
PyBioMed Documentation

Release 1

CBDD Group



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

1	Overview	3
1.1	Who uses PyBioMed?	3
1.2	Goals	3
1.3	Feature overview	4
1.4	The Python programming language	5
2	Getting Started with PyBioMed	7
2.1	Installing the PyBioMed package	7
2.2	Getting molecules	7
2.2.1	Getting molecular structure	8
2.2.2	Reading molecules	8
2.2.3	Getting protein sequence	8
2.2.4	Reading protein sequence	9
2.2.5	Getting DNA sequence	9
2.2.6	Reading DNA sequence	9
2.3	Pretreating structure	9
2.3.1	Pretreating molecules	10
2.3.2	Pretreating protein sequence	10
2.3.3	Pretreating DNA sequence	11
2.4	Calculating molecular descriptors	11
2.4.1	Calculating descriptors	11
2.4.2	Calculating molecular fingerprints	13
2.5	Calculating protein descriptors	14
2.5.1	Calculating protein descriptors via functions	14
2.5.2	Calculate protein descriptors via object	15
2.6	Calculating DNA descriptors	16
2.6.1	Calculating DNA descriptors via functions	16
2.6.2	Calculating DNA descriptors via object	16
2.7	Calculating Interaction descriptors	17
3	Application	21
3.1	Application 1 Prediction of Caco-2 Cell Permeability	21
3.2	Application 2 Prediction of aqueous solubility	23
3.3	Application 3 Prediction of drug–target interaction from the integration of chemical and protein spaces	25
3.4	Application 4 Prediction of protein subcellular location	28
3.5	Application 5 Predicting nucleosome positioning in genomes with dinucleotide-based auto covariance	30
4	PyBioMed API	33
4.1	PyDNA	33
4.1.1	PyDNAac module	33
4.1.2	PyDNAacutil module	34

4.1.3	PyDNAnac module	35
4.1.4	PyDNAnacutil module	35
4.1.5	PyDNApsenac module	36
4.1.6	PyDNApsenacutil module	39
4.1.7	PyDNAutil module	39
4.2	PyMolecule	41
4.2.1	AtomProperty module	41
4.2.2	AtomTypes module	41
4.2.3	basak module	42
4.2.4	bcut module	43
4.2.5	cats2d module	44
4.2.6	charge module	45
4.2.7	connectivity module	50
4.2.8	constitution module	59
4.2.9	estate module	65
4.2.10	fingerprint module	66
4.2.11	geary module	70
4.2.12	ghosecrippen module	71
4.2.13	kappa module	72
4.2.14	moe module	74
4.2.15	molproperty module	76
4.2.16	moran module	78
4.2.17	moreaubroto module	79
4.2.18	topology module	80
4.2.19	Scaffolds module	86
4.3	PyProtein	87
4.3.1	PyProtein module	87
4.3.2	AAComposition module	91
4.3.3	AAIndex module	92
4.3.4	Autocorrelation module	93
4.3.5	CTD module	100
4.3.6	ConjointTriad module	105
4.3.7	GetProteinFromUniprot module	106
4.3.8	GetSubSeq module	107
4.3.9	ProCheck module	107
4.3.10	PseudoAAC module	108
4.3.11	PyProteinAACComposition module	110
4.3.12	PyProteinAAIndex module	111
4.3.13	QuasiSequenceOrder module	112
4.4	PyPretreat	116
4.4.1	PyPretreatDNA module	116
4.4.2	PyPretreatMol module	117
4.4.3	PyPretreatMolutil module	120
4.4.4	PyPretreatPro module	121
4.5	PyInteraction	121
4.5.1	PyInteraction module	121
4.6	PyGetMol	122
4.6.1	GetProtein module	122
4.6.2	GetDNA module	123
4.6.3	Getmol module	124
4.7	test package	125
4.7.1	test module	125
4.7.2	test_PyBioMed module	125
4.7.3	test_PyDNA module	125

4.7.4	test_PyInteration module	126
4.7.5	test_PyMolecule module	126
4.7.6	test_PyPretreat module	126
4.7.7	test_PyProtein module	126
4.7.8	test_PyGetMol module	126
5	Testing	127
5.1	Requirements for testing	127
5.2	Testing an installed package	127
6	Download	129
6.1	Python Package	129
6.2	Documentation	129
6.3	The introduction of descriptors	129
6.3.1	Molecular descriptors introduction	129
6.3.2	Protein descriptors introduction	129
6.3.3	DNA descriptors introduction	129
6.3.4	Interaction descriptors introduction	129

The python package PyBioMed is designed by [CBDD Group](#) (Computational Biology & Drug Design Group), Xiangya School of Pharmaceutical Sciences, Central South University. To develop a powerful model for prediction tasks by machine learning algorithms such as scikit-learn, one of the most important things to consider is how to effectively represent the molecules under investigation such as small molecules, proteins, DNA and even complex interactions, by a descriptor. PyBioMed is a feature-rich package used for the characterization of various complex biological molecules and interaction samples, such as chemicals, proteins, DNA, and their interactions. PyBioMed calculates nine types of features including chemical descriptors or molecular fingerprints, structural and physicochemical features of proteins and peptides from amino acid sequence, composition and physicochemical features of DNA from their primary sequences, chemical-chemical interaction features, chemical-protein interaction features, chemical-DNA interaction features, protein-protein interaction features, protein-DNA interaction features, and DNA-DNA interaction features. We hope that the package can be used for exploring questions concerning structures, functions and interactions of various molecular data in the context of chemoinformatics, bioinformatics, and systems biology.

OVERVIEW

To develop a powerful model for prediction tasks by machine learning algorithms such as scikit-learn, one of the most important things to consider is how to effectively represent the molecules under investigation such as small molecules, proteins, DNA and even complex interactions, by a descriptor. PyBioMed is a feature-rich package used for the characterization of various complex biological molecules and interaction samples, such as chemicals, proteins, DNA, and their interactions. PyBioMed calculates nine types of features including chemical descriptors or molecular fingerprints, structural and physicochemical features of proteins and peptides from amino acid sequence, composition and physicochemical features of DNA from their primary sequences, chemical-chemical interaction features, chemical-protein interaction features, chemical-DNA interaction features, protein-protein interaction features, protein-DNA interaction features, and DNA-DNA interaction features. We hope that the package can be used for exploring questions concerning structures, functions and interactions of various molecular data in the context of chemoinformatics, bioinformatics, and systems biology. The python package PyBioMed is designed by CBDD Group (Computational Biology & Drug Design Group), Xiangya School of Pharmaceutical Sciences, Central South University.

1.1 Who uses PyBioMed?

For those researchers from different biomedical fields, the PyBioMed package can be used to analyze and represent various complex molecular data under investigation. PyBioMed will be helpful when exploring questions concerning structures, functions and interactions of various molecular data in the context of chemoinformatics, bioinformatics, and systems biology.

1.2 Goals

PyBioMed is intended to provide

- Tools for pretreating molecules, proteins sequence and DNA sequence
- Calculating chemical descriptors or molecular fingerprints from molecules' structures
- Calculating structural and physicochemical features of proteins and peptides from amino acid sequence
- Calculating composition and physicochemical features of DNA from their primary sequences
- Calculating interaction features including chemical-chemical interaction features, chemical-protein interaction features, chemical-DNA interaction features, protein-protein interaction features, protein-DNA interaction features and DNA-DNA interaction features.
- Getting molecular structures, protein sequence and DNA sequence from Internet through the molecular ID, protein ID and DNA ID.

1.3 Feature overview

The table below shows the descriptors and the number of the descriptor that PyBioMed can calculate in four modules including PyMolecule, PyProtein, PyDNA and PyInteraction. PyMolecule module can calculate 14 different types of molecular descriptors and 18 different types of molecular fingerprints. PyProtein module can calculate 14 types of protein descriptors. PyDNA module can calculate 14 types of DNA descriptors and the number in the table appears when parameters are 'all_property = True, lamada=2, w=0.05'. PyInteraction module can calculate three types of descriptors.

Table 1.1: The descriptors in the PyBioMed package

Types	Features
PyMolecule	Constitution(30) Connectivity descriptors (44) Topology descriptors (35) Basak descriptors (21) Burden descriptors (64) Kappa descriptors (7) E-state descriptors (237) Moran autocorrelation descriptors (32) Geary autocorrelation descriptors (32) Molecular property descriptors (6) Moreau-Broto autocorrelation descriptors (32) Charge descriptors (25) MOE-type descriptors (60) CATS2D descriptors (150) Daylight-type fingerprints (2048) MACCS fingerprints (166) Atom pairs fingerprints TopologicalTorsion fingerprints E-state fingerprints (79) FP2 fingerprints (1024) FP3 fingerprints (210) FP4 fingerprints (307) ECFP2 fingerprints (1024) ECFP4 fingerprints (1024) ECFP6 fingerprints (1024) Morgan fingerprints (1024) Ghosecrippen fingerprints (110) FCFP2 fingerprints (1024) FCFP4 fingerprints (1024) FCFP6 fingerprints (1024) Pharm2D2point fingerprints (135) Pharm2D3point fingerprints (2135)
PyProtein	Amino acid composition (20) Dipeptide composition (400) Tripeptide composition (8000) CTD composition (21) CTD transition (21) CTD distribution (105) M-B autocorrelation (240) Moran autocorrelation (240)
<i>continued on next page</i>	

<i>continued from previous page</i>	
	Geary autocorrelation (240) Conjoint triad features (343) Quasi-sequence order descriptors (100) Sequence order coupling number (60) Pseudo amino acid composition 1 (50) Pseudo amino acid composition 2 (50)
PyDNA	Basic kmer (16) Reverse compliment kmer (12) DAC (76) DCC (2812) DACC (2888) TAC (24) TCC (264) TACC (288) PseDNC (18) PseKNC (66) PC-PseDNC (18) PC-PseTNC (66) SC-PseDNC (92) SC-PseTNC (88)
PyInteraction	Feature type 1 Feature type 2 Feature type 3

1.4 The Python programming language

Python is a powerful programming language that allows simple and flexible representations of biochemical molecules, and clear and concise expressions of bioinformatics algorithms. Python has a vibrant and growing ecosystem of packages that PyBioMed uses to provide more features such as RDkit and Pybel. In addition, Python is also an excellent “glue” language for putting together pieces of software from other languages which allows reuse of legacy code and engineering of high-performance algorithms. Equally important, Python is free, well-supported, and a joy to use. In order to make full use of PyBioMed, you will want to know how to write basic programs in Python. Among the many guides to Python, we recommend the documentation at <http://www.python.org>.

GETTING STARTED WITH PYBIOMED

This document is intended to provide an overview of how one can use the PyBioMed functionality from Python. If you find mistakes, or have suggestions for improvements, please either fix them yourselves in the source document (the .py file) or send them to the mailing list: oriental-cds@163.com and gadsby@163.com.

This document is intended to provide an overview of how one can use the PyBioMed functionality from Python. If you find mistakes, or have suggestions for improvements, please either fix them yourselves in the source document (the .py file) or send them to the mailing list: oriental-cds@163.com and gadsby@163.com

2.1 Installing the PyBioMed package

PyBioMed has been successfully tested on Linux and Windows systems. The user could download the PyBioMed package via: <http://pybiomed.scbdd.com/download/PyBioMed-1.0.zip> or <https://github.com/gadsbyfly/PyBioMed/blob/master/doc/download/PyBioMed-1.0.zip>. The installation process of PyBioMed is very easy:

Note: You first need to install RDKit and pybel successfully.

On Windows:

- (1): download the PyBioMed-1.0.zip
- (2): extract or uncompress the PyBioMed-1.0.zip file
- (3): cd PyBioMed-1.0
- (4): python setup.py install

On Linux:

- (1): download the PyBioMed package (.tar.gz)
- (2): tar -zxvf PyBioMed-1.0.tar.gz
- (3): cd PyBioMed-1.0
- (4): python setup.py install

2.2 Getting molecules

The PyGetMol provide different formats to get molecular structures, protein sequence and DNA sequence.

2.2.1 Getting molecular structure

In order to be convenient to users, the *Getmol* module provides the tool to get molecular structures by the molecular ID from website including NCBI, EBI, CAS, Kegg and Drugbank.

```
>>> from PyBioMed.PyGetMol import Getmol
>>> DrugBankID = 'DB01014'
>>> smi = Getmol.GetMolFromDrugbank(DrugBankID)
>>> print smi
N[C@@H](CO)C(=O)O
>>> smi=Getmol.GetMolFromCAS(casid="50-12-4")
>>> print smi
CCC1(c2ccccc2)C(=O)N(C)C(=N1)O
>>> smi=Getmol.GetMolFromNCBI(cid="2244")
>>> print smi
CC(=O)Oc1ccccc1C(=O)O
>>> smi=Getmol.GetMolFromKegg(kid="D02176")
>>> print smi
C[N+](C)(C)C[C@H](O)CC(=O)[O-]
```

2.2.2 Reading molecules

The *Getmol* module also provides the tool to read molecules in different formats including SDF, Mol, InChi and Smiles.

Users can read a molecule from string.

```
>>> from PyBioMed.PyGetMol.Getmol import ReadMolFromSmile
>>> mol = ReadMolFromSmile('N[C@@H](CO)C(=O)O')
>>> print mol
<rdkit.Chem.rdchem.Mol object at 0x0D8D3688>
```

Users can also read a molecule from a file.

```
>>> from PyBioMed.PyGetMol.Getmol import ReadMolFromSDF
>>> mol = ReadMolFromSDF('./PyBioMed/test/test_data/test.sdf') #You should change_
↳the path to your own real path
>>> print mol
<rdkit.Chem.rdmolfiles.SDMolSupplier at 0xd8d03f0>
```

2.2.3 Getting protein sequence

The *GetProtein* module provides the tool to get protein sequence by the pdb ID and uniprot ID from website.

```
>>> from PyBioMed.PyGetMol import GetProtein
>>> GetProtein.GetPDB(['1atp'])
>>> seq = GetProtein.GetSeqFromPDB('1atp.pdb')
>>> print seq
GNAAAKKGSEQESVKEFLAKAKEDFLKKWETPSQNTAQLDQFDRIKTLGTGSFGRVMLVKHKESGNHYAMKILDKQKVVKLKQ
IEHTLNEKRILQAVNFPFLVKLEFSFKDNSNLYMVEYVAGGEMFSLRRIGRFSEPHARFYAAQIVLTFEYLHSLDLIYRDLK
PENLLIDQQGYIQVTDGFGFAKRVKGRWTWXLGTPPEYLAPEIILSKGYNKAVDWWALGVLIYEMAAGYPPFFADQPIQIYEKIVS
GKVRFPSPHFSSDLKDLLRNLLQVDLTKRFGNLKNGVNDIKNHKWFATTDWIAIYQRKVEAPFIPKFKGPGDTSNFDDEEEEEIR
VXINEKCGKEFTEFTTYADFIASGRTGRRNAIHD
>>> seq = GetProtein.GetProteinSequence('000560')
>>> print seq
```

```
MSLYPSLEDLKVDKVIQAQTAFSANPANPAILSEASAPIPHDGNLYPRLYPELSQYMGLSLNNEEIRANVAVVSGAPLQGQLVA
RPSSINYMVAPVTGNDVGIIRAEIKQGIREVILCKDQDGKIGRLKSIDNGIFVQLVQANSASLVGLRFGDQVLQINGENCAG
WSSDKAHKVLKQAFGEKITMTIIRDRPFERTITMHKIDSTGHVGFIFKNGKITSIVKDSSAARNGLLTHEHNICEINGQNVIGLKDS
QIADILSTSGTVVTTITIMPAFIFEHIIKRMAPSIMKSLMDHTIPEV
```

2.2.4 Reading protein sequence

```
>>> from PyBioMed.PyGetMol.GetProtein import ReadFasta
>>> f = open('./PyBioMed/test/test_data/protein.fasta') #You should change the path_
↳to your own real path
>>> protein_seq = ReadFasta(f)
>>> print protein
['MLIHQYDHATAQYIASHLADPDPLNDGRWLIPAFATATPLPERPARTWPFFLDGAWVLRPDHRGQRLYRTDTGEAAEIVAAG
IAPEAAGLTPTPRPSDEHRWIDGAWQIDPQIVARARDAAMREFDLRMASARQANAGRADAYAAGLLSDAEIAVFKAWAIYQMD
LVRVVSAAASFDDVQWPAEPDEAAVIEQADGKASAGDAAAA',
'MLIHQYDHATAQYIASHLADPDPLNDGRWLIPAFATATPLPERPARTWPFFLDGAWVLRPDHRGQRLYRTDTGEAAEIVAAGI
APEAAGLTPTPRPSDEHRWIDGAWQIDPQIVARARDAAMREFDLRMASARQANAGRADAYAAGLLSDAEIAVFKAWAIYQMDL
VRVVSAAASFDDVQWPAEPDEAAVIEQADGKASAGDAAAA']
```

2.2.5 Getting DNA sequence

The `GetDNA` module provides the tool to get DNA sequence by the Gene ID from website.

```
>>> from PyBioMed.PyGetMol import GetDNA
>>> seq = GetDNA.GetDNAFromUniGene('AA954964')
>>> print seq
>ENA|AA954964|AA954964.1 op24b10.s1 Soares_NFL_T_GBC_S1 Homo sapiens cDNA clone IMAGE:
↳1577755 3'&apos;;, mRNA sequence.
TTTTAAATATAAAAGGATAACTTTATTGAATATACAAATTCAAGAGCATTCAATTTTTT
TTTAAGATTATGGCATAAGACAGATCAATGGTAATGGTTTATATATCCTATACTTACCAA
ACAGATTAGGTAGATATACTGACCTATCAATGCTCAAAATAACAAAATGAATACATGTCC
CTAAACTATTTCTGTATTCTACTACTAAATGGGAAATCTGTCAGCTGACCACCCACC
AGACTTTTTCCCATAGGAAGTTTGATATGCTGTCATTGATATATACCATTTCTGAATATA
AACCTCTATCTTGGGTCCTTTTCTCTTGCCTACTTCATTATCTGTCTTCCCAACCCACC
TAAGACTTAGTCAAAACAGGATACAGAGATCTGGATGGCTCTACGCAGAG
```

2.2.6 Reading DNA sequence

```
>>> from PyBioMed.PyGetMol.GetDNA import ReadFasta
>>> f = open('./PyBioMed/test/test_data/example.fasta') #You should change the path_
↳to your own real path
>>> dna_seq = ReadFasta(f)
>>> print dna
['GACTGAAGTGCACCTTTGGTTTCATATTATTGCTC']
```

2.3 Pretreating structure

The `PyPretreat` can pretreat the molecular structure, the protein sequence and the DNA sequence.

2.3.1 Pretreating molecules

The `PyPretreatMol` can pretreat the molecular structure. The `PyPretreatMol` provides the following functions:

- Normalization of functional groups to a consistent format.
- Recombination of separated charges.
- Breaking of bonds to metal atoms.
- Competitive reionization to ensure strongest acids ionize first in partially ionize molecules.
- Tautomer enumeration and canonicalization.
- Neutralization of charges.
- Standardization or removal of stereochemistry information.
- Filtering of salt and solvent fragments.
- Generation of fragment, isotope, charge, tautomer or stereochemistry insensitive parent structures.
- Validations to identify molecules with unusual and potentially troublesome characteristics.

The user can diconnect metal ion.

```
>>> from PyBioMed.PyPretreat.PyPretreatMol import StandardizeMol
>>> from rdkit import Chem
>>> mol = Chem.MolFromSmiles(' [Na]OC(=O)c1ccc(C[S+2]([O-])([O-]))cc1')
>>> sdm = StandardizeMol()
>>> mol = sdm.disconnect_metals(mol)
>>> print Chem.MolToSmiles(mol, isomericSmiles=True)
O=C([O-])c1ccc(C[S+2]([O-])[O-])cc1.[Na+]
```

Pretreat the molecular structure using all functions.

```
>>> from PyBioMed.PyPretreat import PyPretreatMol
>>> stdsmi = PyPretreatMol.StandardSmi(' [Na]OC(=O)c1ccc(C[S+2]([O-])([O-]))cc1')
>>> print stdsmi
O=C([O-])c1ccc(C[S](=O)=O)cc1
```

2.3.2 Pretreating protein sequence

The user can check the protein sequence using the `PyPretreatPro`. If the sequence is right, the result is the number of amino acids. If the sequence is wrong, the result is 0.

```
>>> from PyBioMed.PyPretreat import PyPretreatPro
>>> protein="ADGCGVGEGTGQGPMCMCMKWVYADEDAADLESDFSFADEDASLESDFSFPWSNQRVFCSFADEDASU"
>>> print PyPretreatPro.ProteinCheck(protein)
0
>>> from PyBioMed.PyPretreat import PyPretreatPro
>>> protein="ADGCRN"
>>> print PyPretreatPro.ProteinCheck(protein)
6
```


2.3.3 Pretreating DNA sequence

The user can check the DNA sequence using the `PyPretreatDNA`. If the sequence is right, the result is True. If the sequence is wrong, the result is the wrong word.

```
>>> from PyBioMed.PyPretreat import PyPretreatDNA
>>> DNA="ATTTAC"
>>> print PyPretreatDNA.DNAChecks (DNA)
True
>>> DNA= "ATCGUA"
>>> print PyPretreatDNA.DNAChecks (DNA)
U
```

2.4 Calculating molecular descriptors

The PyBioMed package could calculate a large number of molecular descriptors. These descriptors capture and magnify distinct aspects of chemical structures. Generally speaking, all descriptors could be divided into two classes: descriptors and fingerprints. Descriptors only used the property of molecular topology, including constitutional descriptors, topological descriptors, connectivity indices, E-state indices, Basak information indices, Burden descriptors, autocorrelation descriptors, charge descriptors, molecular properties, kappa shape indices, MOE-type descriptors. Molecular fingerprints contain FP2, FP3, FP4, topological fingerprints, Estate, atompairs, torsions, morgan and MACCS.

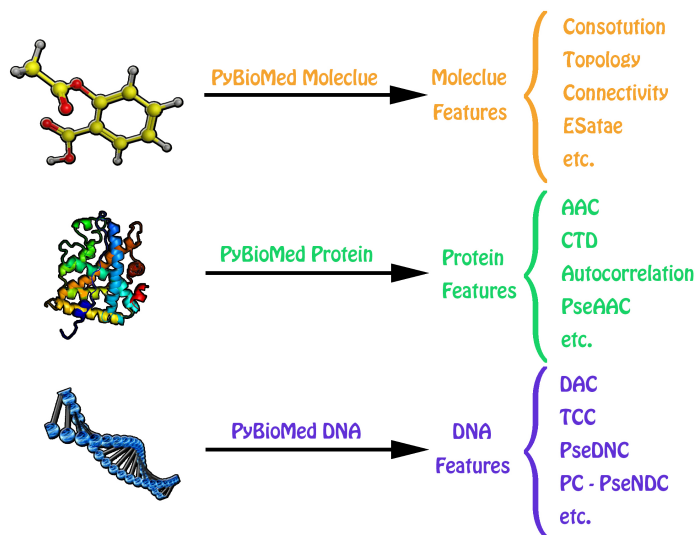


Fig. 2.1: The descriptors could be calculated through PyBioMed package

2.4.1 Calculating descriptors

We could import the corresponding module to calculate the molecular descriptors as need. There is 14 modules to compute descriptors. Moreover, a easier way to compute these descriptors is construct a PyMolecule object, which encapsulates all methods for the calculation of descriptors.

Calculating molecular descriptors via functions

The `GetConnectivity()` function in the `connectivity` module can calculate the connectivity descriptors. The result is given in the form of dictionary.

```
>>> from PyBioMed.PyMolecule import connectivity
>>> from rdkit import Chem
>>> smi = 'CCC1(c2ccccc2)C(=O)N(C)C(=N1)O'
>>> mol = Chem.MolFromSmiles(smi)
>>> molecular_descriptor = connectivity.GetConnectivity(mol)
>>> print molecular_descriptor
{'Chi3ch': 0.0, 'knotp': 2.708, 'dchi3': 3.359, 'dchi2': 2.895, 'dchi1': 2.374, 'dchi0': 2.415, 'Chi5ch': 0.068, 'Chiv4': 1.998, 'Chiv7': 0.259, 'Chiv6': 0.591, 'Chiv1': 5.241, 'Chiv0': 9.344, 'Chiv3': 3.012, 'Chiv2': 3.856, 'Chi4c': 0.083, 'dchi4': 2.96, 'Chiv4pc': 1.472, 'Chiv3c': 0.588, 'Chiv8': 0.118, 'Chi3c': 1.264, 'Chi8': 0.636, 'Chi9': 0.322, 'Chi2': 6.751, 'Chi3': 6.372, 'Chi0': 11.759, 'Chi1': 7.615, 'Chi6': 2.118, 'Chi7': 1.122, 'Chi4': 4.959, 'Chi5': 3.649, 'Chiv5': 1.244, 'Chiv4c': 0.04, 'Chiv9': 0.046, 'Chi4pc': 3.971, 'knotpv': 0.884, 'Chiv5ch': 0.025, 'Chiv3ch': 0.0, 'Chiv10': 0.015, 'Chiv6ch': 0.032, 'Chi10': 0.135, 'Chi4ch': 0.0, 'Chiv4ch': 0.0, 'mChi1': 0.448, 'Chi6ch': 0.102}
```

The function `GetTopology()` in the `topology` module can calculate all topological descriptors.

```
>>> from PyBioMed.PyMolecule import topology
>>> from rdkit import Chem
>>> smi = 'CCC1(c2ccccc2)C(=O)N(C)C(=N1)O'
>>> mol = Chem.MolFromSmiles(smi)
>>> molecular_descriptor = topology.GetTopology(mol)
>>> print len(molecular_descriptor)
25
```

The function `CATS2D()` in the `cats2d` module can calculate all CATS2D descriptors.

```
>>> from PyBioMed.PyMolecule.cats2d import CATS2D
>>> smi = 'CC(N)C(=O)[O-]'
>>> mol = Chem.MolFromSmiles(smi)
>>> cats = CATS2D(mol, PathLength = 10, scale = 3)
>>> print cats
{'CATS_AP4': 0.0, 'CATS_AP3': 1.0, 'CATS_AP6': 0.0, 'CATS_AA8': 0.0, 'CATS_AA9': 0.0, 'CATS_AP1': 0.0, ..., 'CATS_AP5': 0.0}
>>> print len(cats)
150
```

Calculating molecular descriptors via PyMolecule object

The `PyMolecule` class can read molecules in different format including MOL, SMI, InChi and CAS. For example, the user can read a molecule in the format of SMI and calculate the E-state descriptors (316).

```
>>> from PyBioMed import Pymolecule
>>> smi = 'CCC1(c2ccccc2)C(=O)N(C)C(=N1)O'
>>> mol = Pymolecule.PyMolecule()
>>> mol.ReadMolFromSmile(smi)
>>> molecular_descriptor = mol.GetEstate()
>>> print len(molecular_descriptor)
237
```

The object can also read molecules in the format of MOL file and calculate charge descriptors (25).

```
>>> from PyBioMed import Pymolecule
>>> mol = Pymolecule.PyMolecule()
>>> mol.ReadMolFromMol('test/test_data/test.mol')    #change path to the real path in_
↳your own computer
>>> molecular_descriptor = mol.GetCharge()
>>> print molecular_descriptor
{'QNmin': 0, 'QOss': 0.534, 'Mpc': 0.122, 'QHss': 0.108, 'SPP': 0.817, 'LDI': 0.322,
↳'QCmin': -0.061, 'Mac': 0.151, 'Qass': 0.893, 'QNss': 0, 'QCmax': 0.339, 'QOmax': -
↳0.246, 'Tpc': 1.584, 'Qmax': 0.339, 'QOmin': -0.478, 'Tnc': -1.584, 'QHmin': 0.035,
↳'QCss': 0.252, 'QHmax': 0.297, 'QNmax': 0, 'Rnc': 0.302, 'Rpc': 0.214, 'Qmin': -0.
↳478, 'Tac': 3.167, 'Mnc': -0.198}
```

In order to be convenient to users, the object also provides the tool to get molecular structures by the molecular ID from website including NCBI, EBI, CAS, Kegg and Drugbank.

```
>>> from PyBioMed import Pymolecule
>>> DrugBankID = 'DB01014'
>>> mol = Pymolecule.PyMolecule()
>>> smi = mol.GetMolFromDrugbank(DrugBankID)
>>> mol.ReadMolFromSmile(smi)
>>> molecular_descriptor = mol.GetKappa()
>>> print molecular_descriptor
{'phi': 5.989303307692309, 'kappa1': 22.291, 'kappa3': 7.51, 'kappa2': 11.111,
↳'kappam1': 18.587, 'kappam3': 5.395, 'kappam2': 8.378}
```

The code below can calculate all molecular descriptors except fingerprints.

```
>>> from PyBioMed import Pymolecule
>>> smi = 'CCOC=N'
>>> mol = Pymolecule.PyMolecule()
>>> mol.ReadMolFromSmile(smi)
>>> alldes = mol.GetAllDescriptor()
>>> print len(alldes)
765
```

2.4.2 Calculating molecular fingerprints

In the *fingerprint* module, there are eighteen types of molecular fingerprints which are defined by abstracting and magnifying different aspects of molecular topology.

Calculating fingerprint via functions

The CalculateFP2Fingerprint() function calculates the FP2 fingerprint.

```
>>> from PyBioMed.PyMolecule.fingerprint import CalculateFP2Fingerprint
>>> import pybel
>>> smi = 'CCC1(c2ccccc2)C(=O)N(C)C(=N1)O'
>>> mol = pybel.readstring("smi", smi)
>>> mol_fingerprint = CalculateFP2Fingerprint(mol)
>>> print len(mol_fingerprint[1])
103
```

The CalculateEstateFingerprint() function calculates the Estate fingerprint.

```
>>> from PyBioMed.PyMolecule.fingerprint import CalculateEstateFingerprint
>>> smi = 'CCCl(c2ccccc2)C(=O)N(C)C(=N1)O'
>>> mol = Chem.MolFromSmiles(smi)
>>> mol_fingerprint = CalculateEstateFingerprint(mol)
>>> print len(mol_fingerprint[2])
79
```

The function `GhoseCrippenFingerprint()` in the `ghosecrippen` module can calculate all ghosecrippen descriptors.

```
>>> from PyBioMed.PyMolecule.ghosecrippen import GhoseCrippenFingerprint
>>> smi = 'CC(N)C(=O)O'
>>> mol = Chem.MolFromSmiles(smi)
>>> ghoseFP = GhoseCrippenFingerprint(mol)
>>> print ghoseFP
{'S3': 0, 'S2': 0, 'S1': 0, 'S4': 0, ....., 'N9': 0, 'Hal2': 0}
>>> print len(ghoseFP)
110
```

Calculating fingerprint via object

The `PyMolecule` class can calculate eleven kinds of fingerprints. For example, the user can read a molecule in the format of SMI and calculate the ECFP4 fingerprint (1024).

```
>>> from PyBioMed import Pymolecule
>>> smi = 'CCOC=N'
>>> mol = Pymolecule.PyMolecule()
>>> mol.ReadMolFromSmile(smi)
>>> res = mol.GetFingerprint(FPName='ECFP4')
>>> print res
(4294967295L, {3994088662L: 1, 2246728737L: 1, 3542456614L: 1, 2072128742: 1,
↳2222711142L: 1, 2669064385L: 1, 3540009223L: 1, 849275503: 1, 2245384272L: 1,
↳2246703798L: 1, 864674487: 1, 4212523324L: 1, 3340482365L: 1}, <rdkit.DataStructs.
↳cDataStructs.UIntSparseIntVect object at 0x0CA010D8>)
```

2.5 Calculating protein descriptors

`PyProtein` is a tool used for protein feature calculation. `PyProtein` calculates structural and physicochemical features of proteins and peptides from amino acid sequence. These sequence-derived structural and physicochemical features have been widely used in the development of machine learning models for predicting protein structural and functional classes, post-translational modification, subcellular locations and peptides of specific properties. There are two ways to calculate protein descriptors in the `PyProtein` module. One is to directly use the corresponding methods, the other one is firstly to construct a `PyProtein` class and then run their methods to obtain the protein descriptors. It should be noted that the output is a dictionary form, whose keys and values represent the descriptor name and the descriptor value, respectively. The user could clearly understand the meaning of each descriptor.

2.5.1 Calculating protein descriptors via functions

The user can input the protein sequence and calculate the protein descriptors using function.

```
>>> from PyBioMed.PyProtein import AAComposition
>>> protein="ADGCGVGEGTGQGPMCMCMKWVYAEDAADLESDFSFADEDASLESDFSFPWSNQRFCSFADEDAS"
>>> AAC=AAComposition.CalculateAAComposition(protein)
>>> print AAC
{'A': 11.94, 'C': 7.463, 'E': 8.955, 'D': 14.925, 'G': 8.955, 'F': 5.97, 'I': 0.0, 'H': 0.0, 'K': 1.493, 'M': 4.478, 'L': 2.985, 'N': 2.985, 'Q': 2.985, 'P': 2.985, 'S': 11.94, 'R': 1.493, 'T': 1.493, 'W': 2.985, 'V': 4.478, 'Y': 1.493}
```

PyBioMed also provides `getpdb` to get sequence from [PDB](#) website to calculate protein descriptors.

```
>>> from PyBioMed.PyGetMol import GetProtein
>>> GetProtein.GetPDB(['latp', 'lefz', 'lf88'])
>>> seq = GetProtein.GetSeqFromPDB('latp.pdb')
>>> print seq
GNAAAAGKGESEVKEFLAKAKEDFLKKWETPSQNTAQLDQFDRIKTLGTGSFGRVMLVKHKESGNHYAMKILDKQKVVKLQ
IEHTLNEKRILQAVNFPFLVKLEFSFKDNSNLYMVMYVAGGEMFSLRRIGRFSEPHARFYAAQIVLTFFEYLHSLDLIYRDLK
PENLLIDQQGYIQVTDGFGAKRVKGRWTXLCGTPEYLAPEIILSKGYNKAVDWWALGVLIYEMAAGYPPFFADQPIQIYEKIVS
GKVRFPESHFSSDLKDLLRNLQVDLTKRFGNLKNGVNDIKNHKWFATTDWIAIYQRKVEAPFIPKFKGPGDTSNFDDYEEEEIR
VXINEKCGKEFTEFTTYADFIASGRTGRRNAIHD
>>> from PyBioMed.PyProtein import CTD
>>> protein_descriptor = CTD.CalculateC(protein)
>>> print protein_descriptor
{'_NormalizedVDWVC2': 0.224, '_PolarizabilityC2': 0.328, '_PolarizabilityC3': 0.179,
'_ChargeC1': 0.03, '_PolarizabilityC1': 0.493, '_SecondaryStrC2': 0.239, '_SecondaryStrC3': 0.418, '_NormalizedVDWVC3': 0.179, '_SecondaryStrC1': 0.343, '_SolventAccessibilityC1': 0.448, '_SolventAccessibilityC2': 0.328, '_SolventAccessibilityC3': 0.224, '_NormalizedVDWVC1': 0.522, '_HydrophobicityC3': 0.284, '_HydrophobicityC1': 0.328, '_ChargeC3': 0.239, '_PolarityC2': 0.179, '_PolarityC1': 0.299, '_HydrophobicityC2': 0.388, '_PolarityC3': 0.03, '_ChargeC2': 0.731}
```

2.5.2 Calculate protein descriptors via object

The *PyProtein* can calculate all kinds of protein descriptors in the PyBioMed. For example, the *PyProtein* can calculate DPC.

```
>>> from PyBioMed import Pyprotein
>>> protein="ADGCGVGEGTGQGPMCMCMKWVYAEDAADLESDFSFADEDASLESDFSFPWSNQRFCSFADEDAS"
>>> protein_class = Pyprotein.PyProtein(protein)
>>> print len(protein_class.GetDPComp())
400
```

The *PyProtein* also provide the tool to get sequence from [Uniprot](#) through the Uniprot ID.

```
>>> from PyBioMed import Pyprotein
>>> from PyBioMed.PyProtein.GetProteinFromUniprot import GetProteinSequence
>>> uniprotID = 'P48039'
>>> protein_sequence = GetProteinSequence(uniprotID)
>>> print protein_sequence
MEDINFASLAPRHGSRPFMGWTWNEIGTSQLNGGAFWSSSLWSGIKNGFSSIKSFGNKAWSNTGQMLRDKLKDQNFQQKVVDGL
ASGINGVVDIANQALQNLQINQRLNSRQPPVALQQRPPPKVEEVEVEEKLPPLEVAPPLPSKGEKRPDPLEETLVVESREPPS
YEQALKEGASPYPMTKPIGSMARPVYGKESKPVLTLELPPPVPTVPMPAPTTLGTAVSRPTAPTAVATPARRPRGANWQSTLNS
IVGLGVKSLKRRRCY
>>> protein_class = Pyprotein.PyProtein(protein_sequence)
>>> CTD = protein_class.GetCTD()
```

```
>>> print len(CTD)
147
```

The *PyProtein* can calculate all protein descriptors except the tri-peptide composition descriptors.

```
>>> from PyBioMed import Pyprotein
>>> protein="ADGCGVGEGTGQGPNCNMCWKVYADEDAADLESDFSFADEDASLESDFSFPWSNQRVFCSFADEDAS"
>>> protein_class = Pyprotein.PyProtein(protein)
>>> print len(protein_class.GetAll())
10049
```

2.6 Calculating DNA descriptors

The PyDNA module can generate various feature vectors for DNA sequences, this module could:

- Calculating three nucleic acid composition features describing the local sequence information by means of kmers (subsequences of DNA sequences);
- Calculating six autocorrelation features describing the level of correlation between two oligonucleotides along a DNA sequence in terms of their specific physicochemical properties;
- Calculating six pseudo nucleotide composition features, which can be used to represent a DNA sequence with a discrete model or vector yet still keep considerable sequence order information, particularly the global or long-range sequence order information, via the physicochemical properties of its constituent oligonucleotides.

2.6.1 Calculating DNA descriptors via functions

The user can input a DNA sequence and calculate the DNA descriptors using functions.

```
>>> from PyBioMed.PyDNA.PyDNAac import GetDAC
>>> dac = GetDAC('GACTGAAGTGCACCTTTGGTTTCATATTATTGCTC', phyche_index=['Twist','Tilt'])
>>> print(dac)
{'DAC_4': -0.004, 'DAC_1': -0.175, 'DAC_2': -0.185, 'DAC_3': -0.173}
```

The user can check the parameters and calculate the descriptors.

```
>>> from PyBioMed.PyDNA import PyDNApsenac
>>> from PyBioMed.PyDNA.PyDNApsenac import GetPseDNC
>>> dnaseq = 'GACTGAAGTGCACCTTTGGTTTCATATTATTGCTC'
>>> PyDNApsenac.CheckPsenac(lamada = 2, w = 0.05, k = 2)
>>> pseDnc = GetPseDNC('ACCCCA',lamada=2, w=0.05)
>>> print(pseDnc)
{'PseDNC_18': 0.0521, 'PseDNC_16': 0.0, 'PseDNC_17': 0.0391, 'PseDNC_14': 0.0,
↪ 'PseDNC_15': 0.0, 'PseDNC_12': 0.0, 'PseDNC_13': 0.0, 'PseDNC_10': 0.0, 'PseDNC_11':
↪ 0.0, 'PseDNC_4': 0.0, 'PseDNC_5': 0.182, 'PseDNC_6': 0.545, 'PseDNC_7': 0.0,
↪ 'PseDNC_1': 0.0, 'PseDNC_2': 0.182, 'PseDNC_3': 0.0, 'PseDNC_8': 0.0, 'PseDNC_9': 0.
↪ 0}
```

2.6.2 Calculating DNA descriptors via object

The PyDNA can calculate all kinds of protein descriptors in the PyBioMed. For example, the PyDNA can calculate SCPseDNC.

```
>>> from PyBioMed import Pydna
>>> dna = Pydna.PyDNA('GACTGAACTGCACCTTTGGTTTCATATTATTGCTC')
>>> scpsednc = dna.GetSCPseDNC()
>>> print len(scpsednc)
16
```

2.7 Calculating Interaction descriptors

The PyInteraction module can generate six types of interaction descriptors including chemical-chemical interaction features, chemical-protein interaction features, chemical-DNA interaction features, protein-protein interaction features, protein-DNA interaction features, and DNA-DNA interaction features by integrating two groups of features.

The user can choose three different types of methods to calculate interaction descriptors. The function `CalculateInteraction1()` can calculate two interaction features by combining two features.

$$F_{ab} = (F_a, F_b)$$

The function `CalculateInteraction2()` can calculate two interaction features by two multiplied features.

$$F = \{F(k) = F_a(i)F_b(j), i = 1, 2, \dots, p, j = 1, 2, \dots, p, k = (i-1)p + j\}$$

The function `CalculateInteraction3()` can calculate two interaction features by

$$F = [F_a(i) + F_b(i), F_a(i) * F_b(i)]$$

The function `CalculateInteraction3()` is only used in the same type of descriptors including chemical-chemical interaction, protein-protein interaction and DNA-DNA interaction.

The user can calculate chemical-chemical features using three methods.

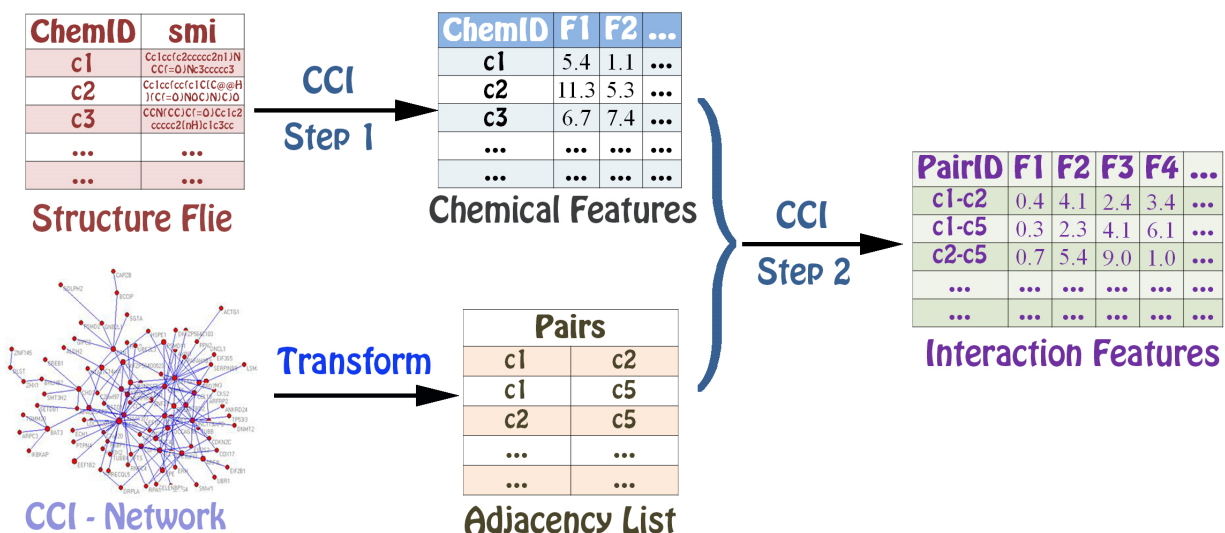


Fig. 2.2: The calculation process for chemical-chemical interaction descriptors.

```
>>> from PyBioMed.PyInteraction import PyInteraction
>>> from PyBioMed.PyMolecule import moe
```

```

>>> from rdkit import Chem
>>> smis = ['CCCC', 'CCCCC', 'CCCCC', 'CC(N)C(=O)O', 'CC(N)C(=O)[O-].[Na+]' ]
>>> m = Chem.MolFromSmiles(smis[3])
>>> mol_des = moe.GetMOE(m)
>>> mol_mol_interaction1 = PyInteraction.CalculateInteraction1(mol_des, mol_des)
>>> print mol_mol_interaction1
{'slogPVSA6ex': 0.0, 'PEOEVSAl0ex': 0.0, ..., 'EstateVSA9ex': 4.795, 'slogPVSA2': 4.
↪ 795, 'slogPVSA3': 0.0, 'slogPVSA0': 5.734, 'slogPVSA1': 17.118, 'slogPVSA6': 0.0,
↪ 'slogPVSA7': 0.0, 'slogPVSA4': 6.924, 'slogPVSA5': 0.0, 'slogPVSA8': 0.0, 'slogPVSA9
↪ ': 0.0}
>>> print len(mol_mol_interaction1)
120
>>> mol_mol_interaction2 = PyInteraction.CalculateInteraction2(mol_des, mol_des)
>>> print len(mol_mol_interaction2)
3600
>>> mol_mol_interaction3 = PyInteraction.CalculateInteraction3(mol_des, mol_des)
{'EstateVSA9+EstateVSA9': 22.992, 'EstateVSA9+EstateVSA9': 9.59, 'PEOEVSAl+PEOEVSAl':
↪ 9.59, 'VSAEstate10*VSAEstate10': 0.0, 'PEOEVSAl*PEOEVSAl': 0.0, 'PEOEVSAl1*PEOEVSAl1
↪ ': 0.0, 'PEOEVSAl4*PEOEVSAl4': 0.0, 'VSAEstate2*VSAEstate2': 0.0, 'MRVSA0+MRVSA0': 19.
↪ 802, 'MRVSA6+MRVSA6': 0.0.....}
>>> print len(mol_mol_interaction3)
120

```

The user can calculate chemical-protein feature using two methods.

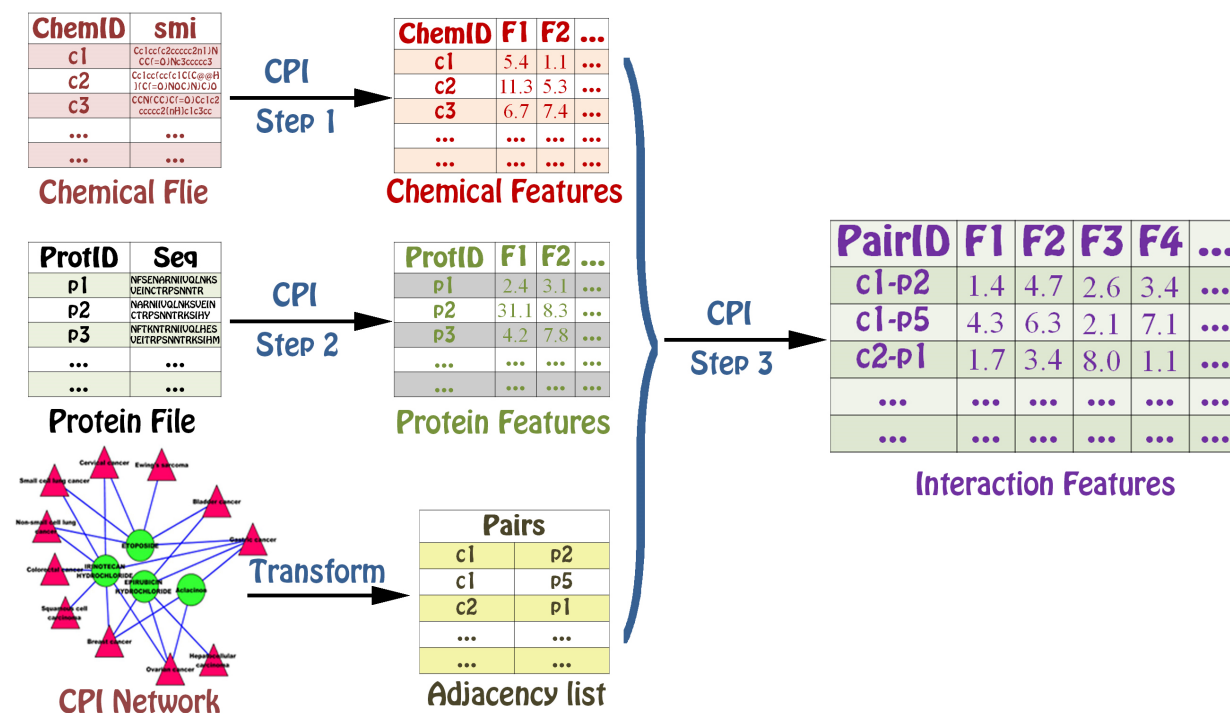


Fig. 2.3: The calculation process for chemical-protein interaction descriptors.

```

>>> from rdkit import Chem
>>> from PyBioMed.PyMolecule import moe
>>> from PyBioMed.PyInteraction.PyInteraction import CalculateInteraction2
>>> smis = ['CCCC', 'CCCCC', 'CCCCC', 'CC(N)C(=O)O', 'CC(N)C(=O)[O-].[Na+]' ]

```



```

>>> m = Chem.MolFromSmiles(smis[3])
>>> mol_des = moe.GetMOE(m)
>>> from PyBioMed.PyDNA.PyDNApsenac import GetPseDNC
>>> protein_des = GetPseDNC('ACCCCA', lamada=2, w=0.05)
>>> pro_mol_interaction1 = PyInteraction.CalculateInteraction1(mol_des, protein_des)
>>> print len(pro_mol_interaction1)
78
>>> pro_mol_interaction2 = CalculateInteraction2(mol_des, protein_des)
>>> print len(pro_mol_interaction2)
1080

```

The user can calculate chemical-DNA feature using two methods.

```

>>> from PyBioMed.PyDNA import PyDNAac
>>> DNA_des = PyDNAac.GetTCC('GACTGAAC TGCACTTTGGTTTCATATTATTGCTC', phyche_index=[
↪ 'Dnase I', 'Nucleosome', 'MW-kg'])
>>> from rdkit import Chem
>>> smis = ['CCCC', 'CCCCC', 'CCCCC', 'CC(N)C(=O)O', 'CC(N)C(=O)[O-].[Na+']
>>> m = Chem.MolFromSmiles(smis[3])
>>> mol_des = moe.GetMOE(m)
>>> mol_DNA_interaction1 = PyInteraction.CalculateInteraction1(mol_des, DNA_des)
>>> print len(mol_DNA_interaction1)
72
>>> mol_DNA_interaction2 = PyInteraction.CalculateInteraction2(mol_des, DNA_des)
>>> print len(mol_DNA_interaction2)
720

```


APPLICATION

The PyBioMed Python package can generate various feature vectors for molecular structure, protein sequences and DNA sequences. The PyBioMed package would be applied to solve many tasks in the field of cheminformatics, bioinformatics and systems biology. We will introduce five examples of its applications including Caco-2 cell permeability, aqueous solubility, drug–target interaction data, protein subcellular location, and nucleosome positioning in genomes. All datasets and Python scripts used in the next five examples can be download on <https://github.com/gadsbyfly/PyBioMed/blob/master/doc/download>.

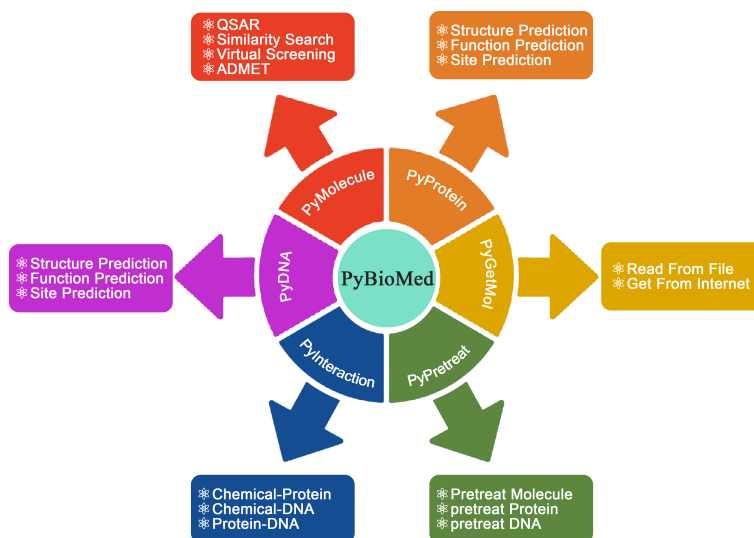


Fig. 3.1: The overview of PyBioMed python package. PyBioMed could calculate various molecular descriptors from chemicals, proteins, DNAs/RNAs and their interactions. PyBioMed can also pretreat molecules, protein sequences and DNA sequences.

3.1 Application 1 Prediction of Caco-2 Cell Permeability

Caco-2 cell monolayer model is a popular surrogate in predicting the in vitro human intestinal permeability of a drug due to its morphological and functional similarity with human enterocytes. Oral administration of drugs is the preferred route and a major goal in the development of new drugs because of its ease and patient compliance. Before an oral drug reaches the systemic circulation, it must pass through intestinal cell membranes via passive diffusion, carrier-mediated uptake or active transport processes. Bioavailability, reflecting the drug proportion in the circulatory system, is a significant index of drug efficacy. Screening for absorption ability is one of the most important parts of assessing oral bioavailability. Caco-2 cell line is a popular surrogate for the human intestinal epithelium to estimate in vivo drug permeability due to their morphological and functional similarities with human enterocytes. To build a

Caco-2 cell permeability prediction model, we use the PyBioMed package to calculate molecular features and then the Random Forest (RF) method was applied to build Caco-2 cell permeability classification model. The benchmark data set for building the Caco-2 cell permeability predictor was taken from (NN Wang et al. 2016). The dataset contains 1272 compounds.

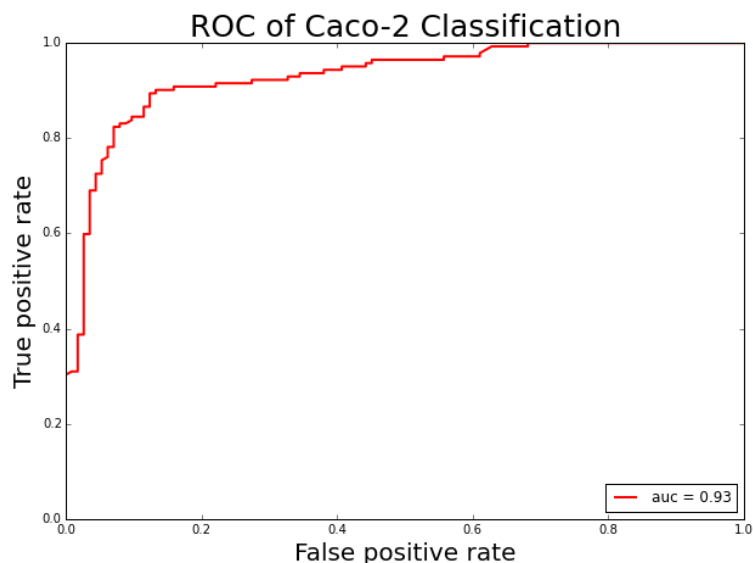


Fig. 3.2: The receiver operating characteristic curve of Caco-2 classification.

```

1  from PyBioMed import Pymolecule
2  import pandas as pd
3  from sklearn.ensemble import RandomForestClassifier
4  from sklearn import metrics
5  from matplotlib import pyplot as plt
6  #=====
7  # load the data
8  #=====
9  train_set = pd.read_excel('D:/PyBioMed/PyBioMed-1.0/PyBioMed/example/caco2/caco2.xlsx
10 ↪',sheetname=0)
11 test_set = pd.read_excel('D:/PyBioMed/PyBioMed-1.0/PyBioMed/example/caco2/caco2.xlsx',
12 ↪sheetname=1)
13
14 train_set_smi = train_set['smi']
15 test_set_smi = test_set['smi']
16
17 train_set_label = train_set[['label']]
18 test_set_label = test_set[['label']]
19 #=====
20 # calculating molecular descriptors to a descriptors dataframe
21 #=====
22 def calculate_des(smi):
23     des = {}
24     drugclass=Pymolecule.Pymolecule()
25     drugclass.ReadMolFromSmile(smi)
26     des.update(drugclass.GetMoran())
27     des.update(drugclass.GetMOE())
28     return pd.DataFrame({smi:(des)}).T
29
30 train_set_des = pd.concat(map(calculate_des,list(train_set_smi)))

```

```

29 test_set_des = pd.concat(map(calculate_des,list(test_set_smi)))
30 #=====
31 # building the model and predicting the test set
32 #=====
33 clf = RandomForestClassifier(n_estimators=500,max_features='sqrt', n_jobs=-1, max_
    ↳depth=None,random_state=0)
34 clf.fit(train_set_des,train_set_label)
35
36 proba = clf.predict_proba(test_set_des)[:,-1]
37 predict_label = clf.predict(test_set_des)
38 #=====
39 # Calculating auc score
40 #=====
41 AUC_score = round(metrics.roc_auc_score(test_set_label, proba),2)
42 TPR = round(metrics.recall_score(test_set_label, predict_label),2)
43 ACC = round(metrics.accuracy_score(test_set_label, predict_label),2)
44 P = float(test_set_label.sum())
45 N = test_set_label.shape[0] - P
46 SPE = round((P/N+1.0)*ACC-TPR*P/N,2)
47 matthews_corrcoef = round(metrics.matthews_corrcoef(test_set_label, predict_label),2)
48 f1_score = round(metrics.f1_score(test_set_label, predict_label), 2)
49 fpr_cv, tpr_cv, thresholds_cv = metrics.roc_curve(test_set_label, proba)
50 #=====
51 # plotting the auc plot
52 #=====
53 plt.figure(figsize = (10,7))
54 plt.plot(fpr_cv, tpr_cv, 'r', label='auc = %0.2f'% AUC_score, lw=2)
55 plt.xlabel('False positive rate',{'fontsize':20});
56 plt.ylabel('True positive rate',{'fontsize':20});
57 plt.title('ROC of Caco-2 Classification',{'fontsize':25})
58 plt.legend(loc="lower right",numpoints=15)
59 plt.show()

```

```

>>> print 'sensitivity:',TPR, 'specificity:', SPE, 'accuracy:', ACC, 'AUC:', AUC_
    ↳score, 'MACCS:', matthews_corrcoef, 'F1:', f1_score
sensitivity: 0.91 specificity: 0.8 accuracy: 0.86 AUC: 0.93 MACCS: 0.72 F1: 0.88

```

3.2 Application 2 Prediction of aqueous solubility

Aqueous solubility is one of the major drug properties to be optimized in drug discovery. Aqueous solubility and membrane permeability are the two key factors that affect a drug's oral bioavailability. Generally, a drug with high solubility and membrane permeability is considered to have bioavailability problems. Otherwise, it is a problematic candidate or needs careful formulation work. To build an aqueous solubility prediction model, we use the PyBioMed package to calculate molecular features and then the random forest (RF) method was applied to build aqueous solubility regression model. The benchmark data set for building the aqueous solubility regression model was taken from (Junmei Wang et al.). The dataset contains 3637 compounds.

```

1 from PyBioMed import Pymolecule
2 import pandas as pd
3 import numpy as np
4 from sklearn import cross_validation
5 from sklearn.ensemble import RandomForestRegressor
6 from matplotlib import pyplot as plt
7 from sklearn.cross_validation import train_test_split
8 from sklearn import metrics

```

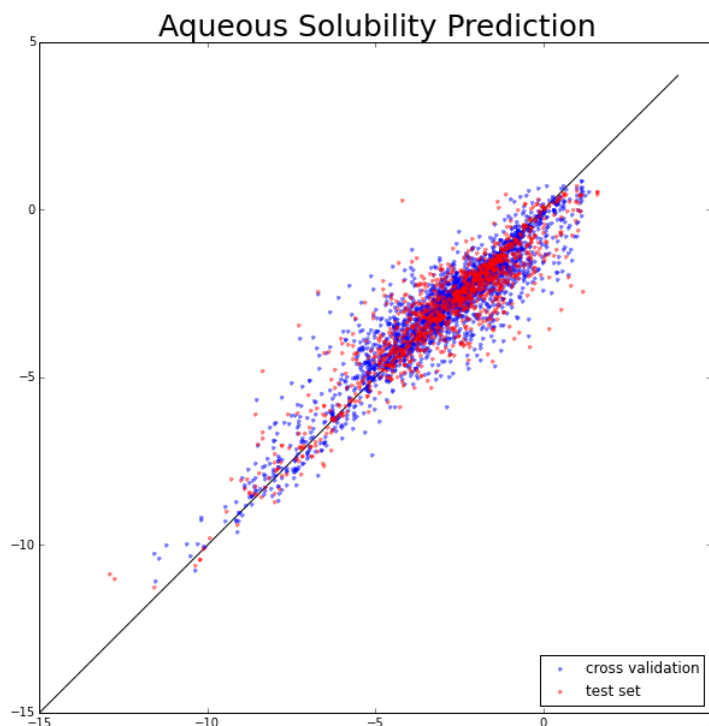


Fig. 3.3: The aqueous solubility prediction. The X-axis represents experimental values and the Y-axis represents predicted values.

```

9  #=====
10 # loading the data
11 #=====
12 solubility_set = pd.read_excel('./PyBioMed/example/solubility/Solubility-total.xlsx',
13                               ↪sheetname = 0) #change the path to the real path
14 smis = solubility_set['SMI']
15 logS = solubility_set['logS']
16 #=====
17 # #calculating molecular descriptors
18 #=====
19 def calculate_des(smi):
20     des = {}
21     drugclass=Pymolecule.PyMolecule()
22     drugclass.ReadMolFromSmile(smi)
23     des.update(drugclass.GetEstate())
24     des.update(drugclass.GetMOE())
25     return pd.DataFrame({smi:(des)}).T
26 solubility_set_des = pd.concat(map(calculate_des,list(smis)))
27 solubility_set_des = np.array(solubility_set_des)
28 logS = np.array(logS)
29 #=====
30 # building the model and predict
31 #=====
32 train_set_des, test_set_des, train_logS, test_logS = train_test_split(solubility_set_
33                               ↪des,
34                               ↪logS, test_size = 0.33, random_state = 42)

```

```

34 kf = cross_validation.KFold(train_set_des.shape[0], n_folds=10, random_state=0)
35 clf = RandomForestRegressor(n_estimators=500, max_features='auto', n_jobs = -1)
36 CV_pred_logS = []
37 VALIDATION_index = []
38 for train_index, validation_index in kf:
39     VALIDATION_index = VALIDATION_index + list(validation_index)
40     clf.fit(train_set_des[train_index, :], train_logS[train_index])
41     pred_logS = clf.predict(train_set_des[validation_index, :])
42     CV_pred_logS = CV_pred_logS + list(pred_logS)
43 CV_true_logS = train_logS[VALIDATION_index]
44 r2_CV = metrics.r2_score(CV_true_logS, CV_pred_logS)
45
46 clf.fit(train_set_des, train_logS)
47 pred_logS_test = clf.predict(test_set_des)
48 r2_test = metrics.r2_score(test_logS, pred_logS_test)
49 #=====
50 # plotting the figure
51 #=====
52 plt.figure(figsize = (10,10))
53 plt.plot(range(-15,5), range(-15,5), 'black')
54 plt.plot(CV_true_logS, CV_pred_logS, 'b.', label = 'cross validation', alpha = 0.5 )
55 plt.plot(test_logS, pred_logS_test, 'r.', label = 'test set', alpha = 0.5)
56 plt.title('Aqueous Solubility Prediction', {'fontsize':25})
57 plt.legend(loc="lower right", numpoints=1)
58 plt.plot()

```

```

>>> print 'CV_R^2:', 'r2_cv', 'Test_R^2:', 'r2_test'
CV_R^2: 0.86 Test_R^2: 0.84

```

3.3 Application 3 Prediction of drug–target interaction from the integration of chemical and protein spaces

Drug-target interactions (DTIs) are central to current drug discovery processes and public health fields. The rapidly increasing amount of publicly available data in biology and chemistry enables researchers to revisit drug-target interaction problems by systematic integration and analysis of heterogeneous data. To identify the interactions between drugs and targets is of important in drug discovery today. Interaction with ligands can modulate the function of many targets in the processes of signal transport, catalytic reaction and so on. With the enrichment of data repository, automatically prediction of target-protein interactions is an alternative method to facilitate drug discovery. Our previous work (Cao et al, 2014) proved that the calculated features perform well in the prediction of chemical-protein interaction. The benchmark data set for building the drug-target interaction predictor was taken from (Yamanishi, Araki et al. 2008). The dataset contains 6888 samples, among them 2922 drug-protein pairs have interactions which are defined as positive dataset and 3966 drug-protein pairs do not have interactions which are defined as negative dataset. To represent each drug-protein pairs, 150 CATS molecular fingerprints and 147 CTD composition, transition and distribution features of protein, a total number of 297 features were used. The random forest (RF) classifier was employed to build model.

```

1 from PyBioMed import Pymolecule
2 from PyBioMed import Pyprotein
3 import pandas as pd
4 import numpy as np
5 from sklearn.ensemble import RandomForestClassifier as RF
6 from sklearn import cross_validation
7 from sklearn import metrics

```

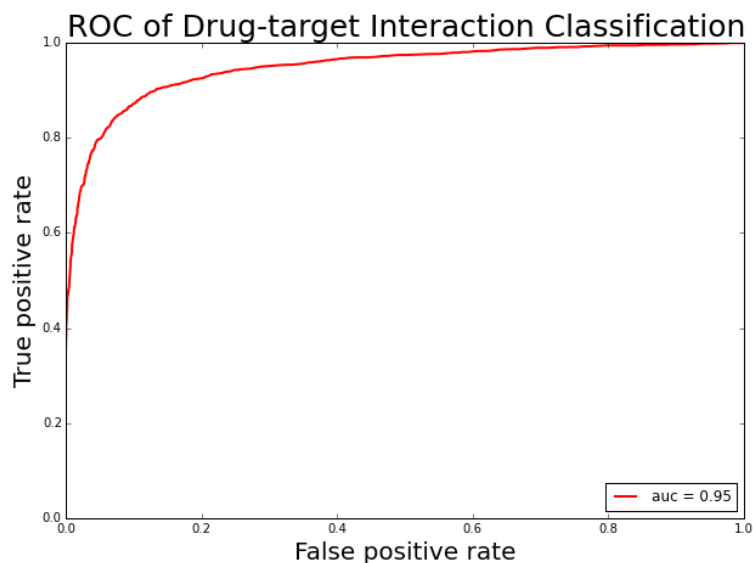


Fig. 3.4: The receiver operating characteristic curve of drug-target interaction classification.

```

8  from matplotlib import pyplot as plt
9  #=====
10 # loading the data
11 #=====
12 path = 'input PyBioMed path in your computer' #input the real path in your own path
13 smis = pd.read_excel(path + 'example/dpi/DPI_SMIs.xlsx')
14 smis.index = smis['Drug']
15 protein_seq = pd.read_table(path + 'example/dpi/hsa_seqs_all.tsv', sep = '\t')
16 protein_seq.index = protein_seq['Protein']
17
18 positive_pairs = pd.read_excel(path + 'example/dpi/Enzyme.xls')
19 positive_pairs = zip(list(positive_pairs['Protein']), list(positive_pairs['Drug']))
20
21 negative_pairs = pd.read_excel(path + 'example/dpi/Enzymedecoy.xls')
22 negative_pairs = zip(list(negative_pairs['Protein']), list(negative_pairs['Drug']))
23 #=====
24 # calculating descriptors
25 #=====
26 def calculate_pair_des(smi, seq):
27     pair_des = {}
28     drugclass = Pymolecule.PyMolecule()
29     drugclass.ReadMolFromSmile(smi)
30     pair_des.update(drugclass.GetCATS2D())
31     proclass = Pyprotein.PyProtein(seq)
32     pair_des.update(proclass.GetCTD())
33     return pair_des
34 positive_pairs_des = {}
35 for n, positive_pair in enumerate(positive_pairs):
36     try:
37         pair_des = calculate_pair_des(smis.ix[positive_pair[1]][1],protein_
38 ↪seq.ix[positive_pair[0]][1])
39         positive_pairs_des[n] = pair_des
40     except:
41         continue

```



```

42 negative_pairs_des = {}
43 for n, negative_pair in enumerate(negative_pairs):
44     try:
45         pair_des = calculate_pair_des(smis.ix[negative_pair[1]][1],protein_
↳seq.ix[negative_pair[0]][1])
46         negative_pairs_des[n] = pair_des
47     except:
48         continue
49 #=====
50 # cross-validation
51 #=====
52 x = np.array(pd.concat([pd.DataFrame(positive_pairs_des).T, pd.DataFrame(negative_
↳pairs_des).T],
53                               join_axes=[pd.DataFrame(positive_
↳pairs_des).T.columns],axis = 0, ignore_index=True))
54
55 positive_count, negative_count = len(positive_pairs_des), len(negative_pairs_des)
56 y = np.array([1]*positive_count+ [0]*negative_count)
57
58 # ROC curve of CV
59 kf = cross_validation.KFold(x.shape[0], n_folds=10, shuffle=True,random_state=5)
60 clf = RF(n_estimators=500, max_features='sqrt', n_jobs=-1, oob_score=True)
61 CV_pred_prob = []
62 CV_pred_label=[]
63 VALIDATION_index = []
64 for train_index, validation_index in kf:
65     VALIDATION_index = VALIDATION_index + list(validation_index)
66     clf.fit(x[train_index :],y[train_index])
67     pred_prob = clf.predict_proba(x[validation_index,:])
68     pred_label = clf.predict(x[validation_index,:])
69     CV_pred_prob = CV_pred_prob + list(pred_prob[:,1])
70     CV_pred_label = CV_pred_label + list(pred_label)
71 fpr_cv, tpr_cv, thresholds_cv = metrics.roc_curve(y[VALIDATION_index], CV_pred_prob)
72 y_true = y[VALIDATION_index]
73 AUC_score = metrics.roc_auc_score(y[VALIDATION_index], CV_pred_prob)
74 TPR = metrics.recall_score(y_true, CV_pred_label)
75 ACC = metrics.accuracy_score(y_true, CV_pred_label)
76 SPE = (float(positive_count)/float(negative_count)+1.0)*ACC-TPR*float(positive_count)/
↳float(negative_count)
77 matthews_corrcoef = metrics.matthews_corrcoef(y_true, CV_pred_label)
78 f1_score = metrics.f1_score(y_true, CV_pred_label)
79 #=====
80 # plotting the figure
81 #=====
82 plt.figure(figsize = (10,7))
83 plt.plot(fpr_cv, tpr_cv, 'r', label='auc = %0.2f'% AUC_score, lw=2)
84 plt.xlabel('False positive rate',{'fontsize':20});
85 plt.ylabel('True positive rate',{'fontsize':20});
86 plt.title('ROC of Drug-target Interaction Classification',{'fontsize':25})
87 plt.legend(loc="lower right",numpoints=15)
88 plt.show()

```

```

>>> print 'sensitivity:',TPR, 'specificity:', SPE, 'accuracy:', ACC, 'AUC:', AUC_
↳score, 'MCC:', matthews_corrcoef, 'F1:', f1_score
sensitivity: 0.84 specificity: 0.93 accuracy: 0.89 AUC: 0.95 MCC: 0.78 F1: 0.87

```

3.4 Application 4 Prediction of protein subcellular location

To identify the functions of proteins in organism is one of the fundamental goals in cell biology and proteomics. The function of a protein in organism is closely linked to its location in a cell. Determination of protein subcellular location (PSL) by experimental methods is expensive and time-consuming. With the enrichment of data repository, automatically prediction of PSL is an alternative method to facilitate the determination of PSL. To build a PSL prediction model, we use PyProtein in PyBioMed to calculate protein features and then the random forest (RF) method was applied to build PSL classification model. The benchmark data set for building the protein subcellular location predictor was taken from (Jia, Qian et al. 2007). The dataset contains 2568 samples, among them 849 proteins were located at Cytoplasm which is defined as positive dataset and 1619 proteins were located at Nucleus which is defined as negative dataset. For each protein, 20 amino acid composition (AAC), 147 CTD composition, transition and distribution and 30 pseudo amino acid composition (PAAC), a total number of 197 features were calculate through the PyBioMed tool.

To build the classification model, the CSV file containing the calculated descriptors was then converted to sample matrix (x_train) and a sample label vector (y_train) is also provided. Then, the python script randomforests.py based on sklearn package was employed to build the classification model (the number of trees is 500, the maximum number of features in each tree is square root of the number of features). The performance of this model was evaluated by using 10-fold cross-validation. The AUC score, accuracy, sensitivity and specificity are 0.90, 0.85, 0.94 and 0.69 respectively

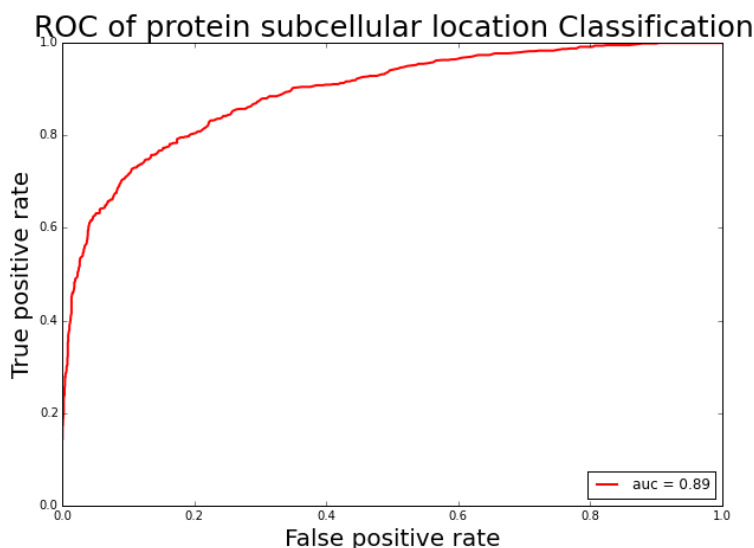


Fig. 3.5: The receiver operating characteristic curve of protein subcellular location classification.

```

1 import pandas as pd
2 from PyBioMed.PyProtein.CTD import CalculateCTD
3 import numpy as np
4 from sklearn.ensemble import RandomForestClassifier as RF
5 from sklearn import cross_validation
6 from sklearn import metrics
7 from matplotlib import pyplot as plt
8
9 #=====
10 # loading the data
11 #=====
12 path = 'input PyBioMed path in your computer' #input the PyBioMed path in your own_
13 ↪computer
14 f = open(path + 'example/subcell/Cytoplasm_seq.txt', 'r')
15 cytoplasm = [line.replace('\n', '') for line in f.readlines() if line != '\n']

```

```

14 f.close()
15 f = open(path + 'example/subcell/Nuclear_seq.txt','r')
16 nuclear = [line.replace('\n','') for line in f.readlines() if line != '\n']
17 f.close()
18 #=====
19 # calculating the descriptors
20 #=====
21 cytoplasm_des = dict(zip(range(len(cytoplasm)),map(CalculateCTD,cytoplasm)))
22 nuclear_des = dict(zip(range(len(nuclear)),map(CalculateCTD,nuclear)))
23 cytoplasm_des_df = pd.DataFrame(cytoplasm_des).T
24 nuclear_des_df = pd.DataFrame(nuclear_des).T
25 #=====
26 # cross-validation
27 #=====
28 x = np.array(pd.concat([cytoplasm_des_df, nuclear_des_df]))
29 positive_count, negative_count = len(cytoplasm_des), len(nuclear_des)
30 y = np.array([1]*positive_count+ [0]*negative_count)
31 kf = cross_validation.KFold(x.shape[0], n_folds=10, shuffle = True, random_state=5)
32 clf = RF(n_estimators=500, max_features='sqrt', n_jobs=-1, oob_score=True)
33 CV_pred_prob = []
34 CV_pred_label=[]
35 VALIDATION_index = []
36 kf = cross_validation.KFold(x.shape[0], n_folds=10, shuffle=True,random_state=5)
37 clf = RF(n_estimators=500, max_features='sqrt', n_jobs=-1, oob_score=True)
38 CV_pred_prob = []
39 CV_pred_label=[]
40 VALIDATION_index = []
41 for train_index, validation_index in kf:
42     VALIDATION_index = VALIDATION_index + list(validation_index)
43     clf.fit(x[train_index :],y[train_index])
44     pred_prob = clf.predict_proba(x[validation_index,:])
45     pred_label = clf.predict(x[validation_index,:])
46     CV_pred_prob = CV_pred_prob + list(pred_prob[:,1])
47     CV_pred_label = CV_pred_label + list(pred_label)
48 fpr_cv, tpr_cv, thresholds_cv = metrics.roc_curve(y[VALIDATION_index], CV_pred_prob)
49 y_true = y[VALIDATION_index]
50 AUC_score = metrics.roc_auc_score(y[VALIDATION_index], CV_pred_prob)
51 TPR = metrics.recall_score(y_true, CV_pred_label)
52 ACC = metrics.accuracy_score(y_true, CV_pred_label)
53 SPE = (float(positive_count)/float(negative_count)+1.0)*ACC-TPR*float(positive_count)/
54     ↳float(negative_count)
55 matthews_corrcoef = metrics.matthews_corrcoef(y_true, CV_pred_label)
56 f1_score = metrics.f1_score(y_true, CV_pred_label)
57 #=====
58 # plotting the figure
59 #=====
60 plt.figure(figsize = (10,7))
61 plt.plot(fpr_cv, tpr_cv, 'r', label='auc = %0.2f'% AUC_score, lw=2)
62 plt.xlabel('False positive rate',{'fontsize':20});
63 plt.ylabel('True positive rate',{'fontsize':20});
64 plt.title('ROC of protein subcellular location Classification',{'fontsize':25})
65 plt.legend(loc="lower right",numpoints=15)
66 plt.show()

```

```

>>> print 'sensitivity:',TPR, 'specificity:', SPE, 'accuracy:', ACC, 'AUC:', AUC_
↳score, 'MACCS:', matthews_corrcoef, 'F1:', f1_score
sensitivity: 0.67 specificity: 0.92 accuracy: 0.84 AUC: 0.89 MACCS: 0.62 F1: 0.74

```

3.5 Application 5 Predicting nucleosome positioning in genomes with dinucleotide-based auto covariance

Nucleosome positioning participates in many cellular activities and plays significant roles in regulating cellular processes (Guo, et al., 2014). Computational methods that can predict nucleosome positioning based on the DNA sequences is highly desired. Here, a computational predictor was constructed by using dinucleotide-based auto covariance and SVMs, and its performance was evaluated by 10-fold cross-validation. The benchmark data set for the *H. sapiens* was taken from (Schones, et al., 2008). Since the *H. sapiens* genome and its nucleosome map contain a huge amount of data, according to Liu's strategy (Liu, et al., 2011) the nucleosome-forming sequence samples (positive data) and the linkers or nucleosome-inhibiting sequence samples (negative data) were extracted from chromosome (Guo, et al., 2014). A file named "H_sapiens_pos.fasta" containing 2,273 nucleosome-forming DNA segments is used as the positive dataset, and a file named "H_sapiens_neg.fasta" containing 2,300 nucleosome-inhibiting DNA segments is used as the negative dataset.

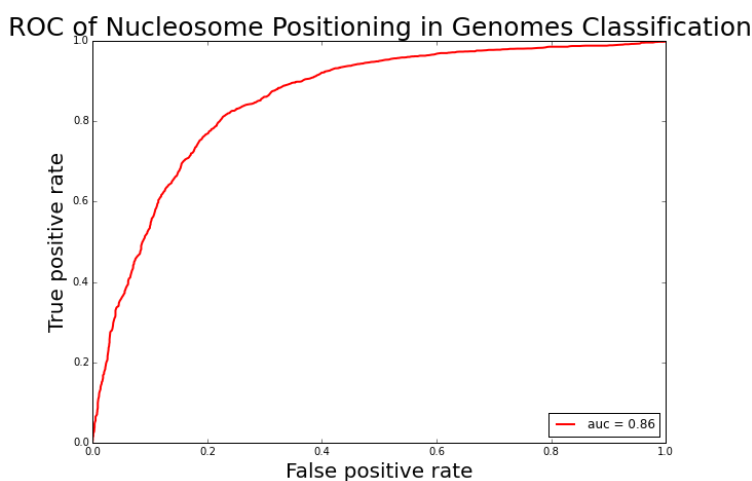


Fig. 3.6: The receiver operating characteristic curve of nucleosome positioning in genomes classification.

```

1 import pandas as pd
2 from PyBioMed import Pydna
3 from PyBioMed.PyGetMol import GetDNA
4 import numpy as np
5 from sklearn.ensemble import RandomForestClassifier as RF
6 from sklearn import cross_validation
7 from sklearn import metrics
8 from matplotlib import pyplot as plt
9
10 # loading data
11 #=====
12 path = 'PyBioMed package real path in your computer' # input the real path in your_
13 ↪own computer
14 seqs_pos = GetDNA.ReadFasta(open(path + '/example/dna/H_sapiens_pos.fasta'))
15 seqs_neg = GetDNA.ReadFasta(open(path + '/example/dna/H_sapiens_neg.fasta'))
16 #=====
17 # calculating descriptors
18 #=====
19 def calculate_des(seq):
20     des = []
21     dnaclass = Pydna.PyDNA(seq)
22     des.extend(dnaclass.GetDAC(all_property=True).values())

```

```

22         des.extend(dnaclass.GetPseDNC(all_property=True,lamada=2, w=0.05).values())
23         des.extend(dnaclass.GetPseKNC(all_property=True,lamada=2, w=0.05).values())
24         des.extend(dnaclass.GetSCPseDNC(all_property=True).values())
25     return des
26 pos_des = []
27 for seq_pos in seqs_pos:
28     pos_des.append(calculate_des(seq_pos))
29 neg_des = []
30 for seq_neg in seqs_neg:
31     neg_des.append(calculate_des(seq_neg))
32 #=====
33 # cross validation
34 #=====
35 x = np.array(pos_des+neg_des)
36 positive_count, negative_count = len(pos_des), len(neg_des)
37 y = np.array([1]*positive_count+ [0]*negative_count)
38 kf = cross_validation.KFold(x.shape[0], n_folds=10, random_state=0)
39 clf = RF(n_estimators=500, max_features='sqrt', n_jobs=-1, oob_score=True)
40 CV_pred_prob = []
41 CV_pred_label=[]
42 VALIDATION_index = []
43 for train_index, validation_index in kf:
44     VALIDATION_index = VALIDATION_index + list(validation_index)
45     clf.fit(x[train_index :],y[train_index])
46     pred_prob = clf.predict_proba(x[validation_index,:])
47     pred_label = clf.predict(x[validation_index,:])
48     CV_pred_prob = CV_pred_prob + list(pred_prob[:,1])
49     CV_pred_label = CV_pred_label + list(pred_label)
50 fpr_cv, tpr_cv, thresholds_cv = metrics.roc_curve(y[VALIDATION_index], CV_pred_prob)
51 # Calculate auc score of cv
52 y_true = y[VALIDATION_index]
53 AUC_score = metrics.roc_auc_score(y[VALIDATION_index], CV_pred_prob)
54 TPR = metrics.recall_score(y_true, CV_pred_label)
55 ACC = metrics.accuracy_score(y_true, CV_pred_label)
56 SPE = (float(positive_count)/float(negative_count)+1.0)*ACC-TPR*float(positive_count)/
57     ↳float(negative_count)
58 matthews_corrcoef = metrics.matthews_corrcoef(y_true, CV_pred_label)
59 fl_score = metrics.f1_score(y_true, CV_pred_label)
60 #=====
61 # plotting the figure
62 #=====
63 plt.figure(figsize = (10,7))
64 plt.plot(fpr_cv, tpr_cv, 'r', label='auc = %0.2f'% AUC_score, lw=2)
65 plt.xlabel('False positive rate',{'fontsize':20});
66 plt.ylabel('True positive rate',{'fontsize':20});
67 plt.title('ROC of Nucleosome Positioning in Genomes Classification',{'fontsize':25})
68 plt.legend(loc="lower right",numpoints=15)
69 plt.show()

```

```

>>> print 'sensitivity:',TPR, 'specificity:', SPE, 'accuracy:', ACC, 'AUC:', AUC_
↳score, 'MACCS:', matthews_corrcoef, 'F1:', fl_score
sensitivity: 0.82 specificity: 0.80 accuracy: 0.81 AUC: 0.88 MACCS: 0.62 F1: 0.81

```

3.5. Application 5 Predicting nucleosome positioning in genomes with dinucleotide-based auto 31 covariance

PYBIOMED API

4.1 PyDNA

4.1.1 PyDNAac module

Created on Thu May 19 01:05:29 2016

@author: yzj

`PyDNAac.CheckAcc(lag, k)`

`PyDNAac.GetDAC(input_data, **kwargs)`
Make DAC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – bool, choose all physicochemical properties or not.
- **extra_phyche_index** – dict, the key is the dinucleotide (string), and its corresponding value is a list. It means user-defined phyche_index.

`PyDNAac.GetDACC(input_data, **kwargs)`
Make DACC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – bool, choose all physicochemical properties or not.
- **extra_phyche_index** – dict, the key is the dinucleotide (string), and its corresponding value is a list. It means user-defined phyche_index.

`PyDNAac.GetDCC(input_data, **kwargs)`
Make DCC vector.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – bool, choose all physicochemical properties or not.

- **extra_phyche_index** – dict, the key is the dinucleotide (string), and its corresponding value is a list. It means user-defined phyche_index.

`PyDNAac.GetTAC(input_data, **kwargs)`

Make TAC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – bool, choose all physicochemical properties or not.
- **extra_phyche_index** – dict, the key is the dinucleotide (string), and its corresponding value is a list. It means user-defined phyche_index.

`PyDNAac.GetTACC(input_data, **kwargs)`

Make TACC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – bool, choose all physicochemical properties or not.
- **extra_phyche_index** – dict, the key is the dinucleotide (string), and its corresponding value is a list. It means user-defined phyche_index.

`PyDNAac.GetTCC(input_data, **kwargs)`

Make TCC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – bool, choose all physicochemical properties or not.
- **extra_phyche_index** – dict, the key is the dinucleotide (string), and its corresponding value is a list. It means user-defined phyche_index.

`PyDNAac.ReadyAcc(input_data, k, phyche_index=None, all_property=False, extra_phyche_index=None)`

4.1.2 PyDNAacutil module

Created on Sun May 22 14:40:13 2016

@author: yzj

`PyDNAacutil.ExtendPhycheIndex(original_index, extend_index)`

Extend {phyche:[value, ...]}

`PyDNAacutil.MakeACVector(sequence_list, lag, phyche_value, k)`

`PyDNAacutil.MakeCCVector(sequence_list, lag, phyche_value, k)`

4.1.3 PyDNAnac module

Created on Sun May 22 14:45:46 2016

@author: yzj

`PyDNAnac.CheckNacPara(k, normalize=False, upto=False, alphabet='ACGT')`

`PyDNAnac.GetIdKmer(data, hs, non_hs, **kwargs)`

Make IDKmer vector.

Parameters

- **data** – Need to processed FASTA file.
- **hs** – Positive FASTA file.
- **non_hs** – Negative FASTA file.
- **k** – int, the k value of kmer, it should be larger than 0.
- **upto** – bool, whether to generate 1-kmer, 2-kmer, ..., k-mer.
- **alphabet** – string.

`PyDNAnac.GetKmer(data, **kwargs)`

Make a kmer dictionary with options k, upto, revcomp, normalize.

Parameters

- **k** – int, the k value of kmer, it should be larger than 0.
- **normalize** – bool, normalize the result vector or not.
- **upto** – bool, whether to generate 1-kmer, 2-kmer, ..., k-mer.
- **alphabet** – string.
- **data** – file object or sequence list.

Returns kmer vector.

`PyDNAnac.GetKmerList(k, upto, alphabet)`

Get the kmer list.

Parameters

- **k** – int, the k value of kmer, it should be larger than 0.
- **upto** – bool, whether to generate 1-kmer, 2-kmer, ..., k-mer.
- **alphabet** – string.

`PyDNAnac.GetRevckmer(data, **kwargs)`

Make a reverse compliment kmer dictionary with options k, upto, normalize.

Parameters **data** – file object or sequence list.

Returns reverse compliment kmer vector.

4.1.4 PyDNAnacutil module

Created on Sun May 22 14:47:55 2016

@author: yzj

`PyDNAnacutil.ComputeBinNum(num_bins, position, k, numbers)`

`PyDNAnacutil.ComputeQuantileBoundaries` (*num_bins, k_values, number_filename*)

`PyDNAnacutil.Diversity` (*vec*)

Calculate diversity.

Parameters *vec* – kmer vec

Returns Diversity(*X*)

`PyDNAnacutil.FindRevcomp` (*sequence, revcomp_dictionary*)

`PyDNAnacutil.Frequency` (*tol_str, tar_str*)

Generate the frequency of *tar_str* in *tol_str*.

Parameters

- **tol_str** – mother string.
- **tar_str** – substring.

`PyDNAnacutil.IdXS` (*vec_x, vec_s, diversity_s*)

Calculate ID(*X*, *S*)

Parameters

- **vec_x** – kmer *X*
- **vec_s** – kmer *S*

Returns ID(*X*, *S*) = Diversity(*X* + *S*) - Diversity(*X*) - Diversity(*S*)

`PyDNAnacutil.MakeIndex` (*k*)

`PyDNAnacutil.MakeIndexUptoK` (*k*)

`PyDNAnacutil.MakeIndexUptoKRevcomp` (*k*)

`PyDNAnacutil.MakeKmerList` (*k, alphabet*)

`PyDNAnacutil.MakeKmerVector` (*seq_list, kmer_list, rev_kmer_list, k, upto, revcomp, normalize*)

`PyDNAnacutil.MakeRevcompKmerList` (*kmer_list*)

`PyDNAnacutil.MakeSequenceVector` (*sequence, numbers, num_bins, revcomp, revcomp_dictionary, normalize_method, k_values, mismatch, alphabet, kmer_list, boundaries, pseudocount*)

`PyDNAnacutil.MakeUptoKmerList` (*k_values, alphabet*)

`PyDNAnacutil.NormalizeVector` (*normalize_method, k_values, vector, kmer_list*)

`PyDNAnacutil.ReadFastaSequence` (*numeric, fasta_file*)

`PyDNAnacutil.ReadSequenceAndNumbers` (*fasta_file, numbers_filename, numbers_file*)

`PyDNAnacutil.Substitute` (*position, letter, string*)

`PyDNAnacutil.cmp` (*a, b*)

4.1.5 PyDNAPsenac module

Created on Thu Jun 02 09:38:15 2016

@author: yzj

`PyDNAPsenac.CheckPsenac` (*lamada, w, k*)

Check the validation of parameter *lamada*, *w* and *k*.

`PyDNApsenac.GetPCPseDNC(input_data, **kwargs)`

Make a PCPseDNC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – choose all physicochemical properties or not.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).

It means the user-defined physicochemical indices.

`PyDNApsenac.GetPCPseTNC(input_data, **kwargs)`

Make a PCPseDNC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – choose all physicochemical properties or not.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).

It means the user-defined physicochemical indices.

`PyDNApsenac.GetPseDNC(input_data, **kwargs)`

Make PseDNC dictionary.

Parameters

- **input_data** – file type or handle.
- **k** – k-tuple.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).

It means the user-defined physicochemical indices.

`PyDNApsenac.GetPseKNC(input_data, **kwargs)`

Make PseKNC dictionary.

Parameters

- **input_data** – file type or handle.
- **k** – k-tuple.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).

It means the user-defined physicochemical indices.

`PyDNApsenac.GetSCPseDNC(input_data, **kwargs)`

Make a SCPseDNC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – choose all physicochemical properties or not.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).
It means the user-defined physicochemical indices.

`PyDNApsenac.GetSCPseTNC(input_data, **kwargs)`

Make a SCPseTNC dictionary.

Parameters

- **input_data** – file object or sequence list.
- **phyche_index** – physicochemical properties list.
- **all_property** – choose all physicochemical properties or not.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).
It means the user-defined physicochemical indices.

`PyDNApsenac.GetSequenceListAndPhycheValue(input_data, k, phyche_index, extra_phyche_index, all_property)`

For PseKNC-general make sequence_list and phyche_value.

Parameters

- **input_data** – file type or handle.
- **k** – int, the value of k-tuple.
- **k** – physicochemical properties list.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).
It means the user-defined physicochemical indices.
- **all_property** – bool, choose all physicochemical properties or not.

`PyDNApsenac.GetSequenceListAndPhycheValuePsednc(input_data, extra_phyche_index=None)`

For PseDNC, PseKNC, make sequence_list and phyche_value.

Parameters

- **input_data** – file type or handle.
- **extra_phyche_index** –
dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).

It means the user-defined physicochemical indices.

`PyDNApsenac.GetSequenceListAndPhycheValuePseknc` (*input_data*, *extra_phyche_index=None*) *ex-*

For PseDNC, PseKNC, make *sequence_list* and *phyche_value*.

Parameters

- **input_data** – file type or handle.
- **extra_phyche_index** – dict, the key is the dinucleotide (string), the value is its physicochemical property value (list).

It means the user-defined physicochemical indices.

4.1.6 PyDNApsenacutil module

Created on Thu Jun 02 10:00:35 2016

@author: yzj

`PyDNApsenacutil.ExtendPhycheIndex` (*original_index*, *extend_index*)
Extend {phyche:[value, ...]}

`PyDNApsenacutil.GetParallelFactor` (*k*, *lamada*, *sequence*, *phyche_value*)
Get the corresponding factor theta list.

`PyDNApsenacutil.GetParallelFactorPsednc` (*lamada*, *sequence*, *phyche_value*)
Get the corresponding factor theta list. This def is just for dinucleotide.

`PyDNApsenacutil.GetPhycheFactorDic` (*k*)
Get all {nucleotide: [(phyche, value), ...]} dict.

`PyDNApsenacutil.GetPhycheIndex` (*k*, *phyche_list*)
get phyche_value according phyche_list.

`PyDNApsenacutil.GetSeriesFactor` (*k*, *lamada*, *sequence*, *phyche_value*)
Get the corresponding series factor theta list.

`PyDNApsenacutil.MakeOldPsekncVector` (*sequence_list*, *lamada*, *w*, *k*, *phyche_value*, *theta_type=1*)
Generate the psekcnc vector.

`PyDNApsenacutil.MakePsekncVector` (*sequence_list*, *lamada*, *w*, *k*, *phyche_value*, *theta_type=1*)
Generate the psekcnc vector.

`PyDNApsenacutil.ParallelCorFunction` (*nucleotide1*, *nucleotide2*, *phyche_index*)
Get the cFactor.(Type1)

`PyDNApsenacutil.SeriesCorFunction` (*nucleotide1*, *nucleotide2*, *big_lamada*, *phyche_value*)
Get the series correlation Factor(Type 2).

4.1.7 PyDNAutil module

Created on Wed May 18 14:06:37 2016

@author: yzj

`PyDNAutil.ALPHABET = 'ACGT'`
Used for process original data.

`PyDNAUtil.ConvertPhycheIndexToDict (phyche_index)`

`PyDNAUtil.DNAChecks (s)`

`PyDNAUtil.Frequency (tol_str, tar_str)`

Generate the frequency of tar_str in tol_str.

Parameters

- **tol_str** – mother string.
- **tar_str** – substring.

`PyDNAUtil.GeneratePhycheValue (k, phyche_index=None, all_property=False, extra_phyche_index=None)`

Combine the user selected phyche_list, is_all_property and extra_phyche_index to a new standard phyche_value. #####

`PyDNAUtil.GetData (input_data, desc=False)`

Get sequence data from file or list with check.

Parameters

- **input_data** – type file or list
- **desc** – with this option, the return value will be a Seq object list(it only works in file object).

Returns sequence data or shutdown.

`PyDNAUtil.GetSequenceCheckDna (f)`

Read the fasta file.

Input: f: HANDLE to input. e.g. sys.stdin, or open(<file>)

`PyDNAUtil.IsFasta (seq)`

Judge the Seq object is in FASTA format. Two situation: 1. No seq name. 2. Seq name is illegal. 3. No sequence.

Parameters **seq** – Seq object.

`PyDNAUtil.IsSequenceList (sequence_list)`

Judge the sequence list is within the scope of alphabet and change the lowercase to capital.

#####

`PyDNAUtil.IsUnderAlphabet (s, alphabet)`

Judge the string is within the scope of the alphabet or not.

Parameters

- **s** – The string.
- **alphabet** – alphabet.

`PyDNAUtil.NormalizeIndex (phyche_index, is_convert_dict=False)`

`PyDNAUtil.ReadFasta (f)`

Read a fasta file.

Parameters **f** – HANDLE to input. e.g. sys.stdin, or open(<file>)

`PyDNAUtil.ReadFastaCheckDna (f)`

Read the fasta file, and check its legality.

Parameters **f** – HANDLE to input. e.g. sys.stdin, or open(<file>)

`PyDNAUtil.ReadFastaYield(f)`

Yields a Seq object.

Parameters `f` – HANDLE to input. e.g. `sys.stdin`, or `open(<file>)`

`class PyDNAUtil.Seq(name, seq, no)`

`PyDNAUtil.StandardDeviation(value_list)`

`PyDNAUtil.WriteLibsvm(vector_list, label_list, write_file)`

4.2 PyMolecule

4.2.1 AtomProperty module

You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

Z: atomic number L: principal quantum number Zv: number of valence electrons Rv: van der Waals atomic radius Rc: covalent radius m: atomic mass V: van der Waals volume En: Sanderson electronegativity alpha: atomic polarizability (10e-24 cm³) IP: ionization potential (eV) EA: electron affinity (eV)

`AtomProperty.GetAbsoluteAtomicProperty(element='C', propertyname='m')`

Get the absolute property value with propertyname for the given atom.

`AtomProperty.GetRelativeAtomicProperty(element='C', propertyname='m')`

Get the absolute property value with propertyname for the given atom.

4.2.2 AtomTypes module

You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

contains SMARTS definitions and calculators for EState atom types

defined in: Hall and Kier JCICS _35_ 1039-1045 (1995) Table 1

`AtomTypes.BuildPatts(rawV=None)`

Internal Use Only

`AtomTypes.GetAtomLabel(mol)`

Obtain the atom index in a molecule for the above given atom types

`AtomTypes.TypeAtoms(mol)`

assigns each atom in a molecule to an EState type

Returns:

list of tuples (atoms can possibly match multiple patterns) with atom types

4.2.3 basak module

topological structure. You can get 21 molecular connectivity descriptors.

You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.

`basak.CalculateBasakCIC0 (mol)`

Obtain the complementary information content with order 0

proposed by Basak

`basak.CalculateBasakCIC1 (mol)`

Obtain the complementary information content with order 1 proposed

by Basak.

`basak.CalculateBasakCIC2 (mol)`

Obtain the complementary information content with order 2 proposed

by Basak.

`basak.CalculateBasakCIC3 (mol)`

Obtain the complementary information content with order 3 proposed

by Basak.

`basak.CalculateBasakCIC4 (mol)`

Obtain the complementary information content with order 4 proposed

by Basak.

`basak.CalculateBasakCIC5 (mol)`

Obtain the complementary information content with order 5 proposed

by Basak.

`basak.CalculateBasakCIC6 (mol)`

Obtain the complementary information content with order 6 proposed

by Basak.

`basak.CalculateBasakIC0 (mol)`

Obtain the information content with order 0 proposed by Basak

`basak.CalculateBasakIC1 (mol)`

Obtain the information content with order 1 proposed by Basak

`basak.CalculateBasakIC2 (mol)`

Obtain the information content with order 2 proposed by Basak

`basak.CalculateBasakIC3 (mol)`

Obtain the information content with order 3 proposed by Basak

`basak.CalculateBasakIC4 (mol)`
Obtain the information content with order 4 proposed by Basak

`basak.CalculateBasakIC5 (mol)`
Obtain the information content with order 5 proposed by Basak

`basak.CalculateBasakIC6 (mol)`
Obtain the information content with order 6 proposed by Basak

`basak.CalculateBasakSIC0 (mol)`
Obtain the structural information content with order 0
proposed by Basak

`basak.CalculateBasakSIC1 (mol)`
Obtain the structural information content with order 1
proposed by Basak.

`basak.CalculateBasakSIC2 (mol)`
Obtain the structural information content with order 2 proposed
by Basak.

`basak.CalculateBasakSIC3 (mol)`
Obtain the structural information content with order 3 proposed
by Basak.

`basak.CalculateBasakSIC4 (mol)`
Obtain the structural information content with order 4 proposed
by Basak.

`basak.CalculateBasakSIC5 (mol)`
Obtain the structural information content with order 5 proposed
by Basak.

`basak.CalculateBasakSIC6 (mol)`
Obtain the structural information content with order 6 proposed
by Basak.

`basak.Getbasak (mol)`

4.2.4 bcut module

The calculation of Burden eigenvalue descriptors. You can get 64 molecular descriptors. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`bcut.CalculateBurdenElectronegativity (mol)`
Calculate Burden descriptors based on atomic electronegativity.

`bcut.CalculateBurdenMass (mol)`
Calculate Burden descriptors based on atomic mass.

`bcut.CalculateBurdenPolarizability (mol)`
Calculate Burden descriptors based on polarizability.

`bcut.CalculateBurdenVDW (mol)`
Calculate Burden descriptors based on atomic volumes

`bcut.GetBurden (mol)`
Calculate all 64 Burden descriptors

4.2.5 cats2d module

CATS2D Potential Pharmacophore Point (PPP) definitions as describes in # Pharmacophores and Pharmacophore Searches 2006 (Eds. T. Langer and R.D. Hoffmann), Chapter 3: # Alignment-free Pharmacophore Patterns - A Correlation-vector Approach. # The last lipophilic pattern on page 55 of the book is realized as a graph search and not # as a SMARTS search. Therefore, the list contains only two lipophilic SMARTS patterns. # The format is tab separated and contains in the first column the PPP type (D = H-bond donor, # A = H-bond acceptor, P = positive, N = negative, L = lipophilic). The second column of each entry # contains the SMARTS pattern(s). The last entry is a description of the molecular feature

D [OH] Oxygen atom of an OH group D [#7H,#7H2] Nitrogen atom of an NH or NH2 group A [O] Oxygen atom A [#7H0] Nitrogen atom not adjacent to a hydrogen atom P [+] *atom with a positive charge* P [#7H2] Nitrogen atom of an NH2 group N [-] Atom with a negative charge N [C&D2&\$(C(=O)O),P&D2&\$(P(=O)O),S&D2&\$(S(=O)O)] Carbon, sulfur or phosphorus atom of a COOH, SOOH or POOH group. This pattern is realized by an graph algorithm L [Cl,Br,I] Chlorine, bromine, or iodine atom L [S;D2;\$(S(C)(C))] Sulfur atom adjacent to exactly two carbon atoms

Created on Thu Sep 1 20:13:38 2016

Authors: Zhijiang Yao and Dongsheng Cao.

Email: gadsby@163.com and oriental-cds@163.com

`cats2d.AssignAtomType (mol)`
Assign the atoms in the mol object into each of the PPP type according to PPP list definition.

`cats2d.CATS2D (mol, PathLength=10, scale=3)`
The main program for calculating the CATS descriptors.

CATS: chemically advanced template search

—> CATS_DA0

Usage:

result=CATS2D(mol,PathLength = 10,scale = 1)

Input: mol is a molecule object.

PathLength is the max topological distance between two atoms.

scale is the normalization method (descriptor scaling method)

scale = 1 indicates that no normalization. That is to say: the

values of the vector represent raw counts (“counts”).

scale = 2 indicates that division by the number of non-hydrogen

atoms (heavy atoms) in the molecule.

scale = 3 indicates that division of each of 15 possible PPP pairs

by the added occurrences of the two respective PPPs.

Output: result is a dict format with the definitions of each descriptor.

`cats2d.CConstructLFromGraphSearch (mol)`

The last lipophilic pattern on page 55 of the book is realized as a graph search and not as a SMARTS search.

“L” carbon atom adjacent only to carbon atoms.

`cats2d.FormCATSDict (AtomDict, CATSLabel)`

Construct the CATS dict.

`cats2d.FormCATSLabel (PathLength=10)`

Construct the CATS label such as AA0, AA1,...,AP3,.....

The result is a list format.

A acceptor; P positive; N negative; L lipophilic; D donor; #####

`cats2d.MatchAtomType (IndexList, AtomTypeDict)`

Mapping two atoms with a certain distance into their atom types

such as AA,AL, DP,LD etc.

4.2.6 charge module

The calculation of Charge descriptors based on Gasteiger/Marseli partial charges(25). You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`charge.CalculateAllMaxNCharge (mol)`

Most negative charge on all atoms

→Qmin

Usage:

result=CalculateAllMaxNCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateAllMaxPCharge (mol)`

Most positive charge on ALL atoms

→Qmax

Usage:

result=CalculateAllMaxPCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateAllSumSquareCharge** (*mol*)

The sum of square charges on all atoms

->Qass

Usage:

result=CalculateAllSumSquareCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateCMaxNCharge** (*mol*)

Most negative charge on C atoms

->QCmin

Usage:

result=CalculateCMaxNCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateCMaxPCharge** (*mol*)

Most positive charge on C atoms

->QCmax

Usage:

result=CalculateCMaxPCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateCSumSquareCharge** (*mol*)

The sum of square charges on all C atoms

->QCss

Usage:

result=CalculateCSumSquareCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateHMaxNCharge** (*mol*)

Most negative charge on H atoms

->QHmin

Usage:

result=CalculateHMaxNCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateHMaxPCharge (mol)`

Most positive charge on H atoms

→QHmax

Usage:

result=CalculateHMaxPCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateHSumSquareCharge (mol)`

The sum of square charges on all H atoms

→QHss

Usage:

result=CalculateHSumSquareCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateLocalDipoleIndex (mol)`

Calculation of local dipole index (D)

→LDI

Usage:

result=CalculateLocalDipoleIndex(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateMeanAbsoulteCharge (mol)`

The average absolute charge

→Mac

Usage:

result=CalculateMeanAbsoulteCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateMeanNCharge (mol)`

The average negative charge

→Mnc

Usage:

result=CalculateMeanNCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateMeanPCharge (mol)`

The average postive charge

→Mpc

Usage:

result=CalculateMeanPCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateNMaxNCharge** (*mol*)

Most negative charge on N atoms

->QNmin

Usage:

result=CalculateNMaxNCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateNMaxPCharge** (*mol*)

Most positive charge on N atoms

->QNmax

Usage:

result=CalculateNMaxPCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateNSumSquareCharge** (*mol*)

The sum of square charges on all N atoms

->QNss

Usage:

result=CalculateNSumSquareCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateOMaxNCharge** (*mol*)

Most negative charge on O atoms

->QOmin

Usage:

result=CalculateOMaxNCharge(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

charge.**CalculateOMaxPCharge** (*mol*)

Most positive charge on O atoms

->QOmax

Usage:

```
result=CalculateOMaxPCharge(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
charge.CalculateOSumSquareCharge (mol)
```

The sum of square charges on all O atoms

->QOss

Usage:

```
result=CalculateOSumSquareCharge(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
charge.CalculateRelativeNCharge (mol)
```

The partial charge of the most negative atom divided
by the total negative charge.

->Rnc

Usage:

```
result=CalculateRelativeNCharge(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
charge.CalculateRelativePCharge (mol)
```

The partial charge of the most positive atom divided by
the total positive charge.

->Rpc

Usage:

```
result=CalculateRelativePCharge(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
charge.CalculateSubmolPolarityPara (mol)
```

Calculation of submolecular polarity parameter(SPP)

->SPP

Usage:

```
result=CalculateSubmolPolarityPara(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
charge.CalculateTotalAbsoulteCharge (mol)
```

The total absolute charge

->Tac

Usage:

```
result=CalculateTotalAbsoulteCharge(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateTotalNCharge (mol)`

The total negative charge

→Tnc

Usage:

```
result=CalculateTotalNCharge(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.CalculateTotalPCharge (mol)`

The total postive charge

→Tpc

Usage:

```
result=CalculateTotalPCharge(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`charge.GetCharge (mol)`

Get the dictionary of constitutional descriptors for given moelcule mol

Usage:

```
result=GetCharge(mol)
```

Input: mol is a molecule object.

Output: result is a dict form containing all charge descriptors.

4.2.7 connectivity module

structure. You can get 44 molecular connectivity descriptors. You can freely use and distribute it. If you hava any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`connectivity.CalculateChi0 (mol)`

Calculation of molecular connectivity chi index for path order 0

→Chi0

Usage:


```
result=CalculateChi0(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi1** (*mol*)

Calculation of molecular connectivity chi index for path order 1

(i.e.,Radich)

—>Chi1

Usage:

```
result=CalculateChi1(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi10p** (*mol*)

Calculation of molecular connectivity chi index for path order 10

—>Chi10

Usage:

```
result=CalculateChi10p(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi2** (*mol*)

Calculation of molecular connectivity chi index for path order 2

—>Chi2

Usage:

```
result=CalculateChi2(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi3c** (*mol*)

Calculation of molecular connectivity chi index for cluster

—>Chi3c

Usage:

```
result=CalculateChi3c(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi3ch** (*mol*)

Calculation of molecular connectivity chi index for cycles of 3

—>Chi3ch

Usage:

```
result=CalculateChi3ch(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi3p** (*mol*)

Calculation of molecular connectivity chi index for path order 3

—>Chi3

Usage:

```
result=CalculateChi3p(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi4c** (*mol*)

Calculation of molecular connectivity chi index for cluster

—>Chi4c

Usage:

```
result=CalculateChi4c(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi4ch** (*mol*)

Calculation of molecular connectivity chi index for cycles of 4

—>Chi4ch

Usage:

```
result=CalculateChi4ch(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi4p** (*mol*)

Calculation of molecular connectivity chi index for path order 4

—>Chi4

Usage:

```
result=CalculateChi4p(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi4pc** (*mol*)

Calculation of molecular connectivity chi index for path/cluster

—>Chi4pc

Usage:

```
result=CalculateChi4pc(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateChi5ch` (*mol*)

Calculation of molecular connectivity chi index for cycles of 5

—>Chi5ch

Usage:

result=CalculateChi5ch(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateChi5p` (*mol*)

Calculation of molecular connectivity chi index for path order 5

—>Chi5

Usage:

result=CalculateChi5p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateChi6ch` (*mol*)

Calculation of molecular connectivity chi index for cycles of 6

—>Chi6ch

Usage:

result=CalculateChi6ch(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateChi6p` (*mol*)

Calculation of molecular connectivity chi index for path order 6

—>Chi6

Usage:

result=CalculateChi6p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateChi7p` (*mol*)

Calculation of molecular connectivity chi index for path order 7

—>Chi7

Usage:

result=CalculateChi7p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateChi8p` (*mol*)

Calculation of molecular connectivity chi index for path order 8

—>Chi8

Usage:

result=CalculateChi8p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChi9p** (*mol*)

Calculation of molecular connectivity chi index for path order 9

—>Chi9

Usage:

result=CalculateChi9p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv0** (*mol*)

Calculation of valence molecular connectivity chi index for

path order 0

—>Chiv0

Usage:

result=CalculateChiv0(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv1** (*mol*)

Calculation of valence molecular connectivity chi index for

path order 1

—>Chiv1

Usage:

result=CalculateChiv1(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv10p** (*mol*)

Calculation of valence molecular connectivity chi index for

path order 10

—>Chiv10

Usage:

result=CalculateChiv10p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv2** (*mol*)

Calculation of valence molecular connectivity chi index for

path order 2

—>Chiv2

Usage:

result=CalculateChiv2(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv3c** (*mol*)

Calculation of valence molecular connectivity chi index for cluster

—>Chiv3c

Usage:

result=CalculateChiv3c(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv3ch** (*mol*)

Calculation of valence molecular connectivity chi index

for cycles of 3

—>Chiv3ch

Usage:

result=CalculateChiv3ch(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv3p** (*mol*)

Calculation of valence molecular connectivity chi index for

path order 3

—>Chiv3

Usage:

result=CalculateChiv3p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv4c** (*mol*)

Calculation of valence molecular connectivity chi index for cluster

—>Chiv4c

Usage:

result=CalculateChiv4c(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv4ch** (*mol*)

Calculation of valence molecular connectivity chi index for

cycles of 4

—>Chiv4ch

Usage:

result=CalculateChiv4ch(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv4p**(mol)

Calculation of valence molecular connectivity chi index for
path order 4

—>Chiv4

Usage:

result=CalculateChiv4p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv4pc**(mol)

Calculation of valence molecular connectivity chi index for
path/cluster

—>Chiv4pc

Usage:

result=CalculateChiv4pc(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv5ch**(mol)

Calculation of valence molecular connectivity chi index for
cycles of 5

—>Chiv5ch

Usage:

result=CalculateChiv5ch(mol)

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateChiv5p**(mol)

Calculation of valence molecular connectivity chi index for
path order 5

—>Chiv5

Usage:

result=CalculateChiv5p(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateChiv6ch(mol)`
Calculation of valence molecular connectivity chi index for
cycles of 6
—>Chiv6ch

Usage:

result=CalculateChiv6ch(mol)
Input: mol is a molecule object.
Output: result is a numeric value

`connectivity.CalculateChiv6p(mol)`
Calculation of valence molecular connectivity chi index for
path order 6
—>Chiv6

Usage:

result=CalculateChiv6p(mol)
Input: mol is a molecule object.
Output: result is a numeric value

`connectivity.CalculateChiv7p(mol)`
Calculation of valence molecular connectivity chi index for
path order 7
—>Chiv7

Usage:

result=CalculateChiv7p(mol)
Input: mol is a molecule object.
Output: result is a numeric value

`connectivity.CalculateChiv8p(mol)`
Calculation of valence molecular connectivity chi index for
path order 8
—>Chiv8

Usage:

result=CalculateChiv8p(mol)
Input: mol is a molecule object.
Output: result is a numeric value

`connectivity.CalculateChiv9p(mol)`
Calculation of valence molecular connectivity chi index for
path order 9
—>Chiv9

Usage:

```
result=CalculateChiv9p(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateDeltaChi0** (*mol*)

Calculation of the difference between chi0v and chi0

—>dchi0

Usage:

```
result=CalculateDeltaChi0(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateDeltaChi1** (*mol*)

Calculation of the difference between chi1v and chi1

—>dchi1

Usage:

```
result=CalculateDeltaChi1(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateDeltaChi2** (*mol*)

Calculation of the difference between chi2v and chi2

—>dchi2

Usage:

```
result=CalculateDeltaChi2(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateDeltaChi3** (*mol*)

Calculation of the difference between chi3v and chi3

—>dchi3

Usage:

```
result=CalculateDeltaChi3(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

connectivity.**CalculateDeltaChi3c4pc** (*mol*)

Calculation of the difference between chi3c and chi4pc

—>knotp

Usage:

```
result=CalculateDeltaChi3c4pc(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateDeltaChi4 (mol)`

Calculation of the difference between chi4v and chi4

—>dchi4

Usage:

result=CalculateDeltaChi4(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateDeltaChiv3c4pc (mol)`

Calculation of the difference between chiv3c and chiv4pc

—>knotpv

Usage:

result=CalculateDeltaChiv3c4pc(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.CalculateMeanRandic (mol)`

Calculation of mean chi1 (Randic) connectivity index.

—>mchi1

Usage:

result=CalculateMeanRandic(mol)

Input: mol is a molecule object.

Output: result is a numeric value

`connectivity.GetConnectivity (mol)`

Get the dictionary of connectivity descriptors for given molecule mol

Usage:

result=GetConnectivity(mol)

Input: mol is a molecule object.

Output: result is a dict form containing all connectivity indices

4.2.8 constitution module

structure. You can get 30 molecular connectivity descriptors. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`constitution.CalculateAllAtomNumber` (*mol*)

Calculation of all atom counts in a molecule

—>nta

Usage:

result=CalculateAllAtomNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateAromaticBondNumber` (*mol*)

Calculation of aromatic bond counts in a molecule

—>naro

Usage:

result=CalculateAromaticBondNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateAverageMolWeight` (*mol*)

Calculation of average molecular weight

Note that not including H

—>AWeight

Usage:

result=CalculateAverageMolWeight(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateBromineNumber` (*mol*)

Calculation of Bromine counts in a molecule

—>ncobr

Usage:

result=CalculateBromineNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateCarbonNumber` (*mol*)

Calculation of Carbon number in a molecule

—>ncarb

Usage:

result=CalculateCarbonNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateChlorinNumber` (*mol*)

Calculation of Chlorin counts in a molecule

—>ncocl

Usage:

result=CalculateChlorinNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateDoubleBondNumber` (*mol*)

Calculation of double bond counts in a molecule

—>ndb

Usage:

result=CalculateDoubleBondNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateFluorinNumber` (*mol*)

Calculation of Fluorin counts in a molecule

—>ncof

Usage:

result=CalculateFluorinNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateHacceptorNumber` (*mol*)

Calculation of Hydrogen bond acceptor counts in a molecule

—>naccr

Usage:

result=CalculateHacceptorNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateHalogenNumber` (*mol*)

Calculation of Halogen counts in a molecule

—>nhal

Usage:

result=CalculateHalogenNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateHdonorNumber` (*mol*)

Calculation of Hydrogen bond donor counts in a molecule

—>ndonr

Usage:

result=CalculateHdonorNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

constitution.**CalculateHeavyAtomNumber** (*mol*)

Calculation of Heavy atom counts in a molecule

—>nhev

Usage:

result=CalculateHeavyAtomNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

constitution.**CalculateHeteroNumber** (*mol*)

Calculation of Hetero counts in a molecule

—>nhet

Usage:

result=CalculateHeteroNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

constitution.**CalculateHydrogenNumber** (*mol*)

Calculation of Number of Hydrogen in a molecule

—>nhyd

Usage:

result=CalculateHydrogenNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

constitution.**CalculateIodineNumber** (*mol*)

Calculation of Iodine counts in a molecule

—>ncoi

Usage:

result=CalculateIodineNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

constitution.**CalculateMolWeight** (*mol*)

Calculation of molecular weight

Note that not including H

—>Weight

Usage:

```
result=CalculateMolWeight(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
constitution.CalculateNitrogenNumber (mol)
```

Calculation of Nitrogen counts in a molecule

—>nnitro

Usage:

```
result=CalculateNitrogenNumber(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
constitution.CalculateOxygenNumber (mol)
```

Calculation of Oxygen counts in a molecule

—>noxy

Usage:

```
result=CalculateOxygenNumber(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
constitution.CalculatePath1 (mol)
```

```
constitution.CalculatePath2 (mol)
```

```
constitution.CalculatePath3 (mol)
```

```
constitution.CalculatePath4 (mol)
```

```
constitution.CalculatePath5 (mol)
```

```
constitution.CalculatePath6 (mol)
```

```
constitution.CalculatePhosphorNumber (mol)
```

Calculation of Phosphor number in a molecule

—>nphos

Usage:

```
result=CalculatePhosphorNumber(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

```
constitution.CalculateRingNumber (mol)
```

Calculation of ring counts in a molecule

—>nring

Usage:

```
result=CalculateRingNumber(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateRotationBondNumber` (*mol*)

Calculation of rotation bonds counts in a molecule

—>nrot

Note that this is the same as calculation of single bond counts in a molecule.

Usage:

result=CalculateRotationBondNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateSingleBondNumber` (*mol*)

Calculation of single bond counts in a molecule

—>nsb

Usage:

result=CalculateSingleBondNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateSulfurNumber` (*mol*)

Calculation of Sulfur counts in a molecule

—>nsulph

Usage:

result=CalculateSulfurNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.CalculateTripleBondNumber` (*mol*)

Calculation of triple bond counts in a molecule

—>ntb

Usage:

result=CalculateTripleBondNumber(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`constitution.GetConstitutional` (*mol*)

Get the dictionary of constitutional descriptors for given molecule mol

Usage:

result=GetConstitutional(mol)

Input: mol is a molecule object.

Output: result is a dict form containing all constitutional values.

4.2.9 estate module

and Hall's paper. If you have any question please contact me via email.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`estate.CalculateEstateFingerprint (mol)`

The Calculation of EState Fingerprints.

It is the number of times each possible atom type is hit.

Usage:

```
result=CalculateEstateFingerprint(mol)
```

Input: mol is a molecule object.

Output: result is a dict form containing 79 estate fragments.

`estate.CalculateEstateValue (mol)`

The Calculate of EState Values.

It is the sum of the Estate indices for atoms of each type.

Usage:

```
result=CalculateEstateValue(mol)
```

Input: mol is a molecule object.

Output: result is a dict form containing 79 estate values.

`estate.CalculateMaxAtomTypeEState (mol)`

Calculation of maximum of E-State value of specified atom type

res—>dict type

Usage:

```
result=CalculateMaxAtomTypeEState(mol)
```

Input: mol is a molecule object.

Output: result is a dict form containing 79 max estate values.

`estate.CalculateMinAtomTypeEState (mol)`

Calculation of minimum of E-State value of specified atom type

res—>dict type

Usage:

```
result=CalculateMinAtomTypeEState(mol)
```

Input: mol is a molecule object.

Output: result is a dict form containing 79 min estate values.

`estate.GetEstate (mol)`

Obtain all descriptors related to Estate.

Usage:

```
result=GetEstate(mol)
```

Input: mol is a molecule object.

Output: result is a dict form containing all estate values.

4.2.10 fingerprint module

fingerprint system. If you have any question please contact me via email.

2016.11.15

@author: Zhijiang Yao and Dongsheng Cao

Email: gadsby@163.com and oriental-cds@163.com

`fingerprint.CalculateAtomPairsFingerprint` (*mol*)

Calculate atom pairs fingerprints

Usage:

```
result=CalculateAtomPairsFingerprint(mol)
```

Input: mol is a molecule object.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateDaylightFingerprint` (*mol*)

Calculate Daylight-like fingerprint or topological fingerprint (2048 bits).

Usage:

```
result=CalculateDaylightFingerprint(mol)
```

Input: mol is a molecule object.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateECFP2Fingerprint` (*mol*, *radius=1*)

Calculate ECFP2

Usage:

```
result=CalculateECFP2Fingerprint(mol)
```

Input: mol is a molecule object.

radius is a radius.

Output: result is a tuple form. The first is the vector of fingerprints. The second is a dict form whose keys are the

position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateECFP4Fingerprint (mol, radius=2)`
Calculate ECFP4

Usage:

`result=CalculateECFP4Fingerprint(mol)`

Input: mol is a molecule object.

radius is a radius.

Output: result is a tuple form. The first is the vector of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateECFP6Fingerprint (mol, radius=3)`
Calculate ECFP6

Usage:

`result=CalculateECFP6Fingerprint(mol)`

Input: mol is a molecule object.

radius is a radius.

Output: result is a tuple form. The first is the vector of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateEstateFingerprint (mol)`
Calculate E-state fingerprints (79 bits).

Usage:

`result=CalculateEstateFingerprint(mol)`

Input: mol is a molecule object.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateFCFP2Fingerprint (mol, radius=1, nBits=1024)`
Calculate FCFP2

Usage:

`result=CalculateFCFP2Fingerprint(mol)`

Input: mol is a molecule object.

radius is a radius.

Output: result is a tuple form. The first is the vector of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateFCFP4Fingerprint (mol, radius=2, nBits=1024)`
Calculate FCFP4

Usage:

`result=CalculateFCFP4Fingerprint(mol)`

Input: mol is a molecule object.

radius is a radius.

Output: result is a tuple form. The first is the vector of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateFCFP6Fingerprint (mol, radius=3, nBits=1024)`
Calculate FCFP6

Usage:

`result=CalculateFCFP4Fingerprint(mol)`

Input: mol is a molecule object.

radius is a radius.

Output: result is a tuple form. The first is the vector of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateFP2Fingerprint (mol)`
Calculate FP2 fingerprints (1024 bits).

Usage:

`result=CalculateFP2Fingerprint(mol)`

Input: mol is a molecule object.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

`fingerprint.CalculateFP3Fingerprint (mol)`
Calculate FP3 fingerprints (210 bits).

Usage:

```
result=CalculateFP3Fingerprint(mol)
```

Input: mol is a molecule object.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

```
fingerprint.CalculateFP4Fingerprint (mol)
```

Calculate FP4 fingerprints (307 bits).

Usage:

```
result=CalculateFP4Fingerprint(mol)
```

Input: mol is a molecule object.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

```
fingerprint.CalculateGhoseCrippenFingerprint (mol, count=False)
```

```
fingerprint.CalculateMACCSFingerprint (mol)
```

Calculate MACCS keys (166 bits).

Usage:

```
result=CalculateMACCSFingerprint(mol)
```

Input: mol is a molecule object.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

```
fingerprint.CalculateMorganFingerprint (mol, radius=2)
```

Calculate Morgan

Usage:

```
result=CalculateMorganFingerprint(mol)
```

Input: mol is a molecule object.

radius is a radius.

Output: result is a tuple form. The first is the number of fingerprints. The second is a dict form whose keys are the position which this molecule has some substructure. The third is the DataStructs which is used for calculating the similarity.

```
fingerprint.CalculatePharm2D2pointFingerprint (mol, featFac-
tury=<rdkit.Chem.rdMolChemicalFeatures.MolChemicalFeatureF
object at 0x045D0FB8>)
```

`fingerprint.CalculatePharm2D3pointFingerprint` (*mol*, *featFactory=<rdkit.Chem.rdMolChemicalFeatures.MolChemicalFeatureFactory object at 0x045D0FB8>*)

`fingerprint.CalculateSimilarityPybel` (*fp1, fp2*)

Calculate Tanimoto similarity between two molecules.

Usage:

`result=CalculateSimilarityPybel(fp1,fp2)`

Input: *fp1* and *fp2* are two DataStructs.

Output: *result* is a Tanimoto similarity value.

`fingerprint.CalculateSimilarityRdkit` (*fp1, fp2, similarity='Tanimoto'*)

Calculate similarity between two molecules.

Usage:

`result=CalculateSimilarity(fp1,fp2)` Users can choose 11 different types: Tanimoto, Dice, Cosine,

Sokal, Russel, RogotGoldberg, AllBit, Kulczynski, McConnaughey, Asymmetric, BraunBlanquet

Input: *fp1* and *fp2* are two DataStructs.

Output: *result* is a similarity value.

`fingerprint.CalculateTopologicalTorsionFingerprint` (*mol*)

Calculate Topological Torsion Fingerprints

Usage:

`result=CalculateTopologicalTorsionFingerprint(mol)`

Input: *mol* is a molecule object.

Output: *result* is a tuple form. The first is the number of

fingerprints. The second is a dict form whose keys are the

position which this molecule has some substructure. The third

is the DataStructs which is used for calculating the similarity.

4.2.11 geary module

structure. You can get 32 molecular autocorrelation descriptors. You can

freely use and distribute it. If you have any problem, you could contact

with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`geary.CalculateGearyAutoElectronegativity` (*mol*)

Calculation of Geary autocorrelation descriptors based on

carbon-scaled atomic Sanderson electronegativity.

Usage:

`res=CalculateGearyAutoElectronegativity(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing eight geary autocorrealtion

`geary.CalculateGearyAutoMass (mol)`

Calculation of Geary autocorrelation descriptors based on carbon-scaled atomic mass.

Usage:

```
res=CalculateMoranAutoMass(mol)
```

Input: mol is a molecule object.

Output: res is a dict form containing eight geary autocorrealtion

`geary.CalculateGearyAutoPolarizability (mol)`

Calculation of Geary autocorrelation descriptors based on carbon-scaled atomic polarizability.

Usage:

```
res=CalculateGearyAutoPolarizability(mol)
```

Input: mol is a molecule object.

Output: res is a dict form containing eight geary autocorrealtion

`geary.CalculateGearyAutoVolume (mol)`

Calculation of Geary autocorrelation descriptors based on carbon-scaled atomic van der Waals volume.

Usage:

```
res=CalculateGearyAutoVolume(mol)
```

Input: mol is a molecule object.

Output: res is a dict form containing eight geary autocorrealtion

`geary.GetGearyAuto (mol)`

Calcualate all Geary autocorrelation descriptors.

(carbon-scaled atomic mass, carbon-scaled atomic van der Waals volume, carbon-scaled atomic Sanderson electronegativity, carbon-scaled atomic polarizability)

Usage:

```
res=GetGearyAuto(mol)
```

Input: mol is a molecule object.

Output: res is a dict form containing all geary autocorrealtion

4.2.12 ghosecrippen module

This module is to calculate the ghosecrippen descriptor. If you have any question please contact me via email.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`ghosecrippen.GhoseCrippenFingerprint (mol, count=False)`

Ghose-Crippen substructures or counts based on the definitions of

SMARTS from Ghose-Crippen's paper. (110 dimension)

4.2.13 kappa module

structure. You can get 7 molecular kappa descriptors. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`kappa.CalculateFlexibility (mol)`

Calculation of Kier molecular flexibility index

—>phi

Usage:

`result=CalculateFlexibility(mol)`

Input: mol is a molecule object.

Output: result is a numeric value.

`kappa.CalculateKappa1 (mol)`

Calculation of molecular shape index for one bonded fragment

—>kappa1

Usage:

`result=CalculateKappa1(mol)`

Input: mol is a molecule object.

Output: result is a numeric value.

`kappa.CalculateKappa2 (mol)`

Calculation of molecular shape index for two bonded fragment

—>kappa2

Usage:

`result=CalculateKappa2(mol)`

Input: mol is a molecule object.

Output: result is a numeric value.

`kappa.CalculateKappa3 (mol)`

Calculation of molecular shape index for three bonded fragment

—>kappa3

Usage:

```
result=CalculateKappa3(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`kappa.CalculateKappaAlpha1 (mol)`

Calculation of molecular shape index for one bonded fragment

with Alpha

—>kappam1

Usage:

```
result=CalculateKappaAlpha1(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`kappa.CalculateKappaAlpha2 (mol)`

Calculation of molecular shape index for two bonded fragment

with Alpha

—>kappam2

Usage:

```
result=CalculateKappaAlpha2(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`kappa.CalculateKappaAlpha3 (mol)`

Calculation of molecular shape index for three bonded fragment

with Alpha

—>kappam3

Usage:

```
result=CalculateKappaAlpha3(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`kappa.GetKappa (mol)`

Calculation of all kappa values.

Usage:

```
result=GetKappa(mol)
```

Input: mol is a molecule object.

Output: result is a dict form containing 6 kappa values.

4.2.14 moe module

include LabuteASA, TPSA, slogPVSA, MRVSA, PEOEVSA, EstateVSA and VSAEstate, respectively (60).

If you have any question about these indices please contact me via email.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`moe.CalculateEstateVSA (mol, bins=None)`

MOE-type descriptors using Estate indices and surface area contributions.

estateBins=[-0.390,0.290,0.717,1.165,1.540,1.807,2.05,4.69,9.17,15.0]

You can specify your own bins to compute some descriptors

Usage:

result=CalculateEstateVSA(mol)

Input: mol is a molecule object

Output: result is a dict form

`moe.CalculateLabuteASA (mol)`

Calculation of Labute's Approximate Surface Area (ASA from MOE)

Usage:

result=CalculateLabuteASA(mol)

Input: mol is a molecule object

Output: result is a dict form

`moe.CalculatePEOEVS (mol, bins=None)`

MOE-type descriptors using partial charges and surface area contributions.

chgBins=[-.3,-.25,-.20,-.15,-.10,-.05,0,.05,.10,.15,.20,.25,.30]

You can specify your own bins to compute some descriptors

Usage:

result=CalculatePEOEVS(mol)

Input: mol is a molecule object

Output: result is a dict form

`moe.CalculateSLOGPVSA (mol, bins=None)`

MOE-type descriptors using LogP contributions and surface area contributions.

logpBins=[-0.4,-0.2,0,0.1,0.15,0.2,0.25,0.3,0.4,0.5,0.6]

You can specify your own bins to compute some descriptors.

Usage:

```
result=CalculateSLOGPVSA(mol)
```

Input: mol is a molecule object

Output: result is a dict form

`moe.CalculateSMRVSA (mol, bins=None)`

MOE-type descriptors using MR contributions and surface area contributions.

```
mrBins=[1.29, 1.82, 2.24, 2.45, 2.75, 3.05, 3.63,3.8,4.0]
```

You can specify your own bins to compute some descriptors.

Usage:

```
result=CalculateSMRVSA(mol)
```

Input: mol is a molecule object

Output: result is a dict form

`moe.CalculateTPSA (mol)`

Calculation of topological polar surface area based on fragments.

Implementation based on the Daylight contrib program tpsa.

Usage:

```
result=CalculateTPSA(mol)
```

Input: mol is a molecule object

Output: result is a dict form

`moe.CalculateVSAEstate (mol, bins=None)`

MOE-type descriptors using Estate indices and surface area contributions.

```
vsaBins=[4.78,5.00,5.410,5.740,6.00,6.07,6.45,7.00,11.0]
```

You can specify your own bins to compute some descriptors

Usage:

```
result=CalculateVSAEstate(mol)
```

Input: mol is a molecule object

Output: result is a dict form

`moe.GetMOE (mol)`

The calculation of MOE-type descriptors (ALL).

Usage:

```
result=GetMOE(mol)
```

Input: mol is a molecule object

Output: result is a dict form

4.2.15 molproperty module

type of approaches(6), including: LogP; LogP2; MR; TPSA, UI and Hy. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`molproperty.CalculateHydrophilicityFactor` (*mol*)

Calculation of hydrophilicity factor. The hydrophilicity

index is described in more detail on page 225 of the

Handbook of Molecular Descriptors (Todeschini and Consonni 2000).

—>Hy

Usage:

result=CalculateHydrophilicityFactor(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.CalculateMolLogP` (*mol*)

Calculation of LogP value based on Crippen method

—>LogP

Usage:

result=CalculateMolLogP(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.CalculateMolLogP2` (*mol*)

Calculation of LogP² value based on Crippen method

—>LogP2

Usage:

result=CalculateMolLogP2(mol)

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.CalculateMolMR` (*mol*)

Calculation of molecular refraction value based on Crippen method

—>MR

Usage:

```
result=CalculateMolMR(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.CalculateTPSA` (*mol*)

calculates the polar surface area of a molecule based upon fragments

Algorithm in:

16.Ertl, B. Rohde, P. Selzer

Fast Calculation of Molecular Polar Surface Area as a Sum of

Fragment-based Contributions and Its Application to the Prediction

of Drug Transport Properties, J.Med.Chem. 43, 3714-3717, 2000

Implementation based on the Daylight contrib program tpsa.

—>TPSA

Usage:

```
result=CalculateTPSA(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.CalculateUnsaturationIndex` (*mol*)

Calculation of unsaturation index.

—>UI

Usage:

```
result=CalculateUnsaturationIndex(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.CalculateXlogP` (*mol*)

Calculation of Wang octanol water partition coefficient.

—>XLogP

Usage:

```
result=CalculateXlogP(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.CalculateXlogP2` (*mol*)

Calculation of Wang octanol water partition coefficient (XLogP^2).

—>XLogP2

Usage:

```
result=CalculateMolLogP(mol)
```

Input: mol is a molecule object.

Output: result is a numeric value.

`molproperty.GetMolecularProperty` (*mol*)

Get the dictionary of constitutional descriptors for
given molecule *mol*

Usage:

`result=GetMolecularProperty(mol)`

Input: *mol* is a molecule object.

Output: *result* is a dict form containing 6 molecular properties.

4.2.16 moran module

The calculation of Moran autocorrelation descriptors. You can get 32 molecular descriptors. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`moran.CalculateMoranAutoElectronegativity` (*mol*)

Calculation of Moran autocorrelation descriptors based on
carbon-scaled atomic Sanderson electronegativity.

Usage:

`res=CalculateMoranAutoElectronegativity(mol)`

Input: *mol* is a molecule object.

Output: *res* is a dict form containing eight moran autocorrelation

`moran.CalculateMoranAutoMass` (*mol*)

Calculation of Moran autocorrelation descriptors based on
carbon-scaled atomic mass.

Usage:

`res=CalculateMoranAutoMass(mol)`

Input: *mol* is a molecule object.

Output: *res* is a dict form containing eight moran autocorrelation

`moran.CalculateMoranAutoPolarizability` (*mol*)

Calculation of Moran autocorrelation descriptors based on
carbon-scaled atomic polarizability.

Usage:

`res=CalculateMoranAutoPolarizability(mol)`

Input: *mol* is a molecule object.

Output: res is a dict form containing eight moran autocorrealtion

`moran.CalculateMoranAutoVolume(mol)`

Calculation of Moran autocorrelation descriptors based on
carbon-scaled atomic van der Waals volume.

Usage:

`res=CalculateMoranAutoVolume(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing eight moran autocorrealtion

`moran.GetMoranAuto(mol)`

Calcualate all Moran autocorrelation descriptors.

(carbon-scaled atomic mass, carbon-scaled atomic van der Waals volume,
carbon-scaled atomic Sanderson electronegativity,
carbon-scaled atomic polarizability)

Usage:

`res=GetMoranAuto(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing all moran autocorrealtion

4.2.17 moreaubroto module

The calculation of Moreau-Broto autocorrelation descriptors. You can get 32
molecular decriptors. You can freely use and distribute it. If you hava
any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`moreaubroto.CalculateMoreauBrotoAutoElectronegativity(mol)`

Calculation of Moreau-Broto autocorrelation descriptors based on
carbon-scaled atomic Sanderson electronegativity.

Usage:

`res=CalculateMoreauBrotoAutoElectronegativity(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing eight moreau broto autocorrealtion

`moreaubroto.CalculateMoreauBrotoAutoMass (mol)`

Calculation of Moreau-Broto autocorrelation descriptors based on carbon-scaled atomic mass.

Usage:

`res=CalculateMoreauBrotoAutoMass(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing eight moreau broto autocorrelation

`moreaubroto.CalculateMoreauBrotoAutoPolarizability (mol)`

Calculation of Moreau-Broto autocorrelation descriptors based on carbon-scaled atomic polarizability.

`res=CalculateMoreauBrotoAutoPolarizability(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing eight moreau broto autocorrelation

`moreaubroto.CalculateMoreauBrotoAutoVolume (mol)`

Calculation of Moreau-Broto autocorrelation descriptors based on carbon-scaled atomic van der Waals volume.

Usage:

`res=CalculateMoreauBrotoAutoVolume(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing eight moreau broto autocorrelation

`moreaubroto.GetMoreauBrotoAuto (mol)`

Calculate all Moreau-Broto autocorrelation descriptors.

(carbon-scaled atomic mass, carbon-scaled atomic van der Waals volume, carbon-scaled atomic Sanderson electronegativity, carbon-scaled atomic polarizability)

Usage:

`res=GetMoreauBrotoAuto(mol)`

Input: mol is a molecule object.

Output: res is a dict form containing all moreau broto autocorrelation

4.2.18 topology module

structure. You can get 25 molecular topological descriptors. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com and oriental-cds@163.com

`topology.CalculateArithmeticTopoIndex` (*mol*)

Arithmetic topological index by Narumi

—>Arto

Usage:

result=CalculateArithmeticTopoIndex(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateBalaban` (*mol*)

Calculation of Balaban index in a molecule

—>J

Usage:

result=CalculateBalaban(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateBertzCT` (*mol*)

A topological index meant to quantify “complexity” of molecules.

Consists of a sum of two terms, one representing the complexity

of the bonding, the other representing the complexity of the

distribution of heteroatoms.

From S. H. Bertz, J. Am. Chem. Soc., vol 103, 3599-3601 (1981)

—>BertzCT(log value)

Usage:

result=CalculateBertzCT(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateDiameter` (*mol*)

Calculation of diameter, which is Largest value

in the distance matrix [Petitjean 1992].

—>diameter

Usage:

result=CalculateDiameter(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateGeometricTopoIndex` (*mol*)

Geometric topological index by Narumi

—>Geto

Usage:

```
result=CalculateGeometricTopoIndex(mol)
```

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateGraphDistance` (*mol*)

Calculation of graph distance index

—>Tigdi(log value)

Usage:

```
result=CalculateGraphDistance(mol)
```

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateGutmanTopo` (*mol*)

Calculation of Gutman molecular topological index based on
simple vertex degree

—>GMTI(log value)

Usage:

```
result=CalculateGutmanTopo(mol)
```

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateHarary` (*mol*)

Calculation of Harary number

—>Thara

Usage:

```
result=CalculateHarary(mol)
```

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateHarmonicTopoIndex` (*mol*)

Calculation of harmonic topological index proposed by Narnumi.

—>Hato

Usage:

```
result=CalculateHarmonicTopoIndex(mol)
```

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateIpc` (*mol*)

This returns the information content of the coefficients of the
characteristic polynomial of the adjacency matrix of a
hydrogen-suppressed graph of a molecule.

‘avg = 1’ returns the information content divided by the total
population.

From D. Bonchev & N. Trinajstić, J. Chem. Phys. vol 67,
4517-4533 (1977)

—>Ipc(log value)

Usage:

result=CalculateIpc(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateMZagreb1` (*mol*)

Calculation of Modified Zagreb index with order 1 in a molecule

—>MZM1

Usage:

result=CalculateMZagreb1(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateMZagreb2` (*mol*)

Calculation of Modified Zagreb index with order 2 in a molecule

—>MZM2

Usage:

result=CalculateMZagreb2(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateMeanWeiner` (*mol*)

Calculation of Mean Wiener number in a molecule

—>AW

Usage:

result=CalculateWeiner(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculatePetitjean` (*mol*)

Calculation of Petitjean based on topology.

Value of (diameter - radius) / diameter as defined in [Petitjean 1992].

—>petitjeant

Usage:

result=CalculatePetitjean(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculatePlatt` (*mol*)

Calculation of Platt number in a molecule

—>Platt

Usage:

result=CalculatePlatt(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculatePoglianiIndex` (*mol*)

Calculation of Poglicani index

The Pogliani index (Dz) is the sum over all non-hydrogen atoms of a modified vertex degree calculated as the ratio of the number of valence electrons over the principal quantum number of an atom [L. Pogliani, J.Phys.Chem. 1996, 100, 18065-18077].

—>DZ

Usage:

result=CalculatePoglianiIndex(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculatePolarityNumber` (*mol*)

Calculation of Polarity number.

It is the number of pairs of vertexes at distance matrix equal to 3

—>Pol

Usage:

result=CalculatePolarityNumber(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateQuadratic` (*mol*)

Calculation of Quadratic index in a molecule

—>Qindex

Usage:

result=CalculateQuadratic(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateRadius` (*mol*)

Calculation of radius based on topology.

It is :If ri is the largest matrix entry in row i of the distance

matrix D, then the radius is defined as the smallest of the r_i
[Petitjean 1992].

—>radius

Usage:

result=CalculateRadius(mol)

Input: mol is a molecule object

Output: result is a numeric value

topology.**CalculateSchiultz** (*mol*)

Calculation of Schiultz number

—>Tsch(log value)

Usage:

result=CalculateSchiultz(mol)

Input: mol is a molecule object

Output: result is a numeric value

topology.**CalculateSimpleTopoIndex** (*mol*)

Calculation of the logarithm of the simple topological index by Narumi,
which is defined as the product of the vertex degree.

—>Sito

Usage:

result=CalculateSimpleTopoIndex(mol)

Input: mol is a molecule object

Output: result is a numeric value

topology.**CalculateWeiner** (*mol*)

Calculation of Wiener number in a molecule

—>W

Usage:

result=CalculateWeiner(mol)

Input: mol is a molecule object

Output: result is a numeric value

topology.**CalculateXuIndex** (*mol*)

Calculation of Xu index

—>Xu

Usage:

result=CalculateXuIndex(mol)

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateZagreb1 (mol)`

Calculation of Zagreb index with order 1 in a molecule

—>ZM1

Usage:

`result=CalculateZagreb1(mol)`

Input: mol is a molecule object

Output: result is a numeric value

`topology.CalculateZagreb2 (mol)`

Calculation of Zagreb index with order 2 in a molecule

—>ZM2

Usage:

`result=CalculateZagreb2(mol)`

Input: mol is a molecule object

Output: result is a numeric value

`topology.GetTopology (mol)`

Get the dictionary of constitutional descriptors for given

molecule mol

Usage:

`result=CalculateWeiner(mol)`

Input: mol is a molecule object

Output: result is a dict form containing all topological indices.

4.2.19 Scaffolds module

If you have any question please contact me via email.

Bemis, G. W.; Murcko, M. A. "The Properties of Known Drugs. 1. Molecular Frameworks." J. Med. Chem. 39:2887-93 (1996).

2016.11.15

@author: Zhijiang Yao and Dongsheng Cao

Email: gadsby@163.com and oriental-cds@163.com

`Scaffolds.GetScaffold (mol, generic_framework=False)`

Calculate Scaffold

Usage:

`result = GetScaffold(mol)`

Input: mol is a molecule object.

generic_framework is boolean value. If the generic_framework is True, the result returns a generic framework.

Output: result is a string form of the molecule's scaffold.

`Scaffolds.GetUniqueScaffold(mols, generic_framework=False)`

Calculate molecules' unique scaffolds

Usage:

```
result = GetUniqueScaffold(mols)
```

Input: mols is molecules object.

`generic_framework` is boolean value. If the `generic_framework` is True, the result returns a generic framework dictionary.

Output: result is a tuple form. The first is the list of unique scaffolds. The second is a dict form whose keys are the scaffolds and the values are the scaffolds' count.

4.3 PyProtein

4.3.1 PyProtein module

A class used for computing different types of protein descriptors!

You can freely use and distribute it. If you have any problem, you could contact with us timely.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`class PyProtein.PyProtein(ProteinSequence='')`

This `GetProDes` class aims at collecting all descriptor calculation modules into a simple class.

`AALetter = ['A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']`

`GetAAComp()`

amino acid composition descriptors (20)

Usage:

```
result = GetAAComp()
```

`GetAAindex1(name, path='.')`

Get the amino acid property values from `aaindex1`

Usage:

```
result=GetAAIndex1(name)
```

Input: name is the name of amino acid property (e.g., KRIW790103)

Output: result is a dict form containing the properties of 20 amino acids

`GetAAindex23(name, path='.')`

Get the amino acid property values from `aaindex2` and `aaindex3`

Usage:

```
result=GetAAIndex23(name)
```

Input: name is the name of amino acid property (e.g.,TANS760101,GRAR740104)

Output: result is a dict form containing the properties of 400 amino acid pairs

GetALL ()

Calculate all descriptors except tri-peptide descriptors

GetAPAAC (lamda=10, weight=0.5)

Amphiphilic (Type II) Pseudo amino acid composition descriptors

default is 30

Usage:

result = GetAPAAC(lamda=10,weight=0.5)

lamda factor reflects the rank of correlation and is a non-Negative integer, such as 15.

Note that (1)lamda should NOT be larger than the length of input protein sequence;

2.lamda must be non-Negative integer, such as 0, 1, 2, ...; (3) when lamda =0, the output of PseAA server is the 20-D amino acid composition.

weight factor is designed for the users to put weight on the additional PseAA components with respect to the conventional AA components. The user can select any value within the region from 0.05 to 0.7 for the weight factor.

GetCTD ()

Composition Transition Distribution descriptors (147)

Usage:

result = GetCTD()

GetDPComp ()

dipeptide composition descriptors (400)

Usage:

result = GetDPComp()

GetGearyAuto ()

Geary autocorrelation descriptors (240)

Usage:

result = GetGearyAuto()

GetGearyAutop (AAP={}, AAPName='p')

Geary autocorrelation descriptors for the given property (30)

Usage:

result = GetGearyAutop(AAP={},AAPName='p')

AAP is a dict containing physicochemical properties of 20 amino acids

GetMoranAuto ()

Moran autocorrelation descriptors (240)

Usage:

result = GetMoranAuto()

GetMoranAutop (*AAP={}*, *AAPName='p'*)

Moran autocorrelation descriptors for the given property (30)

Usage:

```
result = GetMoranAutop(AAP={},AAPName='p')
```

AAP is a dict containing physicochemical properties of 20 amino acids

GetMoreauBrotoAuto ()

Normalized Moreau-Broto autocorrelation descriptors (240)

Usage:

```
result = GetMoreauBrotoAuto()
```

GetMoreauBrotoAutop (*AAP={}*, *AAPName='p'*)

Normalized Moreau-Broto autocorrelation descriptors for the given property (30)

Usage:

```
result = GetMoreauBrotoAutop(AAP={},AAPName='p')
```

AAP is a dict containing physicochemical properties of 20 amino acids

GetPAAC (*lamda=10*, *weight=0.05*)

Type I Pseudo amino acid composition descriptors (default is 30)

Usage:

```
result = GetPAAC(lamda=10,weight=0.05)
```

lamda factor reflects the rank of correlation and is a non-Negative integer, such as 15.

Note that (1)lamda should NOT be larger than the length of input protein sequence;

2.lamda must be non-Negative integer, such as 0, 1, 2, ...; (3) when lamda =0, the output of PseAA server is the 20-D amino acid composition.

weight factor is designed for the users to put weight on the additional PseAA components with respect to the conventional AA components. The user can select any value within the region from 0.05 to 0.7 for the weight factor.

GetPAACp (*lamda=10*, *weight=0.05*, *AAP=[]*)

Type I Pseudo amino acid composition descriptors for the given properties (default is 30)

Usage:

```
result = GetPAACp(lamda=10,weight=0.05,AAP=[])
```

lamda factor reflects the rank of correlation and is a non-Negative integer, such as 15.

Note that (1)lamda should NOT be larger than the length of input protein sequence;

2.lamda must be non-Negative integer, such as 0, 1, 2, ...; (3) when lamda =0, the output of PseAA server is the 20-D amino acid composition.

weight factor is designed for the users to put weight on the additional PseAA components with respect to the conventional AA components. The user can select any value within the region from 0.05 to 0.7 for the weight factor.

AAP is a list form containing the properties, each of which is a dict form.

GetQSO (*maxlag=30, weight=0.1*)

Quasi sequence order descriptors default is 50

result = GetQSO(maxlag=30, weight=0.1)

maxlag is the maximum lag and the length of the protein should be larger than maxlag. default is 45.

GetQSOp (*maxlag=30, weight=0.1, distancematrix={}*)

Quasi sequence order descriptors default is 50

result = GetQSO(maxlag=30, weight=0.1)

maxlag is the maximum lag and the length of the protein should be larger than maxlag. default is 45.

distancematrix is a dict form containing 400 distance values

GetSOCN (*maxlag=45*)

Sequence order coupling numbers default is 45

Usage:

result = GetSOCN(maxlag=45)

maxlag is the maximum lag and the length of the protein should be larger than maxlag. default is 45.

GetSOCNp (*maxlag=45, distancematrix={}*)

Sequence order coupling numbers default is 45

Usage:

result = GetSOCN(maxlag=45)

maxlag is the maximum lag and the length of the protein should be larger than maxlag. default is 45.

distancematrix is a dict form containing 400 distance values

GetSubSeq (*ToAA='S', window=3*)

obtain the sub sequences wit length 2*window+1, whose central point is ToAA

Usage:

result = GetSubSeq(ToAA='S',window=3)

ToAA is the central (query point) amino acid in the sub-sequence.

window is the span.

GetTPComp ()

tri-peptide composition descriptors (8000)

Usage:

result = GetTPComp()

GetTriad ()

Calculate the conjoint triad features from protein sequence.

Usage:

res = GetTriad()

Output is a dict form containing all 343 conjoint triad features.

Version = 1.0

4.3.2 AAComposition module

The module is used for computing the composition of amino acids, dipetide and 3-mers (tri-peptide) for a given protein sequence. You can get 8420 descriptors for a given protein sequence. You can freely use and distribute it. If you have any problem, you could contact with us timely!

References:

- [1]: Reczko, M. and Bohr, H. (1994) The DEF data base of sequence based protein fold class predictions. Nucleic Acids Res, 22, 3616-3619.
- [2]: Hua, S. and Sun, Z. (2001) Support vector machine approach for protein subcellular localization prediction. Bioinformatics, 17, 721-728.
- [3]: Grassmann, J., Reczko, M., Suhai, S. and Edler, L. (1999) Protein fold class prediction: new methods of statistical classification. Proc Int Conf Intell Syst Mol Biol, 106-112.

Authors: Dongsheng Cao and Yizeng Liang.

Date: 2012.3.27

Email: oriental-cds@163.com

AAComposition.CalculateAAComposition (*ProteinSequence*)

Calculate the composition of Amino acids
for a given protein sequence.

Usage:

result=CalculateAAComposition(protein)

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition of

AAComposition.CalculateAADipeptideComposition (*ProteinSequence*)

Calculate the composition of AADs, dipeptide and 3-mers for a
given protein sequence.

Usage:

result=CalculateAADipeptideComposition(protein)

Input: protein is a pure protein sequence.

Output: result is a dict form containing all composition values of

`AAComposition.CalculateDipeptideComposition (ProteinSequence)`

Calculate the composition of dipeptide for a given protein sequence.

Usage:

```
result=CalculateDipeptideComposition(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition of

`AAComposition.GetSpectrumDict (proteinsequence)`

Calculate the spectrum descriptors of 3-mers for a given protein.

Usage:

```
result=GetSpectrumDict(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition values of 8000

`AAComposition.Getkmers ()`

Get the amino acid list of 3-mers.

Usage:

```
result=Getkmers()
```

Output: result is a list form containing 8000 tri-peptides.

4.3.3 AAIndex module

This module is used for obtaining the properties of amino acids or their pairs from the aaindex database. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`AAIndex.GetAAIndex1 (name, path='.')`

Get the amino acid property values from aaindex1

Usage:

```
result=GetAAIndex1(name)
```

Input: name is the name of amino acid property (e.g., KRIW790103)

Output: result is a dict form containing the properties of 20 amino acids

`AAIndex.GetAAIndex23 (name, path='.')`

Get the amino acid property values from aaindex2 and aaindex3

Usage:

```
result=GetAAIndex23(name)
```

Input: name is the name of amino acid property (e.g., TANS760101, GRAR740104)

Output: result is a dict form containing the properties of 400 amino acid pairs

```
class AAIndex.MatrixRecord
```

```
    Bases: AAIndex.Record
```

```
    Matrix record for mutation matrices or pair-wise contact potentials
```

```
    extend (row)
```

```
    get (aai, aaq, d=None)
```

```
    median ()
```

```
class AAIndex.Record
```

```
    Amino acid index (AAindex) Record
```

```
    aakeys = 'ARNDCQEGHILKMFPSTWYV'
```

```
    extend (row)
```

```
    get (aai, aaq=None, d=None)
```

```
    median ()
```

```
AAIndex.get (key)
```

```
    Get record for key
```

```
AAIndex.grep (pattern)
```

```
    Search for pattern in title and description of all records (case insensitive) and print results on standard output.
```

```
AAIndex.init (path=None, index='123')
```

```
    Read in the aaindex files. You need to run this (once) before you can access any records. If the files are not within the current directory, you need to specify the correct directory path. By default all three aaindex files are read in.
```

```
AAIndex.init_from_file (filename, type=<class AAIndex.Record at 0x0380B998>)
```

```
AAIndex.search (pattern, searchtitle=True, casesensitive=False)
```

```
    Search for pattern in description and title (optional) of all records and return matched records as list. By default search case insensitive.
```

4.3.4 Autocorrelation module

This module is used for computing the Autocorrelation descriptors based different

properties of AADs. You can also input your properties of AADs, then it can help you to compute Autocorrelation descriptors based on the property of AADs. Currently, You can get 720 descriptors for a given protein sequence based on our provided physicochemical properties of AADs. You can freely use and distribute it. If you have any problem, you could contact with us timely!

References:

[1]: <http://www.genome.ad.jp/dbget/aaindex.html>

[2]: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.

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

[4]:Sokal, R.R. and Thomson, B.A. (2006) Population structure inferred by local spatial autocorrelation: an Usage from an Amerindian tribal population. Am J Phys Anthropol, 129, 121-131.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`Autocorrelation.CalculateAutoTotal` (*ProteinSequence*)

A method used for computing all autocorrelation descriptors based on 8 properties of AADs.

Usage:

`result=CalculateGearyAutoTotal(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30*8*3=720 normalized Moreau Broto, Moran, and Geary

`Autocorrelation.CalculateEachGearyAuto` (*ProteinSequence, AAP, AAPName*)

you can use the function to compute GearyAuto

descriptors for different properties based on AADs.

Usage:

`result=CalculateEachGearyAuto(protein,AAP,AAPName)`

Input: protein is a pure protein sequence.

AAP is a dict form containing the properties of 20 amino acids (e.g., `_AvFlexibility`).

AAPName is a string used for indicating the property (e.g., `'_AvFlexibility'`).

Output: result is a dict form containing 30 Geary autocorrelation

`Autocorrelation.CalculateEachMoranAuto` (*ProteinSequence, AAP, AAPName*)

you can use the function to compute MoranAuto

descriptors for different properties based on AADs.

Usage:

`result=CalculateEachMoranAuto(protein,AAP,AAPName)`

Input: protein is a pure protein sequence.

AAP is a dict form containing the properties of 20 amino acids (e.g., `_AvFlexibility`).

AAPName is a string used for indicating the property (e.g., `'_AvFlexibility'`).

Output: result is a dict form containing 30 Moran autocorrelation

`Autocorrelation.CalculateEachNormalizedMoreauBrotoAuto` (*ProteinSequence, AAP, AAPName*)

you can use the function to compute MoreauBrotoAuto

descriptors for different properties based on AADs.

Usage:

`result=CalculateEachNormalizedMoreauBrotoAuto(protein,AAP,AAPName)`

Input: protein is a pure protein sequence.

AAP is a dict form containing the properties of 20 amino acids (e.g., `_AvFlexibility`).

AAPName is a string used for indicating the property (e.g., `'_AvFlexibility'`).

Output: result is a dict form containing 30 Normalized Moreau-Broto autocorrelation

`Autocorrelation.CalculateGearyAuto` (*ProteinSequence*, *AAPProperty*, *AAPPropertyName*)

A method used for computing GearyAuto for all properties

Usage:

`result=CalculateGearyAuto(protein,AAP,AAPName)`

Input: protein is a pure protein sequence.

AAPProperty is a list or tuple form containing the properties of 20 amino acids (e.g., `_AAPProperty`).

AAPName is a list or tuple form used for indicating the property (e.g., `'_AAPPropertyName'`).

Output: result is a dict form containing 30*p Geary autocorrelation

`Autocorrelation.CalculateGearyAutoAvFlexibility` (*ProteinSequence*)

Calculate the GearyAuto Autocorrelation descriptors based on

`AvFlexibility`.

Usage: `result=CalculateGearyAutoAvFlexibility(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoFreeEnergy` (*ProteinSequence*)

Calculate the GearyAuto Autocorrelation descriptors based on

`FreeEnergy`.

Usage:

`result=CalculateGearyAutoFreeEnergy(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoHydrophobicity` (*ProteinSequence*)

Calculate the GearyAuto Autocorrelation descriptors based on

`hydrophobicity`.

Usage:

`result=CalculateGearyAutoHydrophobicity(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoMutability` (*ProteinSequence*)

Calculate the GearyAuto Autocorrelation descriptors based on

`Mutability`.

Usage:

`result=CalculateGearyAutoMutability(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoPolarizability (ProteinSequence)`

Calculate the GearyAuto Autocorrelation descriptors based on Polarizability.

Usage:

`result=CalculateGearyAutoPolarizability(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoResidueASA (ProteinSequence)`

Calculate the GearyAuto Autocorrelation descriptors based on ResidueASA.

Usage:

`result=CalculateGearyAutoResidueASA(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoResidueVol (ProteinSequence)`

Calculate the GearyAuto Autocorrelation descriptors based on ResidueVol.

Usage:

`result=CalculateGearyAutoResidueVol(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoSteric (ProteinSequence)`

Calculate the GearyAuto Autocorrelation descriptors based on Steric.

Usage:

`result=CalculateGearyAutoSteric(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Geary Autocorrelation

`Autocorrelation.CalculateGearyAutoTotal (ProteinSequence)`

A method used for computing Geary autocorrelation descriptors based on 8 properties of AADs.

Usage:

`result=CalculateGearyAutoTotal(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing $30 \times 8 = 240$ Geary

`Autocorrelation.CalculateMoranAuto (ProteinSequence, AAPProperty, AAPPropertyName)`

A method used for computing MoranAuto for all properties

Usage:

`result=CalculateMoranAuto(protein,AAP,AAPName)`

Input: protein is a pure protein sequence.

AAProperty is a list or tuple form containing the properties of 20 amino acids (e.g., `_AAProperty`).

AAPName is a list or tuple form used for indicating the property (e.g., `'_AAPropertyName'`).

Output: result is a dict form containing 30*p Moran autocorrelation

`Autocorrelation.CalculateMoranAutoAvFlexibility (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on

AvFlexibility.

Usage:

`result=CalculateMoranAutoAvFlexibility(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoFreeEnergy (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on

FreeEnergy.

Usage:

`result=CalculateMoranAutoFreeEnergy(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoHydrophobicity (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on hydrophobicity.

Usage:

`result=CalculateMoranAutoHydrophobicity(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoMutability (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on

Mutability.

Usage:

`result=CalculateMoranAutoMutability(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoPolarizability (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on

Polarizability.

Usage:

`result=CalculateMoranAutoPolarizability(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoResidueASA (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on ResidueASA.

Usage:

`result=CalculateMoranAutoResidueASA(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoResidueVol (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on ResidueVol.

Usage:

`result=CalculateMoranAutoResidueVol(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoSteric (ProteinSequence)`

Calculate the MoranAuto Autocorrelation descriptors based on AutoSteric.

Usage:

`result=CalculateMoranAutoSteric(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Moran Autocorrelation

`Autocorrelation.CalculateMoranAutoTotal (ProteinSequence)`

A method used for computing Moran autocorrelation descriptors based on 8 properties of AADs.

Usage:

`result=CalculateMoranAutoTotal(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing $30 \times 8 = 240$ Moran

`Autocorrelation.CalculateNormalizedMoreauBrotoAuto (ProteinSequence, AAPProperty, AAPPropertyName)`

A method used for computing MoreauBrotoAuto for all properties.

Usage:

`result=CalculateNormalizedMoreauBrotoAuto(protein,AAP,AAPName)`

Input: protein is a pure protein sequence.

AAPProperty is a list or tuple form containing the properties of 20 amino acids (e.g., `_AAPProperty`).

AAPName is a list or tuple form used for indicating the property (e.g., `'_AAPPropertyName'`).

Output: result is a dict form containing $30 \times p$ Normalized Moreau-Broto autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoAvFlexibility (ProteinSequence)`

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on AvFlexibility.

Usage:

```
result=CalculateNormalizedMoreauBrotoAutoAvFlexibility(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoFreeEnergy` (*ProteinSequence*)

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on

FreeEnergy.

Usage:

```
result=CalculateNormalizedMoreauBrotoAutoFreeEnergy(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoHydrophobicity` (*ProteinSequence*)

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on

hydrophobicity.

Usage:

```
result=CalculateNormalizedMoreauBrotoAutoHydrophobicity(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoMutability` (*ProteinSequence*)

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on Mutability.

Usage:

```
result=CalculateNormalizedMoreauBrotoAutoMutability(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoPolarizability` (*ProteinSequence*)

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on

Polarizability.

Usage:

```
result=CalculateNormalizedMoreauBrotoAutoPolarizability(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoResidueASA` (*ProteinSequence*)

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on

ResidueASA.

Usage:

```
result=CalculateNormalizedMoreauBrotoAutoResidueASA(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoResidueVol (ProteinSequence)`

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on ResidueVol.

Usage:

`result=CalculateNormalizedMoreauBrotoAutoResidueVol(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoSteric (ProteinSequence)`

Calculate the NormalizedMoreauBorto Autocorrelation descriptors based on Steric.

Usage:

`result=CalculateNormalizedMoreauBrotoAutoSteric(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing 30 Normalized Moreau-Broto Autocorrelation

`Autocorrelation.CalculateNormalizedMoreauBrotoAutoTotal (ProteinSequence)`

A method used for computing normalized Moreau Broto autocorrelation descriptors based on 8 proteties of AADs.

Usage:

`result=CalculateNormalizedMoreauBrotoAutoTotal(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing $30 \times 8 = 240$ normalized Moreau Broto

`Autocorrelation.NormalizeEachAAP (AAP)`

All of the amino acid indices are centralized and standardized before the calculation.

Usage:

`result=NormalizeEachAAP(AAP)`

Input: AAP is a dict form containing the properties of 20 amino acids.

Output: result is the a dict form containing the normalized properties

4.3.5 CTD module

This module is used for computing the composition, transition and distribution descriptors based on the different properties of AADs. The AADs with the same properties is marked as the same number. You can get 147 descriptors for a given protein sequence. You can freely use and distribute it. If you hava any problem, you could contact with us timely!

References:

[1]: Inna Dubchak, Ilya Muchink, Stephen R.Holbrook and Sung-Hou Kim. Prediction

of protein folding class using global description of amino acid sequence. Proc.Natl. Acad.Sci.USA, 1995, 92, 8700-8704.

[2]:Inna Dubchak, Ilya Muchink, Christopher Mayor, Igor Dralyuk and Sung-Hou Kim.

Recognition of a Protein Fold in the Context of the SCOP classification. Proteins: Structure, Function and Genetics,1999,35,401-407.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

CTD.**CalculateC** (*ProteinSequence*)

Calculate all composition descriptors based seven different properties of AADs.

Usage:

result=CalculateC(protein)

Input:protein is a pure protein sequence.

CTD.**CalculateCTD** (*ProteinSequence*)

Calculate all CTD descriptors based seven different properties of AADs.

Usage:

result=CalculateCTD(protein)

Input:protein is a pure protein sequence.

CTD.**CalculateComposition** (*ProteinSequence, AAPProperty, AAPName*)

A method used for computing composition descriptors.

Usage:

result=CalculateComposition(protein,AAPProperty,AAPName)

Input: protein is a pure protein sequence.

AAPProperty is a dict form containing classification of amino acids such as _Polarizability.

AAPName is a string used for indicating a AAP name.

CTD.**CalculateCompositionCharge** (*ProteinSequence*)

A method used for calculating composition descriptors based on Charge of AADs.

Usage:

result=CalculateCompositionCharge(protein)

Input:protein is a pure protein sequence.

CTD.**CalculateCompositionHydrophobicity** (*ProteinSequence*)

A method used for calculating composition descriptors based on Hydrophobicity of AADs.

Usage:

result=CalculateCompositionHydrophobicity(protein)

Input:protein is a pure protein sequence.

CTD.**CalculateCompositionNormalizedVDWV** (*ProteinSequence*)

A method used for calculating composition descriptors based on NormalizedVDWV of AADs.

Usage:

```
result=CalculateCompositionNormalizedVDWV(protein)
```

Input:protein is a pure protein sequence.

CTD.**CalculateCompositionPolarity** (*ProteinSequence*)

A method used for calculating composition descriptors based on Polarity of AADs.

Usage:

```
result=CalculateCompositionPolarity(protein)
```

Input:protein is a pure protein sequence.

CTD.**CalculateCompositionPolarizability** (*ProteinSequence*)

A method used for calculating composition descriptors based on Polarizability of AADs.

Usage:

```
result=CalculateCompositionPolarizability(protein)
```

Input:protein is a pure protein sequence.

CTD.**CalculateCompositionSecondaryStr** (*ProteinSequence*)

A method used for calculating composition descriptors based on SecondaryStr of AADs.

Usage:

```
result=CalculateCompositionSecondaryStr(protein)
```

Input:protein is a pure protein sequence.

CTD.**CalculateCompositionSolventAccessibility** (*ProteinSequence*)

A method used for calculating composition descriptors based on SolventAccessibility of AADs.

Usage:

```
result=CalculateCompositionSolventAccessibility(protein)
```

Input:protein is a pure protein sequence.

CTD.**CalculateD** (*ProteinSequence*)

Calculate all distribution descriptors based seven different properties of AADs.

Usage:

```
result=CalculateD(protein)
```

Input:protein is a pure protein sequence.

CTD.**CalculateDistribution** (*ProteinSequence*, *AAPProperty*, *AAPName*)

A method used for computing distribution descriptors.

Usage:

```
result=CalculateDistribution(protein,AAPProperty,AAPName)
```

Input:protein is a pure protein sequence.

AAPProperty is a dict form containing classification of amino acids such as _Polarizability.

AAPName is a string used for indicating a AAP name.

CTD. **CalculateDistributionCharge** (*ProteinSequence*)

A method used for calculating Distribution descriptors based on Charge of

AADs.

Usage:

```
result=CalculateDistributionCharge(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateDistributionHydrophobicity** (*ProteinSequence*)

A method used for calculating Distribution descriptors based on Hydrophobicity of

AADs.

Usage:

```
result=CalculateDistributionHydrophobicity(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateDistributionNormalizedVDWV** (*ProteinSequence*)

A method used for calculating Distribution descriptors based on NormalizedVDWV of

AADs.

Usage:

```
result=CalculateDistributionNormalizedVDWV(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateDistributionPolarity** (*ProteinSequence*)

A method used for calculating Distribution descriptors based on Polarity of

AADs.

Usage:

```
result=CalculateDistributionPolarity(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateDistributionPolarizability** (*ProteinSequence*)

A method used for calculating Distribution descriptors based on Polarizability of

AADs.

Usage:

```
result=CalculateDistributionPolarizability(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateDistributionSecondaryStr** (*ProteinSequence*)

A method used for calculating Distribution descriptors based on SecondaryStr of

AADs.

Usage:

```
result=CalculateDistributionSecondaryStr(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateDistributionSolventAccessibility** (*ProteinSequence*)

A method used for calculating Distribution descriptors based on SolventAccessibility of AADs.

Usage:

```
result=CalculateDistributionSolventAccessibility(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateT** (*ProteinSequence*)

Calculate all transition descriptors based seven different properties of AADs.

Usage:

```
result=CalculateT(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateTransition** (*ProteinSequence*, *AAPProperty*, *AAPName*)

A method used for computing transition descriptors

Usage:

```
result=CalculateTransition(protein,AAPProperty,AAPName)
```

Input:protein is a pure protein sequence.

AAPProperty is a dict form containing classification of amino acids such as `_Polarizability`.

AAPName is a string used for indicating a AAP name.

CTD. **CalculateTransitionCharge** (*ProteinSequence*)

A method used for calculating Transition descriptors based on Charge of AADs.

Usage:

```
result=CalculateTransitionCharge(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateTransitionHydrophobicity** (*ProteinSequence*)

A method used for calculating Transition descriptors based on Hydrophobicity of AADs.

Usage:

```
result=CalculateTransitionHydrophobicity(protein)
```

Input:protein is a pure protein sequence.

CTD. **CalculateTransitionNormalizedVDWV** (*ProteinSequence*)

A method used for calculating Transition descriptors based on NormalizedVDWV of AADs.

Usage:

```
result=CalculateTransitionNormalizedVDWV(protein)
```

Input:protein is a pure protein sequence.

CTD.CalculateTransitionPolarity (*ProteinSequence*)

A method used for calculating Transition descriptors based on Polarity of AADs.

Usage:

```
result=CalculateTransitionPolarity(protein)
```

Input:protein is a pure protein sequence.

CTD.CalculateTransitionPolarizability (*ProteinSequence*)

A method used for calculating Transition descriptors based on Polarizability of AADs.

Usage:

```
result=CalculateTransitionPolarizability(protein)
```

Input:protein is a pure protein sequence.

CTD.CalculateTransitionSecondaryStr (*ProteinSequence*)

A method used for calculating Transition descriptors based on SecondaryStr of AADs.

Usage:

```
result=CalculateTransitionSecondaryStr(protein)
```

Input:protein is a pure protein sequence.

CTD.CalculateTransitionSolventAccessibility (*ProteinSequence*)

A method used for calculating Transition descriptors based on SolventAccessibility of AADs.

Usage:

```
result=CalculateTransitionSolventAccessibility(protein)
```

Input:protein is a pure protein sequence.

CTD.StringtoNum (*ProteinSequence, AAPProperty*)

Tranform the protein sequence into the string form such as 32123223132121123.

Usage:

```
result=StringtoNum(protein,AAPProperty)
```

Input: protein is a pure protein sequence.

AAPProperty is a dict form containing classifciation of amino acids such as _Polarizability.

4.3.6 ConjointTriad module

protein sequence information. You can get $7*7*7=343$ features. You can freely use and distribute it. If you hava any problem, you could contact with us timely!

Reference:

Juwen Shen, Jian Zhang, Xiaomin Luo, Weiliang Zhu, Kunqian Yu, Kaixian Chen,

Yixue Li, Huanliang Jiang. Predicting proten-protein interactions based only

on sequences inforamtion. PNAS. 2007 (104) 4337-4341.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`ConjointTriad.CalculateConjointTriad` (*proteinsequence*)

Calculate the conjoint triad features from protein sequence.

Usage:

`res = CalculateConjointTriad(protein)`

Input: protein is a pure protein sequence.

Output is a dict form containing all 343 conjoint triad features.

4.3.7 GetProteinFromUniprot module

This module is used to download the protein sequence from the uniprot (<http://www.uniprot.org/>) website. You can only need input a protein ID or prepare a file (ID.txt) related to ID. You can

obtain a .txt (ProteinSequence.txt) file saving protein sequence you need. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`GetProteinFromUniprot.GetProteinSequence` (*ProteinID*)

Get the protein sequence from the uniprot website by ID.

Usage:

`result=GetProteinSequence(ProteinID)`

Input: ProteinID is a string indicating ID such as "P48039".

`GetProteinFromUniprot.GetProteinSequenceFromTxt` (*path, openfile, savefile*)

Get the protein sequence from the uniprot website by the file containing ID.

Usage:

`result=GetProteinSequenceFromTxt(path,openfile,savefile)`

Input: path is a directory path containing the ID file such as "/home/orient/protein/"

openfile is the ID file such as "proteinID.txt"

4.3.8 GetSubSeq module

The prediction of functional sites (e.g.,methylation) of proteins usually needs to split the total protein into a set of segments around specific amino acid. Given a specific window size p , we can obtain all segments of length equal to $(2*p+1)$ very easily. Note that the output of the method is a list form. You can freely use and distribute it. If you have any problem, you could contact with us timely.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`GetSubSeq`. **GetSubSequence** (*ProteinSequence*, *ToAA='S'*, *window=3*)

Get all $2*window+1$ sub-sequences whose center is ToAA in a protein.

Usage:

`result=GetSubSequence(protein,ToAA>window)`

Input:protein is a pure protein sequence.

ToAA is the central (query point) amino acid in the sub-sequence.

window is the span.

4.3.9 ProCheck module

sequence. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`ProCheck`. **ProteinCheck** (*ProteinSequence*)

Check whether the protein sequence is a valid amino acid sequence or not

Usage:

`result=ProteinCheck(protein)`

Input: protein is a pure protein sequence.

Output: if the check is no problem, result will return the length of protein.

4.3.10 PseudoAAC module

Instead of using the conventional 20-D amino acid composition to represent the sample of a protein, Prof. Kuo-Chen Chou proposed the pseudo amino acid (PseAA) composition in order for including the sequence-order information. Based on the concept of Chou's pseudo amino acid composition, the server PseAA was designed in a flexible way, allowing users to generate various kinds of pseudo amino acid composition for a given protein sequence by selecting different parameters and their combinations. This module aims at computing two types of PseAA descriptors: Type I and Type II.

You can freely use and distribute it. If you have any problem, you could contact with us timely.

References:

- [1]: Kuo-Chen Chou. Prediction of Protein Cellular Attributes Using Pseudo-Amino Acid Composition. *PROTEINS: Structure, Function, and Genetics*, 2001, 43: 246-255.
- [2]: <http://www.csbio.sjtu.edu.cn/bioinf/PseAAC/>
- [3]: <http://www.csbio.sjtu.edu.cn/bioinf/PseAAC/type2.htm>
- [4]: Kuo-Chen Chou. Using amphiphilic pseudo amino acid composition to predict enzyme subfamily classes. *Bioinformatics*, 2005,21,10-19.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

The hydrophobicity values are from JACS, 1962, 84: 4240-4246. (C. Tanford).

The hydrophilicity values are from PNAS, 1981, 78:3824-3828 (T.P.Hopp & K.R.Woods).

The side-chain mass for each of the 20 amino acids.

CRC Handbook of Chemistry and Physics, 66th ed., CRC Press, Boca Raton, Florida (1985).

R.M.C. Dawson, D.C. Elliott, W.H. Elliott, K.M. Jones, Data for Biochemical Research 3rd ed., Clarendon Press Oxford (1986).

PseudoAAC.**GetAACComposition** (*ProteinSequence*)

Calculate the composition of Amino acids

for a given protein sequence.

Usage:

result=CalculateAACComposition(protein)

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition of

`PseudoAAC.GetAPseudoAAC (ProteinSequence, lamda=30, weight=0.5)`

Computing all of type II pseudo-amino acid composition descriptors based on the given properties. Note that the number of PAAC strongly depends on the lamda value. if lamda = 20, we can obtain 20+20=40 PAAC descriptors. The size of these values depends on the choice of lamda and weight simultaneously.

Usage:

```
result=GetAPseudoAAC(protein,lamda,weight)
```

Input: protein is a pure protein sequence.

lamda factor reflects the rank of correlation and is a non-Negative integer, such as 15.

Note that (1)lamda should NOT be larger than the length of input protein sequence;

2.lamda must be non-Negative integer, such as 0, 1, 2, ...; (3) when lamda =0, the output of PseAA server is the 20-D amino acid composition.

weight factor is designed for the users to put weight on the additional PseAA components with respect to the conventional AA components. The user can select any value within the region from 0.05 to 0.7 for the weight factor.

`PseudoAAC.GetAPseudoAAC1 (ProteinSequence, lamda=30, weight=0.5)`

Computing the first 20 of type II pseudo-amino acid composition descriptors based on

`PseudoAAC.GetAPseudoAAC2 (ProteinSequence, lamda=30, weight=0.5)`

Computing the last lamda of type II pseudo-amino acid composition descriptors based on

`PseudoAAC.GetCorrelationFunction (Ri='S', Rj='D', AAP=[])`

Computing the correlation between two given amino acids using the given properties.

Usage:

```
result=GetCorrelationFunction(Ri,Rj,AAP)
```

Input: Ri and Rj are the amino acids, respectively.

AAP is a list form containing the properties, each of which is a dict form.

`PseudoAAC.GetPseudoAAC (ProteinSequence, lamda=30, weight=0.05, AAP=[])`

Computing all of type I pseudo-amino acid composition descriptors based on the given properties. Note that the number of PAAC strongly depends on the lamda value. if lamda = 20, we can obtain 20+20=40 PAAC descriptors. The size of these values depends on the choice of lamda and weight simultaneously. You must specify some properties into AAP.

Usage:

```
result=GetPseudoAAC(protein,lamda,weight)
```

Input: protein is a pure protein sequence.

lamda factor reflects the rank of correlation and is a non-Negative integer, such as 15.

Note that (1)lamda should NOT be larger than the length of input protein sequence;

2.lamda must be non-Negative integer, such as 0, 1, 2, ...; (3) when lamda =0, the

output of PseAA server is the 20-D amino acid composition.

weight factor is designed for the users to put weight on the additional PseAA components with respect to the conventional AA components. The user can select any value within the region from 0.05 to 0.7 for the weight factor.

AAP is a list form containing the properties, each of which is a dict form.

`PseudoAAC.GetPseudoAAC1 (ProteinSequence, lamda=30, weight=0.05, AAP=[])`

Computing the first 20 of type I pseudo-amino acid composition descriptors based on the given

`PseudoAAC.GetPseudoAAC2 (ProteinSequence, lamda=30, weight=0.05, AAP=[])`

Computing the last lamda of type I pseudo-amino acid composition descriptors based on the given

`PseudoAAC.GetSequenceOrderCorrelationFactor (ProteinSequence, k=1, AAP=[])`

Computing the Sequence order correlation factor with gap equal to k based on the given properties.

Usage:

`result=GetSequenceOrderCorrelationFactor(protein,k,AAP)`

Input: protein is a pure protein sequence.

k is the gap.

AAP is a list form containing the properties, each of which is a dict form.

`PseudoAAC.GetSequenceOrderCorrelationFactorForAPAAC (ProteinSequence, k=1)`

Computing the Sequence order correlation factor with gap equal to k based on

[_Hydrophobicity,_hydrophilicity] for APAAC (type II PseAAC) .

Usage:

`result=GetSequenceOrderCorrelationFactorForAPAAC(protein,k)`

Input: protein is a pure protein sequence.

k is the gap.

`PseudoAAC.NormalizeEachAAP (AAP)`

All of the amino acid indices are centralized and standardized before the calculation.

Usage:

`result=NormalizeEachAAP(AAP)`

Input: AAP is a dict form containing the properties of 20 amino acids.

Output: result is the a dict form containing the normalized properties

4.3.11 PyProteinAACComposition module

Created on Thu Jun 02 10:00:35 2016

@author: yzj

`PyProteinAACComposition.CalculateAACComposition (ProteinSequence)`

Calculate the composition of Amino acids for a given protein sequence.

Usage:

```
result=CalculateAAComposition(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition of

`PyProteinAAComposition.CalculateAADipeptideComposition (ProteinSequence)`

Calculate the composition of AADs, dipeptide and 3-mers for a given protein sequence.

Usage:

```
result=CalculateAADipeptideComposition(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing all composition values of

`PyProteinAAComposition.CalculateDipeptideComposition (ProteinSequence)`

Calculate the composition of dipeptide for a given protein sequence.

Usage:

```
result=CalculateDipeptideComposition(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition of

`PyProteinAAComposition.GetSpectrumDict (proteinsequence)`

Calculate the spectrum descriptors of 3-mers for a given protein.

Usage:

```
result=GetSpectrumDict(protein)
```

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition values of 8000

`PyProteinAAComposition.Getkmers ()`

Get the amino acid list of 3-mers.

Usage:

```
result=Getkmers()
```

Output: result is a list form containing 8000 tri-peptides.

4.3.12 PyProteinAAIndex module

Created on Thu Jun 02 15:38:19 2016

@author: yzj

`PyProteinAAIndex.GetAAIndex1 (name, path='.')`

Get the amino acid property values from aaindex1

Usage:

```
result=GetAAIndex1(name)
```

Input: name is the name of amino acid property (e.g., KRIW790103)

Output: result is a dict form containing the properties of 20 amino acids

`PyProteinAAIndex.GetAAIndex23 (name, path='.')`

Get the amino acid property values from aaindex2 and aaindex3

Usage:

`result=GetAAIndex23(name)`

Input: name is the name of amino acid property (e.g.,TANS760101,GRAR740104)

Output: result is a dict form containing the properties of 400 amino acid pairs

class `PyProteinAAIndex.MatrixRecord`

Bases: `PyProteinAAIndex.Record`

Matrix record for mutation matrices or pair-wise contact potentials

extend (*row*)

get (*aai, aaj, d=None*)

median ()

class `PyProteinAAIndex.Record`

Amino acid index (AAindex) Record

aakeys = 'ARNDCQEGHILKMFPSTWYV'

extend (*row*)

get (*aai, aaj=None, d=None*)

median ()

`PyProteinAAIndex.get (key)`

Get record for key

`PyProteinAAIndex.grep (pattern)`

Search for pattern in title and description of all records (case insensitive) and print results on standard output.

`PyProteinAAIndex.init (path=None, index='123')`

Read in the aaindex files. You need to run this (once) before you can access any records. If the files are not within the current directory, you need to specify the correct directory path. By default all three aaindex files are read in.

`PyProteinAAIndex.init_from_file (filename, type=<class PyProteinAAIndex.Record>)`

`PyProteinAAIndex.search (pattern, searchtitle=True, casesensitive=False)`

Search for pattern in description and title (optional) of all records and return matched records as list. By default search case insensitive.

4.3.13 QuasiSequenceOrder module

given protein sequence. We can obtain two types of descriptors: Sequence-order-coupling number and quasi-sequence-order descriptors. Two distance matrixes between 20 amino acids are employed. You can freely use and distribute it. If you have any problem, please contact us immediately.

References:

[1]:Kuo-Chen Chou. Prediction of Protein Subcellar Locations by Incorporating

Quasi-Sequence-Order Effect. Biochemical and Biophysical Research Communications

2000, 278, 477-483.

[2]: Kuo-Chen Chou and Yu-Dong Cai. Prediction of Protein subcellular locations by GO-FunD-PseAA predictor, Biochemical and Biophysical Research Communications, 2004, 320, 1236-1239.

[3]: Gisbert Schneider and Paul Wrede. The Rational Design of Amino Acid Sequences by Artificial Neural Networks and Simulated Molecular Evolution: Do Novo Design of an Idealized Leader Cleavage Site. Biophys Journal, 1994, 66, 335-344.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`QuasiSequenceOrder.GetAAComposition (ProteinSequence)`

Calculate the composition of Amino acids

for a given protein sequence.

Usage:

`result=CalculateAAComposition(protein)`

Input: protein is a pure protein sequence.

Output: result is a dict form containing the composition of

`QuasiSequenceOrder.GetQuasiSequenceOrder (ProteinSequence, maxlag=30, weight=0.1)`

Computing quasi-sequence-order descriptors for a given protein.

[1]: Kuo-Chen Chou. Prediction of Protein Subcellular Locations by Incorporating Quasi-Sequence-Order Effect. Biochemical and Biophysical Research Communications 2000, 278, 477-483.

Usage:

`result = GetQuasiSequenceOrder(protein,maxlag,weight)`

Input: protein is a pure protein sequence

maxlag is the maximum lag and the length of the protein should be larger than maxlag. default is 30.

weight is a weight factor. please see reference 1 for its choice. default is 0.1.

`QuasiSequenceOrder.GetQuasiSequenceOrder1 (ProteinSequence, maxlag=30, weight=0.1, distanceMatrix={})`

Computing the first 20 quasi-sequence-order descriptors for a given protein sequence.

Usage:

`result = GetQuasiSequenceOrder1(protein,maxlag,weight)`

`QuasiSequenceOrder.GetQuasiSequenceOrder1Grant` (*ProteinSequence*, *maxlag=30*,
weight=0.1, *distancematrix=Distancematrix*)

Computing the first 20 quasi-sequence-order descriptors for
a given protein sequence.

Usage:

```
result = GetQuasiSequenceOrder1Grant(protein,maxlag,weight)
```

`QuasiSequenceOrder.GetQuasiSequenceOrder1SW` (*ProteinSequence*, *maxlag=30*, *weight=0.1*,
distancematrix=Distancematrix)

Computing the first 20 quasi-sequence-order descriptors for
a given protein sequence.

Usage:

```
result = GetQuasiSequenceOrder1SW(protein,maxlag,weight)
```

`QuasiSequenceOrder.GetQuasiSequenceOrder2` (*ProteinSequence*, *maxlag=30*, *weight=0.1*, *distancematrix={}*)

Computing the last maxlag quasi-sequence-order descriptors for
a given protein sequence.

Usage:

```
result = GetQuasiSequenceOrder2(protein,maxlag,weight)
```

`QuasiSequenceOrder.GetQuasiSequenceOrder2Grant` (*ProteinSequence*, *maxlag=30*,
weight=0.1, *distancematrix=Distancematrix*)

Computing the last maxlag quasi-sequence-order descriptors for
a given protein sequence.

Usage:

```
result = GetQuasiSequenceOrder2Grant(protein,maxlag,weight)
```

`QuasiSequenceOrder.GetQuasiSequenceOrder2SW` (*ProteinSequence*, *maxlag=30*, *weight=0.1*,
distancematrix=Distancematrix)

Computing the last maxlag quasi-sequence-order descriptors for
a given protein sequence.

Usage:

```
result = GetQuasiSequenceOrder2SW(protein,maxlag,weight)
```

`QuasiSequenceOrder.GetQuasiSequenceOrderp` (*ProteinSequence*, *maxlag=30*, *weight=0.1*, *distancematrix={}*)

Computing quasi-sequence-order descriptors for a given protein.

[1]:Kuo-Chen Chou. Prediction of Protein Subcellar Locations by

Incorporating Quasi-Sequence-Order Effect. Biochemical and Biophysical

Research Communications 2000, 278, 477-483.

Usage:

```
result = GetQuasiSequenceOrderp(protein,maxlag,weight,distancematrix)
```

Input: protein is a pure protein sequence

maxlag is the maximum lag and the length of the protein should be larger

than maxlag. default is 30.

weight is a weight factor. please see reference 1 for its choice. default is 0.1.

distancematrix is a dict form containing 400 distance values

Output: result is a dict form containing all quasi-sequence-order descriptors

`QuasiSequenceOrder.GetSequenceOrderCouplingNumber` (*ProteinSequence*, *d=1*, *distancematrix=Distancematrix*)

Computing the dth-rank sequence order coupling number for a protein.

Usage:

```
result = GetSequenceOrderCouplingNumber(protein,d)
```

Input: protein is a pure protein sequence.

d is the gap between two amino acids.

Output: result is numeric value.

`QuasiSequenceOrder.GetSequenceOrderCouplingNumberGrant` (*ProteinSequence*,
maxlag=30, *distancematrix=Distancematrix*)

Computing the sequence order coupling numbers from 1 to maxlag
for a given protein sequence based on the Grantham chemical distance
matrix.

Usage:

```
result = GetSequenceOrderCouplingNumberGrant(protein, maxlag,distancematrix)
```

Input: protein is a pure protein sequence

maxlag is the maximum lag and the length of the protein should be larger
than maxlag. default is 30.

distancematrix is a dict form containing Grantham chemical distance
matrix. omitted!

Output: result is a dict form containing all sequence order coupling numbers

`QuasiSequenceOrder.GetSequenceOrderCouplingNumberSW` (*ProteinSequence*,
maxlag=30, *distancematrix=Distancematrix*)

Computing the sequence order coupling numbers from 1 to maxlag
for a given protein sequence based on the Schneider-Wrede physicochemical
distance matrix

Usage:

```
result = GetSequenceOrderCouplingNumberSW(protein, maxlag,distancematrix)
```

Input: protein is a pure protein sequence

maxlag is the maximum lag and the length of the protein should be larger
than maxlag. default is 30.

distancematrix is a dict form containing Schneider-Wrede physicochemical
distance matrix. omitted!

Output: result is a dict form containing all sequence order coupling numbers based

`QuasiSequenceOrder.GetSequenceOrderCouplingNumberTotal` (*ProteinSequence*,
maxlag=30)

Computing the sequence order coupling numbers from 1 to maxlag for a given protein sequence. Usage: result = GetSequenceOrderCouplingNumberTotal(protein, maxlag)

Input: protein is a pure protein sequence

maxlag is the maximum lag and the length of the protein should be larger

than maxlag. default is 30.

`QuasiSequenceOrder.GetSequenceOrderCouplingNumberp` (*ProteinSequence*, *maxlag*=30, *distancematrix*={})

Computing the sequence order coupling numbers from 1 to maxlag

for a given protein sequence based on the user-defined property.

Usage:

```
result = GetSequenceOrderCouplingNumberp(protein, maxlag,distancematrix)
```

Input: protein is a pure protein sequence

maxlag is the maximum lag and the length of the protein should be larger

than maxlag. default is 30.

distancematrix is the a dict form containing 400 distance values

Output: result is a dict form containing all sequence order coupling numbers based

4.4 PyPretreat

4.4.1 PyPretreatDNA module

Created on Wed May 18 14:06:37 2016

@author: yzj

```
PyPretreatDNA.ALPHABET = 'ACGT'
```

Used for process original data.

PyPretreatDNA.**ConvertPhycheIndexToDict** (*phyche_index*)

PyPretreatDNA.DNAChecks (s)

PyPretreatDNA.**Frequency** (*tol_str*, *tar_str*)

Generate the frequency of tar_str in tol_str.

Parameters

- **tol_str** – mother string.
- **tar_str** – substring.

```
PyPretreatDNA.GeneratePhycheValue(k, phyche_index=None, all_property=False, ex-
tra_phyche_index=None)
```

Combine the user selected `pyche_list`, `is_all_property` and `extra_pyche_index` to a new standard `pyche` value. #####

PyPretreatDNA.**GetData** (*input_data*, *desc=False*)

Get sequence data from file or list with check.

Parameters

- **input_data** – type file or list
- **desc** – with this option, the return value will be a Seq object list(it only works in file object).

Returns sequence data or shutdown.

`PyPretreatDNA.GetSequenceCheckDna(f)`

Read the fasta file.

Input: f: HANDLE to input. e.g. sys.stdin, or open(<file>)

`PyPretreatDNA.IsFasta(seq)`

Judge the Seq object is in FASTA format. Two situation: 1. No seq name. 2. Seq name is illegal. 3. No sequence.

Parameters **seq** – Seq object.

`PyPretreatDNA.IsSequenceList(sequence_list)`

Judge the sequence list is within the scope of alphabet and change the lowercase to capital.

#####

`PyPretreatDNA.IsUnderAlphabet(s, alphabet)`

Judge the string is within the scope of the alphabet or not.

Parameters

- **s** – The string.
- **alphabet** – alphabet.

`PyPretreatDNA.NormalizeIndex(phyche_index, is_convert_dict=False)`

`PyPretreatDNA.ReadFasta(f)`

Read a fasta file.

Parameters **f** – HANDLE to input. e.g. sys.stdin, or open(<file>)

`PyPretreatDNA.ReadFastaCheckDna(f)`

Read the fasta file, and check its legality.

Parameters **f** – HANDLE to input. e.g. sys.stdin, or open(<file>)

`PyPretreatDNA.ReadFastaYield(f)`

Yields a Seq object.

Parameters **f** – HANDLE to input. e.g. sys.stdin, or open(<file>)

`class PyPretreatDNA.Seq(name, seq, no)`

`PyPretreatDNA.StandardDeviation(value_list)`

`PyPretreatDNA.WriteLibsvm(vector_list, label_list, write_file)`

4.4.2 PyPretreatMol module

`PyPretreatMol.StandardMol(mol)`

The function for performing standardization of molecules and deriving parent molecules. The function contains derive fragment, charge, tautomer, isotope and stereo parent molecules. The primary usage is:

```
mol1 = Chem.MolFromSmiles('C1=CC=CC=C1')
mol2 = s.standardize(mol1)
```

PyPretreatMol.**StandardSmi** (*smi*)

The function for performing standardization of molecules and deriving parent molecules. The function contains derive fragment, charge, tautomer, isotope and stereo parent molecules. The primary usage is:

```
smi = StandardSmi('C[n+]1c([N-](C))cccc1')
```

```

class PyPretreatMol.StandardizeMol (normalizations=(Normalization(u'Nitro to N+(O-)=O',
u'[*:1][N,P,As,Sb:2](=[O,S,Se,Te:3])=[O,S,Se,Te:4]>>[*:1][*+1:2]([*-
1:3])=[*:4]'), Normalization(u'Sulfone to S(=O)(=O)',
u'[S+2:1]([O-:2])([O-:3])>>[S+0:1]([O-
0:2])(=[O-0:3]')), Normalization(u'Pyridine ox-
ide to n+O-', u'[n:1]=[O:2]>>[n+:1][O-
:2]'), Normalization(u'Azide to N=N+=N-',
u'[*:H:1][N:2]=[N:3]#[N:4]>>[*:H:1][N:2]=[N+:3]=[N-
:4]'), Normalization(u'Diazo/azo to =N+=N-
', u'[*:1]=[N:2]#[N:3]>>[*:1]=[N+:2]=[N-
:3]'), Normalization(u'Sulfoxide to -S+(O-)-',
u'![O:1][S+0:X3:2]([O:3])![O:4]>>[*:1][S+1:2]([O-
:3])[*:4]'), Normalization(u'Phosphate to P(O-
)=O', u'[O,S,Se,Te;-1:1][P+;D4:2][O,S,Se,Te;-
1:3]>>[*+0:1]=[P+0;D5:2][*-1:3]'), Nor-
malization(u'Amidinium to C(=NH2+)NH2',
u'[C,S;X3+1:1]([NX3!H0:3])>>[*+0:1]([N:2])=[N+:3]'),
Normalization(u'Normalize hydrazine-
diazonium', u'[CX4:1][NX3H:2]-
[NX3H:3][CX4:4][NX2+:5]#[NX1:6]>>[CX4:1][NH0:2]=[NH+:3][C:4][N+0:5]=
Normalization(u'Recombine 1,3-separated
charges', u'[N,P,As,Sb,O,S,Se,Te;-1:1]-
[A:2]=[N,P,As,Sb,O,S,Se,Te;+1:3]>>[*-0:1]=[*:2]-
[*+0:3]'), Normalization(u'Recombine 1,3-separated
charges', u'[n,o,p,s;-1:1]:[a:2]=[N,O,P,S;+1:3]>>[*-
0:1]:[*:2]-[*+0:3]'), Normalization(u'Recombine
1,3-separated charges', u'[N,O,P,S;-1:1]-
[a:2]:[n,o,p,s;+1:3]>>[*-0:1]:[*:2]:[*+0:3]'), Nor-
malization(u'Recombine 1,5-separated charges',
u'[N,P,As,Sb,O,S,Se,Te;-1:1]-[A+0:2]=[A:3]-
[A:4]=[N,P,As,Sb,O,S,Se,Te;+1:5]>>[*-
0:1]:[*:2]-[*:3]:[*:4]-[*+0:5]'), Normaliza-
tion(u'Recombine 1,5-separated charges', u'[n,o,p,s;-
1:1]:[a:2]:[a:3]:[c:4]=[N,O,P,S;+1:5]>>[*-
0:1]:[*:2]:[*:3]:[c:4]-[*+0:5]'), Normaliza-
tion(u'Recombine 1,5-separated charges', u'[N,O,P,S;-
1:1]-[c:2]:[a:3]:[a:4]:[n,o,p,s;+1:5]>>[*-
0:1]=[c:2]:[*:3]:[*:4]:[*+0:5]'), Normaliza-
tion(u'Normalize 1,3 conjugated cation', u'[N,O;+0!H0:1]-
[A:2]=[N!$(*[O-]),O;+1H0:3]>>[*+1:1]:[*:2]-
[*+0:3]'), Normalization(u'Normalize 1,3 con-
jugated cation', u'[n;+0!H0:1]:[c:2]=[N!$(*[O-
]),O;+1H0:3]>>[*+1:1]:[*:2]-[*+0:3]'), Normaliza-
tion(u'Normalize 1,3 conjugated cation', u'[N,O;+0!H0:1]-
[c:2]:[n!$(*[O-]),o;+1H0:3]>>[*+1:1]:[*:2]:[*+0:3]'),
Normalization(u'Normalize 1,5 conjugated cation',
u'[N,O;+0!H0:1]-[A:2]=[A:3]-[A:4]=[N!$(*[O-
]),O;+1H0:5]>>[*+1:1]:[*:2]-[*:3]:[*:4]-
[*+0:5]'), Normalization(u'Normalize 1,5 conjugated
cation', u'[n;+0!H0:1]:[a:2]:[a:3]:[c:4]=[N!$(*[O-
]),O;+1H0:5]>>[n+1:1]:[*:2]:[*:3]:[*:4]-[*+0:5]'),
Normalization(u'Normalize 1,5 conjugated cation',
u'[N,O;+0!H0:1]-[c:2]:[a:3]:[a:4]:[n!$(*[O-
]),o;+1H0:5]>>[*+1:1]:[c:2]:[*:3]:[*:4]:[*+0:5]'),
Normalization(u'Normalize 1,5 conjugated cation',
u'[n;+0!H0:1]:[a:2]:[a:3]:[a:4]:[n!$(*[O-
]),+1H0:5]>>[n+1:1]:[*:2]:[*:3]:[*:4]:[n+0:5]!'),
Normalization(u'Normalize 1,5 conjugated
cation', u'[n;+0!H0:1]:[a:2]:[a:3]:[a:4]:[n!$(*[O-
]),+1H0:5]>>[n+1:1]:[*:2]:[*:3]:[*:4]:[n+0:5]'),

```

Bases: `object`

The main class for performing standardization of molecules and deriving parent molecules.

The primary usage is via the `standardize()` method:

```
s = Standardizer()
mol1 = Chem.MolFromSmiles('C1=CC=CC=C1')
mol2 = s.standardize(mol1)
```

There are separate methods to derive fragment, charge, tautomer, isotope and stereo parent molecules.

addhs (*mol*)

canonicalize_tautomer

Returns A callable TautomerCanonicalizer instance.

disconnect_metals

Returns A callable MetalDisconnector instance.

largest_fragment

Returns A callable LargestFragmentChooser instance.

normalize

Returns A callable Normalizer instance.

reionize

Returns A callable Reionizer instance.

rmhs (*mol*)

uncharge

Returns A callable Uncharger instance.

`PyPretreatMol.ValidatorMol` (*mol*)

Return log messages for a given SMILES string using the default validations.

Note: This is a convenience function for quickly validating a single SMILES string.

Parameters **smiles** (*string*) – The SMILES for the molecule.

Returns A list of log messages.

Return type list of strings.

`PyPretreatMol.ValidatorSmi` (*smi*)

Return log messages for a given SMILES string using the default validations.

Note: This is a convenience function for quickly validating a single SMILES string.

Parameters **smiles** (*string*) – The SMILES for the molecule.

Returns A list of log messages.

Return type list of strings.

4.4.3 PyPretreatMolutil module

Created on Wed Jun 15 10:13:55 2016

@author: yzj molvs.tautomer ~~~~~

This module contains tools for enumerating tautomers and determining a canonical tautomer.

copyright Copyright 2014 by Matt Swain.

license MIT, see LICENSE file for more details.

4.4.4 PyPretreatPro module

sequence. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`PyPretreatPro.ProteinCheck` (*ProteinSequence*)

Check whether the protein sequence is a valid amino acid sequence or not

Usage:

`result=ProteinCheck(protein)`

Input: protein is a pure protein sequence.

Output: if the check is no problem, result will return the length of protein.

4.5 PyInteraction

4.5.1 PyInteraction module

The calculation of interaction descriptors. You can choose three types of interaction descriptors. You can freely use and distribute it. If you have any problem, you could contact with us timely!

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.14

Email: gadsby@163.com

`PyInteraction.CalculateInteraction1` (*dict1={}, dict2={}*)

Calculate the two interaction features by combining two different features.

Usage:

`res=CalculateInteraction(dict1,dict2)`

Input: dict1 is a dict form containing features.

dict2 is a dict form containing features.

Output: res is a dict form containing interaction features.

`PyInteraction.CalculateInteraction2 (dict1={}, dict2={})`

Calculate the two interaction features by combining two different features.

Usage:

`res=CalculateInteraction(dict1,dict2)`

Input: dict1 is a dict form containing features.

dict2 is a dict form containing features.

Output: res is a dict form containing interaction features.

`PyInteraction.CalculateInteraction3 (dict1={}, dict2={})`

Calculate the two interaction features by

$F=[Fa(i)+Fb(i)),Fa(i)*Fb(i)]$ (2n)

It's used in same type of descriptors.

Usage:

`res=CalculateInteraction(dict1,dict2)`

Input: dict1 is a dict form containing features.

dict2 is a dict form containing features.

Output: res is a dict form containing interaction features.

4.6 PyGetMol

4.6.1 GetProtein module

Created on Sat Jul 13 11:18:26 2013

This module is used for downloading the PDB file from RCSB PDB web and extract its amino acid sequence. This module can also download the protein sequence from the uniprot (<http://www.uniprot.org/>) website. You can only need input a protein ID or prepare a file (ID.txt) related to ID. You can obtain a .txt (ProteinSequence.txt) file saving protein sequence you need.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`GetProtein.GetPDB (pdbidlist=[])`

Download the PDB file from PDB FTP server by providing a list of pdb id.

`GetProtein.GetProteinSequence (ProteinID)`

Get the protein sequence from the uniprot website by ID.

Usage:


```
result=GetProteinSequence(ProteinID)
```

Input: ProteinID is a string indicating ID such as “P48039”.

```
GetProtein.GetProteinSequenceFromTxt (path, openfile, savefile)
```

Get the protein sequence from the uniprot website by the file containing ID.

Usage:

```
result=GetProteinSequenceFromTxt(path,openfile,savefile)
```

Input: path is a directory path containing the ID file such as “/home/orient/protein/”

openfile is the ID file such as “proteinID.txt”

```
GetProtein.GetSeqFromPDB (pdbfile='')
```

Get the amino acids sequences from pdb file.

```
GetProtein.IsFasta (seq)
```

Judge the Seq object is in FASTA format. Two situation: 1. No seq name. 2. Seq name is illegal. 3. No sequence.

Parameters *seq* – Seq object.

```
GetProtein.ReadFasta (f)
```

Read a fasta file.

Parameters *f* – HANDLE to input. e.g. sys.stdin, or open(<file>)

```
class GetProtein.Seq (name, seq, no)
```

```
GetProtein.pdbDownload (file_list, hostname='ftp.wwpdb.org', directory='/pub/pdb/data/structures/all/pdb/', prefix='pdb', suffix='.ent.gz')
```

Download all pdb files in file_list and unzip them.

```
GetProtein.pdbSeq (pdb, use_atoms=False)
```

Parse the SEQRES entries in a pdb file. If this fails, use the ATOM entries. Return dictionary of sequences keyed to chain and type of sequence used.

```
GetProtein.unZip (some_file, some_output)
```

Unzip some_file using the gzip library and write to some_output.

4.6.2 GetDNA module

This module is used for downloading the DNA sequence from ncbi web. You can only need input a DNA ID.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.11.04

Email: gadsby@163.com

```
GetDNA.GetDNAFromUniGene (SeqID='')
```

This module is used for downloading the DNA sequence from ncbi web. You can only need input a DNA ID.

```
GetDNA.IsFasta (seq)
```

Judge the Seq object is in FASTA format. Two situation: 1. No seq name. 2. Seq name is illegal. 3. No sequence.

Parameters *seq* – Seq object.

`GetDNA.IsUnderAlphabet` (*s*, *alphabet*)

Judge the string is within the scope of the alphabet or not.

Parameters

- **s** – The string.
- **alphabet** – alphabet.

`GetDNA.ReadFasta` (*f*)

Read a fasta file.

Parameters **f** – HANDLE to input. e.g. sys.stdin, or open(<file>)

class `GetDNA.Seq` (*name*, *seq*, *no*)

4.6.3 Getmol module

This module is to get different formats of molecules from file and web. If you have any question please contact me via email.

Authors: Zhijiang Yao and Dongsheng Cao.

Date: 2016.06.04

Email: gadsby@163.com

`Getmol.GetMolFromCAS` (*casid*='')

Downloading the molecules from <http://www.chemnet.com/cas/> by CAS ID (*casid*). if you want to use this function, you must be install pybel.

`Getmol.GetMolFromDrugbank` (*dbid*='')

Downloading the molecules from <http://www.drugbank.ca/> by *dbid* (*dbid*).

`Getmol.GetMolFromEBI` ()

`Getmol.GetMolFromKegg` (*kid*='')

Downloading the molecules from <http://www.genome.jp/> by kegg id (*kid*).

`Getmol.GetMolFromNCBI` (*cid*='')

Downloading the molecules from <http://pubchem.ncbi.nlm.nih.gov/> by *cid* (*cid*).

`Getmol.ReadMolFromInchi` (*inchi*='')

Read a molecule by Inchi string.

Usage:

`res=ReadMolFromInchi(inchi)`

Input: *inchi* is a InChi string.

Output: *res* is a molecule object.

`Getmol.ReadMolFromMOL` (*filename*='')

Read a molecule by mol file format.

Usage:

`res=ReadMolFromMOL(filename)`

Input: *filename* is a file name with path.

Output: *res* is a molecular object.

`Getmol.ReadMolFromMol (filename='')`

Read a molecule with mol file format.

Usage:

`res=ReadMolFromMol(filename)`

Input: filename is a file name.

Output: res is a molecule object.

`Getmol.ReadMolFromSDF (filename='')`

Read a set of molecules by SDF file format.

Note: the output of this function is a set of molecular objects.

You need to use for statement to call each object.

Usage:

`res=ReadMolFromSDF(filename)`

Input: filename is a file name with path.

Output: res is a set of molecular object.

`Getmol.ReadMolFromSmile (smi='')`

Read a molecule by SMILES string.

Usage:

`res=ReadMolFromSmile(smi)`

Input: smi is a SMILES string.

Output: res is a molecule object.

4.7 test package

4.7.1 test module

4.7.2 test_PyBioMed module

Created on Mon Oct 24 09:05:01 2016

@author: Gadsby

`test_PyBioMed.test_pybiomed()`

4.7.3 test_PyDNA module

Created on Mon Oct 24 09:16:55 2016

@author: Gadsby

`test_PyDNA.test_pydna()`

4.7.4 test_PyInteration module

Created on Thu Nov 03 15:08:29 2016

@author: Gadsby

```
test_PyInteration.test_pyinteration()
```

4.7.5 test_PyMolecule module

Created on Thu Oct 27 11:13:27 2016

@author: Gadsby

```
test_PyMolecule.test_pymolecule()
```

4.7.6 test_PyPretreat module

Created on Mon Oct 31 14:12:12 2016

@author: Gadsby

```
test_PyPretreat.test_pypretreat()
```

4.7.7 test_PyProtein module

Created on Thu Oct 27 09:38:03 2016

@author: Gadsby

```
test_PyProtein.test_pyprotein()
```

4.7.8 test_PyGetMol module

Created on Tue Oct 25 15:07:23 2016

@author: Gadsby

```
test_PyGetMol.test_pygetmol()
```

5.1 Requirements for testing

PyBioMed requires RDKit and pybel packages. If you don't already have the packages installed, follow the directions here https://openbabel.org/docs/dev/UseTheLibrary/Python_Pybel.html

<http://www.rdkit.org/>

5.2 Testing an installed package

If you have a file-based (not a Python egg) installation you can test the installed package with

```
>>> from PyBioMed.test import test_PyBioMed
>>> test_PyBioMed.test_pybiomed()
```


DOWNLOAD

6.1 Python Package

Source and binary releases: <https://pypi.python.org/pypi/pybiomed/>

Github (latest development): <https://github.com/gadsbyfly/PyBioMed/>

6.2 Documentation

PDF

[https://github.com/gadsbyfly/PyBioMed/blob/master/doc/Descriptor introduction/PyBioMedDocumentation.pdf](https://github.com/gadsbyfly/PyBioMed/blob/master/doc/Descriptor%20introduction/PyBioMedDocumentation.pdf)

6.3 The introduction of descriptors

PDF

6.3.1 Molecular descriptors introduction

<https://github.com/gadsbyfly/PyBioMed/blob/master/doc/DescriptorIntroduction/PyBioMedChem.pdf>

6.3.2 Protein descriptors introduction

<https://github.com/gadsbyfly/PyBioMed/blob/master/doc/DescriptorIntroduction/PyBioMedProtein.pdf>

6.3.3 DNA descriptors introduction

<https://github.com/gadsbyfly/PyBioMed/blob/master/doc/DescriptorIntroduction/PyBioMedDNA.pdf>

6.3.4 Interaction descriptors introduction

<https://github.com/gadsbyfly/PyBioMed/blob/master/doc/DescriptorIntroduction/PyBioMedInteraction.pdf>

HTML in zip file

<https://github.com/gadsbyfly/PyBioMed/blob/master/doc/DescriptorIntroduction/PyBioMedDocumentation.zip>