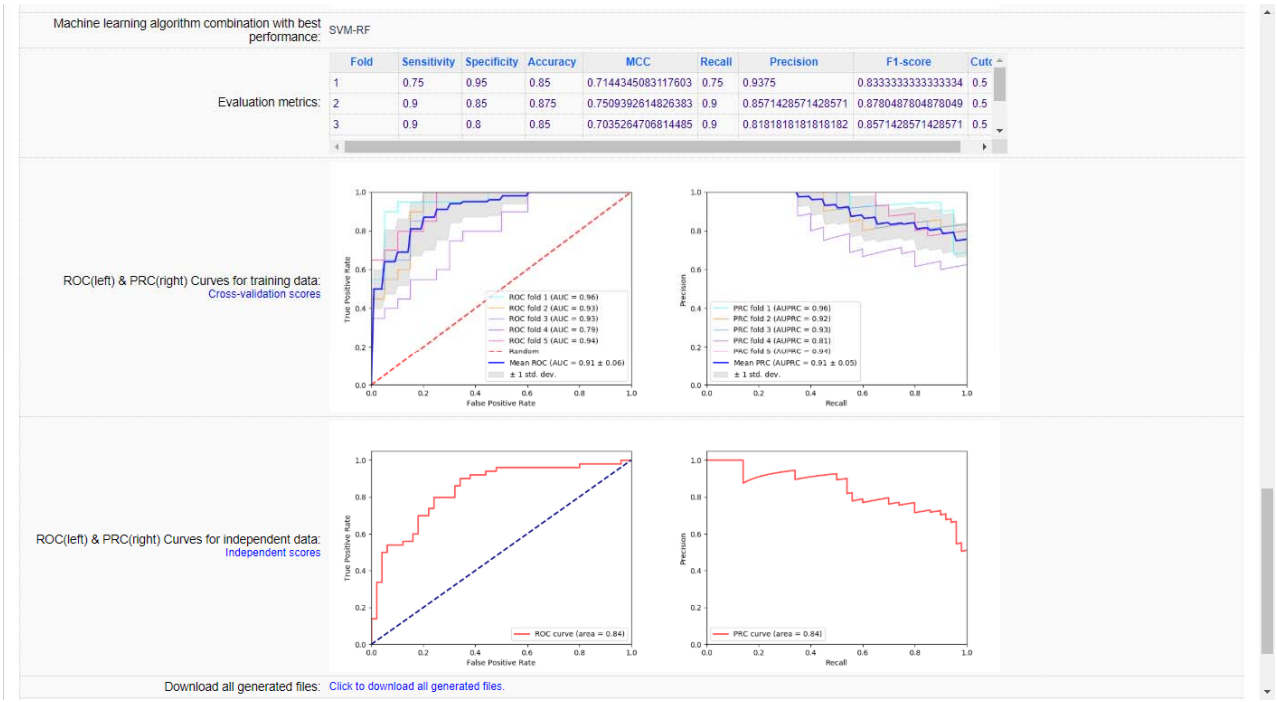
Dimension reduction method: pca

Dimension reduced features:
View all

| samples | pc.1 | pc.2 | pc.3 |
|---|---|---|---|
| s.1 | -1.8188724403949879 | 0.013317368852855215 | 0.2856022684437705 |
| s.2 | -1.5715417287142333 | -0.06268988028744293 | 0.5708894815434701 |
| s.3 | 0.41994266145840553 | 1.407722446046383 | 0.8383975636532028 |
| s.4 | -1.2755358796012276 | -0.3109798154279577 | -0.5753608090194846 |
| s.5 | 0.6748651478267704 | -0.41764576046357654 | 1.025079099299388 |

Dimension reduction plot(left - 2d plot, right - 3d plot):






After a few seconds, the result should displayed in the result page. For each job, *iLearnWeb* will generate a job ID, your calculation result will be stored for a week. With a week, you can query your result by searching your job ID.

## 7. 11. Summary

In summary, *iLearn* has been extensively benchmarked to guarantee correctness of computations, and was deliberately designed to ensure workflow efficiency. To the best of our knowledge, this is the first universal toolkit for integrated feature calculation, clustering, selection analysis, model construction and result visualization. We will integrate more analysis, clustering and machine learning algorithms to enable interactive analysis and machine learning-based modeling in future work. It is anticipated that *iLearn* will be widely used as a powerful tool in bioinformatics, computational biology, systems biology and proteome research.

## 12. Acknowledgments

# 13. References


1. Chen, Z., Zhao, P., Li, F., Leier, A., Marquez-Lago, T.T., Wang, Y., Webb, G.I., Smith, A.I., Daly, R.J., Chou, K.C. *et al.* (2018) iFeature: a Python package and web server for features extraction and selection from protein and peptide sequences. *Bioinformatics*, **34**, 2499-2502.
2. Noble, W.S., Kuehn, S., Thurman, R., Yu, M. and Stamatoyannopoulos, J. (2005) Predicting the in vivo signature of human gene regulatory sequences. *Bioinformatics*, **21 Suppl 1**, i338-343.
3. Lee, D., Karchin, R. and Beer, M.A. (2011) Discriminative prediction of mammalian enhancers from DNA sequence. *Genome Res*, **21**, 2167-2180.
4. Gupta, S., Dennis, J., Thurman, R.E., Kingston, R., Stamatoyannopoulos, J.A. and Noble, W.S. (2008) Predicting human nucleosome occupancy from primary sequence. *PLoS Comput Biol*, **4**, e1000134.
5. Chen, W., Tran, H., Liang, Z., Lin, H. and Zhang, L. (2015) Identification and analysis of the N(6)-methyladenosine in the Saccharomyces cerevisiae transcriptome. *Sci Rep*, **5**, 13859.
6. He, W., Jia, C., Duan, Y. and Zou, Q. (2018) 70ProPred: a predictor for discovering sigma70 promoters based on combining multiple features. *BMC Syst Biol*, **12**, 44.
7. He, W., Jia, C. and Zou, Q. (2018) 4mCPred: Machine Learning Methods for DNA N4-methylcytosine sites Prediction. *Bioinformatics*.
8. Nair, A.S. and Sreenadhan, S.P. (2006) A coding measure scheme employing electron-ion interaction pseudopotential (EIIP). *Bioinformation*, **1**, 197-202.
9. Liu, B., Liu, F., Fang, L., Wang, X. and Chou, K.C. (2015) repDNA: a Python package to generate various modes of feature vectors for DNA sequences by incorporating user-defined physicochemical properties and sequence-order effects. *Bioinformatics*, **31**, 1307-1309.
10. Chen, W., Feng, P.M., Lin, H. and Chou, K.C. (2013) iRSpot-PseDNC: identify recombination spots with pseudo dinucleotide composition. *Nucleic Acids Res*, **41**, e68.
11. Guo, S.H., Deng, E.Z., Xu, L.Q., Ding, H., Lin, H., Chen, W. and Chou, K.C. (2014) iNuc-PseKNC: a sequence-based predictor for predicting nucleosome positioning in genomes with pseudo k-tuple nucleotide composition. *Bioinformatics*, **30**, 1522-1529.
12. Chen, W., Lei, T.Y., Jin, D.C., Lin, H. and Chou, K.C. (2014) PseKNC: a flexible web server for generating pseudo K-tuple nucleotide composition. *Anal Biochem*, **456**, 53-60.
13. Qiu, W.R., Xiao, X. and Chou, K.C. (2014) iRSpot-TNCPseAAC: identify recombination spots with trinucleotide composition and pseudo amino acid components. *Int J Mol Sci*, **15**, 1746-1766.
14. Bhasin, M. and Raghava, G.P. (2004) Classification of nuclear receptors based on amino acid composition and dipeptide composition. *J Biol Chem*, **279**, 23262-23266.
15. Chen, K., Jiang, Y., Du, L. and Kurgan, L. (2009) Prediction of integral membrane protein type by collocated hydrophobic amino acid pairs. *J Comput Chem*, **30**, 163-172.
16. Chen, K., Kurgan, L. and Rahbari, M. (2007) Prediction of protein crystallization using collocation of amino acid pairs. *Biochem Biophys Res Commun*, **355**, 764-769.
17. Chen, K., Kurgan, L.A. and Ruan, J. (2007) Prediction of flexible/rigid regions from protein sequences using k-spaced amino acid pairs. *BMC Struct Biol*, **7**, 25.
18. Chen, K., Kurgan, L.A. and Ruan, J. (2008) Prediction of protein structural class using novel evolutionary collocation-based sequence representation. *J Comput Chem*, **29**, 1596-1604.
19. Lee, T.Y., Lin, Z.Q., Hsieh, S.J., Bretana, N.A. and Lu, C.T. (2011) Exploiting maximal dependence decomposition to identify conserved motifs from a group of aligned signal sequences.

*Bioinformatics*, **27**, 1780-1787.

20. Chen, Z., Zhou, Y., Song, J. and Zhang, Z. (2013) hCKSAAP_UbSite: improved prediction of human ubiquitination sites by exploiting amino acid pattern and properties. *Biochim Biophys Acta*, **1834**, 1461-1467.
21. Chen, Z., Chen, Y.Z., Wang, X.F., Wang, C., Yan, R.X. and Zhang, Z. (2011) Prediction of ubiquitination sites by using the composition of k-spaced amino acid pairs. *PLoS One*, **6**, e22930.
22. Feng, Z.P. and Zhang, C.T. (2000) Prediction of membrane protein types based on the hydrophobic index of amino acids. *J Protein Chem*, **19**, 269-275.
23. Horne, D.S. (1988) Prediction of protein helix content from an autocorrelation analysis of sequence hydrophobicities. *Biopolymers*, **27**, 451-477.
24. Sokal, R.R. and Thomson, B.A. (2006) Population structure inferred by local spatial autocorrelation: an example from an Amerindian tribal population. *Am J Phys Anthropol*, **129**, 121-131.
25. Kawashima, S., Pokarowski, P., Pokarowska, M., Kolinski, A., Katayama, T. and Kanehisa, M. (2008) AAindex: amino acid index database, progress report 2008. *Nucleic Acids Res*, **36**, D202-205.
26. Xiao, N., Cao, D.S., Zhu, M.F. and Xu, Q.S. (2015) protr/ProtrWeb: R package and web server for generating various numerical representation schemes of protein sequences. *Bioinformatics*, **31**, 1857-1859.
27. Dubchak, I., Muchnik, I., Holbrook, S.R. and Kim, S.H. (1995) Prediction of protein folding class using global description of amino acid sequence. *Proc Natl Acad Sci U S A*, **92**, 8700-8704.
28. Dubchak, I., Muchnik, I., Mayor, C., Dralyuk, I. and Kim, S.H. (1999) Recognition of a protein fold in the context of the Structural Classification of Proteins (SCOP) classification. *Proteins*, **35**, 401-407.
29. Cai, C.Z., Han, L.Y., Ji, Z.L., Chen, X. and Chen, Y.Z. (2003) SVM-Prot: Web-based support vector machine software for functional classification of a protein from its primary sequence. *Nucleic Acids Res*, **31**, 3692-3697.
30. Cai, C.Z., Han, L.Y., Ji, Z.L. and Chen, Y.Z. (2004) Enzyme family classification by support vector machines. *Proteins*, **55**, 66-76.
31. Han, L.Y., Cai, C.Z., Lo, S.L., Chung, M.C. and Chen, Y.Z. (2004) Prediction of RNA-binding proteins from primary sequence by a support vector machine approach. *RNA*, **10**, 355-368.
32. Tomii, K. and Kanehisa, M. (1996) Analysis of amino acid indices and mutation matrices for sequence comparison and structure prediction of proteins. *Protein Eng*, **9**, 27-36.
33. Shen, J., Zhang, J., Luo, X., Zhu, W., Yu, K., Chen, K., Li, Y. and Jiang, H. (2007) Predicting protein-protein interactions based only on sequences information. *Proc Natl Acad Sci U S A*, **104**, 4337-4341.
34. Chou, K.C. (2005) Using amphiphilic pseudo amino acid composition to predict enzyme subfamily classes. *Bioinformatics*, **21**, 10-19.
35. Chou, K.C. (2001) Prediction of protein cellular attributes using pseudo-amino acid composition. *Proteins*, **43**, 246-255.
36. Chen, X., Qiu, J.D., Shi, S.P., Suo, S.B., Huang, S.Y. and Liang, R.P. (2013) Incorporating key position and amino acid residue features to identify general and species-specific Ubiquitin conjugation sites. *Bioinformatics*, **29**, 1614-1622.
37. Needleman, S.B. and Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, **48**, 443-453.

38. Cai, Y., Huang, T., Hu, L., Shi, X., Xie, L. and Li, Y. (2012) Prediction of lysine ubiquitination with mRMR feature selection and analysis. *Amino Acids*, **42**, 1387-1395.
39. Radivojac, P., Vacic, V., Haynes, C., Cocklin, R.R., Mohan, A., Heyen, J.W., Goebl, M.G. and Iakoucheva, L.M. (2010) Identification, analysis, and prediction of protein ubiquitination sites. *Proteins*, **78**, 365-380.
40. Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, **25**, 3389-3402.
41. Tung, C.W. and Ho, S.Y. (2008) Computational identification of ubiquitylation sites from protein sequences. *BMC Bioinformatics*, **9**, 310.
42. Lee, T.Y., Chen, S.A., Hung, H.Y. and Ou, Y.Y. (2011) Incorporating distant sequence features and radial basis function networks to identify ubiquitin conjugation sites. *PLoS One*, **6**, e17331.
43. Jones, D.T. (1999) Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol*, **292**, 195-202.
44. Peng, K., Radivojac, P., Vucetic, S., Dunker, A.K. and Obradovic, Z. (2006) Length-dependent prediction of protein intrinsic disorder. *BMC Bioinformatics*, **7**, 208.
45. Obradovic, Z., Peng, K., Vucetic, S., Radivojac, P. and Dunker, A.K. (2005) Exploiting heterogeneous sequence properties improves prediction of protein disorder. *Proteins*, **61 Suppl 7**, 176-182.
46. Liu, B., Liu, F., Wang, X., Chen, J., Fang, L. and Chou, K.C. (2015) Pse-in-One: a web server for generating various modes of pseudo components of DNA, RNA, and protein sequences. *Nucleic Acids Res*, **43**, W65-71.
47. Faraggi, E., Yang, Y., Zhang, S. and Zhou, Y. (2009) Predicting continuous local structure and the effect of its substitution for secondary structure in fragment-free protein structure prediction. *Structure*, **17**, 1515-1527.
48. Heffernan, R., Dehzangi, A., Lyons, J., Paliwal, K., Sharma, A., Wang, J., Sattar, A., Zhou, Y. and Yang, Y. (2016) Highly accurate sequence-based prediction of half-sphere exposures of amino acid residues in proteins. *Bioinformatics*, **32**, 843-849.
49. Heffernan, R., Paliwal, K., Lyons, J., Dehzangi, A., Sharma, A., Wang, J., Sattar, A., Yang, Y. and Zhou, Y. (2015) Improving prediction of secondary structure, local backbone angles, and solvent accessible surface area of proteins by iterative deep learning. *Sci Rep*, **5**, 11476.
50. Sandberg, M., Eriksson, L., Jonsson, J., Sjostrom, M. and Wold, S. (1998) New chemical descriptors relevant for the design of biologically active peptides. A multivariate characterization of 87 amino acids. *J Med Chem*, **41**, 2481-2491.
51. Chen, Y.Z., Chen, Z., Gong, Y.A. and Ying, G. (2012) SUMOhydro: a novel method for the prediction of sumoylation sites based on hydrophobic properties. *PLoS One*, **7**, e39195.
52. Jain, A.K., Murty, M.N. and Flynn, P.J. (1999) Data clustering: A review. *Acm Comput Surv*, **31**, 264-323.
53. Rokach, L. and Maimon, O. (2005) In Maimon, O. and Rokach, L. (eds.), *Data Mining and Knowledge Discovery Handbook*. Springer US, Boston, MA, pp. 321-352.
54. Jain, A.K. (2010) Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, **31**, 651-666.
55. Frey, B.J. and Dueck, D. (2007) Clustering by passing messages between data points. *Science*, **315**, 972-976.

56. Cheng, Y.Z. (1995) Mean Shift, Mode Seeking, and Clustering. *Ieee T Pattern Anal*, **17**, 790-799.
57. Ester, M., Kriegel, H.-P., #246, Sander, r. and Xu, X. (1996), *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, Portland, Oregon, pp. 226-231.
58. Chen, J., Guo, M., Wang, X. and Liu, B. (2018) A comprehensive review and comparison of different computational methods for protein remote homology detection. *Brief Bioinform*, **19**, 231-244.
59. Pearson, K. (1901) LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **2**, 559-572.
60. Blei, D.M., Ng, A.Y. and Jordan, M.I. (2003) Latent Dirichlet allocation. *Journal of Machine Learning Research*, **3**, 993-1022.
61. Vapnik, V.N. (1995) *The nature of statistical learning theory*. Springer-Verlag New York, Inc.
62. Vapnik, V.N. (1999) An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, **10**, 988-999.
63. Breiman, L. (2001) Random Forests. *Machine Learning*, **45**, 5-32.
64. Li, Y., Wang, M., Wang, H., Tan, H., Zhang, Z., Webb, G.I. and Song, J. (2014) Accurate in silico identification of species-specific acetylation sites by integrating protein sequence-derived and functional features. *Sci Rep*, **4**, 5765.
65. Jia, J., Liu, Z., Xiao, X., Liu, B. and Chou, K.C. (2016) iSuc-PseOpt: Identifying lysine succinylation sites in proteins by incorporating sequence-coupling effects into pseudo components and optimizing imbalanced training dataset. *Anal Biochem*, **497**, 48-56.
66. Jia, J., Liu, Z., Xiao, X., Liu, B. and Chou, K.C. (2016) pSuc-Lys: Predict lysine succinylation sites in proteins with PseAAC and ensemble random forest approach. *J Theor Biol*, **394**, 223-230.
67. Hasan, M.M., Khatun, M.S., Mollah, M.N.H., Yong, C. and Guo, D. (2017) A systematic identification of species-specific protein succinylation sites using joint element features information. *Int J Nanomedicine*, **12**, 6303-6315.
68. Wang, S.-C. (2003), *Interdisciplinary Computing in Java Programming*. Springer US, Boston, MA, pp. 81-100.
69. Song, J., Li, F., Leier, A., Marquez-Lago, T.T., Akutsu, T., Haffari, G., Chou, K.C., Webb, G.I., Pike, R.N. and Hancock, J. (2018) PROSPERous: high-throughput prediction of substrate cleavage sites for 90 proteases with improved accuracy. *Bioinformatics*, **34**, 684-687.
70. Li, F., Li, C., Marquez-Lago, T.T., Leier, A., Akutsu, T., Purcell, A.W., Smith, A.I., Lithgow, T., Daly, R.J., Song, J. *et al.* (2018) Quokka: a comprehensive tool for rapid and accurate prediction of kinase family-specific phosphorylation sites in the human proteome. *Bioinformatics*.
71. Hou, T., Zheng, G., Zhang, P., Jia, J., Li, J., Xie, L., Wei, C. and Li, Y. (2014) LAceP: lysine acetylation site prediction using logistic regression classifiers. *PLoS One*, **9**, e89575.
72. Chen, Y.Z., Tang, Y.R., Sheng, Z.Y. and Zhang, Z. (2008) Prediction of mucin-type O-glycosylation sites in mammalian proteins using the composition of k-spaced amino acid pairs. *BMC Bioinformatics*, **9**, 101.
73. Chen, Z., Wang, Y., Zhai, Y.F., Song, J. and Zhang, Z. (2013) ZincExplorer: an accurate hybrid method to improve the prediction of zinc-binding sites from protein sequences. *Mol Biosyst*, **9**, 2213-2222.
74. Tu, J.V. (1996) Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *J Clin Epidemiol*, **49**, 1225-1231.

75. Liu, B., Yang, F., Huang, D.S. and Chou, K.C. (2018) iPromoter-2L: a two-layer predictor for identifying promoters and their types by multi-window-based PseKNC. *Bioinformatics*, **34**, 33-40.
76. Feng, P.M., Chen, W., Lin, H. and Chou, K.C. (2013) iHSP-PseRAAAC: Identifying the heat shock protein families using pseudo reduced amino acid alphabet composition. *Anal Biochem*, **442**, 118-125.