Sequence analysis

iFeature: a python package and web server for features extraction and selection from protein and peptide sequences

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

[1]School of Basic Medical Science, Qingdao University, Qingdao, China, [2]State Key Laboratory of Cotton Biology, Institute of Cotton Research of Chinese Academy of Agricultural Sciences (CAAS), Anyang, 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]Institute of Image Processing and Pattern Recognition, Shanghai Jiao Tong University, Shanghai, China, [7]Monash Centre for Data Science, Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia, [8]Gordon Research Institute, Boston, MA 02478, USA, [9]Center for Informational Biology, University of Electronic Science and Technology of China, Chengdu, 610054, China

[†]These two authors contributed equally to this work. *To whom correspondence should be addressed.

[†]These two authors contributed equally to this work. *To whom correspondence should be addressed.

Supplementary Material

Package Version: 1.0

Content

Brief introduction

iFeature is a comprehensive Python-based toolkit for generating various numerical feature representation schemes from protein or peptide sequences. *iFeature* is capable of calculating and extracting a wide spectrum of 18 major sequence encoding schemes that encompass 53 different types of feature descriptors. Among the different feature groups, it also allows users to extract specific physiochemical properties of amino acids from the AAindex database (Kawashima, et al., 2008). Furthermore, *iFeature* also integrates five kinds of frequently used feature clustering algorithms, four feature selection algorithms and three dimensionality reduction algorithms. In order to facilitate users' interpretability of outcomes, the clustering and dimensionality reduction results generated by *iFeature* can be further visualized in form of scatter diagrams. This makes *iFeature* a unique and powerful tool that greatly facilitates feature generation, analysis, training and benchmarking of machine-learning models and predictions.

1. Installation

iFeature 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 *iFeature*, 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 software can be freely downloaded from https://www.anaconda.com/download/.

2. The Full Workflow of *iFeature*

Here, we provide step-by-step user instruction illustrating the full workflow of the *iFeature* toolkit by running the example provided in the directory of "examples". The examples provided include both protein and peptide sequences.
iFeature includes four main programs: "iFeature.py", "iFeaturePseKRAAC.py", "cluster.py" and "feaSelector.py". All the functions regarding feature extraction, feature or sample clustering and feature selection analysis can be executed through these four main programs by specifying the parameter '--type'.

- "iFeature.py" is the main program used to extract 37 different types of feature descriptors: Usage: `tcsh% python iFeature.py --help`
- "iFeaturePseKRAAC.py" is the program used to extract the 16 types of pseudo K-tuple reduced amino acid composition (PseKRAAC) feature descriptors: Usage: `tcsh% python iFeaturePseKRAAC.py --help`
- "cluster.py" is the program used for running the feature or sample clustering algorithms: Usage: `tcsh% python cluster.py --help`
- "feaSelector.py" is the fourth main program used to implement the feature selection

algorithms: Usage: `tcsh% python feaSelector.py --help`

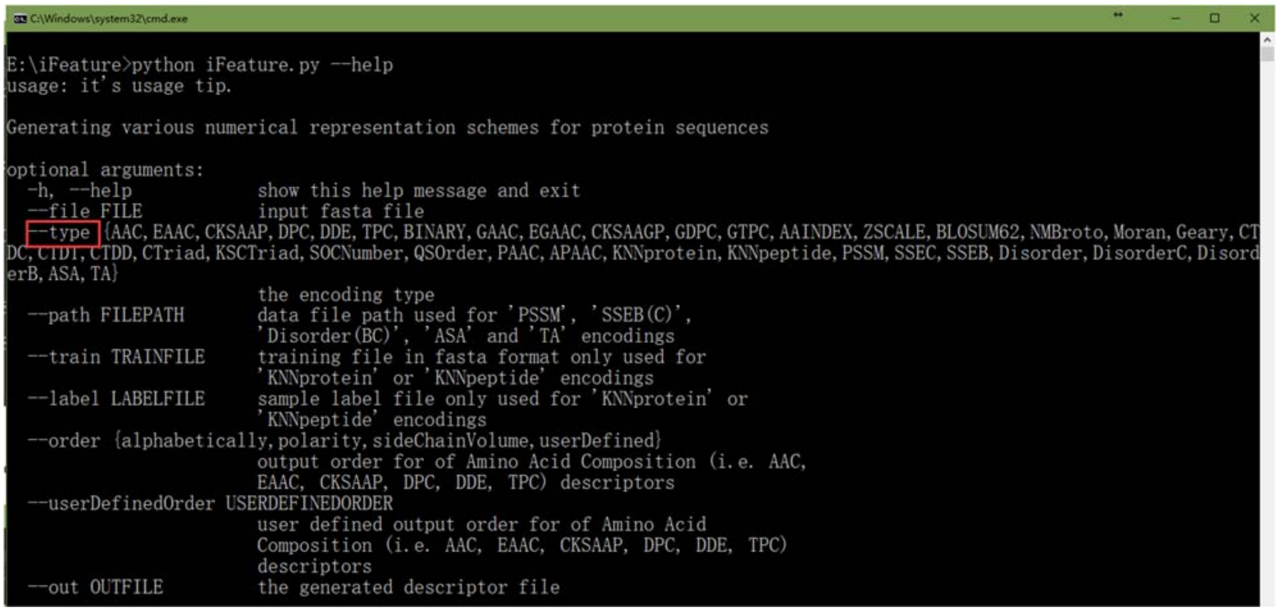
Furthermore, the *iFeature* package contains other Python scripts to generate the position-specific scoring matrix (PSSM) profiles, predicted protein secondary structure and predicted protein disorder, which have also been often used to improve the prediction performance of machine learning-based classifiers in conjunction with sequence-derived information. The three dimensionality reduction algorithms are also included in the 'scripts' directory.
The detailed usage of these scripts is described in section 4 below ("**Commonly Used Feature Descriptors**").

3. Software Package Overview

iFeature 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. A complete list of the 18 major encoding schemes included in *iFeature* is summarized in **Table 1** in the main manuscript. Moreover, *iFeature* also integrates a variety of commonly used feature clustering, selection, and dimensionality reduction algorithms, which greatly facilitates feature generation, importance analysis, model training and performance evaluation experiments. We describe the detailed functions of *iFeature* below.

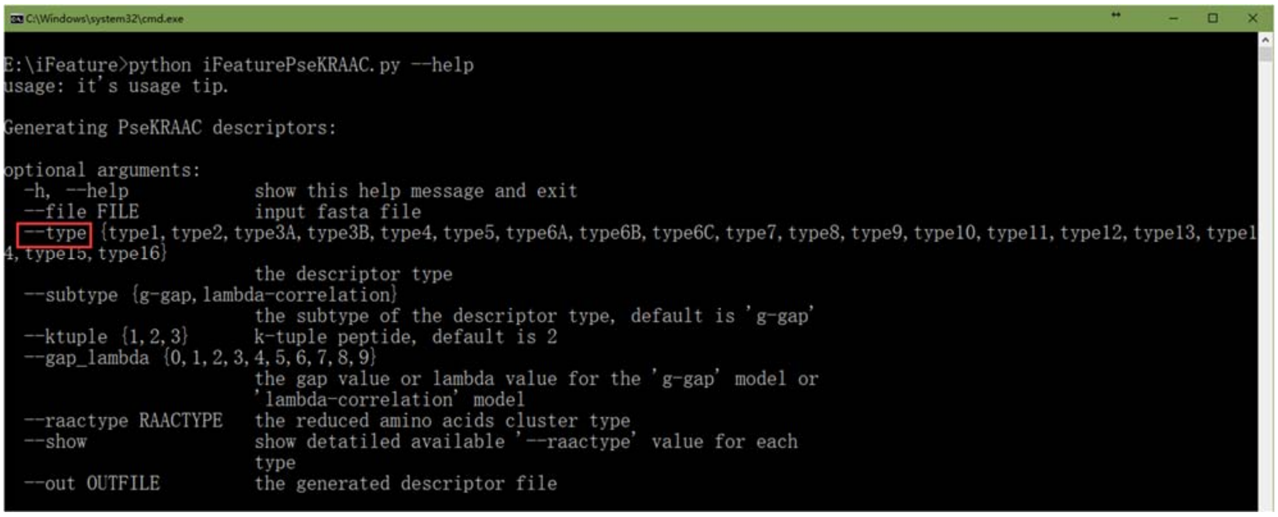
Feature descriptor extraction:
Generally, each type of feature descriptor can be calculated using the main programs "iFeature.py" and "iFeaturePseKRAAC.py" implemented in the *iFeature* toolkit. Users are advised to specify the descriptor type by using the parameter `'--type'`.

```
tcsh% python iFeature.py --help
```



```
E:\iFeature>python iFeature.py --help
usage: it's usage tip.

Generating various numerical representation schemes for protein sequences

optional arguments:
  -h, --help            show this help message and exit
  --file FILE           input fasta file
  --type {AAC,EAAC,CKSAAP,DPC,DDE,TPC,BINARY,GAAC,EGAAC,CKSAAGP,GDPC,GTPC,AAINDEX,ZSCALE,BLOSUM62,NMBroto,Moran,Geary,CT
DC,CTDT,CTDD,CTriad,KSCTriad,SOCNumber,QSOrder,PAAC,APAAC,KNNprotein,KNNpeptide,PSSM,SSEC,SSEB,Disorder,DisorderC,Disord
erB,ASA,TA}
                        the encoding type
  --path FILEPATH       data file path used for 'PSSM', 'SSEB(C)',
                        'Disorder(BC)', 'ASA' and 'TA' encodings
  --train TRAINFILE     training file in fasta format only used for
                        'KNNprotein' or 'KNNpeptide' encodings
  --label LABELFILE     sample label file only used for 'KNNprotein' or
                        'KNNpeptide' encodings
  --order {alphabetically,polarity,sideChainVolume,userDefined}
                        output order for of Amino Acid Composition (i.e. AAC,
                        EAAC, CKSAAP, DPC, DDE, TPC) descriptors
  --userDefinedOrder USERDEFINEDORDER
                        user defined output order for of Amino Acid
                        Composition (i.e. AAC, EAAC, CKSAAP, DPC, DDE, TPC)
                        descriptors
  --out OUTFILE         the generated descriptor file
```

```
tcsh% python iFeaturePseKRAAC.py --help
```

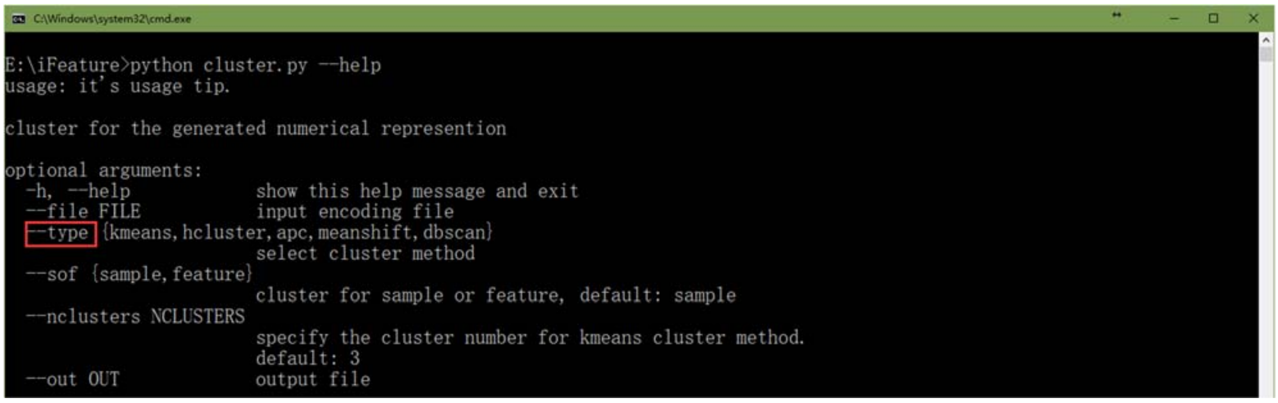


```
E:\iFeature>python iFeaturePseKRAAC.py --help
usage: it's usage tip.

Generating PseKRAAC descriptors:

optional arguments:
  -h, --help            show this help message and exit
  --file FILE           input fasta file
  --type {type1,type2,type3A,type3B,type4,type5,type6A,type6B,type6C,type7,type8,type9,type10,type11,type12,type13,type1
4,type15,type16}
                        the descriptor type
  --subtype {g-gap,lambda-correlation}
                        the subtype of the descriptor type, default is 'g-gap'
  --ktuple {1,2,3}      k-tuple peptide, default is 2
  --gap_lambda {0,1,2,3,4,5,6,7,8,9}
                        the gap value or lambda value for the 'g-gap' model or
                        'lambda-correlation' model
  --raactype RAACTYPE   the reduced amino acids cluster type
  --show                show detatiled available '--raactype' value for each
                        type
  --out OUTFILE         the generated descriptor file
```

Feature clustering:

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

```
tcsh% python cluster.py --help
```

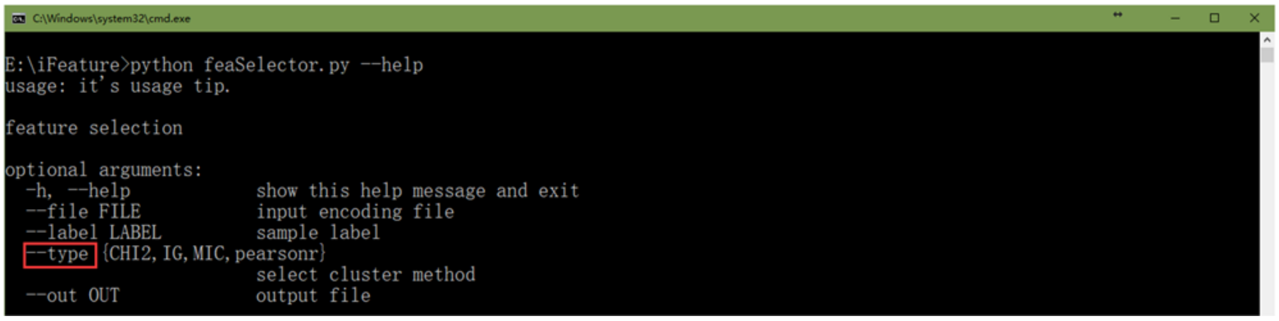


```
E:\iFeature>python cluster.py --help
usage: it's usage tip.

cluster for the generated numerical represention

optional arguments:
  -h, --help            show this help message and exit
  --file FILE           input encoding file
  --type {kmeans,hcluster,apc,meanshift,dbscan}
                        select cluster method
  --sof {sample,feature}
                        cluster for sample or feature, default: sample
  --nclusters NCLUSTERS
                        specify the cluster number for kmeans cluster method.
                        default: 3
  --out OUT             output file
```

Feature selection:

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

```
tcsh% python feaSelector.py --help
```



```
E:\iFeature>python feaSelector.py --help
usage: it's usage tip.

feature selection

optional arguments:
  -h, --help            show this help message and exit
  --file FILE           input encoding file
  --label LABEL         sample label
  --type {CHI2,IG,MIC,pearsonr}
                        select cluster method
  --out OUT             output file
```

4. Commonly Used Feature Descriptors

Let us assume that a protein or peptide sequence with *N* amino acid residues can be generally represented as $\{R_1, R_2, \ldots, R_n\}$, where R_i represents the residue at the *i*-th position in the sequence. The following commonly used feature descriptors can be calculated and extracted using *iFeature*.

4.1 Amino Acid Composition (AAC)

The Amino Acid Composition (AAC) encoding (Bhasin and Raghava, 2004) calculates the frequency of each amino acid type in a protein or peptide sequence. The frequencies of all 20 natural amino acids (i.e. "ACDEFGHIKLMNPQRSTVWY") can be calculated as:

$$f(t) = \frac{N(t)}{N}, \quad t \in \{A, C, D, \ldots, Y\}$$

where *N(t)* is the number of amino acid type *t*, while *N* is the length of a protein or peptide sequence.

Use the following command to extract the AAC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
AAC
```

4.2 Enhanced Amino Acid Composition (EAAC)

The Enhanced Amino Acid Composition (EAAC) feature type is introduced here for the first time. It 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. An illustrated example of this encoding scheme is provided in the following **Figure S1**.

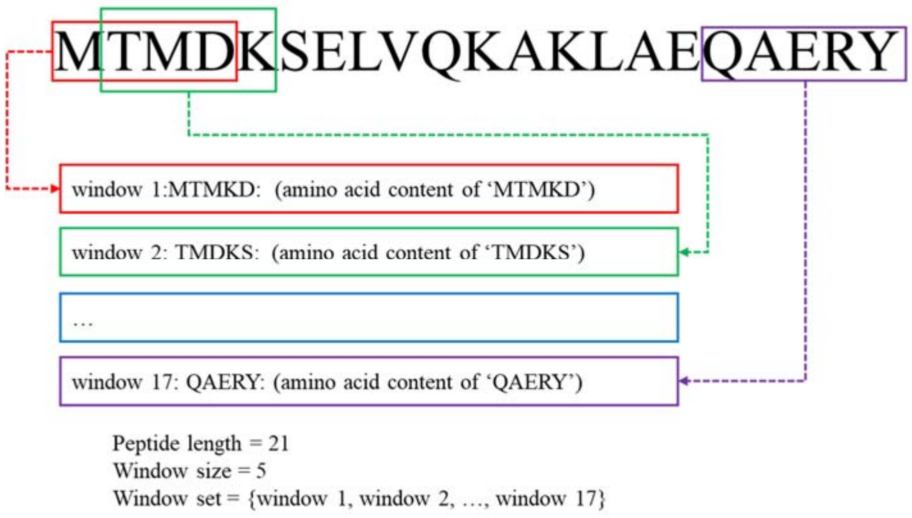

MTMDKSELVQKAKLAEQAERY
window 1:MTMKD: (amino acid content of 'MTMKD')
window 2: TMDKS: (amino acid content of 'TMDKS')
...
window 17: QAERY: (amino acid content of 'QAERY')
Peptide length = 21
Window size = 5
Window set = {window 1, window 2, ..., window 17}

Figure S1. An illustrated example of the EAAC descriptor.

The EAAC can be calculated as:

$$f(t,\ win) = \frac{N(t, win)}{N(win)}, \quad t \in \{A, C, D, ..., Y\}, \quad win \in \{window1, window2, ..., window17\}$$

where *N(t,win)* is the number of amino acid type *t* in the sliding window *win* and *N(win)* is the size of the sliding window *win*.

Use the following command to extract the EAAC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
EAAC
```

Advanced users can adjust the size of the sliding window to <*N*> (the default is 5) by running the following Python command:

```
tcsh% python codes/EAAC.py examples/test-peptide.txt <N> EAAC.tsv
```

Here, <*N*> is a placeholder for the integer defining the window size.

4.3 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) (Chen, et al., 2009; Chen, et al., 2007a; Chen, et al., 2007b; Chen, et al., 2008). Taking $k = 0$ as an example, there are 400 0-spaced residue pairs (i.e., AA, AC, AD,…, YY.). Then, a feature vector can be defined as:

$$\left(\frac{N_{AA}}{N_{total}}, \frac{N_{AC}}{N_{total}}, \frac{N_{AD}}{N_{total}}, ..., \frac{N_{YY}}{N_{total}}\right)_{400}$$

The value of each descriptor denotes the composition of the corresponding residue pair in the protein or peptide sequence. For instance, if the residue pair AA appears *m* times in the protein, the composition of the residue pair AA is equal to *m* divided by the total number of *0*-spaced residue pairs (N_{total}) in the protein. 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 protein of length *P*, respectively. An illustrated example of this encoding scheme (k=0) is provided in the following **Figure S2**.

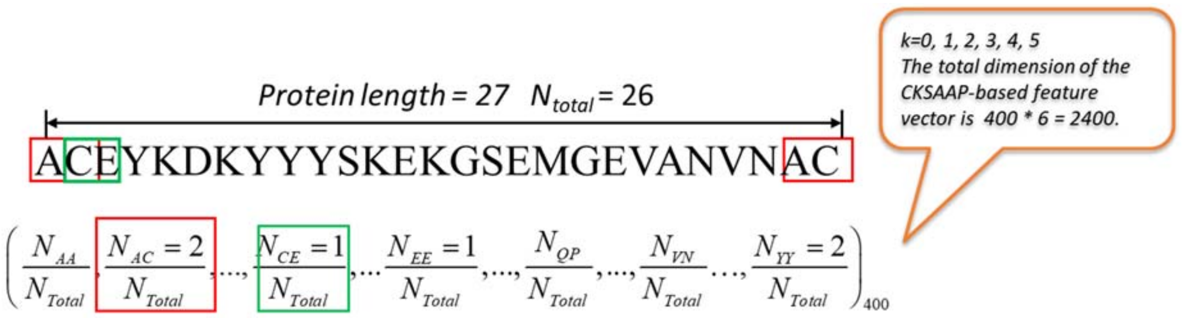

k=0, 1, 2, 3, 4, 5
The total dimension of the CKSAAP-based feature vector is 400 * 6 = 2400.
Protein length = 27 N_total = 26
ACEYKDKYYYSKEKGSEMGEVANVNAC
N_AA/N_Total, N_AC = 2/N_Total, ..., N_CE = 1/N_Total, ..., N_EE = 1/N_Total, ..., N_QP/N_Total, ..., N_VN/N_Total ..., N_YY = 2/N_Total
400

Figure S2. An illustrated example of the CKSAAP (k = 0) descriptor.

Use the following command to extract the CKSAAP feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
CKSAAP
```

Advanced users can adjust the value of *k* to <*k*> (the default is 5) by running the following Python command:

```
tcsh% python codes/CKSAAP.py examples/test-protein.txt <k>
CKSAAP.tsv
```

Here, <*k*> is a placeholder for the integer. $k = 0, 1, 2, \dots, 5$.

4.4 Tri-Peptide Composition (TPC)

The Tripeptide Composition (TPC) (Bhasin and Raghava, 2004) gives 8000 descriptors, defined as:

$$f(r,s,t) = \frac{N_{rst}}{N-2}, \qquad r,s,t \in \{A,C,D,...,Y\}$$

where N_{rst} is the number of tripeptides represented by amino acid types *r*, *s* and *t.*

Use the following command to extract the TPC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
TPC
```

4.5 Di-Peptide Composition (DPC)

The Dipeptide Composition (Saravanan and Gautham, 2015) gives 400 descriptors. It is defined as:

$$D(r,s) = \frac{N_{rs}}{N-1}, \qquad r,s \in \{A,C,D,\dots Y\}$$

where N_{rs} is the number of dipeptides represented by amino acid types *r* and *s.*

Use the following command to extract the DPC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
DPC
```

4.6 Dipeptide Deviation from Expected Mean (DDE)

The Dipeptide Deviation from Expected Mean feature vector (Saravanan and Gautham, 2015) is constructed by computing three parameters, i.e. dipeptide composition (D_c), theoretical mean (T_m), and theoretical variance (T_v). The above three parameters and the DDE are computed as follows. $D_c(r,s)$, the dipeptide composition measure for the dipeptide '*rs*', is given as

$$D_c(r,s) = \frac{N_{rs}}{N-1}, \qquad r,s \in \{A,C,D,\dots Y\}$$

where N_{rs} is the number of dipeptides represented by amino acid types *r* and *s* and *N* is the length of the protein or peptide. $T_m(r,s)$, the theoretical mean, is given by:

$$T_m(r,s) = \frac{C_r}{C_N} \times \frac{C_s}{C_N}$$

where C_r is the number of codons that code for the first amino acid and C_s is the number of codons that code for the second amino acid in the given dipeptide '*rs*'. C_N is the total number of possible codons, excluding the three stop codons (i.e., 61). T_v (*r,s*), the theoretical variance of the dipeptide '*rs*', is given by:

$$T_v(r,s) = \frac{T_m(r,s)(1-T_m(r,s))}{N-1}$$

Finally, *DDE(r,s)* is calculated as:

$$DDE(r,s) = \frac{D_c(r,s) - T_m(r,s)}{\sqrt{T_v(r,s)}}$$

Use the following command to extract the DDE feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
DDE
```

Furthermore, for feature descriptors described in sections 4.1 - 4.6, the output order (the default is alphabetically) can be selected or defined. In particular, three different orders of amino acids (i.e. alphabetically, by polarity and by side chain volume) are now available in *iFeature*. In addition, *iFeature* allows users to define an output order that best suits their purposes:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
AAC --order polarity

tcsh% python iFeature.py --file examples/test-protein.txt --type
AAC --order userDefined --userDefinedOrder YWVTSRQPNMLKIHGFEDCA
```

4.7 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 (Lee, et al., 2011b). 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). GAAC descriptor is the frequency of each amino acid group, which is defined as:

$$f(g) = \frac{N(g)}{N}, \quad g \in \{g1, g2, g3, g4, g5\}$$

$$N(g_t) = \sum N(t), \qquad t \in g$$

where *N(g)* is the number of amino acid in group g, *N(t)* is the number of amino acid type *t,* and *N* is the length of the protein/peptide sequence.

Use the following command to extract the GAAC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
GAAC
```

4.8 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.

$$f(g,\ win) = \frac{N(g, win)}{N(win)}, \quad g \in \{g1, g2, g3, g4, g5\}, \quad win \in \{window1, window2, ..., window17\}$$

where *N(g, win)* is the number of amino acids in group *g* within the sliding window *win* and *N(win)* is the size of the sliding window *win*.

Use the following command to extract the EGAAC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
EGAAC
```

Advanced users can adjust the size of the sliding window to <*N*> (the default is 5) by running the following Python command:

```
tcsh% python codes/EGAAC.py examples/test-peptide.txt <N>
EGAAC.tsv
```

Here, <*N*> is a placeholder for the integer defining the window size.

4.9 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). Taking $k = 0$ as an example, there are 25 0-spaced group pairs (i.e., *g1g1, g1g2, g1g3, ... g5g5*). Thus, a feature vector of CKSAAGP can be defined as:

$$\left(\frac{N_{g1g1}}{N_{total}}, \frac{N_{g1g2}}{N_{total}}, \frac{N_{g1g3}}{N_{total}}, ..., \frac{N_{g5g5}}{N_{total}}\right)_{25}$$

The value of each descriptor denotes the composition of the corresponding residue group pair in a protein or peptide sequence. For instance, if the residue group pair *g1g1* appears *m* times in the protein, the composition of the residue pair *g1g1* is equal to *m* divided by the total number of *0*-spaced residue pairs (N_{total}) in the protein. For $k = 0, 1, 2, 3, 4$ and 5, the values of N_{total} are $P – 1$, $P – 2$, $P – 3$, $P – 4$, $P – 5$ and $P – 6$ respectively, for a protein of length *P*.

Use the following command to extract the CKSAAGP feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
CKSAAGP
```

Advanced users can adjust the value of *k* to <*k*> (the default is 5) by running the following Python command:

```
tcsh% python codes/CKSAAGP.py examples/test-protein.txt <k>
CKSAAGP.tsv
```

Here, <*k*> is a placeholder for the integer. $k = 0, 1, 2, \dots, 5$.

4.10 Grouped Di-Peptide Composition (GDPC)

The Grouped Di-Peptide Composition encoding is another variation of the DPC descriptor. It is composed of a total of 25 descriptors that are defined as:

$$f(r,s)=\frac{N_{rs}}{N-1}, \qquad r,s\in\{g1,g2,g3,g4,g5\}$$

where N_{rs} is the number of tripeptides represented by amino acid type groups r and s, N is the length of a protein or peptide sequence.

Use the following command to extract the GDPC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
GDPC
```

4.11 Grouped Tri-Peptide Composition (GTPC)

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

$$f(r,s,t)=\frac{N_{rst}}{N-2}, \qquad r,s,t\in\{g1,g2,g3,g4,g5\}$$

where N_{rst} is the number of tripeptides represented by amino acid type groups r, s and t. $\underline{N}$ is the length of a protein or peptide sequence.

Use the following command to extract the GTPC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
GTPC
```

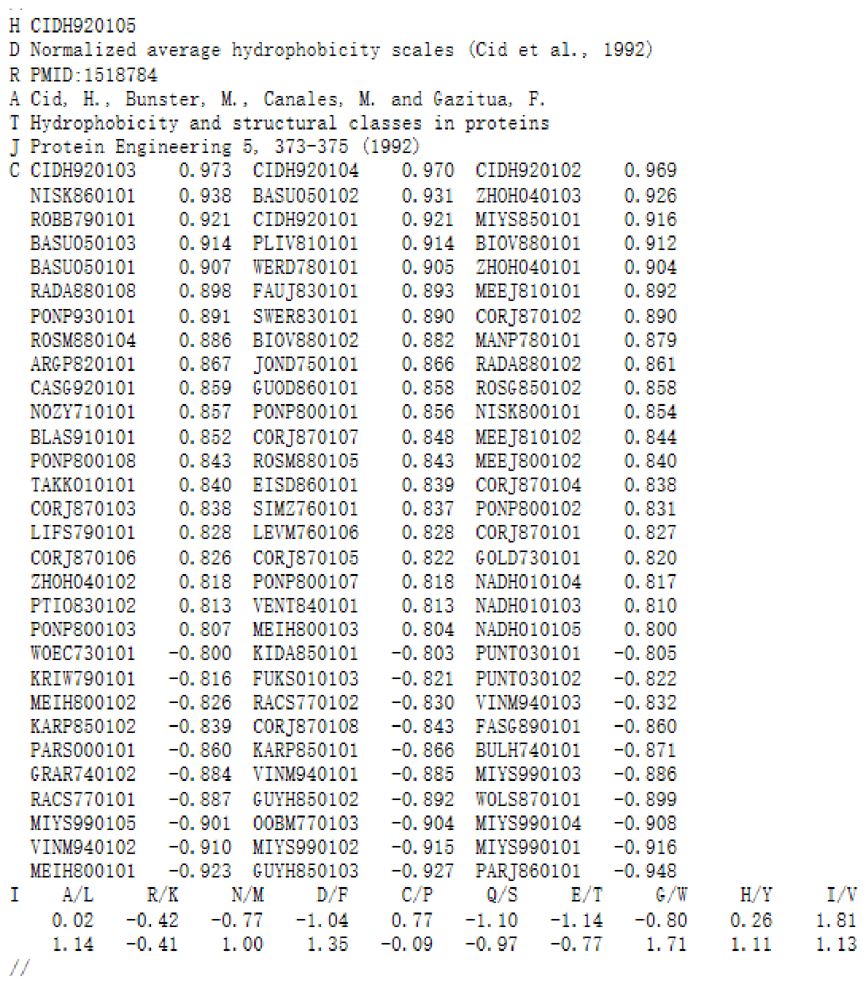
4.12 Binary (BINARY)

In the Binary encoding (Chen, et al., 2011; Chen, et al., 2013b), 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 iFeature.py --file examples/test-peptide.txt --type
BINARY
```

4.13 Moran correlation (Moran)



```
H CIDH920105
D Normalized average hydrophobicity scales (Cid et al., 1992)
R PMID:1518784
A Cid, H., Bunster, M., Canales, M. and Gazitua, F.
T Hydrophobicity and structural classes in proteins
J Protein Engineering 5, 373-375 (1992)
C CIDH920103    0.973  CIDH920104    0.970  CIDH920102    0.969
  NISK860101    0.938  BASU050102    0.931  ZHOH040103    0.926
  ROBB790101    0.921  CIDH920101    0.921  MIYS850101    0.916
  BASU050103    0.914  PLIV810101    0.914  BIOV880101    0.912
  BASU050101    0.907  WERD780101    0.905  ZHOH040101    0.904
  RADA880108    0.898  FAUJ830101    0.893  MEEJ810101    0.892
  PONP930101    0.891  SWER830101    0.890  CORJ870102    0.890
  ROSM880104    0.886  BIOV880102    0.882  MANP780101    0.879
  ARGP820101    0.867  JOND750101    0.866  RADA880102    0.861
  CASG920101    0.859  GUOD860101    0.858  ROSG850102    0.858
  NOZY710101    0.857  PONP800101    0.856  NISK800101    0.854
  BLAS910101    0.852  CORJ870107    0.848  MEEJ810102    0.844
  PONP800108    0.843  ROSM880105    0.843  MEEJ800102    0.840
  TAKK010101    0.840  EISD860101    0.839  CORJ870104    0.838
  CORJ870103    0.838  SIMZ760101    0.837  PONP800102    0.831
  LIFS790101    0.828  LEVM760106    0.828  CORJ870101    0.827
  CORJ870106    0.826  CORJ870105    0.822  GOLD730101    0.820
  ZHOH040102    0.818  PONP800107    0.818  NADH010104    0.817
  PTIO830102    0.813  VENT840101    0.813  NADH010103    0.810
  PONP800103    0.807  MEIH800103    0.804  NADH010105    0.800
  WOEC730101   -0.800  KIDA850101   -0.803  PUNT030101   -0.805
  KRIW790101   -0.816  FUKS010103   -0.821  PUNT030102   -0.822
  MEIH800102   -0.826  RACS770102   -0.830  VINM940103   -0.832
  KARP850102   -0.839  CORJ870108   -0.843  FASG890101   -0.860
  PARS000101   -0.860  KARP850101   -0.866  BULH740101   -0.871
  GRAR740102   -0.884  VINM940101   -0.885  MIYS990103   -0.886
  RACS770101   -0.887  GUYH850102   -0.892  WOLS870101   -0.899
  MIYS990105   -0.901  OOBM770103   -0.904  MIYS990104   -0.908
  VINM940102   -0.910  MIYS990102   -0.915  MIYS990101   -0.916
  MEIH800101   -0.923  GUYH850103   -0.927  PARJ860101   -0.948
I    A/L     R/K     N/M     D/F     C/P     Q/S     E/T     G/W     H/Y     I/V
    0.02   -0.42   -0.77   -1.04    0.77   -1.10   -1.14   -0.80    0.26    1.81
    1.14   -0.41    1.00    1.35   -0.09   -0.97   -0.77    1.71    1.11    1.13
//
```

Figure S3. An illustrated example of amino acid physicochemical properties in the AAIndex database (Kawashima, et al., 2008).

The autocorrelation descriptors are defined based on the distribution of amino acid properties along the sequence (Feng and Zhang, 2000; Horne, 1988; Sokal and Thomson, 2006). The amino acid properties used here are different types of amino acids index, which is retrieved from the AAindex Database (Kawashima, et al., 2008) available at http://www.genome.jp/dbget/aaindex.html/. The eight indices 'CIDH920105', 'BHAR880101', 'CHAM820101', 'CHAM820102', 'CHOC760101', 'BIGC670101', 'CHAM810101', 'DAYM780201' are used (Xiao, et al., 2015). An illustrated example of this amino acid physicochemical properties from the AAIndex database is provided in **Figure S3**.

All the amino acid indices are centralized and standardized prior to the calculation:

$$P_r = \frac{P_r - \overline{P}}{\sigma}$$

where $\bar{P}$ is the average of the properties of the 20 amino acids and σ is the standard deviation of the properties of the 20 amino acids. $\bar{P}$ and σ can be calculated as follows:

$$\bar{P} = \frac{\sum_{r=1}^{20} P_r}{20}, \qquad \sigma = \sqrt{\frac{1}{20}\sum_{r=1}^{20}(P_r - \bar{P})^2}$$

The Moran autocorrelation descriptors (Feng and Zhang, 2000; Lin and Pan, 2001) can thus be defined as:

$$I(d) = \frac{\frac{1}{N-d}\sum_{i=1}^{N-d}(P_i - \bar{P}')(P_{i+d} - \bar{P}')}{\frac{1}{N}\sum_{i=1}^{N}(P_i - \bar{P}')^2}, \qquad d = 1, 2, 3 \ldots, nlag$$

where *d* is the lag of the autocorrelation, *nlag* is the maximum value of the lag (default value: 30), P_i and P_{i+d} are the properties of the amino acids at positions *i* and *i* + *d*, respectively. $\bar{P}'$ is the average of the considered property *P* over the entire sequence of length *N* and is calculated as:

$$\bar{P}' = \frac{\sum_{i=1}^{N} P_i}{N}$$

Use the following command to calculate the Moran feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
Moran
```

Advanced users can adjust the property index to <index> and the maximum value of the *nlag* to <*N*> (the default is 30) by running the following Python command:

```
tcsh% python codes/Moran.py --file examples/test-protein.txt --
props <index> --nlag <N> --out Moran.tsv
```

Here, '`--props` <index>' specifies the amino acid property index from the AAindex database, where multiple indices can be used here, e.g. "CIDH920105" and "BHAR880101", separated by ":", '`--nlag` <*N*>' is a placeholder for the integer defining the maximum value of the lag. '`--out`' specifies the name of the result output file. In the example provided below, "Moran.tsv" is the result output file.

For example:

```
tcsh% python codes/Moran.py --file examples/test-protein.txt --
props CIDH920105:BHAR880101 --nlag 15 --out Moran.tsv
```

4.14 Geary correlation (Geary)

The Geary autocorrelation descriptors (Sokal and Thomson, 2006) for a protein or peptide sequence are defined as:

$$C(d)=\frac{\frac{1}{2(N-d)}\sum_{i=1}^{N-d}(P_i-P_{i+d})^2}{\frac{1}{N-1}\sum_{i=1}^{N}(P_i-\bar{P}')^2}, \qquad d=1,2,...,nlag$$

where d, P, P_i and P_{i+d}, $nlag$ have the same definitions as described above.

Use the following command to extract the Geary feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
Geary
```

Advanced users can adjust the property index to <index> and the maximum value of the *nlag* to <*N*> (the default is 30) by running the following Python command:

```
tcsh% python codes/Geary.py --file examples/test-protein.txt --
props <index> --nlag <N> --out Geary.tsv
```

Here, '`--props` <index>' specifies the amino acid property index from the AAindex database, where multiple indices can be used here, e.g. "CIDH920105" and "BHAR880101", separated by ":", '`--nlag` <*N*>' is a placeholder for the integer defining the maximum value of the lag. '`--out`' specifies the name of the result output file.

For example:

```
tcsh% python codes/Geary.py --file examples/test-protein.txt --
props CIDH920105:BHAR880101 --nlag 15 --out Geary.tsv
```

4.15 Normalized Moreau-Broto Autocorrelation (NMBroto)

The Moreau-Broto autocorrelation descriptors (Horne, 1988) are defined as follows:

$$AC(d)=\sum_{i=1}^{N-d}P_i\times P_{i+d}, \qquad d=1,2,...,nlag$$

The normalized Moreau-Broto autocorrelation descriptors are thus defined as:

$$ATS(d)=\frac{AC(d)}{N-d}, \qquad d=1,2,...,nlag$$

Use the following command to extract the NMBroto feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
NMBroto
```

Advanced users can adjust the property index to <index> and the maximum value of the *nlag* to <*N*> (the default is 30) by running the following Python command:

```
tcsh% python codes/NMBroto.py --file examples/test-protein.txt --
props <index> --nlag <N> --out NMBroto.tsv
```

Here, '`--props` <index>' specifies the amino acid property index from the AAindex database, where multiple indices can be used here, e.g. "CIDH920105" and "BHAR880101", separated by ":", '`--nlag` <*N*>' is a placeholder for the integer defining the maximum value of the lag. '`--out`' specifies the name of the output file.

For example:

```
tcsh% python codes/NMBroto.py --file examples/test-protein.txt --
props CIDH920105:BHAR880101 --nlag 15 --out NMBroto.tsv
```

4.16 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 (Cai, et al., 2003; Cai, et al., 2004; Dubchak, et al., 1995; Dubchak, et al., 1999; Han, et al., 2004). 13 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 (Tomii and Kanehisa, 1996). The groups of amino acids are listed in **Table S1**.

Table S1. Amino acid physicochemical attributes and the division of the amino acids into three groups according to each attribute.

Attribute	Division		
Hydrophobicity_PRAM900101	Polar: RKEDQN	Neutral: GASTPHY	Hydrophobicity: CLVIMFW
Hydrophobicity_ARGP820101	Polar: QSTNGDE	Neutral: RAHCKMV	Hydrophobicity: LYPFIW
Hydrophobicity_ZIMJ680101	Polar: QNGSWTDERA	Neutral: HMCKV	Hydrophobicity: LPFYI
Hydrophobicity_PONP930101	Polar: KPDESNQT	Neutral: GRHA	Hydrophobicity: YMFWLCVI
Hydrophobicity_CASG920101	Polar: KDEQPSRNTG	Neutral: AHYMLV	Hydrophobicity: FIWC
Hydrophobicity_ENGD860101	Polar: RDKENQHYP	Neutral :SGTAW	Hydrophobicity: CVLIMF
Hydrophobicity_FASG890101	Polar: KERSQD	Neutral: NTPG	Hydrophobicity:

			AYHWVMFLIC
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: 4.9-6.2 LIFWCMVY	Polarity value: 8.0-9.2 PATGS	Polarity value: 10.4-13.0 HQRKNED
Polarizability	Polarizability value: 0-1.08 GASDT	Polarizability value: 0.128-120.186 GPNVEQIL	Polarizability value: 0.219-0.409 KMHFRYW
Charge	Positive: KR	Neutral: ANCQGHILMFPSTWYV	Negative: DE
Secondary structure	Helix: EALMQKRH	Strand: VIYCWFT	Coil: GNPSD
Solvent accessibility	Buried: ALFCGIVW	Exposed: PKQEND	Intermediate: MPSTHY

4.16.1 CTDC

Taking the hydrophobicity attribute as an example, all amino acids are divided into three groups: polar, neutral and hydrophobic (**Table S1**). The Composition descriptor consists of three values: the global compositions (percentage) of polar, neutral and hydrophobic residues of the protein. An illustrated example of this encoding scheme is provided in the following **Figure S4**. The Composition descriptor can be calculated as follows:

$$C(r)=\frac{N(r)}{N},\quad r\in\{polar,neutral,hydrophobic\}$$

where *N(r)* is the number of amino acid type *r* in the encoded sequence and *N* is the length of the sequence.

Use the following command to extract the CTDC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
CTDC
```

Considering that categorizing these physicochemical properties into three groups is arbitrary, *iFeature* also allows a user-defined classification. Accordingly, we provided three additional Python scripts to address this issue by allowing users to specify their own amino acid groups and subsequently calculate the respective CTDC, CTDT and CTDD features (please refer to the following sections for that). For example, use the following command to extract the user-defined CTDC feature descriptors:

```
tcsh%    python    codes/CTDCClass.py    examples/test-protein.txt
CTDCClass.tsv RKEDQN GASTPHY CLV IMFW
```

4.16.2 CTDT

The Transition descriptor T also consists of three values (Dubchak, et al., 1995; Dubchak, et al., 1999): 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. Transitions between the neutral group and the hydrophobic group and those between the hydrophobic group and the polar group are defined in a similar way. The transition descriptor can then be calculated as:

$$T(r,s) = \frac{N(r,s)+N(s,r)}{N-1}, \quad r,s \in \{(polar, neutral),(neutral, hydrophobic),(hydrophobic, polar)\}$$

where *N(r,s)* and *N(s,r)* are the numbers of dipeptides encoded as "*rs*" and "*sr*" respectively in the sequence, while *N* is the length of the sequence. An illustrated example of this encoding scheme is provided in the following **Figure S4**.

Use the following command to extract the CTDT descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
CTDT
```

Use the following command to extract the user-defined CTDT feature descriptors:

```
tcsh%   python   codes/CTDTClass.py   examples/test-protein.txt
CTDTClass.tsv RKEDQN GASTPHY CLV IMFW
```

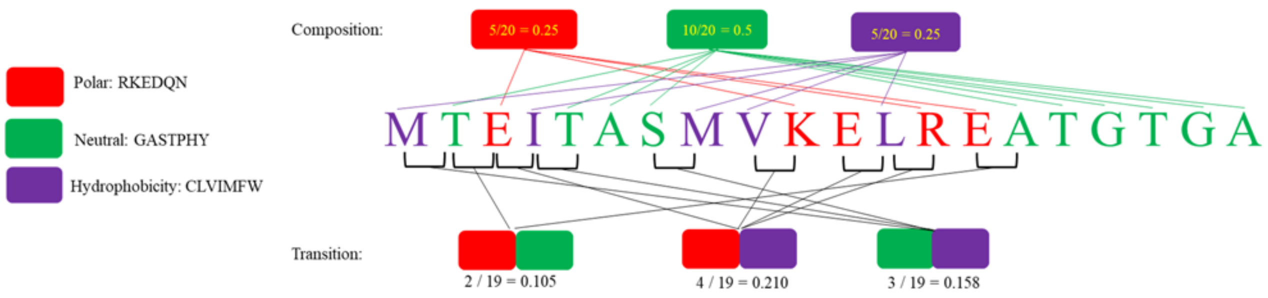


Figure S4. An illustrated example of the calculation of composition and transition descriptors. This example uses the hydrophobicity attribute.

4.16.3 CTDD

The Distribution descriptor consists of five values for each of the three groups (polar, neutral and hydrophobic) (Dubchak, et al., 1995; Dubchak, et al., 1999), 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.
For example, we start with the first residue up to and including the residue that marks 25/50/75/100% of occurrences for residues of any given group and then we simply divide the position of this residue by the length of the entire sequence.

Use the following command to extract the CTDD descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
CTDD
```

Use the following command to extract the user-defined CTDD feature descriptors:

```
tcsh%    python    codes/CTDDClass.py    examples/test-protein.txt
CTDDClass.tsv RKEDQN GASTPHY CLV IMFW
```

4.17 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 (Shen, et al., 2007). First, the protein sequence is represented by a binary space (*V, F*), where *V* denotes the vector space of the sequence features, and each feature (V_i) represents a sort of triad type; *F* is the number vector corresponding to *V*, where f_i, the value of the *i*-th dimension of *F*, is the number of type V_i appearing in the protein sequence.

For the amino acids that have been catalogued into seven classes, the size of *V* should be equal to 7 x 7 x 7=343. Accordingly, $i = 1, 2, 3, \ldots, 343$. An illustrated example of this encoding scheme is provided in the following **Figure S5**.
In principle, the longer a protein sequence, the higher the probability to have larger values of f_i, confounding the comparison of proteins with different lengths. Thus, we define a new parameter, d_i, by normalizing f_i with the following equation:

$$d_i = \frac{f_i - \min\{f_1, f_2, \ldots, f_{343}\}}{\max\{f_1, f_2, \ldots, f_{343}\}}$$

Use the following command to extract the CTriad feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
CTriad
```

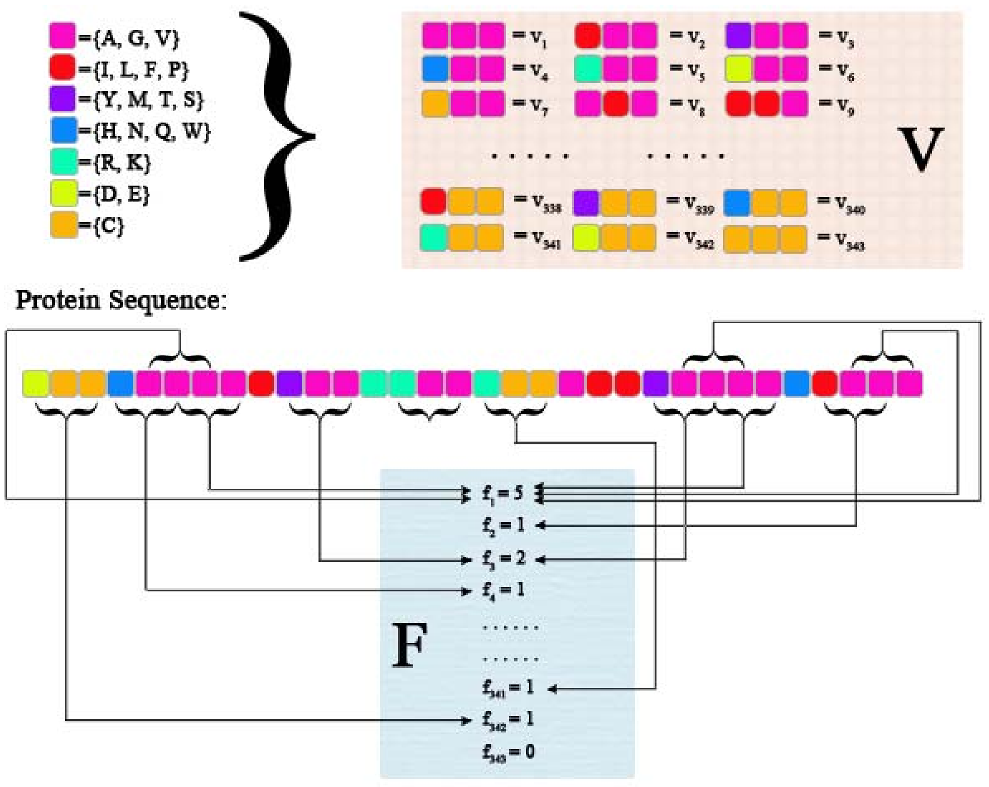


Figure S5. Schematic diagram for constructing the vector space (*V, F*) of a given protein sequence. *V* is the vector space of the sequence features; each feature (V_i) represents a triad composed of three consecutive amino acids; *F* is the number vector corresponding to *V*, and the value of the *i*-th entry of *F*, denoted f_i, is the number of occurrences that the triad associated with V_i appearing in the protein sequence. The figure was adapted from the Supplementary Figure in (Shen, et al., 2007).

4.18 *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). For example, AxRxT is a 1-spaced triad. Thus, the dimensionality of the KSCTriad encoded feature vector is 343 (*k*+1). An illustrated example of this encoding scheme is provided in the following **Figure S6**.

Use the following command to extract the KSCTriad feature descriptors:
```
tcsh% python iFeature.py --file examples/test-protein.txt --type
KSCTriad
```

Advanced users can adjust the value of *k* to <*k*> (the default is 5) by running the following Python command:
```
tcsh% python codes/KSCTriad.py examples/test-protein.txt <k>
KSCTriad.tsv
```

Here, <*k*> is a placeholder for the integer. $k = 0, 1, 2, \ldots, 5$.

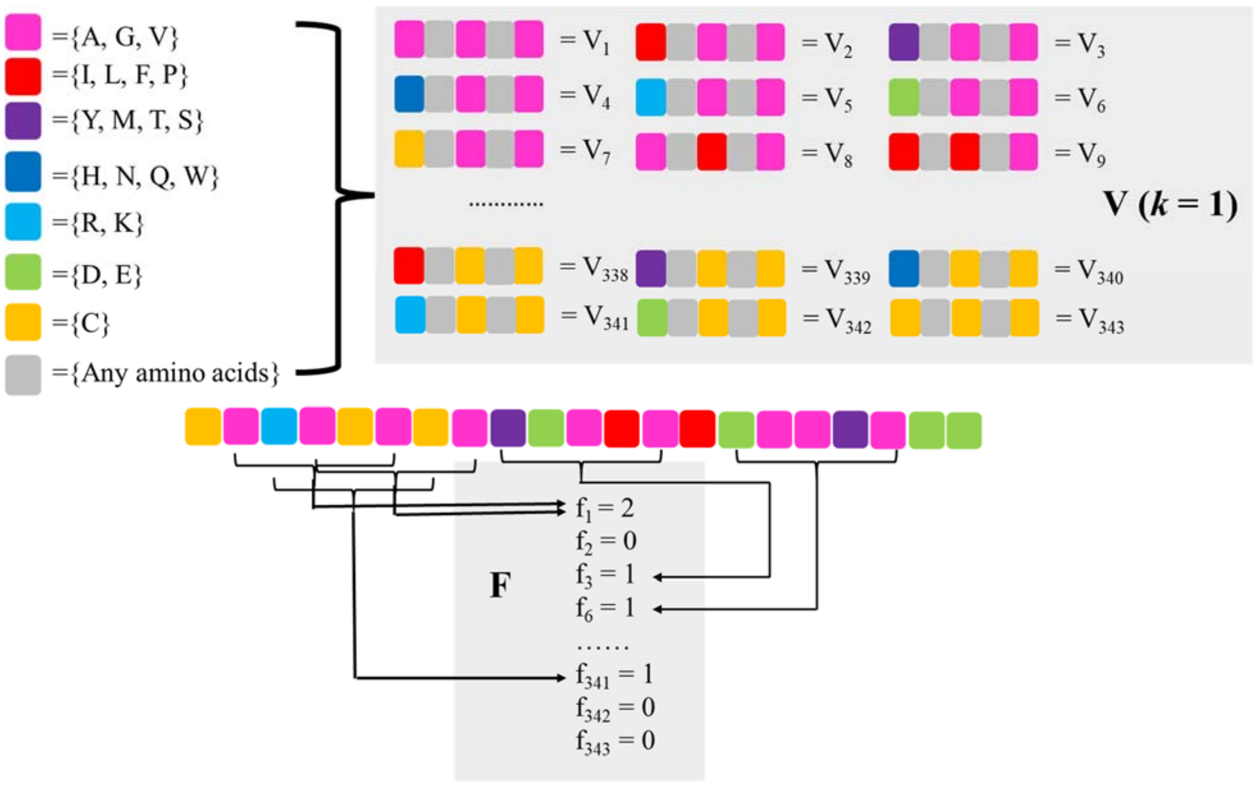


Figure S6. Schematic diagram for constructing the vector space (V, F) of protein sequence (k =1).

4.19 Sequence-Order-Coupling Number (SOCNumber)

The d-th rank sequence-order-coupling number is defined as:

$$\tau_d = \sum_{i=1}^{N-d} (d_{i,i+d})^2, \qquad d = 1, 2, 3, \ldots, nlag$$

where $d_{i,i+d}$ is the entry in a given distance matrix describing a distance between the two amino acids at position i and $i + d$, *nlag* denotes the maximum value of the lag (default value: 30) and N is the length of a protein or peptide sequence. As distance matrix both the Schneider-Wrede physicochemical distance matrix (Schneider and Wrede, 1994) used by Kuo-Chen Chou, and the chemical distance matrix by Grantham (Grantham, 1974) are used. Accordingly, the descriptor dimension will be *nlag* x 2. The quasi-sequence-order descriptors described next also utilizes the two matrices. An illustrated example of this encoding scheme is provided in the following **Figure S7**.

Note: the length of the protein must be not less than the maximum value of *nlag*.

Use the following command to extract the SOCNumber feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
SOCNumber
```

Advanced users can adjust the maximum value of the *nlag* to <*N*> (the default is 30) by running the following Python command:

```
tcsh% python codes/SOCNumber.py examples/test-protein.txt <N>
SOCNumber.tsv
```

Here, <*N*> is a placeholder for the integer defining the maximum value of *nlag*.

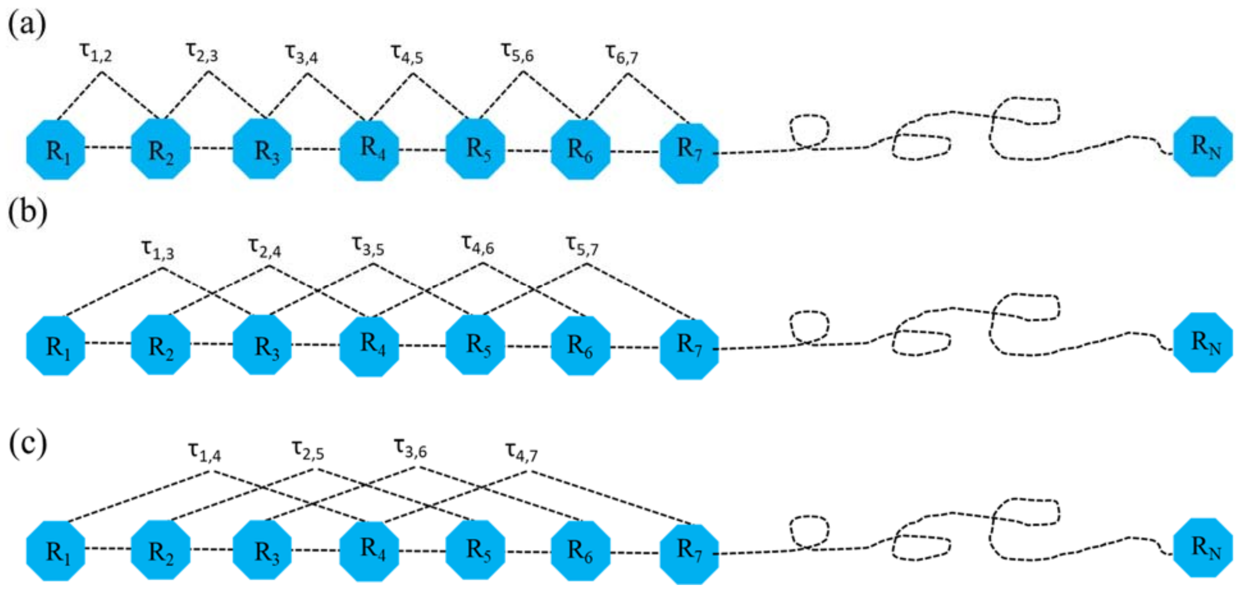


Figure S7. A schematic drawing to show (a) the 1st-rank, (b) the 2nd-rank, and (3) the 3rd-rank sequence-order-coupling mode along a protein sequence. (a) reflects the coupling mode between all the most adjacent residues, (b) shows the coupling between the adjacent plus one residues, and (c) shows the coupling between the adjacent plus two residues. This figure is adapted from (Chou, 2000).

4.20 Quasi-sequence-order (QSOrder)

For each amino acid type, a quasi-sequence-order descriptor can be defined as:

$$X_r = \frac{f_r}{\sum_{r=1}^{20} f_r + w\sum_{d=1}^{nlag} \tau_d}, \qquad r = 1, 2, ..., 20$$

where f_r is the normalized occurrence of amino acid type r and w is a weighting factor (w = 0.1), *nlag* and τ_d have the same definitions as described above. These are the first 20 quasi-sequence-order descriptors. The other 30 quasi-sequence-order descriptors are defined as:

$$X_d = \frac{w\tau_d - 20}{\sum_{r=1}^{20} f_r + w\sum_{d=1}^{nlag} \tau_d}, \qquad d = 21, 22, ..., 20 + nlag$$

Extract the QSOrder descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
QSOrder
```

Advanced users can adjust the maximum value of the *nlag* to <*N*> (the default is 30) by running the following Python command:

```
tcsh% python codes/QSOrder.py examples/test-protein.txt <N>
QSOrder.tsv
```

Here, <*N*> is a placeholder for the integer defining the maximum value of *nlag*.

4.21 Pseudo-Amino Acid Composition (PAAC)

This group of descriptors has been proposed in (Chou, 2001; Chou, 2005). Let $H_1^o(i)$, $H_2^o(i)$, $M^o(i)$ for *i* = 1, 2, 3, … 20 be the original hydrophobicity values, the original hydrophilicity values and the original side chain masses of the 20 natural amino acids, respectively. They are converted to the following quantities by a standard conversion:

$$H_1(i) = \frac{H_1^o(i) - \frac{1}{20}\sum_{i=1}^{20} H_1^o(i)}{\sqrt{\frac{\sum_{i=1}^{20}[H_1^o(i) - \frac{1}{20}\sum_{i=1}^{20} H_1^o(i)]^2}{20}}}$$

where $H_2^o(i)$ and $M^o(i)$ are normalized as $H_2(i)$ and $M(i)$ in the same manner.

Next, a correlation function can be defined as:

$$\Theta(R_i, R_j) = \frac{1}{3}\{[H_1(R_i) - H_1(R_j)]^2 + [H_2(R_i) - H_2(R_j)]^2 + [M(R_i) - M(R_j)]^2\}$$

This correlation function is actually an averaged value for the three amino acid properties: hydrophobicity value, hydrophilicity value and side chain mass. Therefore, we can extend this definition of correlation function for one amino acid property or for a set of *n* amino acid properties.

For one amino acid property, the correlation can be defined as:

$$\Theta(R_i, R_j) = [H_1(R_i) - H_1(R_j)]^2$$

where *H(R_i)* is the amino acid property of amino acid R_i after standardization.

An illustrated example of the correlation function is provided in the following **Figure S8**.

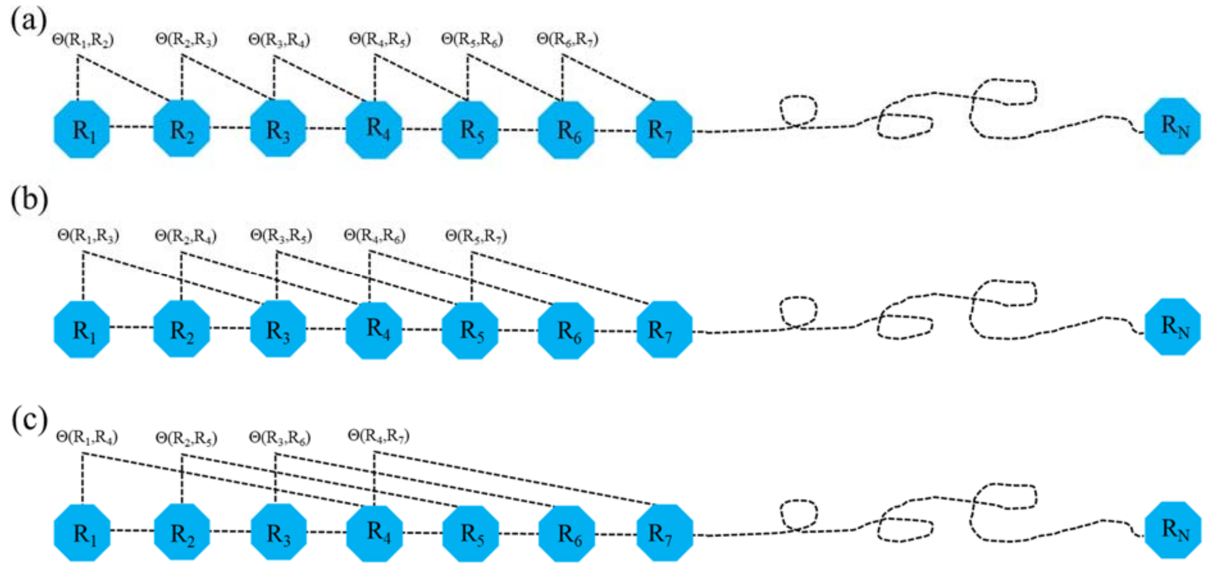


Figure S8. A schematic drawing to show (a) the first-tier, (b) the second-tier, and (3) the third-tier sequence order correlation mode along a protein sequence. (a) reflects the coupling mode between all the most adjacent residues, (b) shows the coupling between the adjacent plus one residues, and (c) shows the coupling between the adjacent plus two residues. This figure is adapted from (Chou, 2001) for illustration purposes.

For a set of *n* amino acid properties, it can be defined as:

$$\Theta(R_i, R_j) = \frac{1}{n}\sum_{n=1}^{n}[H_k(R_i) - H_k(R_j)]^2$$

where $H_k(R_i)$ is the *k*-th property in the amino acid property set for amino acid R_i.

A set of descriptors called sequence order-correlated factors are defined as:

$$\theta_1 = \frac{1}{N-1}\sum_{i=1}^{N-1}\Theta(R_i, R_{i+1})$$

$$\theta_2 = \frac{1}{N-2}\sum_{i=1}^{N-2}\Theta(R_i, R_{i+2})$$

$$\theta_3 = \frac{1}{N-3}\sum_{i=1}^{N-3}\Theta(R_i, R_{i+3})$$

$$\ldots$$

$$\theta_\lambda = \frac{1}{N-\lambda}\sum_{i=1}^{N-\lambda}\Theta(R_i, R_{i+\lambda})$$

where λ ($\lambda < N$) is an integer parameter to be chosen. Let f_i be the normalized occurrence frequency of amino acid *i* in the protein sequence. Then, a set of $20 + \lambda$ descriptors called the pseudo-amino acid composition for a protein sequence can be defines as:

$$X_c = \frac{f_c}{\sum_{r=1}^{20} f_r + w\sum_{j=1}^{\lambda} \theta_j}, \qquad (1 < c < 20)$$

$$X_c = \frac{w\theta_{c-20}}{\sum_{r=1}^{20} f_r + w\sum_{j=1}^{\lambda} \theta_j}, \qquad (21 < c < 20 + \lambda)$$

where w is the weighting factor for the sequence-order effect and is set to $w = 0.05$ in *iFeature* as suggested by Chou *et al.* (Chou, 2001).

Use the following command to extract the PAAC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
PAAC
```

Advanced users can adjust the maximum value of the λ to <*N*> (the default is 30) by running the following Python command:

```
tcsh% python codes/PAAC.py examples/test-protein.txt <N> PAAC.tsv
```

Here, <*N*> is a placeholder for the integer defining the maximum value of λ.

4.22 Amphiphilic Pseudo-Amino Acid Composition (APAAC)

Amphiphilic Pseudo-Amino Acid Composition (APAAC) was proposed in (Chou, 2001; Chou, 2005). The definition of this set of features is similar to the PAAC descriptors. Using $H_1(i)$ and $H_2(j)$ as previously defined, the hydrophobicity and hydrophilicity correlation functions are defined as:

$$H^1_{i,j} = H_1(i)H_1(j)$$
$$H^2_{i,j} = H_2(i)H_2(j),$$

respectively. An illustrated example of the correlation functions is provided in the following **Figure S9**.

Thus, sequence order factors can be defined as:

$$\tau_1 = \frac{1}{N-1}\sum_{i=1}^{N-1} H_{i,i+1}^{1}$$

$$\tau_2 = \frac{1}{N-1}\sum_{i=1}^{N-1} H_{i,i+1}^{2}$$

$$\tau_3 = \frac{1}{N-2}\sum_{i=1}^{N-2} H_{i,i+2}^{1}$$

$$\tau_4 = \frac{1}{N-2}\sum_{i=1}^{N-2} H_{i,i+2}^{2}$$

$$\ldots$$

$$\tau_{2\lambda-1} = \frac{1}{N-\lambda}\sum_{i=1}^{N-\lambda} H_{i,i+\lambda}^{1}$$

$$\tau_{2\lambda} = \frac{1}{N-\lambda}\sum_{i=1}^{N-\lambda} H_{i,i+\lambda}^{2}$$

Then, a set of descriptors, called Amphiphilic Pseudo-Amino Acid Composition (APAAC), is defined as:

$$P_c = \frac{f_c}{\sum_{r=1}^{20} f_r + w\sum_{j=1}^{2\lambda} \tau_j}, \qquad (1 < c < 20)$$

$$P_c = \frac{\omega\tau_u}{\sum_{r=1}^{20} f_r + w\sum_{j=1}^{2\lambda} \tau_j}, \qquad (21 < u < 20 + 2\lambda)$$

where w is the weighting factor. In *iFeature* this factor is set to $w = 0.5$ as described in Chou's work (Chou, 2001).

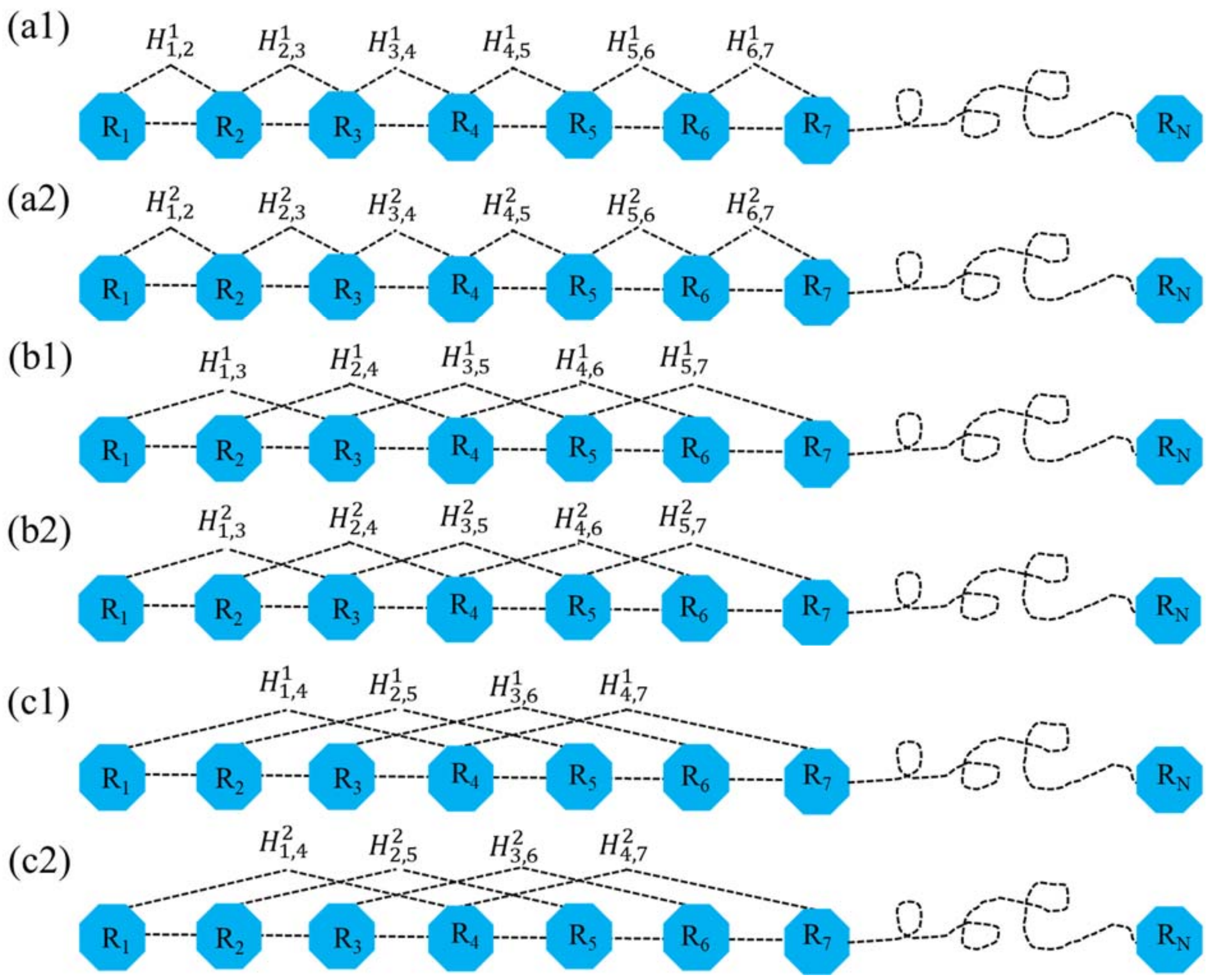

(a1)
$H^1_{1,2}$ $H^1_{2,3}$ $H^1_{3,4}$ $H^1_{4,5}$ $H^1_{5,6}$ $H^1_{6,7}$
(a2)
$H^2_{1,2}$ $H^2_{2,3}$ $H^2_{3,4}$ $H^2_{4,5}$ $H^2_{5,6}$ $H^2_{6,7}$
(b1)
$H^1_{1,3}$ $H^1_{2,4}$ $H^1_{3,5}$ $H^1_{4,6}$ $H^1_{5,7}$
(b2)
$H^2_{1,3}$ $H^2_{2,4}$ $H^2_{3,5}$ $H^2_{4,6}$ $H^2_{5,7}$
(c1)
$H^1_{1,4}$ $H^1_{2,5}$ $H^1_{3,6}$ $H^1_{4,7}$
(c2)
$H^2_{1,4}$ $H^2_{2,5}$ $H^2_{3,6}$ $H^2_{4,7}$
R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_N

Figure S9. A schematic diagram to show (a1/a2) the first-rank, (b1/b2) the second-rank and (c1/c2) the third-rank sequence-order-coupling mode along a protein sequence through a hydrophobicity/hydrophilicity correlation function, where $H^1_{i,j}$ and $H^2_{i,j}$ are given by the aforementioned equation. Panels (a1/a2) reflects the coupling mode between the most adjacent residues, panels (b1/b2) shows the coupling between the adjacent plus one residues, and panels (c1/c2) shows the coupling between the adjacent plus two residues. This figure is adapted from (Chou, 2005) for illustration purposes.

Use the following command to extract the APAAC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
APAAC
```

Advanced users can adjust the maximum value of the λ to <*N*> (the default is 30) by running the following Python command:

```
tcsh% python codes/APAAC.py examples/test-protein.txt <N> APAAC.tsv
```

Here, <*N*> is a placeholder for the integer defining the maximum value of λ.

4.23 *K*-Nearest Neighbor for peptides (KNNpeptide)

The *K*-Nearest Neighbor for peptides (KNNpeptide) descriptor (Chen, et al., 2013a) 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. Here, the similarity score between two peptides is defined as:

$$Score = \sum_{i=1}^{n} S(P_{1,i}, P_{2,i})$$

$$S(a,b) = \{ \begin{matrix} BLOSUM62(a,b), & if\,(BLOSUM62) > 0 \\ 0, & if\,(BLOSUM62) \le 0 \end{matrix}$$

where *P* is the peptide with *n* amino acids, *i* is the sequence position, and *BLOSUM62(a,b)* is the corresponding element value for amino acids *a* and *b* in the BLOSUM62 matrix. The label file divides the protein sequences in the training file into different classes (positive samples and negative samples or multiclass). Then, the ratios of different sample classes for the top *K*-Nearest Neighbor peptides will be calculated. The default *K* values in KNNpeptide are set based on a set of values (i.e. 1%, 2%, 3%, …, 30% of the total numbers of samples in the training file). KNNpeptide descriptor can be applied to encode peptides of equal length.

Use the following command to extract the KNNpeptide descriptors:
```
tcsh% python iFeature.py --file examples/test-peptide.txt --type KNNpeptide --train examples/train-peptide.txt --label examples/label.txt
```

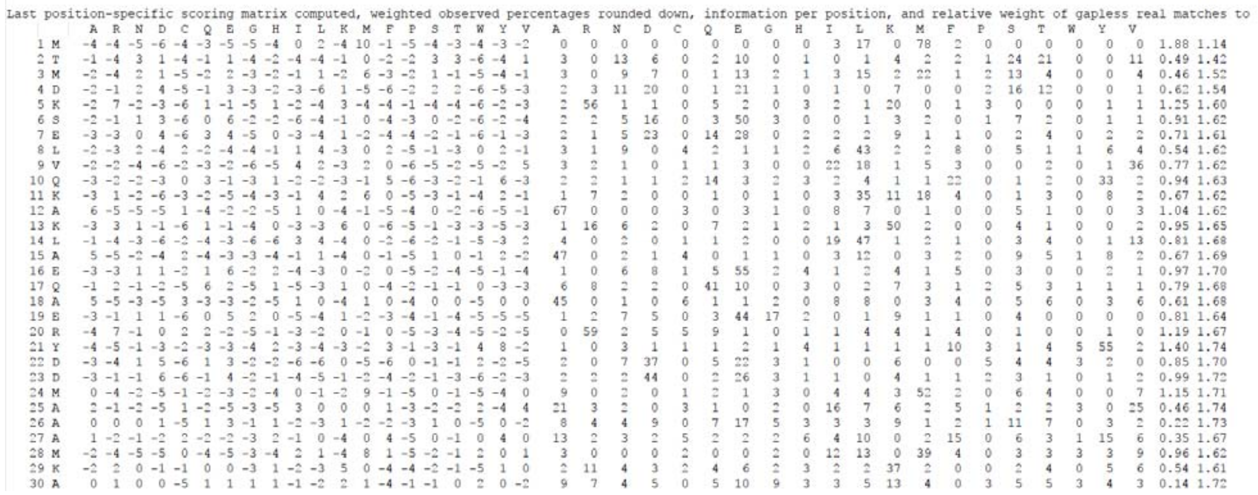
4.24 *K*-Nearest Neighbor for proteins (KNNprotein)

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 (Needleman and Wunsch, 1970).

Use the following command to extract the KNNprotein feature descriptors:
```
tcsh% python iFeature.py --file examples/test-protein.txt --type KNNprotein --train examples/train-protein.txt --label examples/label.txt
```

4.25 PSSM profile (PSSM)

This feature descriptor (Cai, et al., 2012; Radivojac, et al., 2010) is extracted from the Position-Specific Scoring Matrix (PSSM) profile. An illustrated example of PSSM profile is provided in **Figure S10**. The PSSM profile can be obtained by running PSI-BLAST (Altschul, et al., 1997) 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.



```
Last position-specific scoring matrix computed, weighted observed percentages rounded down, information per position, and relative weight of gapless real matches to
         A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V     A   R   N   D   C   Q   E   G   H   I   L   K   M   F   P   S   T   W   Y   V
   1 M  -4 -4 -5 -6 -4 -3 -5 -5 -4  0  2 -4 10 -1 -5 -4 -3 -4 -3 -2     0   0   0   0   0   0   0   0   0   3  17   0  78   2   0   0   0   0   0   0  1.88 1.14
   2 T  -1 -4  3  1 -4 -1  1 -4 -2 -4 -4 -1  0 -2 -2  3  3 -6 -4  1     3   0  13   6   0   2  10   0   1   0   1   4   2   2   1  24  21   0   0  11  0.49 1.42
   3 M  -2 -4  2  1 -5 -2  2 -3 -2 -1  1 -2  6 -3 -2  1 -1 -5 -4 -1     3   0   9   7   0   1  13   2   1   3  15   2  22   1   2  13   4   0   0   4  0.46 1.52
   4 D  -2 -1  2  4 -5 -1  3 -3 -2 -3 -6  1 -5 -6 -2  2  2 -6 -5 -3     2   3  11  20   0   1  21   1   0   1   0   7   0   0   2  16  12   0   0   1  0.62 1.54
   5 K  -2  7 -2 -3 -6  1 -1 -5  1 -2 -4  3 -4 -4 -1 -4 -4 -6 -2 -3     2  56   1   1   0   5   2   0   3   2   1  20   0   1   3   0   0   0   1   1  1.25 1.60
   6 S  -2 -1  1  3 -6  0  6 -2 -2 -6 -4 -1  0 -4 -3  0 -2 -6 -2 -4     2   2   5  16   0   3  50   3   0   0   1   3   2   0   1   7   2   0   1   1  0.91 1.62
   7 E  -3 -3  0  4 -6  3  4 -5  0 -3 -4  1 -2 -4 -4 -2 -1 -6 -1 -3     2   1   5  23   0  14  28   0   2   2   2   9   1   1   0   2   4   0   2   2  0.71 1.61
   8 L  -2 -3  2 -4  2 -2 -4 -4 -1  1  4 -3  0  2 -5 -1 -3  0  2 -1     3   1   9   0   4   2   1   1   2   6  43   2   2   8   0   5   1   1   6   4  0.54 1.62
   9 V  -2 -2 -4 -6 -2 -3 -2 -6 -5  4  2 -3  2  0 -6 -5 -2 -5 -2  5     3   2   1   0   1   1   3   0   0  22  18   1   5   3   0   0   2   0   1  36  0.77 1.62
  10 Q  -3 -2 -2 -3  0  3 -1 -3  1 -2 -2 -3 -1  5 -6 -3 -2 -1  6 -3     2   2   1   1   2  14   3   2   3   2   4   1   1  22   0   1   2   0  33   2  0.94 1.63
  11 K  -3  1 -2 -6 -3 -2 -5 -4 -3 -1  4  2  6  0 -5 -3 -1 -4  2 -1     1   7   2   0   0   1   0   1   0   3  35  11  18   4   0   1   3   0   8   2  0.67 1.62
  12 A   6 -5 -5 -5  1 -4 -2 -2 -5  1  0 -4 -1 -5 -4  0 -2 -6 -5 -1    67   0   0   0   3   0   3   1   0   8   7   0   1   0   0   5   1   0   0   3  1.04 1.62
  13 K  -3  3  1 -1 -6  1 -1 -4  0 -3 -3  6  0 -6 -5 -1 -3 -3 -5 -3     1  16   6   2   0   7   2   1   2   1   3  50   2   0   0   4   1   0   0   2  0.95 1.65
  14 L  -1 -4 -3 -6 -2 -4 -3 -6 -6  3  4 -4  0 -2 -6 -2 -1 -5 -3  2     4   0   2   0   1   1   2   0   0  19  47   1   2   1   0   3   4   0   1  13  0.81 1.68
  15 A   5 -5 -2 -4  2 -4 -3 -3 -4 -1  1 -4  0 -1 -5  1  0 -1  2 -2    47   0   2   1   4   0   1   1   0   3  12   0   3   2   0   9   5   1   8   2  0.67 1.69
  16 E  -3 -3  1  1 -2  1  6 -2  2 -4 -3  0 -2  0 -5 -2 -4 -5 -1 -4     1   0   6   8   1   5  55   2   4   1   2   4   1   5   0   3   0   0   2   1  0.97 1.70
  17 Q  -1  2 -1 -2 -5  6  2 -5  1 -5 -3  1  0 -4 -2 -1 -1  0 -3 -3     6   8   2   2   0  41  10   0   3   0   2   7   3   1   2   5   3   1   1   1  0.79 1.68
  18 A   5 -5 -3 -5  3 -3 -3 -2 -5  1  0 -4  1  0 -4  0  0 -5  0  0    45   0   1   0   6   1   1   2   0   8   8   0   3   4   0   5   6   0   3   6  0.61 1.68
  19 E  -3 -1  1  1 -6  0  5  2  0 -5 -4  1 -2 -3 -4 -1 -4 -5 -5 -5     1   2   7   5   0   3  44  17   2   0   1   9   1   1   0   4   0   0   0   0  0.81 1.64
  20 R  -4  7 -1  0  2  2 -2 -5 -1 -3 -2  0 -1  0 -5 -3 -4 -5 -2 -5     0  59   2   5   5   9   1   0   1   1   4   4   1   4   0   1   0   0   1   0  1.19 1.67
  21 Y  -4 -5 -1 -3 -2 -3 -3 -4  2 -3 -4 -3 -2  3 -1 -3 -1  4  8 -2     1   0   3   1   1   1   2   1   4   1   1   1   1  10   3   1   4   5  55   2  1.40 1.74
  22 D  -3 -4  1  5 -6  1  3 -2 -2 -6 -6  0 -5 -6  0 -1 -1  2 -2 -5     2   0   7  37   0   5  22   3   1   0   0   6   0   0   5   4   4   3   2   0  0.85 1.70
  23 D  -3 -1 -1  6 -6 -1  4 -2 -1 -4 -5 -1 -2 -4 -2 -1 -3 -6 -2 -3     2   2   2  44   0   2  26   3   1   1   0   4   1   1   2   3   1   0   1   2  0.99 1.72
  24 M   0 -4 -2 -5 -1 -2 -3 -2 -4  0 -1 -2  9 -1 -5  0 -1 -5 -4  0     9   0   2   0   1   2   1   3   0   4   4   3  52   2   0   6   4   0   0   7  1.15 1.71
  25 A   2 -1 -2 -5  1 -2 -5 -3 -5  3  0  0  0  1 -3 -2 -2  2 -4  4    21   3   2   0   3   1   0   2   0  16   7   6   2   5   1   2   2   3   0  25  0.46 1.74
  26 A   0  0  0  1 -5  1  3 -1  1 -2 -3  1 -2 -2 -3  1  0 -5  0 -2     8   4   4   9   0   7  17   5   3   3   3   9   1   2   1  11   7   0   3   2  0.22 1.73
  27 A   1 -2 -1 -2  2 -2 -2 -3  2 -1  0 -4  0  4 -5  0 -1  0  4  0    13   2   3   2   5   2   2   2   6   4  10   0   2  15   0   6   3   1  15   6  0.35 1.67
  28 M  -2 -4 -5 -5  0 -4 -5 -3 -4  2  1 -4  8  1 -5 -2 -1  2  0  1     3   0   0   0   2   0   0   2   0  12  13   0  39   4   0   3   3   3   3   9  0.96 1.62
  29 K  -2  2  0 -1 -1  0  0 -3  1 -2 -3  5  0 -4 -4 -2 -1 -5  1  0     2  11   4   3   2   4   6   2   3   2   2  37   2   0   0   2   4   0   5   6  0.54 1.61
  30 A   0  1  0  0 -5  1  1  1  1 -1 -2  2  1 -4 -1 -1  0  2  0 -2     9   7   4   5   0   5  10   9   3   3   5  13   4   0   3   5   5   3   4   3  0.14 1.72
```

Figure S10. Example of a PSSM profile.

Use the following command to extract the PSSM feature descriptors:

```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
PSSM --path examples/predictedProteinProperty
```

'`--path`' specifies the path of the protein PSSM profile files and the PSSM profiles can be obtained by running the script "generatePSSMProfile.py" in the scripts directory.

4.26 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 (Kawashima, et al., 2008) 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 (Tung and Ho, 2008) can be applied to encode peptides of equal length.

Use the following command to perform extract the AAINDEX feature descriptors:

```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
AAINDEX
```

4.27 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 (Lee, et al., 2011a) can be applied to encode peptides of equal length.

Use the following command to extract the BLOSUM62 descriptors:
```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
BLOSUM62
```

4.28 Secondary Structure Elements Content (SSEC)

Protein secondary structure was first predicted by the PSIPRED V4.0 software (Jones, 1999). Here, the content of three types of secondary structure elements is calculated.

$$f(sse) = \frac{N(sse)}{N}, \qquad sse \in \{Helix, Strand, Coil\}$$

where *N(sse)* is the number of the secondary structure element *sse* and *N* is the length of the protein/peptide sequence.

Use the following command to extract the SSEC feature descriptors:
```
tcsh% python iFeature.py --file examples/test-protein.txt --type
SSEC --path examples/predictedProteinProperty
```

'`--path`' specifies the path of the predicted protein secondary structure files, and the predicted secondary structure file can be obtained by running the script "generateSecondaryStructure.py" in the scripts directory.

4.29 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 descriptors:
```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
SSEB --path examples/predictedProteinProperty
```

4.30 Disorder (Disorder)

Protein disorder information was first predicted by the VSL2 software (Obradovic, et al., 2005; Peng, et al., 2006). The predicted probability value is taken as the feature. The Disorder descriptor (Cai, et al., 2012; Chen, et al., 2015) can be applied to encode peptides of equal length.

Use the following command to extract the Disorder descriptors:
```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
Disorder --path examples/predictedProteinProperty
```

'`--path`' specifies the path of the predicted protein disorder files.

4.31 DisorderC (DisorderC)

For this descriptor, the content of disorder and order is calculated.

$$f(d) = \frac{N(d)}{N}, \quad d \in \{order, disorder\}$$

where *N(d)* is the number of ordered or disordered residues and *N* is the length of the protein/peptide sequence.

Use the following command to extract the DisorderC feature descriptors:

```
tcsh% python iFeature.py --file examples/test-protein.txt --type
DisorderC --path examples/predictedProteinProperty
```

4.32 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 iFeature.py --file examples/test-peptide.txt --type
DisorderB --path examples/predictedProteinProperty
```

4.33 Accessible Solvent accessibility (ASA)

The protein Accessible Solvent Accessibility information was first predicted by the SPINE-X software (Faraggi, et al., 2009; Heffernan, et al., 2016; Heffernan, et al., 2015). 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 iFeature.py --file examples/test-peptide.txt --type
ASA --path examples/predictedProteinProperty
```

'`--path`' specifies the path of the SPINE-X output files.

4.34 Torsion angle (TA)

The protein Torsion Angle information was also introduced first by the SPINE-X software (Faraggi, et al., 2009; Heffernan, et al., 2016; Heffernan, et al., 2015). 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 iFeature.py --file examples/test-peptide.txt --type TA
--path examples/predictedProteinProperty
```

'`--path`' specifies the path of the SPINE-X output files.

4.35 Z-Scale (ZSCALE)

For this descriptor, each amino acid is characterized by five physicochemical descriptor variables (cf. **Table S2**), which were developed by Sandberg et al. in 1998 (Sandberg, et al., 1998). The ZSCALE descriptor (Chen, et al., 2012) can be applied to encode peptides of equal length.

Use the following command to extract the ZSCALE feature descriptors:

```
tcsh% python iFeature.py --file examples/test-peptide.txt --type
ZSCALE
```

Table S2. Z-scale for the 20 amino acids.

Amino acid	Z1	Z2	Z3	Z4	Z5	Amino Acid	Z1	Z2	Z3	Z4	Z5
A	0.24	-2.32	0.60	-0.14	1.30	M	-2.85	-0.22	0.47	1.94	-0.98
C	0.84	-1.67	3.71	0.18	-2.65	N	3.05	1.60	1.04	-1.15	1.61
D	3.98	0.93	1.93	-2.46	0.75	P	-1.66	0.27	1.84	0.70	2.00
E	3.11	0.26	-0.11	-3.04	-0.25	Q	1.75	0.50	-1.44	-1.34	0.66
F	-4.22	1.94	1.06	0.54	-0.62	R	3.52	2.50	-3.50	1.99	-0.17
G	2.05	4.06	0.36	-0.82	-0.38	S	2.39	-1.07	1.15	-1.39	0.67
H	2.47	1.95	0.26	3.90	0.09	T	0.75	-2.18	-1.12	-1.46	-0.40
I	-3.89	-1.73	-1.71	-0.84	0.26	V	-2.59	-2.64	-1.54	-0.85	-0.02
K	2.29	0.89	-2.49	1.49	0.31	W	-4.36	3.94	0.59	3.44	-1.59
L	-4.28	-1.30	-1.49	-0.72	0.84	Y	-2.54	2.44	0.43	0.04	-1.47

4.36 48 pseudo *K*-tuple reduced amino acids composition (PseKRAAC)

Previous studies indicate that certain residues are similar in their physicochemical features, and can be clustered into groups because they play similar structural or functional roles in proteins (Wang and Wang, 1999). By implementing reduced amino acid alphabets, the protein complexity can be significantly simplified, which reduces information redundancy and decreases the risk of overfitting. The Pseudo *K*-tuple Reduced Amino Acids Composition (PseKRAAC) descriptor (Zuo, et al., 2017) includes two different feature types for protein sequence analysis: *g-gap* and λ-correlation PseKRAAC (Chou, 2001).
The *g-gap* PseKRAAC is used to represent a protein sequence with a vector containing RAAC^K components, where *g* represents the gap between each *K*-tuple peptides (Liu, et al., 2015a; Liu, et al., 2015b; Liu, et al., 2015c; Wang, et al., 2016). A *g-gap* of *n* reflects the sequence-order information for all *K*-tuple peptides with the starting residues separated by *n* residues. An illustrated

example of this encoding scheme ($K = 2$) is provided in the following **Figure S11A**.
The λ-correlation PseKRAAC is used to represent a protein sequence with a vector containing $RAAC^K$ components, where λ is an integer that represents the correlation tier and is less than N-K, where N is the sequence length. The n-th-tier correlation factor ($\lambda = n$) reflects the sequence-order correlation between the n-th nearest residues. An illustrated example of this encoding scheme (K = 2) is provided in the following **Figure S11B**.

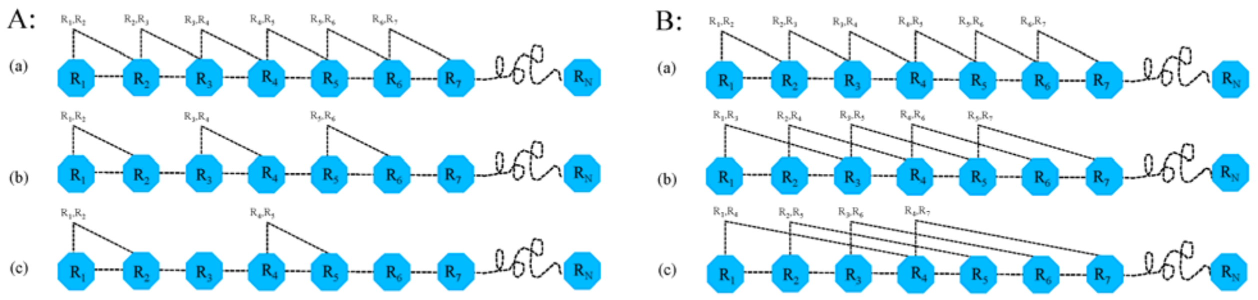


Figure S11. A schematic diagram to show: (A) *g-gap* definition of dipeptide, and (B) λ-correlation definition of dipeptide A: (a) *g-gap* of 0 reflects the sequence-order information between all adjacent dipeptides, i.e. separated by zero residues, (b) *g-gap* of 1 reflects the sequence-order information for all dipeptides with the starting residues separated by one residue, and (c) *g-gap* of 2 reflects the sequence-order information for all dipeptides with the starting residues separated by two residues; B: (a) the first-tier correlation factor reflects the sequence-order correlation between the nearest residues along a protein chain, (b) the second-tier correlation factor reflects the sequence-order correlation between the second nearest residues, (c) the third-tier correlation factor reflects the sequence-order correlation between the 3rd nearest residues, and so forth. The figure is adapted from (Zuo, et al., 2017).

The 16 types of reduced amino acid alphabets with different clustering approaces can be used to generate different versions of pseudo reduced amino acid compositions (PseRAACs) (**Table S3**).

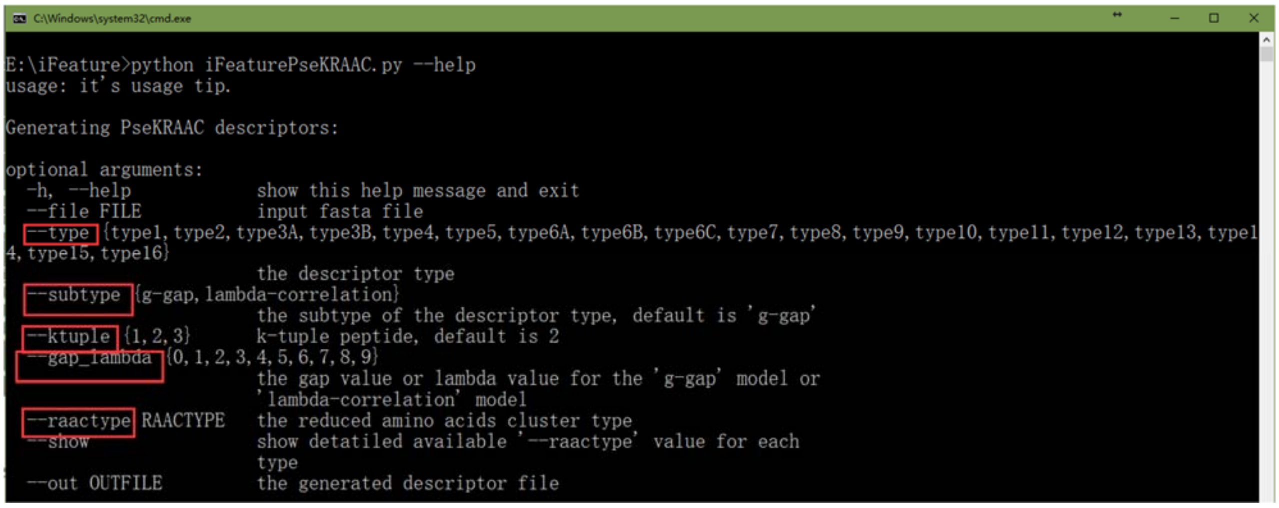
Table S3. A list of 16 types of reduced amino acid alphabets for proteins (Zuo, et al., 2017).

Type	Description	Cluster	Reference
1	RedPSSM	2-19	(Liang, et al., 2015)
2	BLOSUM 62 matrix	2-6, 8, 15	(Ogul and Mumcuoglu, 2007)
3	PAM matrix (3A) and WAG matrix (3B)	2-19	(Kosiol, et al., 2004)
4	Protein Blocks	5,8,9,11,13	(Zuo and Li, 2010)
5	BLOSUM50 matrix	3,4,8,10,15	(Zuo and Li, 2010)
6	Multiple cluster	4,5A,5B,5C	(Melo and Marti-Renom, 2006)
7	Metric multi-dimensional scaling	2-19	(Rakshit and Ananthasuresh, 2008)
8	Grantham Distance Matrix	2-19	(Susko and Roger, 2007)
9	Grantham Distance Matrix	2-19	(Susko and Roger, 2007)
10	BLOSUM matrix for SWISS-PROT	2-19	(Liu, et al., 2002)
11	BLOSUM matrix for SWISS-PROT	2-19	(Liu, et al., 2002)
12	BLOSUM matrix for DAPS	2-18	(Li and Wang, 2007)
13	Coarse-graining substitution	4,12,17	(Peterson, et al., 2009)

	matrices		
14	Alphabet Simplifer	2-19	(Cannata, et al., 2002)
15	MJ matrix	2-16	(Li, et al., 2003)
16	BLOSUM50 matrix	2-16	(Li, et al., 2003)

Use the following command to obtain the help information:

```
tcsh% python iFeaturePseKRAAC.py --help
```



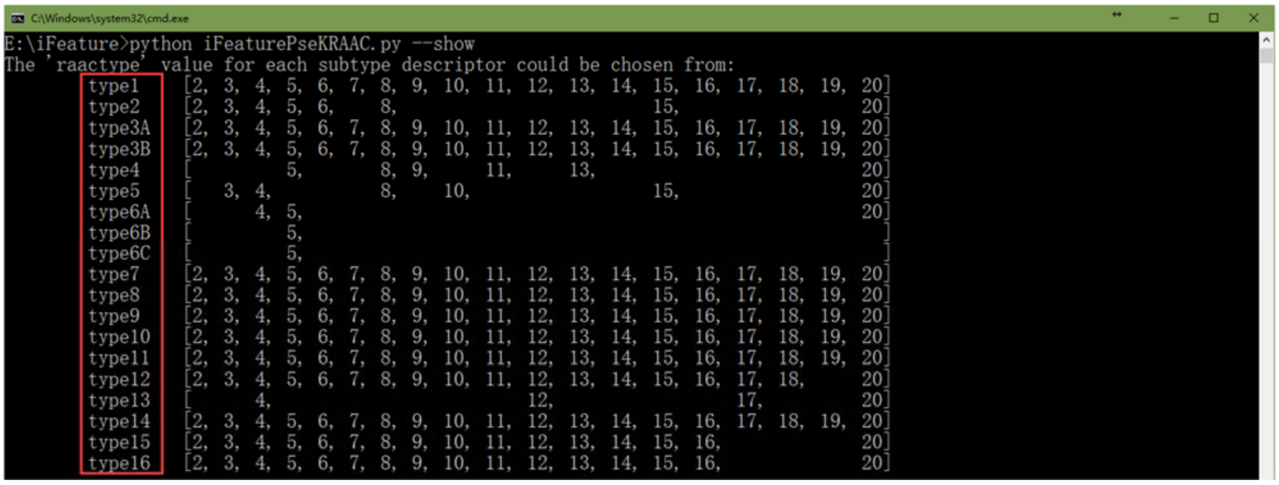
```
E:\iFeature>python iFeaturePseKRAAC.py --help
usage: it's usage tip.

Generating PseKRAAC descriptors:

optional arguments:
  -h, --help            show this help message and exit
  --file FILE           input fasta file
  --type {type1,type2,type3A,type3B,type4,type5,type6A,type6B,type6C,type7,type8,type9,type10,type11,type12,type13,type1
4,type15,type16}
                        the descriptor type
  --subtype {g-gap,lambda-correlation}
                        the subtype of the descriptor type, default is 'g-gap'
  --ktuple {1,2,3}      k-tuple peptide, default is 2
  --gap_lambda {0,1,2,3,4,5,6,7,8,9}
                        the gap value or lambda value for the 'g-gap' model or
                        'lambda-correlation' model
  --raactype RAACTYPE   the reduced amino acids cluster type
  --show                show detatiled available '--raactype' value for each
                        type
  --out OUTFILE         the generated descriptor file
```

The following parameters are required by 'iFeaturePseKRAAC.py':

- `--file` protein/peptide sequence file in fasta format
- `--type` descriptor type in the PseKRAAC feature group
- `--subtype` 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)
- `--raactype` the reduced amino acids cluster type.

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

```
tcsh% python iFeaturePseKRAAC.py --show
```



```
C:\Windows\system32\cmd.exe
E:\iFeature>python iFeaturePseKRAAC.py --show
The 'raactype' value for each subtype descriptor could be chosen from:
        type1   [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type2   [2,  3,  4,  5,  6,      8,                          15,                 20]
        type3A  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type3B  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type4   [            5,          8,  9,      11,     13,                         20]
        type5   [    3,  4,              8,      10,                 15,                 20]
        type6A  [        4,  5,                                                          20]
        type6B  [            5,                                                            ]
        type6C  [            5,                                                            ]
        type7   [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type8   [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type9   [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type10  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type11  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type12  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18,     20]
        type13  [        4,                              12,                 17,         20]
        type14  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
        type15  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16,             20]
        type16  [2,  3,  4,  5,  6,  7,  8,  9,  10, 11, 12, 13, 14, 15, 16,             20]
```

Use the following command to extract the PseKRAAC feature descriptors:

```
tcsh% python iFeaturePseKRAAC.py --file examples/test-protein.txt
--type type1 --subtype lambda-correlation --ktuple 2 --gap_lambda
2 --raactype 5
```

5. Feature selection analysis using *iFeature*

iFeature integrates several commonly used (and very useful) feature selection, clustering, and dimensionality reduction algorithms. In order to facilitate the interpretation of the results for non-expert users, a scatter diagram will be plotted, showing the distribution of the clusters. The clustering result will also be stored in a text file. The clustering algorithms can be run using the following command:

```
tcsh%    python    cluster.py    --file    descriptor.tsv    --type
<clustering_algorithm> --sof <sample/feature>
```

'--file' is the descriptor output file, which is generated by 'iFeature.py' or 'iFeaturePseKRAAC.py', '--type' specifies the clustering algorithm (kmeans, hcluster, apc, meanshift and dbscan). '--sof' specifies the option for performing clustering for samples or features (default: samples).

5.1 *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 (Jain, et al., 1999; Rokach and Maimon, 2005). 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.
The K-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster "centroids". Note that they are not, in general, points from the set X, although they live in the same space. The

K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(\|x_j - \mu_j\|^2)$$

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

```
tcsh% python cluster.py --file examples/example.tsv --type kmeans
--sof feature --nclusters 2
```

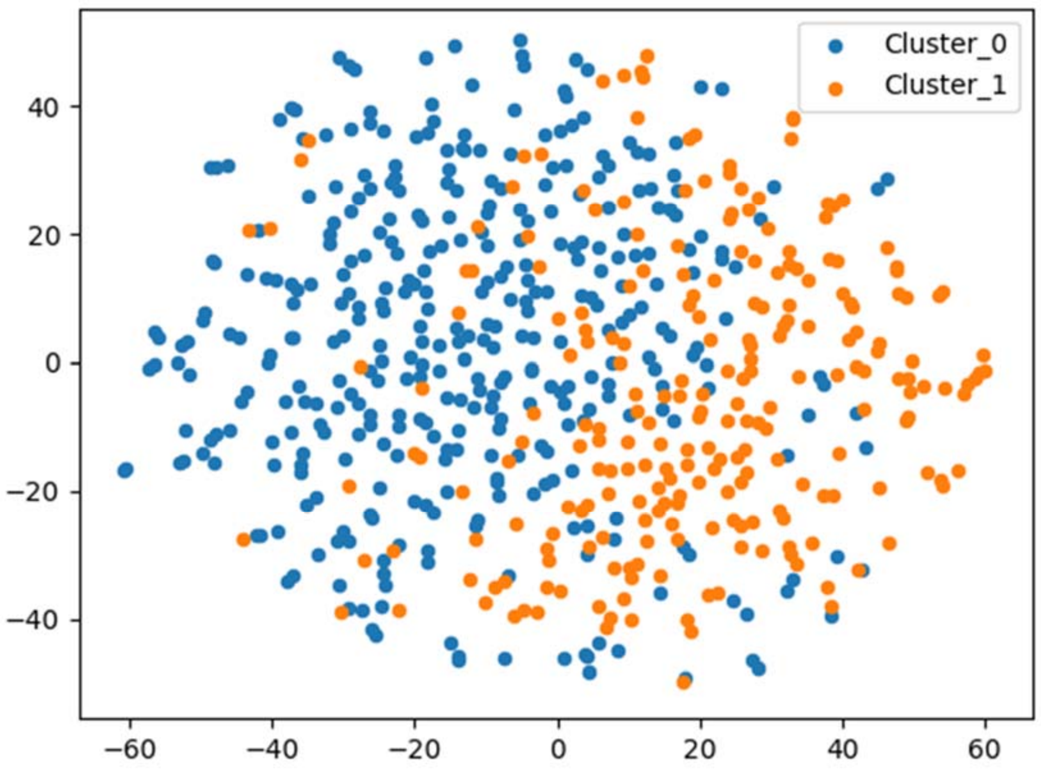


Figure S12. An example of the scatter diagram generated by *iFeature* for 'kmeans' clustering. The dataset is composed of samples of malonylation and non-malonylation sites in mammals and the feature extraction method is EAAC.

5.2 Hierarchical clustering (hcluster)

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively (Jain, 2010; Jain, et al., 1999; Rokach and Maimon, 2005). 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 hierarchical clustering:

```
tcsh% python cluster.py --file examples/example.tsv --type hcluster
--sof feature
```

In addition to the scatter diagram, for 'hcluster' clustering, a hierarchical cluster diagram can also be alternatively plotted. An example is shown in Figure S13 below.

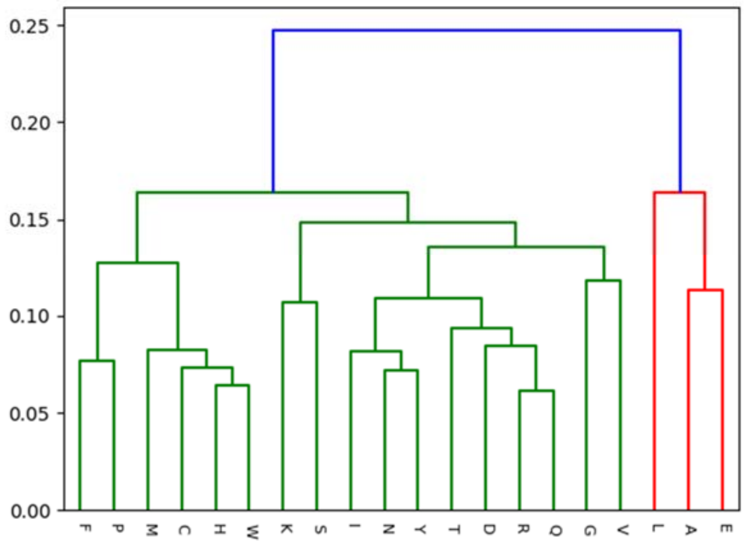


Figure S13. Example of a hierarchical cluster diagram for 'hcluster' clustering.

5.3 Affinity Propagation clustering (apc)

Affinity Propagation creates clusters by sending messages between pairs of samples until convergence (Frey and Dueck, 2007). 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 Affinity Propagation clustering:

```
tcsh% python cluster.py --file examples/example.tsv --type apc --
sof feature
```

5.4 Mean Shift clustering (meanshift)

MeanShift clustering aims to discover blobs in a smooth density of samples (Cheng, 1995). 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 Mean Shift clustering:

```
tcsh% python cluster.py --file examples/example.tsv --type meanshift --sof feature
```

5.5 DBSCAN clustering (dbscan)

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density (Ester, et al., 1996). Due to this rather generic view, clusters found by DBSCAN can have any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component of the DBSCAN algorithm is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples). The algorithm has two parameters, min_samples and eps, which define formally what we mean when we say 'dense'. Higher min_samples or lower eps indicate a higher density necessary to form a cluster.

Use the following command to perform the DBSCAN clustering:

```
tcsh% python cluster.py --file examples/example.tsv --type dbscan --sof feature
```

The feature selection algorithms can be run by the following command:

```
tcsh% python feaSelector.py --file descriptor.tsv --type <feature_selection_algorithm> --label sample_label_file
```

'**--**label' specify the sample class of the samples in 'descriptor.tsv'.

5.6 Chi-Square feature selection (CHI2)

In statistics, the χ^2 test is applied to test the independence of two events. χ^2 is a measure of how much expected counts *E* and observed counts *N* deviate from each other (Chen, et al., 2009). A high value of χ^2 indicates that the hypothesis of independence, which implies that expected and observed counts are similar, is incorrect. This score can be used to select the *n* features with the highest values for the test chi-square statistic from *x*, which must contain only non-negative features such as Booleans or frequencies, relative to the classes.

$$x^2 = \sum \frac{(A-E)^2}{E}$$

where *A* is the observed value and *E* is the expected value.

Use the following command to perform the Chi-Square feature selection:

```
tcsh% python feaSelector.py --file examples/example.tsv --type CHI2 --label examples/label.txt --out CHI2_feature.txt
```

5.7 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 (Chen, et al., 2009; Chen, et al., 2007b). For any variable *X* from the features, its information entropy is defined as:

$$I(X) = -\sum_i P(x_i)\log_2(P(x_i))$$

where x_i represents a set of values of *X*, and $P(x_i)$ denotes the prior probability of x_i. The conditional entropy of *X* under the condition of *Y* is defined as:

$$I(X \mid Y) = -\sum_j P(y_j)\sum_i P(x_i \mid y_i)\log_2(P(x_i \mid y_j))$$

where $P(x_i|y_j)$ is the posterior probability of x_i given the value y_j of *Y*. Then, information gain IG(X|Y) is given by:

$$IG(X \mid Y) = I(X) - I(Y)$$

Use the following command to perform the Information Gain feature selection:

```
tcsh% python feaSelector.py --file examples/example.tsv --type IG
--label examples/label.txt --out IG_feature.txt
```

5.8 Mutual Information feature selection (MI)

Given two random variables *X* and *Y*, their mutual information (Peng, et al., 2005) can be defined based on their probabilities *P(X), P(Y)* and *P(X, Y)*:

$$I(X,Y) = \sum_{x\in X}\sum_{y\in Y} P(x,y)\log\frac{P(x,y)}{P(x)p(y)}$$

Mutual Information feature selection can be executed using the following command:

```
tcsh% python feaSelector.py --file examples/example.tsv --type MIC
--label examples/label.txt --out MIC_feature.txt
```

5.9 Pearson Correlation coefficient feature selection (pearsonr)

The Pearson correlation coefficient (M. Stigler, 1989) *r* for a pair of variables (*X*, *Y*) is calculated as follows:

$$r = \frac{\sum_i (x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_i (x_i - \bar{x}_i)^2 \sum_i (y_i - \bar{y}_i)^2}}$$

where $\bar{x}i$ is the mean of *X*, and $\bar{y}_i$ the mean of *Y*. The value of *r* is bounded within the [-1, 1] interval. A higher absolute value of *r* corresponds to a higher correlation between *X* and *Y*.

Use the following command to perform the Pearson Correlation feature selection:

```
tcsh% python feaSelector.py --file examples/example.tsv --type
pearsonr --label examples/label.txt --out pearsonr_feature.txt
```

After running the feature selection algorithm (as discussed in sections 5.6 - 5.9), the feature descriptors will be ranked according to their importance. The higher the ranking is, the more important the feature descriptor is.

In addition, three dimensionality reduction algorithms (PCA, LDA, and t-SNE) have been implemented in the *iFeature* package and can be run using the following commands (explained in sections 5.10 - 5.12 below). To facilitate users' understanding and interpretability of results, the dimensionality reduction results can be visualized in the form of a scatter diagram.

5.10 Principal Component Analysis (PCA)

PCA (Pearson, 1901) 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 analysis:

```
tcsh% python scripts/pcaAnalysis.py --file examples/example.tsv --
ncomponents 3 --out pcaResult.txt
```

'--ncomponents' specifies the number of principal components, '--out' specifies the name of the output file of PCA.

5.11 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (Blei, et al., 2003) is a generative probabilistic model for collections of discrete dataset such as text corpora.

Use the following command to perform the LDA analysis:

```
tcsh% python scripts/ldaAnalysis.py --file examples/example.tsv --
ncomponents 3 --label examples/label.txt  --out ldaResult.txt
```

'--label' is the label file, which divides the protein sequences in the training file into different classes (positive samples and negative samples).

5.12 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten, 2014) is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets.

Use the following command to perform the t-SNE analysis:

```
tcsh% python scripts/tsneAnalysis.py --file examples/example.tsv -
-label examples/label.txt  --out tSNEResult.txt
```

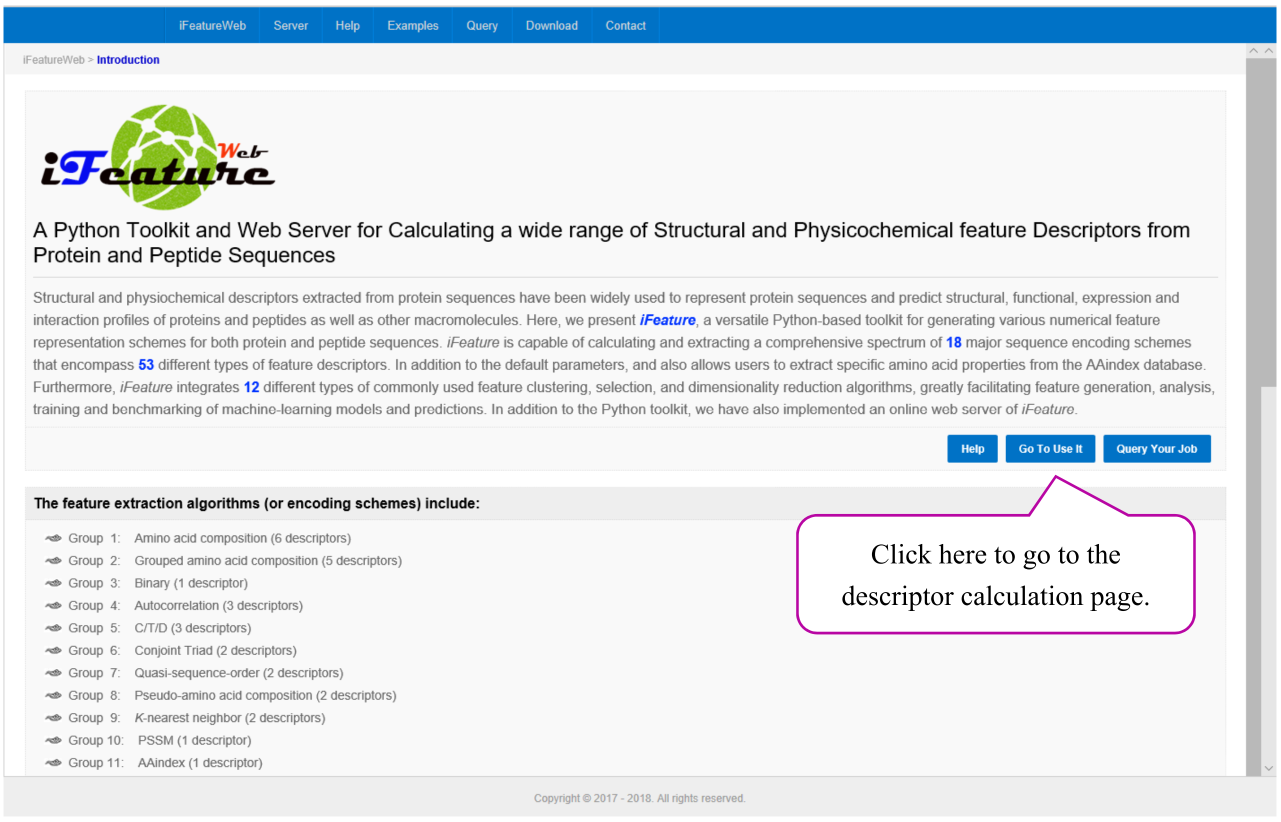
'--label' is the label file, which divides the protein sequences in the training file into different classes (positive samples and negative samples).

6. Online Web Server

Moreover, for users that are not familiar with computer programming using Python we also implemented an online web server of *iFeature*, which is publicly available at http://ifeature.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 *iFeature* web server (called *iFeatureWeb*) is a user-friendly online platform for computing the protein/peptide descriptors presented in the *iFeature* package. This is a short tutorial for using *iFeatureWeb*.

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



A Python Toolkit and Web Server for Calculating a wide range of Structural and Physicochemical feature Descriptors from Protein and Peptide Sequences

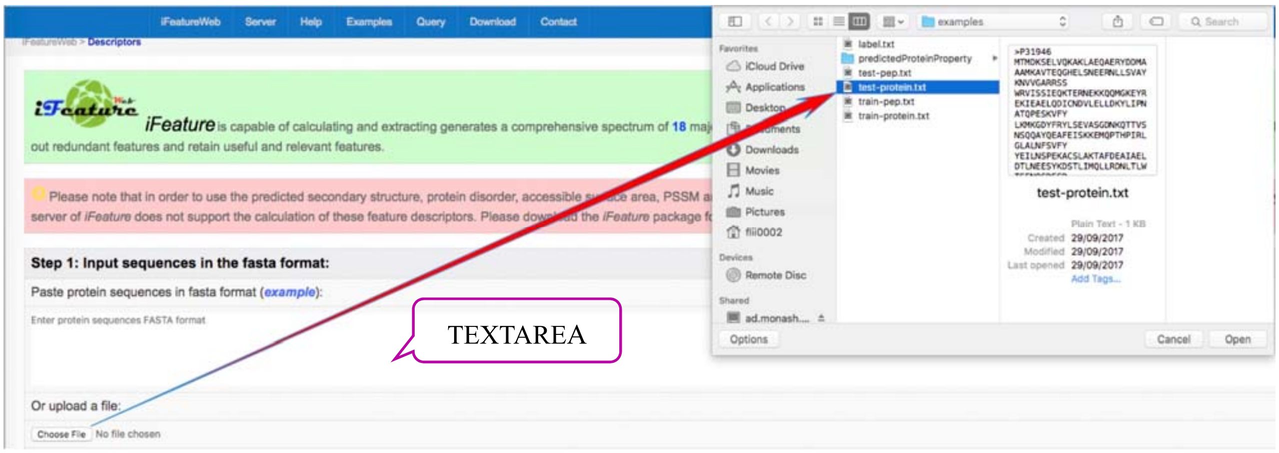
Structural and physiochemical descriptors extracted from protein sequences have been widely used to represent protein sequences and predict structural, functional, expression and interaction profiles of proteins and peptides as well as other macromolecules. Here, we present ***iFeature***, a versatile Python-based toolkit for generating various numerical feature representation schemes for both protein and peptide sequences. *iFeature* is capable of calculating and extracting a comprehensive spectrum of **18** major sequence encoding schemes that encompass **53** different types of feature descriptors. In addition to the default parameters, and also allows users to extract specific amino acid properties from the AAindex database. Furthermore, *iFeature* integrates **12** different types of commonly used feature clustering, selection, and dimensionality reduction algorithms, greatly facilitating feature generation, analysis, training and benchmarking of machine-learning models and predictions. In addition to the Python toolkit, we have also implemented an online web server of *iFeature*.

The feature extraction algorithms (or encoding schemes) include:

- Group 1: Amino acid composition (6 descriptors)
- Group 2: Grouped amino acid composition (5 descriptors)
- Group 3: Binary (1 descriptor)
- Group 4: Autocorrelation (3 descriptors)
- Group 5: C/T/D (3 descriptors)
- Group 6: Conjoint Triad (2 descriptors)
- Group 7: Quasi-sequence-order (2 descriptors)
- Group 8: Pseudo-amino acid composition (2 descriptors)
- Group 9: *K*-nearest neighbor (2 descriptors)
- Group 10: PSSM (1 descriptor)
- Group 11: AAindex (1 descriptor)

Copyright © 2017 - 2018. All rights reserved.

Step 1. Input your fasta sequences in the designated text area or upload a file that includes the sequences in fasta format.



iFeatureWeb > Descriptors

iFeature is capable of calculating and extracting generates a comprehensive spectrum of 18 maj
out redundant features and retain useful and relevant features.

Please note that in order to use the predicted secondary structure, protein disorder, accessible surface area, PSSM a
server of *iFeature* does not support the calculation of these feature descriptors. Please download the *iFeature* package f

Step 1: Input sequences in the fasta format:

Paste protein sequences in fasta format (*example*):

Enter protein sequences FASTA format

Or upload a file:

No file chosen

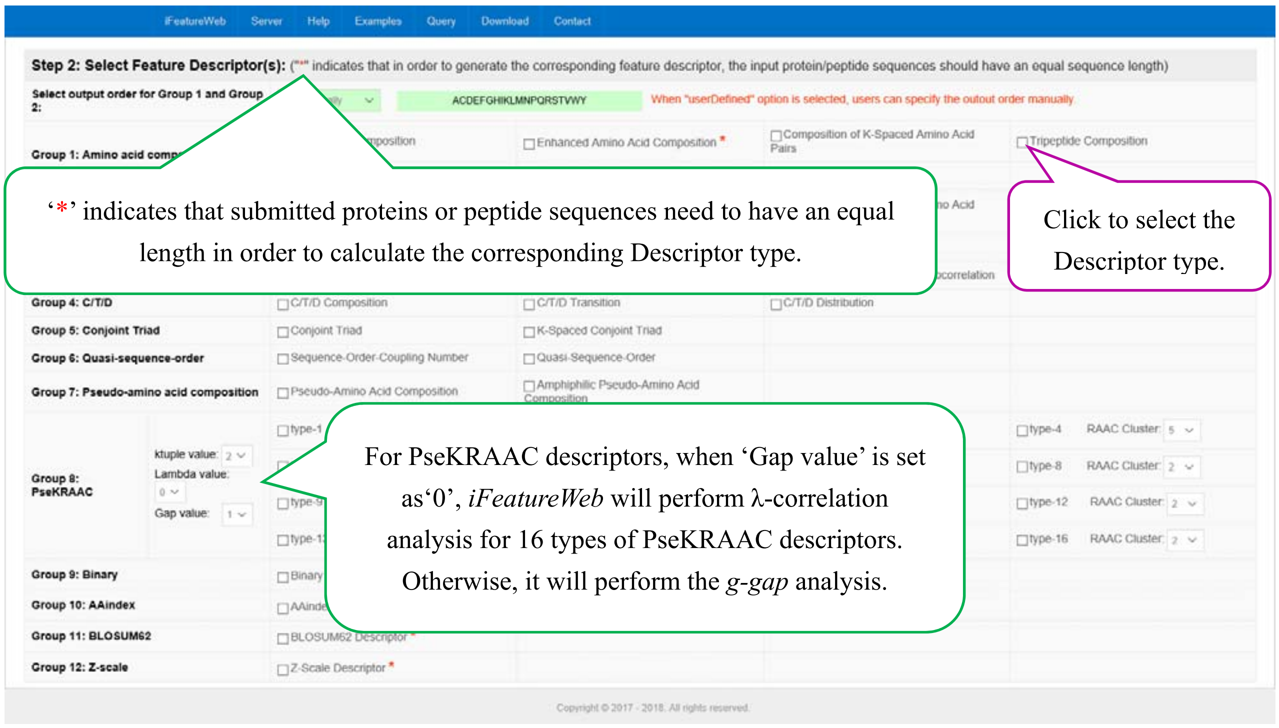
test-protein.txt

Plain Text - 1 KB

Created 29/09/2017
Modified 29/09/2017
Last opened 29/09/2017

Note: Paste your protein (or peptide) sequences in the 'TEXTAREA' or upload a file that includes the sequences. The protein sequences must be in 'FASTA' format. *iFeatureWeb* was designed to accept at most 100 sequences at once.

Step 2. Select descriptor(s).



Step 2: Select Feature Descriptor(s): ("*" indicates that in order to generate the corresponding feature descriptor, the input protein/peptide sequences should have an equal sequence length)

Select output order for Group 1 and Group 2: ACDEFGHIKLMNPQRSTVWY When "userDefined" option is selected, users can specify the outout order manually

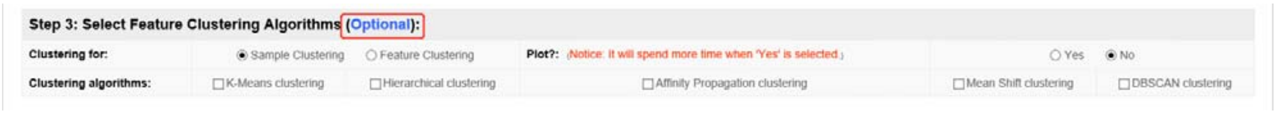
Group 1: Amino acid comp	☐ ...mposition	☐ Enhanced Amino Acid Composition *	☐ Composition of K-Spaced Amino Acid Pairs	☐ Tripeptide Composition
Group 4: C/T/D	☐ C/T/D Composition	☐ C/T/D Transition	☐ C/T/D Distribution	
Group 5: Conjoint Triad	☐ Conjoint Triad	☐ K-Spaced Conjoint Triad		
Group 6: Quasi-sequence-order	☐ Sequence-Order-Coupling Number	☐ Quasi-Sequence-Order		
Group 7: Pseudo-amino acid composition	☐ Pseudo-Amino Acid Composition	☐ Amphiphilic Pseudo-Amino Acid Composition		
Group 8: PseKRAAC (ktuple value: 2; Lambda value: 0; Gap value: 1)	☐ type-1			☐ type-4 RAAC Cluster: 5
				☐ type-8 RAAC Cluster: 2
	☐ type-9			☐ type-12 RAAC Cluster: 2
	☐ type-13			☐ type-16 RAAC Cluster: 2
Group 9: Binary	☐ Binary			
Group 10: AAindex	☐ AAindex			
Group 11: BLOSUM62	☐ BLOSUM62 Descriptor *			
Group 12: Z-scale	☐ Z-Scale Descriptor *			

Copyright © 2017 - 2018. All rights reserved.

iFeatureWeb
Server
Help
Examples
Query
Download
Contact
'*' indicates that submitted proteins or peptide sequences need to have an equal length in order to calculate the corresponding Descriptor type.
Click to select the Descriptor type.
For PseKRAAC descriptors, when 'Gap value' is set as'0', iFeatureWeb will perform λ-correlation analysis for 16 types of PseKRAAC descriptors. Otherwise, it will perform the g-gap analysis.

One or more descriptor types should be chosen. Then you can click the 'Submit' button at the bottom of the page to calculate the selected descriptor(s) or go on to select the clustering algorithms for sample clustering or feature clustering.

Step 3. Select the clustering algorithms.


Step 3: Select Feature Clustering Algorithms (Optional):

Clustering for: [✓ Sample Clustering] Feature Clustering

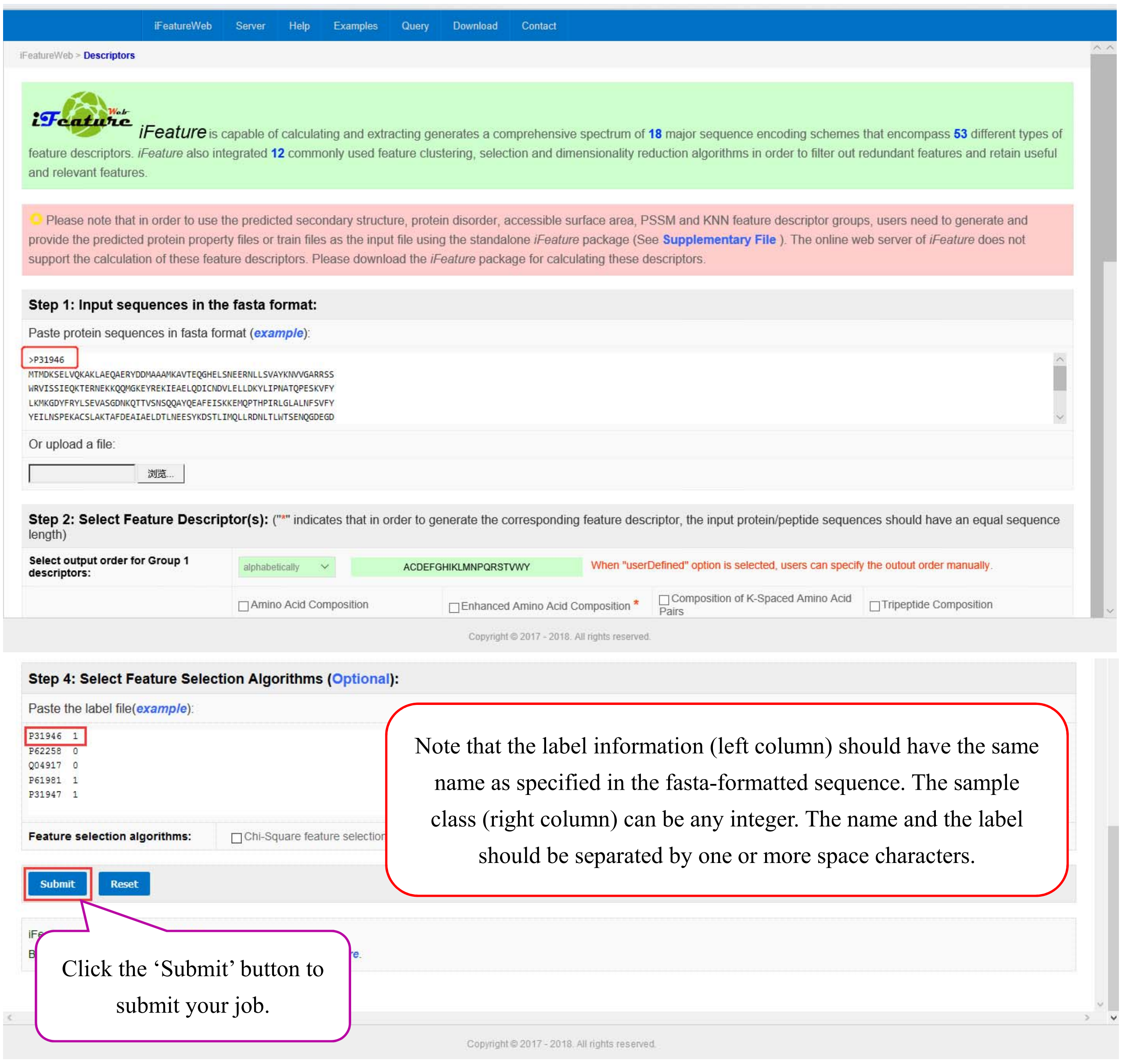
Plot?: (Notice: It will spend more time when 'Yes' is selected.) Yes [✓ No]

Clustering algorithms: ☐ K-Means clustering ☐ Hierarchical clustering ☐ Affinity Propagation clustering ☐ Mean Shift clustering ☐ DBSCAN clustering

In this step, *iFeatureWeb* allows users to select clustering algorithms. Users can select clustering for 'sample' or for 'feature'. Five commonly used clustering algorithms are supported by *iFeatureWeb*. If you do not need to perform a clustering analysis, just skip this step.

Step 4. Select the feature selection algorithms.

In this step, with additional 'sample label' information, *iFeatureWeb* can identify the most characterizing features according to the feature selection algorithms. Four commonly used feature selection algorithms are supported by *iFeatureWeb*. Like Step 3, this step is also optional.



iFeature is capable of calculating and extracting generates a comprehensive spectrum of **18** major sequence encoding schemes that encompass **53** different types of feature descriptors. *iFeature* also integrated **12** commonly used feature clustering, selection and dimensionality reduction algorithms in order to filter out redundant features and retain useful and relevant features.

Please note that in order to use the predicted secondary structure, protein disorder, accessible surface area, PSSM and KNN feature descriptor groups, users need to generate and provide the predicted protein property files or train files as the input file using the standalone *iFeature* package (See **Supplementary File**). The online web server of *iFeature* does not support the calculation of these feature descriptors. Please download the *iFeature* package for calculating these descriptors.

Step 1: Input sequences in the fasta format:

Paste protein sequences in fasta format (*example*):

```
>P31946
MTMDKSELVQKAKLAEQAERYDDMAAAMKAVTEQGHELSNEERNLLSVAYKNVVGARRSS
WRVISSIEQKTERNEKKQQMGKEYREKIEAELQDICNDVLELLDKYLIPNATQPESKVFY
LKMKGDYFRYLSEVASGDNKQTTVSNSQQAYQEAFEISKKEMQPTHPIRLGLALNFSVFY
YEILNSPEKACSLAKTAFDEAIAELDTLNEESYKDSTLIMQLLRDNLTLWTSENQGDEGD
```

Or upload a file:

Step 2: Select Feature Descriptor(s): ("*" indicates that in order to generate the corresponding feature descriptor, the input protein/peptide sequences should have an equal sequence length)

Select output order for Group 1 descriptors: alphabetically ACDEFGHIKLMNPQRSTVWY When "userDefined" option is selected, users can specify the outout order manually.

☐ Amino Acid Composition ☐ Enhanced Amino Acid Composition * ☐ Composition of K-Spaced Amino Acid Pairs ☐ Tripeptide Composition

Copyright © 2017 - 2018. All rights reserved.

Step 4: Select Feature Selection Algorithms (Optional):

Paste the label file(*example*):

```
P31946  1
P62258  0
Q04917  0
P61981  1
P31947  1
```

Feature selection algorithms: ☐ Chi-Square feature selection

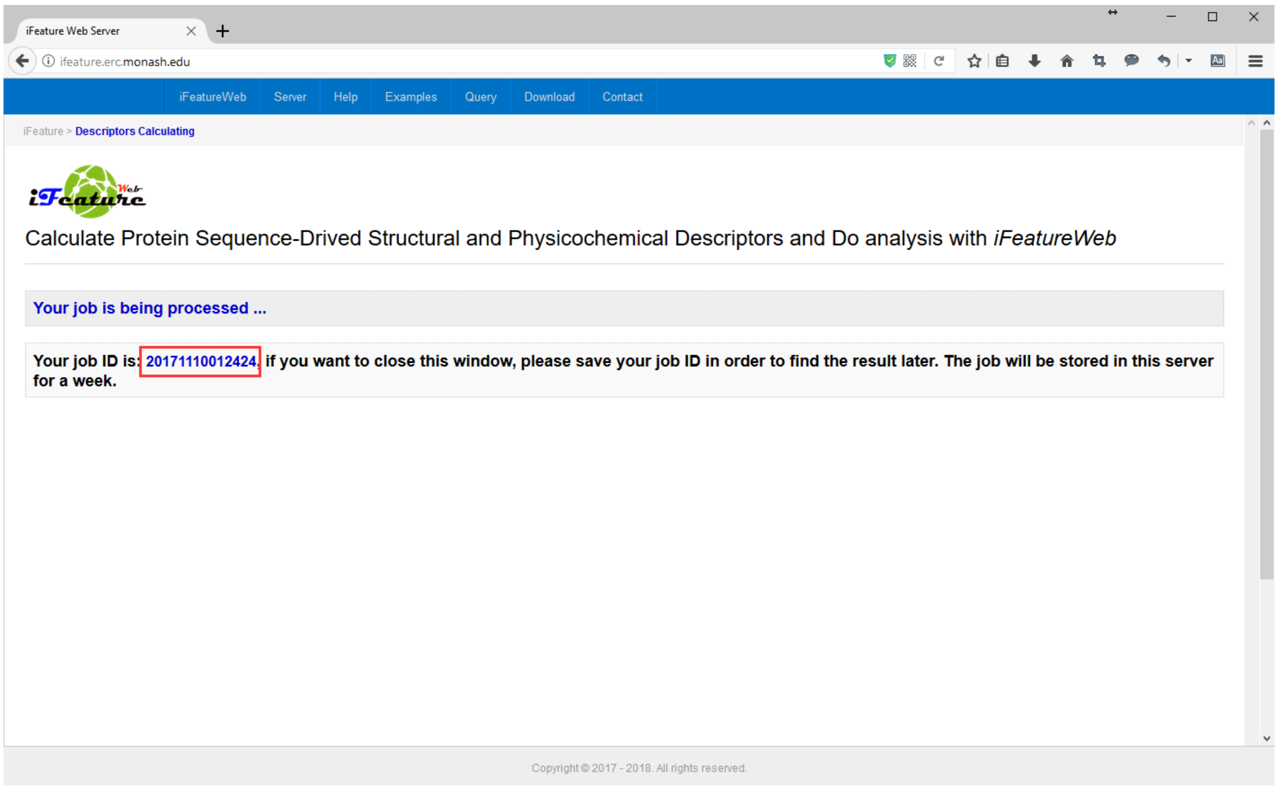
Note that the label information (left column) should have the same name as specified in the fasta-formatted sequence. The sample class (right column) can be any integer. The name and the label should be separated by one or more space characters.

Click the 'Submit' button to submit your job.

Copyright © 2017 - 2018. All rights reserved.

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

Step 5. Waiting for your result.

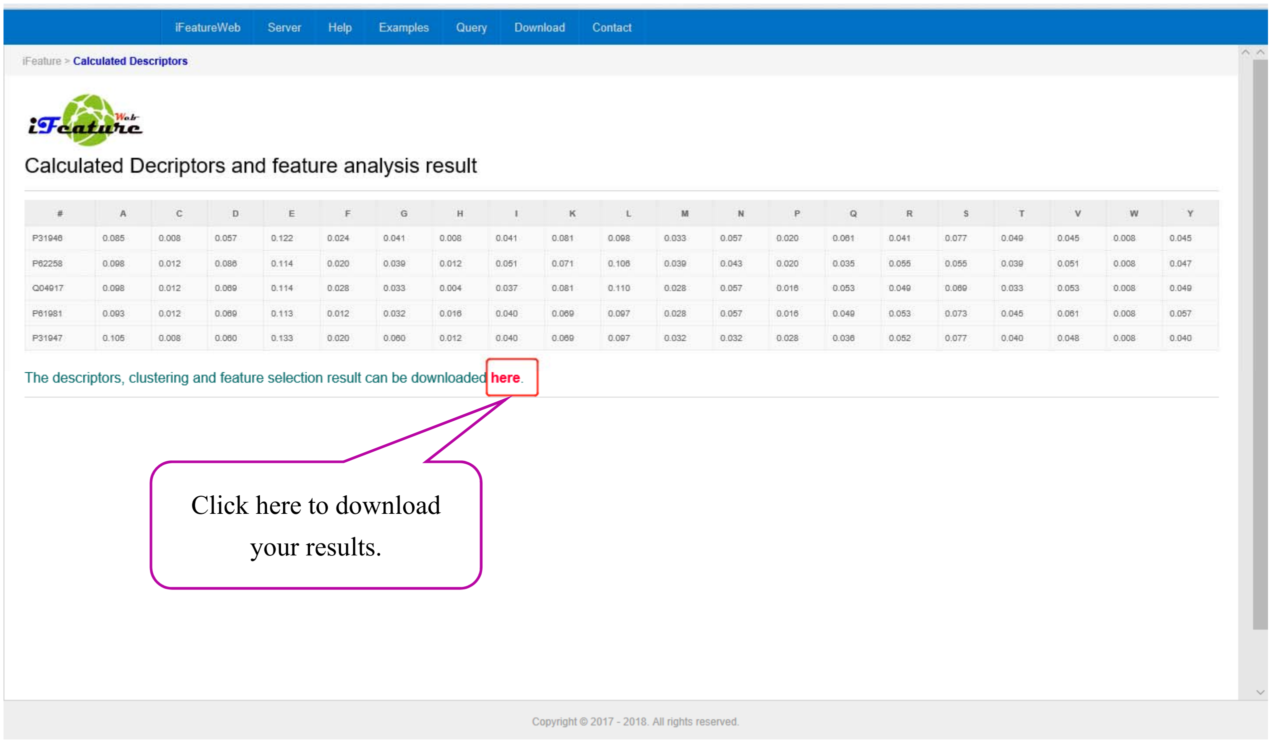


Calculate Protein Sequence-Drived Structural and Physicochemical Descriptors and Do analysis with *iFeatureWeb*

Your job is being processed ...

Your job ID is: 20171110012424, if you want to close this window, please save your job ID in order to find the result later. The job will be stored in this server for a week.

Copyright © 2017 - 2018. All rights reserved.



iFeatureWeb
Server
Help
Examples
Query
Download
Contact
iFeature > Calculated Descriptors
Click here to download your results.

Calculated Decriptors and feature analysis result

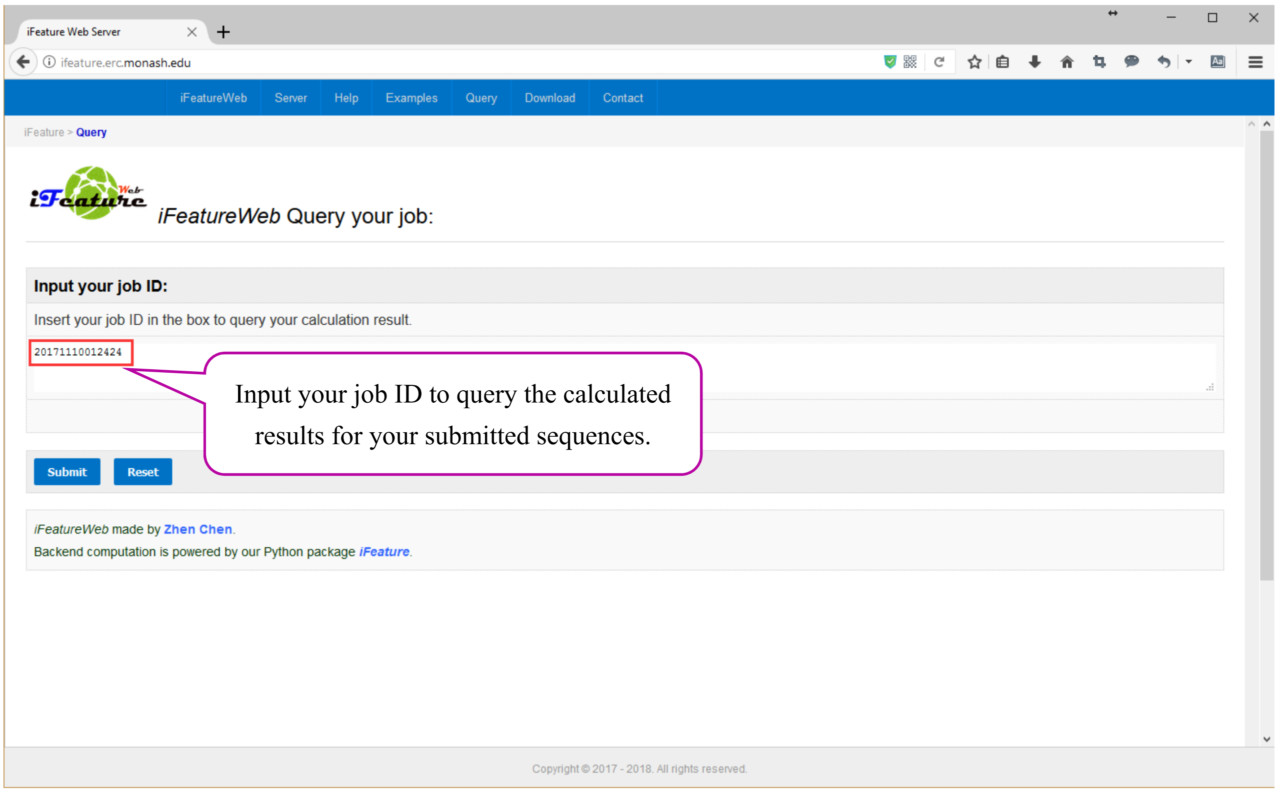
#	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
P31946	0.085	0.008	0.057	0.122	0.024	0.041	0.008	0.041	0.081	0.098	0.033	0.057	0.020	0.061	0.041	0.077	0.049	0.045	0.008	0.045
P62258	0.098	0.012	0.086	0.114	0.020	0.039	0.012	0.051	0.071	0.106	0.039	0.043	0.020	0.035	0.055	0.055	0.039	0.051	0.008	0.047
Q04917	0.098	0.012	0.069	0.114	0.028	0.033	0.004	0.037	0.081	0.110	0.028	0.057	0.016	0.053	0.049	0.069	0.033	0.053	0.008	0.049
P61981	0.093	0.012	0.069	0.113	0.012	0.032	0.016	0.040	0.069	0.097	0.028	0.057	0.016	0.049	0.053	0.073	0.045	0.061	0.008	0.057
P31947	0.105	0.008	0.060	0.133	0.020	0.060	0.012	0.040	0.069	0.097	0.032	0.032	0.028	0.036	0.052	0.077	0.040	0.048	0.008	0.040

The descriptors, clustering and feature selection result can be downloaded **here**.

Copyright © 2017 - 2018. All rights reserved.

After a few seconds, the calculated descriptors should appear as a table, where rows represent protein sequences and columns represent descriptors. The descriptors, clustering and feature selection results are available for download.

For each job, *iFeatureWeb* 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.


iFeature Web Server
ifeature.erc.monash.edu
iFeatureWeb
Server
Help
Examples
Query
Download
Contact
iFeature > Query
iFeatureWeb Query your job:
Input your job ID:
Insert your job ID in the box to query your calculation result.
20171110012424
Input your job ID to query the calculated results for your submitted sequences.
Submit
Reset
iFeatureWeb made by Zhen Chen.
Backend computation is powered by our Python package iFeature.
Copyright © 2017 - 2018. All rights reserved.

7. Summary

In summary, *iFeature* 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 and selection analysis. We will integrate more analysis and clustering algorithms to enable interactive analysis and machine learning-based modeling in future work. It is anticipated that *iFeature* will be widely used as a powerful tool in bioinformatics, computational biology, systems biology and proteome research. The functionality of iFeature is made freely available via an online web server (http://iFeature.erc.monash.edu/) and stand-alone toolkit (https://github.com/Superzchen/iFeature/).

8. Acknowledgments

This work was financially supported by grants from the Australian Research Council (ARC) (LP110200333 and DP120104460), the National Natural Science Foundation of China (Grant No. 31701142), the National Health of Medical Research Council of Australia (NHMRC), the National Institute of Allergy and Infectious Diseases of the National Institutes of Health (R01 AI111965), and a Major Inter-Disciplinary Research (IDR) Grant awarded by Monash University. AL and TML were supported by Informatics startup packages through the UAB School of Medicine.

9. References

Altschul, S.F., *et al.* (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res*, 25, 3389-3402.

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.

Blei, D.M., Ng, A.Y. and Jordan, M.I. (2003) Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3, 993-1022.

Cai, C.Z., *et al.* (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.

Cai, C.Z., *et al.* (2004) Enzyme family classification by support vector machines. *Proteins*, 55, 66-76.

Cai, Y., *et al.* (2012) Prediction of lysine ubiquitination with mRMR feature selection and analysis. *Amino Acids*, 42, 1387-1395.

Cannata, N., *et al.* (2002) Simplifying amino acid alphabets by means of a branch and bound algorithm and substitution matrices. *Bioinformatics*, 18, 1102-1108.

Chen, K., *et al.* (2009) Prediction of integral membrane protein type by collocated hydrophobic amino acid pairs. *J Comput Chem*, 30, 163-172.

Chen, K., Kurgan, L. and Rahbari, M. (2007a) Prediction of protein crystallization using collocation of amino acid pairs. *Biochem Biophys Res Commun*, 355, 764-769.

Chen, K., Kurgan, L.A. and Ruan, J. (2007b) Prediction of flexible/rigid regions from protein sequences using k-spaced amino acid pairs. *BMC Struct Biol*, 7, 25.

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.

Chen, X., *et al.* (2013a) Incorporating key position and amino acid residue features to identify general and species-specific Ubiquitin conjugation sites. *Bioinformatics*, 29, 1614-1622.

Chen, Y.Z., *et al.* (2012) SUMOhydro: a novel method for the prediction of sumoylation sites based on hydrophobic properties. *PLoS One*, 7, e39195.

Chen, Z., *et al.* (2011) Prediction of ubiquitination sites by using the composition of k-spaced amino acid pairs. *PLoS One*, 6, e22930.

Chen, Z., *et al.* (2013b) hCKSAAP_UbSite: improved prediction of human ubiquitination sites by exploiting amino acid pattern and properties. *Biochim Biophys Acta*, 1834, 1461-1467.

Chen, Z., *et al.* (2015) Towards more accurate prediction of ubiquitination sites: a comprehensive review of current methods, tools and features. *Brief Bioinform*, 16, 640-657.

Cheng, Y.Z. (1995) Mean Shift, Mode Seeking, and Clustering. *Ieee T Pattern Anal*, 17, 790-799.

Chou, K.C. (2000) Prediction of protein subcellular locations by incorporating quasi-sequence-order effect. *Biochem Biophys Res Commun*, 278, 477-483.

Chou, K.C. (2001) Prediction of protein cellular attributes using pseudo-amino acid composition. *Proteins*, 43, 246-255.

Chou, K.C. (2005) Using amphiphilic pseudo amino acid composition to predict enzyme subfamily classes. *Bioinformatics*, 21, 10-19.

Dubchak, I., *et al.* (1995) Prediction of protein folding class using global description of amino acid sequence. *Proc Natl Acad Sci U S A*, 92, 8700-8704.

Dubchak, I., *et al.* (1999) Recognition of a protein fold in the context of the Structural Classification of Proteins (SCOP) classification. *Proteins*, 35, 401-407.

Ester, M., *et al.* A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. Portland, Oregon: AAAI Press; 1996. p. 226-231.

Faraggi, E., *et al.* (2009) Predicting continuous local structure and the effect of its substitution for secondary structure in fragment-free protein structure prediction. *Structure*, 17, 1515-1527.

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.

Frey, B.J. and Dueck, D. (2007) Clustering by passing messages between data points. *Science*, 315, 972-976.

Grantham, R. (1974) Amino acid difference formula to help explain protein evolution. *Science*, 185, 862-864.

Han, L.Y., *et al.* (2004) Prediction of RNA-binding proteins from primary sequence by a support vector machine approach. *RNA*, 10, 355-368.

Heffernan, R., *et al.* (2016) Highly accurate sequence-based prediction of half-sphere exposures of amino acid residues in proteins. *Bioinformatics*, 32, 843-849.

Heffernan, R., *et al.* (2015) Improving prediction of secondary structure, local backbone angles, and solvent accessible surface area of proteins by iterative deep learning. *Sci Rep*, 5, 11476.

Horne, D.S. (1988) Prediction of protein helix content from an autocorrelation analysis of sequence hydrophobicities. *Biopolymers*, 27, 451-477.

Jain, A.K. (2010) Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31, 651-666.

Jain, A.K., Murty, M.N. and Flynn, P.J. (1999) Data clustering: A review. *Acm Comput Surv*, 31, 264-323.

Jones, D.T. (1999) Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol*, 292, 195-202.

Kawashima, S., *et al.* (2008) AAindex: amino acid index database, progress report 2008. *Nucleic Acids Res*, 36, D202-205.

Kosiol, C., Goldman, N. and Buttimore, N.H. (2004) A new criterion and method for amino acid classification. *J Theor Biol*, 228, 97-106.

Lee, T.Y., *et al.* (2011a) Incorporating distant sequence features and radial basis function networks to identify ubiquitin conjugation sites. *PLoS One*, 6, e17331.

Lee, T.Y., *et al.* (2011b) Exploiting maximal dependence decomposition to identify conserved motifs from a group of aligned signal sequences. *Bioinformatics*, 27, 1780-1787.

Li, J. and Wang, W. (2007) Grouping of amino acids and recognition of protein structurally conserved regions by reduced alphabets of amino acids. *Sci China C Life Sci*, 50, 392-402.

Li, T., *et al.* (2003) Reduction of protein sequence complexity by residue grouping. *Protein Eng*, 16, 323-330.

Liang, Y., Liu, S. and Zhang, S. (2015) Prediction of Protein Structural Classes for Low-Similarity Sequences Based on Consensus Sequence and Segmented PSSM. *Comput Math Methods Med*, 2015, 370756.

Lin, Z. and Pan, X.M. (2001) Accurate prediction of protein secondary structural content. *J Protein Chem*, 20, 217-220.

Liu, B., *et al.* (2015a) Identification of microRNA precursor with the degenerate K-tuple or Kmer strategy. *J Theor Biol*, 385, 153-159.

Liu, B., *et al.* (2015b) 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.

Liu, B., *et al.* (2015c) 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.

Liu, X., *et al.* (2002) Simplified amino acid alphabets based on deviation of conditional probability from random background. *Phys Rev E Stat Nonlin Soft Matter Phys*, 66, 021906.

M. Stigler, S. Francis Galton's Account of the Invention of Correlation. 1989.

Maaten, L.V.D. (2014) Accelerating t-SNE using tree-based algorithms. *J. Mach. Learn. Res.*, 15, 3221-3245.

Melo, F. and Marti-Renom, M.A. (2006) Accuracy of sequence alignment and fold assessment using reduced amino acid alphabets. *Proteins*, 63, 986-995.

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.

Obradovic, Z.*, et al.* (2005) Exploiting heterogeneous sequence properties improves prediction of protein disorder. *Proteins*, 61 Suppl 7, 176-182.

Ogul, H. and Mumcuoglu, E.U. (2007) A discriminative method for remote homology detection based on n-peptide compositions with reduced amino acid alphabets. *Biosystems*, 87, 75-81.

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.

Peng, H.C., Long, F.H. and Ding, C. (2005) Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *Ieee T Pattern Anal*, 27, 1226-1238.

Peng, K.*, et al.* (2006) Length-dependent prediction of protein intrinsic disorder. *BMC Bioinformatics*, 7, 208.

Peterson, E.L.*, et al.* (2009) Reduced amino acid alphabets exhibit an improved sensitivity and selectivity in fold assignment. *Bioinformatics*, 25, 1356-1362.

Radivojac, P.*, et al.* (2010) Identification, analysis, and prediction of protein ubiquitination sites. *Proteins*, 78, 365-380.

Rakshit, S. and Ananthasuresh, G.K. (2008) An amino acid map of inter-residue contact energies using metric multi-dimensional scaling. *J Theor Biol*, 250, 291-297.

Rokach, L. and Maimon, O. Clustering Methods. In: Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US; 2005. p. 321-352.

Sandberg, M.*, et al.* (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.

Saravanan, V. and Gautham, N. (2015) Harnessing Computational Biology for Exact Linear B-Cell Epitope Prediction: A Novel Amino Acid Composition-Based Feature Descriptor. *OMICS*, 19, 648-658.

Schneider, G. and Wrede, P. (1994) The rational design of amino acid sequences by artificial neural networks and simulated molecular evolution: de novo design of an idealized leader peptidase cleavage site. *Biophys J*, 66, 335-344.

Shen, J.*, et al.* (2007) Predicting protein-protein interactions based only on sequences information. *Proc Natl Acad Sci U S A*, 104, 4337-4341.

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.

Susko, E. and Roger, A.J. (2007) On reduced amino acid alphabets for phylogenetic inference. *Mol Biol Evol*, 24, 2139-2150.

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.

Tung, C.W. and Ho, S.Y. (2008) Computational identification of ubiquitylation sites from protein sequences. *BMC Bioinformatics*, 9, 310.

Wang, J. and Wang, W. (1999) A computational approach to simplifying the protein folding alphabet. *Nat Struct Biol*, 6, 1033-1038.

Wang, R., Xu, Y. and Liu, B. (2016) Recombination spot identification Based on gapped k-mers. *Sci Rep*, 6, 23934.

Xiao, N., *et al.* (2015) protr/ProtrWeb: R package and web server for generating various numerical representation schemes of protein sequences. *Bioinformatics*, 31, 1857-1859.

Zuo, Y., *et al.* (2017) PseKRAAC: a flexible web server for generating pseudo K-tuple reduced amino acids composition. *Bioinformatics*, 33, 122-124.

Zuo, Y.C. and Li, Q.Z. (2010) Using K-minimum increment of diversity to predict secretory proteins of malaria parasite based on groupings of amino acids. *Amino Acids*, 38, 859-867.