# BioMedR: R/CRAN Package for generating various molecular representations for chemicals, proteins, DNAs/RNAs and their interactions

Minfeng Zhu, Jie Dong, Dongsheng Cao

Package Version: Release 1
2019-07-03

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

# Contents

# 1. Introduction

The **BioMedR** package presented in this manual offers an R/CRAN package for generating various molecular representations for chemicals, proteins, DNAs/RNAs and their interactions.

**BioMedR** implemented and integrated the state-of-the-art protein sequence descriptors, molecular descriptors/fingerprints and DNA/RNAdescriptors with R. For small molecules, the **BioMedR** package could

- Calculate 307 molecular descriptors (2D/3D), including constitutional, topological, geometrical, and electronic descriptors, etc.
- Calculate more than ten types of molecular fingerprints, including FP4 keys, E-state fingerprints, MACCS keys, etc., and parallelized chemical similarity search.
- Parallelized pairwise similarity computation derived by fingerprints and maximum common substructure search within a list of small molecules.
- Clustering of Molecules Based on Structural Similarities.

For protein sequences, the **BioMedR** package could

- Calculate six protein descriptor groups composed of fourteen types of commonly used structural and physicochemical descriptors that include 9,920 descriptors.
- Calculate profile-based protein representation derived by PSSM (Position-Specific Scoring Matrix).
- Calculate six types of generalized scales-based descriptors derived by various dimensionality reduction methods for proteochemometric (PCM) modeling.
- Parallellized pairwise similarity computation derived by protein sequence alignment and Gene Ontology (GO) semantic similarity measures within a list of proteins.

For DNA sequences, the **BioMedR** package could

- Calculate three nucleic acid composition features describing the local sequence information by means of kmers (subsequences of DNA sequences);
- Calculate six autocorrelation features describing the level of correlation between two oligonucleotides along a DNA sequence in terms of their specific physicochemical properties;
- Calculate two 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.
- Parallellized pairwise similarity computation derived by protein sequence alignment and Gene Ontology (GO) semantic similarity measures within a list of proteins.

By combining various types of descriptors for drugs, proteins and DNA/RNA in different methods, interaction descriptors representing protein-protein, compound-compound, DNA-DNA, compound-DNA compound-protein and DNA-protein interactions could be conveniently generated with **BioMedR**, including:

- Two types of compound-protein interaction (CPI) descriptors
- Two types of compound-DNA interaction (CDI) descriptors
- Two types of DNA-protein interaction (DPI) descriptors
- Three types of protein-protein interaction (PPI) descriptors
- Three types of compound-compound interaction (CCI) descriptors
- Three types of DNA-DNA interaction (DDI) descriptors

Several useful auxiliary utilities are also shipped with **BioMedR**:

- Parallelized molecule, protein and DNA sequence retrieval from several online databases, like PubChem, ChEMBL, KEGG, DrugBank, UniProt, RCSB PDB, genBank, etc.
- Loading molecules stored in SMILES/SDF files and loading protein/DNA/RNA sequences from FASTA/PDB files
- Molecular file format conversion

The computed protein sequence descriptors, molecular descriptors/fingerprints, DNA/RNA sequence descriptors, interaction descriptors and pairwise similarities are widely used in various research fields relevant to drug disvery, primarily bioinformatics, chemoinformatics, proteochemometrics and chemogenomics. In this part, there are several examples using aforementioned descriptors and their detailed information are as follows:

- Regression Modeling in QSAR Study of logD. A prediction model for logD based on molecular descriptors.
- Classification Modeling in QSAR Study of hERG. A prediction model for hERG based on molecular fingerprint.
- Chemical Similarity Searching. A similarity searching study for bcl2 database.
- Clustering of Molecules Based on Structural Similarity. A clustering process was carried out for bcl2 and resuled in a heatmap and a hierarchical cluster result.
- Predicting Protein Subcellular Localization. The protein descriptors and random forest were used to build a model for protein subcellular localization.
- Predicting nucleosome positioning in genomes. The DNA descriptors and random forest were applied to construct the prediction model for the nucleosome positioning in genomes.

- Predicting Drug-Target Interaction by Integrating Chemical and Genomic Spaces.

To install the **BioMedR** package in R, simply type

```
install.packages('BioMedR')
```

Several dependencies of the **BioMedR** package may require some system-level libraries, check the corresponding manuals of these packages for detailed installation guides.

## 2. Miscellaneous Tools

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

### 2.1. Retrieve small molecules from PubChem, ChEMBL, CAS, KEGG, DrugBank

This function `BMgetDrugKEGG()` get drug molecules from KEGG by drug ID(s), The input `ID` is a character vector specifying the drug ID(s). The returned sequences are stored in a list:

```
> ids = c('D00496', 'D00411')
> drugseq = BMgetDrugSmiKEGG(ids)
> print(drugseq)

[1] "C(C(C(S)(C)C)N)(O)=O"
[2] "C=12C(CC=3C(=CC(C(NS(C=4C(=CC=CC4)C)(=O)=O)=O)=CC3)OC)=CN(C1C=CC(=C2)NC
(OC5CCCC5)=O)C"
```

If the connection is slow or accidentally interrupts, just try more times until success.

The functions in **BioMedR** named after `BMgetDrugMol...()` and `BMgetDrugSmi...()` supports the parallelized retrieval of (drug) molecules from PubChem, ChEMBL, CAS, KEGG, and DrugBank.

### 2.2. Retrieve protein Sequences from Uniprot, KEGG, RCSBPDB

This function `BMgetProtseqUniprot()` get protein sequences from uniprot.org by protein ID(s), The input `ID` is a character vector specifying the protein ID(s). The returned sequences are stored in a list:

```
> ids = c('P00750', 'P00752')
> protseq = BMgetProtSeqUniprot(ids)
> print(protseq)

[[1]]
[[1]]$`sp|P00750|TPA_HUMAN`
[1] "MDAMKRGLCCVLLLCGAVFVSPSQEIHARFRRGARSYQVICRDEKTQMIYQQHQSWLRPVLRSNRVEYCWCN
SGRAQCHSVPVKSCSEPRCFNGGTCQQALYFSDFVCQCPEGFAGKCCEIDTRATCYEDQGISYRGTWSTAESGAECT
```

```
NWNSSALAQKPYSGRRPDAIRLGLGNHNYCRNPDRDSKPWCYVFKAGKYSSEFCSTPACSEGNSDCYFGNGSAYRGT
HSLTESGASCLPWNSMILIGKVYTAQNPSAQALGLGKHNYCRNPDGDAKPWCHVLKNRRLTWEYCDVPSCSTCGLRQ
YSQPQFRIKGGLFADIASHPWQAAIFAKHRRSPGERFLCGGILISSCWILSAAHCFQERFPPHHLTVILGRTYRVVP
GEEEQKFEVEKYIVHKEFDDDTYDNDIALLQLKSDSSRCAQESSVVRTVCLPPADLQLPDWTECELSGYGKHEALSP
FYSERLKEAHVRLYPSSRCTSQHLLNRTVTDNMLCAGDTRSGGPQANLHDACQGDSGGPLVCLNDGRMTLVGIISWG
LGCGQKDVPGVYTKVTNYLDWIRDNMRP"

[[2]]
[[2]]$`sp|P00752|KLK_PIG`
[1] "APPIQSRIIGGRECEKNSHPWQVAIYHYSSFQCGGVLVNPKWVLTAAHCKNDNYEVWLGRHNLFENENTAQF
FGVTADFPHPGFNLSLLKXHTKADGKDYSHDLMLLRLQSPAKITDAVKVLELPTQEPELGSTCEASGWGSIEPGPDB
FEFPDEIQCVQLTLLQNTFCABAHPBKVTESMLCAGYLPGGKDTCMGDSGGPLICNGMWQGITSWGHTPCGSANKPS
IYTKLIFYLDWINDTITENP"
```

The functions named after `BMgetProtSeq...()`, `BMgetProtFASTA...()` and `BMgetProtPDB...()` supports the parallelized retrieval of proteins from UniProt, KEGG and RCSB PDB.

## 2.3. Retrieve DNA/RNA Sequences from GenBank

This function `BMgetDNAGenBank()` get DNA/RNA sequences from GenBank by GI ID(s), The input `ID` is a character vector specifying the GI ID(s). The returned sequences are stored in a list:

```
> ids = c('2', '392893239')
> DNAseq = BMgetDNAGenBank(ids)
> print(DNAseq)
```

```
$gi_2
[1] "AATTCATGCGTCCGGACTTCTGCCTCGAGCCGCCGTACACTGGGCCCTGCAAAGCTCGTATCATCCGTTACT
TCTACAATGCAAAGGCAGGCCTGTGTCAGACCTTCGTATACGGCGGTTGCCGTGCTAAGCGTAACAACTTCAAATCC
GCGGAAGACTGCGAACGTACTTGCGGTGGTCCTTAGTAAAGCTTG"

$gi_392893239
[1] "GCCTAAAGACGACCGCGACGCGGCCGCTCGCACTCATAGACTACGCTAGTGGTGAGATACGCAGAGAAAAAG
ACGAGAGAGTATTGAGAGAATGGAGACATCACTACATCTAACATAGGGTCGCCAGTCGTCACCGAATTATTGGATTC
AAATTTAGGTCCC"
```

## 2.4. Read FASTA, PDB Format files

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

The Protein Data Bank (pdb) file format is a textual file format describing the three dimensional structures of protein. The `readPDB()` function provides the function to read protein sequences stored in PDB format files. See ?readPDB for details.

## 2.5. Sanity Check of the Deoxyribonucleic Acid Types

The `checkDNA()` function checks if the DNA sequence's deoxyribonucleic acids types are in the 4 default types, which returns a `TRUE` if all the deoxyribonucleic acids in the sequence belongs to the 4 default types; The `protcheck()` function checks if the protein sequence's amino acid types are in the 20 default types, which returns a `TRUE` if all the amino acids in the sequence belongs to the 20 default types:

```
require(BioMedR)
x = readFASTA(system.file('protseq/P00750.fasta', package = 'BioMedR'))[[1]]
# A real sequence
checkProt(x)
## [1] TRUE
# An artificial sequence
checkProt(paste(x, 'Z', sep = ''))
## [1] FALSE
```

## 2.6. Protein Sequence Partition

The `segProt()` function partitions the protein sequences to create sliding windows. This is usually required when creating feature vectors for machine learning tasks. Users could specify a sequence `x`, and a character `aa`, one of the 20 amino acid types, and a positive integer $k$, which controls the window size (half of the window).

This function returns a named list, each component contains one of the segmentations (a character string), names of the list components are the positions of the specified amino acid in the sequence. See the example below:

```
segProt(x, aa = 'M', k = 5)

## $`48`
## [1] "DEKTQMIYQQH"
##
## $`242`
## [1] "LPWNSMILIGK"
##
## $`490`
## [1] "TVTDNMLCAGD"
##
## $`525`
## [1] "LNDGRMTLVGI"
##
```

The **BioMedR** package also integrated the functionality of converting molecular file formats. For example, we could convert the SDF files to SMILES files using `convMolFormat()`.

## 2.7. Molecular data manipulation

we load the **BioMedR** package, and read the molecules stored in a SMILES file:

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'BioMedR')
mol = readMolFromSmi(smi, type = 'mol')
```

The `readMolFromSmi()` function is used for reading molecules from SMILES files, for molecules stored in SDF files, use `readMolFromSDF()` instead.

## 2.8. Summary

The summary of the Retrieving data from various online databases in the **BioMedR** package is listed in table 1, 2 and 3.

Table 1: Retrieving drug molecular data from various online databases

| Function name | Function description |
|---|---|
| `BMgetDrug()` | Retrieve drug molecules in MOL format and SMILES format from various online databases |
| `BMgetDrugMolDrugBank()` | Retrieve drug molecules in MOL format from DrugBank |
| `BMgetDrugMolPubChem()` | Retrieve drug molecules in MOL format from PubChem |
| `BMgetDrugMolChEMBL()` | Retrieve drug molecules in MOL format from ChEMBL |
| `BMgetDrugMolKEGG()` | Retrieve drug molecules in MOL format from the KEGG |
| `BMgetDrugMolCAS()` | Retrieve drug molecules in InChI format from CAS |
| `BMgetDrugSmiDrugBank()` | Retrieve drug molecules in SMILES format from DrugBank |
| `BMgetDrugSmiPubChem()` | Retrieve drug molecules in SMILES format from PubChem |
| `BMgetDrugSmiChEMBL()` | Retrieve drug molecules in SMILES format from ChEMBL |
| `BMgetDrugSmiKEGG()` | Retrieve drug molecules in SMILES format from KEGG |

Table 2: Retrieving protein sequence data from various online databases

| Function name | Function description |
|---|---|
| `BMgetProt()` | Retrieve protein sequence in FASTA format or PDB format from various online databases |
| `BMgetProtFASTAFUniProt()` | Retrieve protein sequence in FASTA format from UniProt |
| `BMgetProtFASTAKEGG()` | Retrieve protein sequence in FASTA format from KEGG |
| `BMgetProtPDBRCSBPDB()` | Retrieve protein sequence in PDB Format from RCSB PDB |
| `BMgetProtSeqUniProt()` | Retrieve protein sequence from UniProt |
| `BMgetProtSeqKEGG()` | Retrieve protein sequence from KEGG |
| `BMgetProtSeqRCSBPDB()` | Retrieve protein sequence from RCSB PDB |

Table 3: Retrieving DNA/RNA sequences data from various online databases

| Function name | Function description |
|---|---|
| `BMgetDNAGenBank()` | Retrieve DNA/RNA sequences in FASTA format from GenBank |

The summary of the data manipulation in the **BioMedR** package is listed in table 4 and 5.

# 3. Calculating Drug Molecular Descriptors and Fingerprints

Table 4: Protein/DNA sequence data manipulation

| Function name | Function description |
|---|---|
| `readFASTA()` | Read protein sequences in FASTA format |
| `readPDB()` | Read protein sequences in PDB format |
| `segProt()` | Protein sequence segmentation |
| `checkProt()` | Check if the protein sequence's amino acid types are the 20 default types |
| `checkDNA()` | Check if the DNA/RNA sequence's deoxyribonucleic acid types are the 4 default types |

Table 5: Molecular data manipulation

| Function name | Function description |
|---|---|
| `readMolFromSDF()` | Read molecules from SDF files and return parsed Java molecular object |
| `readMolFromSmi()` | Read molecules from SMILES files and return parsed Java molecular object or plain text list |
| `convMolFormat()` | Chemical file formats conversion |

To reasonably and conveniently use the **BioMedR** package, the users should intelligently evaluate the underlying details of the descriptors provided rather than using this package with their data blindly, especially for these more flexible descriptor types. For a drug molecule, its molecular descriptors and chemical fragments can be calculated by **rcdk** (Steinbeck *et al.* 2003) package and the **ChemmineOB** package (Horan and Girke 2013).

## 3.1. Calculating Drug Molecular Descriptors

### *logP*

The atomic parameters was developed to successfully evaluate the molecular water-octanol partition coefficient, which is a measure of hydrophobicity (Ghose and Crippen 1986); The atomic physicochemical parameters was developed to model the dispersive and hydrophobic interactions (Ghose and Crippen 1987).

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'BioMedR')
mol = readMolFromSmi(smi, type = 'mol')
logp = extrDrugALOGP(mol)
head(logp)
```

```
##                                                         ALogP      ALogp2
## c1ccc2c(c1)oc(=O)c(C(CC(=O)C)c1ccc([N+](=O)[O-])cc1)c2O  1.4501 2.10279001
## c1(ccc(cc1)O)NC(=O)C                                      -0.2314 0.05354596
## c1ccccc1NC(=O)C                                            0.3317 0.11002489
## n1nc(S(=O)(=O)N)sc1NC(=O)C                                 0.1354 0.01833316
## O=S(=O)(c1ccc(cc1)C(=O)C)NC(=O)NC1CCCCC1                  -2.0362 4.14611044
## CC(=O)NO                                                  -1.0852 1.17765904
##                                                               AMR
## c1ccc2c(c1)oc(=O)c(C(CC(=O)C)c1ccc([N+](=O)[O-])cc1)c2O 98.1118
## c1(ccc(cc1)O)NC(=O)C                                    44.7549
## c1ccccc1NC(=O)C                                         43.1494
## n1nc(S(=O)(=O)N)sc1NC(=O)C                              44.2004
## O=S(=O)(c1ccc(cc1)C(=O)C)NC(=O)NC1CCCCC1                76.3739
## CC(=O)NO                                                15.4305
```

This function returns three columns named `ALogP`, `ALogp2` and `AMR`. Note the underlying code in CDK assumes that aromaticity has been detected before evaluating this descriptor. The code also expects that the molecule will have hydrogens explicitly set. For SD files, this is usually not a problem since hydrogens are explicit. But for the case of molecules obtained from SMILES, hydrogens must be made explicit.

### *WienerNumbers*

The wiener polarity number p is defined as the number of pairs of carbon atoms which are separated by three carbon-carbon bonds. The wiener path number w is defined as the sum of the tiistmces between any two carbon atoms in the molecule. They were proposed by Harry Wiener in 1947 to study the correlation between the boiling points of the paraffins and the structural variables (Wiener 1947).

```
wiener = extrDrugWienerNumbers(mol)
head(wiener)
```

```
##                                                         WPATH WPOL
## c1ccc2c(c1)oc(=O)c(C(CC(=O)C)c1ccc([N+](=O)[O-])cc1)c2O  1592   44
## c1(ccc(cc1)O)NC(=O)C                                      166   11
## c1ccccc1NC(=O)C                                           126    9
## n1nc(S(=O)(=O)N)sc1NC(=O)C                                257   14
## O=S(=O)(c1ccc(cc1)C(=O)C)NC(=O)NC1CCCCC1                 1176   31
## CC(=O)NO                                                   18    2
```

This descriptor calculates the Wiener numbers, including the Wiener Path number and the Wiener Polarity Number. Wiener path number: half the sum of all the distance matrix entries; Wiener polarity number: half the sum of all the distance matrix entries with a value of 3.

## *BCUT*

The BCUT value based on the activity-seeded and structure-based clustering was proposed by R. S. Pearlman in 1999 (Pearlman and Smith 1999). It has been proven to be an ideal approach to validate either high- or low-dimensional chemistry-space metrics when validation by computer-graphic visualization is not possible.

```
bcut = extrDrugBCUT(mol)
head(bcut)
```

```
##                                                         BCUTw.1l BCUTw.1h
## c1ccc2c(c1)oc(=O)c(C(CC(=O)C)c1ccc([N+](=O)[O-])cc1)c2O 11.99616 16.00080
## c1(ccc(cc1)O)NC(=O)C                                    11.89000 15.99592
## c1ccccc1NC(=O)C                                         11.89000 15.99492
## n1nc(S(=O)(=O)N)sc1NC(=O)C                              11.89000 31.97307
## O=S(=O)(c1ccc(cc1)C(=O)C)NC(=O)NC1CCCCC1                11.90000 31.97207
## CC(=O)NO                                                11.99900 16.00115
##                                                           BCUTc.1l
## c1ccc2c(c1)oc(=O)c(C(CC(=O)C)c1ccc([N+](=O)[O-])cc1)c2O -0.3848947
## c1(ccc(cc1)O)NC(=O)C                                    -0.3607188
## c1ccccc1NC(=O)C                                         -0.2810520
## n1nc(S(=O)(=O)N)sc1NC(=O)C                              -0.2805235
## O=S(=O)(c1ccc(cc1)C(=O)C)NC(=O)NC1CCCCC1                -0.2941177
## CC(=O)NO                                                -0.3632885
##                                                          BCUTc.1h BCUTp.1l
## c1ccc2c(c1)oc(=O)c(C(CC(=O)C)c1ccc([N+](=O)[O-])cc1)c2O 0.2972769 3.726860
## c1(ccc(cc1)O)NC(=O)C                                    0.2511029 4.199143
## c1ccccc1NC(=O)C                                         0.2510941 4.191793
## n1nc(S(=O)(=O)N)sc1NC(=O)C                              0.2643700 4.464387
## O=S(=O)(c1ccc(cc1)C(=O)C)NC(=O)NC1CCCCC1                0.2858059 4.592529
## CC(=O)NO                                                0.2197801 3.520334
##                                                          BCUTp.1h
## c1ccc2c(c1)oc(=O)c(C(CC(=O)C)c1ccc([N+](=O)[O-])cc1)c2O 11.858618
## c1(ccc(cc1)O)NC(=O)C                                     8.415819
## c1ccccc1NC(=O)C                                          8.357006
## n1nc(S(=O)(=O)N)sc1NC(=O)C                              12.193660
## O=S(=O)(c1ccc(cc1)C(=O)C)NC(=O)NC1CCCCC1                11.513438
## CC(=O)NO                                                 5.198756
```

This function return a data frame, each row represents one of the molecules, each column represents one feature. This function returns 6 columns, See `?extrDrugBCUT` for details.

By default, the descriptor will return the highest and lowest eigenvalues for the three classes of descriptor in a single ArrayList (in the order shown above). However it is also possible to supply a parameter list indicating how many the highest and lowest eigenvalues (for each class of descriptor) are required. The descriptor works with the hydrogen depleted molecule.

A side effect of specifying the number of highest and lowest eigenvalues is that it is possible to get two copies of all the eigenvalues. That is, if a molecule has 5 heavy atoms, then specifying the 5 highest eigenvalues returns all of them, and specifying the 5 lowest eigenvalues returns all of them, resulting in two copies of all the eigenvalues.

Note that it is possible to specify an arbitrarily large number of eigenvalues to be returned. However if the number (i.e., nhigh or nlow) is larger than the number of heavy atoms, the remaining eignevalues will be `NaN`.

Given the above description, if the aim is to get all the eigenvalues for a molecule, you should set nlow to 0 and specify the number of heavy atoms (or some large number) for nhigh (or vice versa).

## 3.2. Calculating Drug Molecular Fingerprints

### *Estate*

Estate was an electrotopoligicl state indice for atom types combining the electronic, topological and valence state information proposed by Lowell H. Hall in 1995. It is useful for database characterization, molecular similarity analysis, and QSAR.

```
estate = extrDrugEstate(mol)
head(estate)
```

```
## $`79`
##  [1]  7  9 12 13 16 17 18 31 34 35 37
##
## $`79`
## [1]  7 12 16 17 24 34 35
##
## $`79`
## [1]  7 12 16 17 24 35
##
## $`79`
## [1]  7 16 17 21 24 29 35 51 53
##
## $`79`
## [1]  7  9 12 13 16 17 24 35 53
##
## $`79`
## [1]  7 16 24 34 35
```

### *MACCS*

The MACCS fingerprint uses a dictionary of MDL keys, which contains a set of 166 mostly common substructure features. These are referred to as the MDL public MACCS keys.

```
maccs = extrDrugMACCS(mol)
head(maccs)
```

```
## $`166`
##  [1]  24  49  56  57  63  70  71  89  91  94  98 102 105 113 115 119 122 123 124 125 12
## [22] 130 132 133 135 136 137 139 140 143 144 145 146 148 150 152 154 155 156 157 158 15
## [43] 160 161 162 163 164 165
##
## $`166`
## [1]  92 110 113 117 127 131 133 135 139 143 151 152 154 156 157 158 159 160 161 162 163
## [22] 164 165
##
## $`166`
## [1]  92 110 117 133 135 151 154 156 158 160 161 162 163 164 165
##
## $`166`
## [1]  32  33  36  47  51  52  55  58  59  60  61  64  65  67  69  73  77  79  80  81  83
## [22]  84  88  92  94  95  96  97 102 106 110 112 117 120 121 124 130 131 133 135 136 13
## [43] 142 146 148 151 154 156 158 159 160 161 162 164 165
##
## $`166`
## [1]  32  33  37  43  51  55  58  59  60  61  64  67  69  73  77  81  88  89  90  91  94
## [22]  97 102 104 106 110 111 112 117 118 124 129 130 131 133 136 140 142 145 146 147 14
## [43] 151 152 154 156 158 159 160 161 162 163 164 165
##
## $`166`
## [1]  24  68  69  71  72  92  94 102 110 117 124 131 139 151 154 156 158 159 160 161 164
```

`extrDrugMACCS` and `extrDrugEstate` return a list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

what's more, we can calculate several selected molecular descriptors. The corresponding functions for molecular descriptor calculation are all named after `extrDrug...()` in **BioMedR**:

```
# calculate selected molecular descriptors
x = suppressWarnings(cbind(
    extrDrugALOGP(mol),
    extrDrugApol(mol),
    extrDrugECI(mol),
    extrDrugTPSA(mol),
    extrDrug...(),
    ...)
```

## 3.3. Summary

The summary of the drug molecule descriptors in the **BioMedR** package is listed in table 6.

The summary of the drug molecule fingerprint in the **BioMedR** package is listed in table 7.

Table 6: Molecular descriptors

| Function name | Descriptor name |
|---|---|
| `extrDrugAIO()` | All the molecular descriptors in the **BioMedR** package |
| `extrDrugALOGP()` | Atom additive logP and molar refractivity values descriptor |
| `extrDrugAminoAcidCount()` | Number of amino acids |
| `extrDrugApol()` | Sum of the atomic polarizabilities |
| `extrDrugAromaticAtomsCount()` | Number of aromatic atoms |
| `extrDrugAromaticBondsCount()` | Number of aromatic bonds |
| `extrDrugAtomCount()` | Number of atom descriptor |
| `extrDrugAutocorrelationCharge()` | Moreau-Broto autocorrelation descriptors using partial charges |
| `extrDrugAutocorrelationMass()` | Moreau-Broto autocorrelation descriptors using atomic weight |
| `extrDrugAutocorrelationPolarizability()` | Moreau-Broto autocorrelation descriptors using polarizability |
| `extrDrugBCUT()` | BCUT, the eigenvalue based descriptor |
| `extrDrugBondCount()` | Number of bonds of a certain bond order |
| `extrDrugBPol()` | Sum of the absolute value of the difference between atomic polarizabilities of all bonded atoms in the molecule |
| `extrDrugCarbonTypes()` | Topological descriptor characterizing the carbon connectivity in terms of hybridization |
| `extrDrugChiChain()` | Kier & Hall Chi chain indices of orders 3, 4, 5, 6 and 7 |
| `extrDrugChiCluster()` | Kier & Hall Chi cluster indices of orders 3, 4, 5 and 6 |
| `extrDrugChiPath()` | Kier & Hall Chi path indices of orders 0 to 7 |
| `extrDrugChiPathCluster()` | Kier & Hall Chi path cluster indices of orders 4, 5 and 6 |
| `extrDrugCPSA()` | Descriptors combining surface area and partial charge information |
| `extrDrugDescOB()` | Molecular descriptors provided by OpenBabel |
| `extrDrugECI()` | Eccentric connectivity index descriptor |
| `extrDrugFMF()` | FMF descriptor |
| `extrDrugFragmentComplexity()` | Complexity of a system |
| `extrDrugGravitationalIndex()` | Mass distribution of the molecule |
| `extrDrugHBondAcceptorCount()` | Number of hydrogen bond acceptors |
| `extrDrugHBondDonorCount()` | Number of hydrogen bond donors |
| `extrDrugHybridizationRatio()` | Molecular complexity in terms of carbon hybridization states |
| `extrDrugIPMolecularLearning()` | Ionization potential |
| `extrDrugKappaShapeIndices()` | Kier & Hall Kappa molecular shape indices |
| `extrDrugKierHallSmarts()` | Number of occurrences of the E-State fragments |
| `extrDrugLargestChain()` | Number of atoms in the largest chain |
| `extrDrugLargestPiSystem()` | Number of atoms in the largest Pi chain |
| `extrDrugLengthOverBreadth()` | Ratio of length to breadth descriptor |
| `extrDrugLongestAliphaticChain()` | Number of atoms in the longest aliphatic chain |
| `extrDrugMannholdLogP()` | LogP based on the number of carbons and hetero atoms |
| `extrDrugMDE()` | Molecular Distance Edge (MDE) descriptors for C, N and O |
| `extrDrugMomentOfInertia()` | Principal moments of inertia and ratios of the principal moments |
| `extrDrugPetitjeanNumber()` | Petitjean number of a molecule |
| `extrDrugPetitjeanShapeIndex()` | Petitjean shape indices |
| `extrDrugRotatableBondsCount()` | Number of non-rotatable bonds on a molecule |
| `extrDrugRuleOfFive()` | Number failures of the Lipinski's Rule Of Five |
| `extrDrugTPSA()` | Topological Polar Surface Area (TPSA) |
| `extrDrugVABC()` | Volume of a molecule |
| `extrDrugVAdjMa()` | Vertex adjacency information of a molecule |
| `extrDrugWeight()` | Total weight of atoms |
| `extrDrugWeightedPath()` | Weighted path (Molecular ID) |
| `extrDrugWHIM()` | Holistic descriptors described by Todeschini et al. |
| `extrDrugWienerNumbers()` | Wiener path number and wiener polarity number |
| `extrDrugXLogP()` | Prediction of logP based on the atom-type method called XLogP |
| `extrDrugZagrebIndex()` | Sum of the squared atom degrees of all heavy atoms |

Table 7: Molecular fingerprints

| Function name | Fingerprint type |
|---|---|
| `extrDrugStandard()` | Standard molecular fingerprints (in compact format) |
| `extrDrugStandardComplete()` | Standard molecular fingerprints (in complete format) |
| `extrDrugExtended()` | Extended molecular fingerprints (in compact format) |
| `extrDrugExtendedComplete()` | Extended molecular fingerprints (in complete format) |
| `extrDrugGraph()` | Graph molecular fingerprints (in compact format) |
| `extrDrugGraphComplete()` | Graph molecular fingerprints (in complete format) |
| `extrDrugHybridization()` | Hybridization molecular fingerprints (in compact format) |
| `extrDrugHybridizationComplete()` | Hybridization molecular fingerprints (in complete format) |
| `extrDrugMACCS()` | MACCS molecular fingerprints (in compact format) |
| `extrDrugMACCSComplete()` | MACCS molecular fingerprints (in complete format) |
| `extrDrugEstate()` | E-State molecular fingerprints (in compact format) |
| `extrDrugEstateComplete()` | E-State molecular fingerprints (in complete format) |
| `extrDrugPubChem()` | PubChem molecular fingerprints (in compact format) |
| `extrDrugPubChemComplete()` | PubChem molecular fingerprints (in complete format) |
| `extrDrugKR()` | KR (Klekota and Roth) molecular fingerprints (in compact format) |
| `extrDrugKRComplete()` | KR (Klekota and Roth) molecular fingerprints (in complete format) |
| `extrDrugShortestPath()` | Shortest Path molecular fingerprints (in compact format) |
| `extrDrugShortestPathComplete()` | Shortest Path molecular fingerprints (in complete format) |
| `extrDrugOBFP2()` | FP2 molecular fingerprints |
| `extrDrugOBFP3()` | FP3 molecular fingerprints |
| `extrDrugOBFP4()` | FP4 molecular fingerprints |
| `extrDrugOBMACCS()` | MACCS molecular fingerprints |
| `extrDrugAP()` | atom pair fingerprints |

# 4. Calculating Commonly Used Protein Descriptors

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

A protein or peptide sequence with $N$ amino acid residues could be generally represented as $\{ R_1, R_2, \ldots, R_n \}$, where $R_i$ represents the residue at the $i$-th position in the sequence. The labels $i$ and $j$ are used to index amino acid position in a sequence, and $r$, $s$, $t$ are used to represent the amino acid type. The computed descriptors are roughly divided into 4 groups according to their known applications described in the literature.

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

## 4.1. Amino Acid Composition (AAC)

The Amino Acid Composition (AAC) is the fraction of each amino acid type within a protein. The fractions of all 20 natural amino acids are calculated as:

$$f(r) = \frac{N_r}{N} \quad r = 1, 2, \ldots, 20.$$

where $N_r$ is the number of the amino acid type $r$ and $N$ is the length of the sequence.

As was described above, we could use the function `extrProtAAC()` to extrProt the descriptors (features) from protein sequences:

```
require(BioMedR)
x = readFASTA(system.file('protseq/P00750.fasta', package = 'BioMedR'))[[1]]
extrProtAAC(x)

##          Q          G          H          I          L          K
## 0.04804270 0.08185053 0.03024911 0.03558719 0.07651246 0.03914591
##          M          F          P          S          T          W
## 0.01245552 0.03202847 0.05338078 0.08896797 0.04448399 0.02313167
##          Y          V
## 0.04270463 0.04982206
```

Here with the function `readFASTA()` we loaded a single protein sequence (P00750, Tissue-type plasminogen activator) from a FASTA format file. Then extrProted the AAC descriptors with `extrProtAAC()`. The result returned is a named vector, whose elements are tagged with the name of each amino acid.

## 4.2. Dipeptide Composition (DC)

The Dipeptide Composition (DC) gives 400 descriptors, defined as:

$$f(r,s) = \frac{N_{rs}}{N-1} \quad r,s = 1,2,\ldots,20.$$

where $N_{rs}$ is the number of dipeptide represented by amino acid type $r$ and type $s$. Similar to `extrProtAAC()`, here we use `extrProtDC()` to compute the descriptors:

```
dc = extrProtDC(x)
head(dc, n = 18L)

##          AA          RA          NA          DA          CA          EA
## 0.003565062 0.003565062 0.000000000 0.007130125 0.003565062 0.003565062
##          QA          GA          HA          IA          LA          KA
## 0.007130125 0.007130125 0.001782531 0.003565062 0.001782531 0.001782531
##          MA          FA          PA          SA          TA          WA
## 0.000000000 0.005347594 0.003565062 0.007130125 0.003565062 0.000000000
```

Here we only showed the first 30 elements of the result vector and omitted the rest of the result. The element names of the returned vector are self-explanatory as before.

## 4.3. Tripeptide Composition (TC)

The Tripeptide Composition (TC) gives 8000 descriptors, defined as:

$$f(r,s,t) = \frac{N_{rst}}{N-2} \quad r,s,t = 1,2,\ldots,20$$

where $N_{rst}$ is the number of tripeptides represented by amino acid type $r$, $s$ and $t$. With function `extrProtTC()`, we could easily obtain the length-8000 descriptor, to save some space, here we also omitted the tedious outputs:

```
tc = extrProtTC(x)
head(tc, n = 12L)
```

```
##         AAA         RAA         NAA         DAA         CAA         EAA
## 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##         QAA         GAA         HAA         IAA         LAA         KAA
## 0.001785714 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
```

## 4.4. Autocorrelation Descriptors

Autocorrelation descriptors are defined based on the distribution of amino acid properties along the sequence. The amino acid properties used here are various types of amino acids index (Retrieved from AAindex Database: `http://www.genome.jp/dbget/aaindex.html`, see Kawashima *et al.* (1999), Kawashima and Kanehisa (2000), and Kawashima *et al.* (2008)). Three types of autocorrelation descriptors are defined here and described below.

All the amino acid indices are centralized and standardized before the calculation, i.e.

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

where $\bar{P}$ is the average of the property of the 20 amino acids:

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

### *Normalized Moreau-Broto Autocorrelation Descriptors*

Moreau-Broto autocorrelation descriptors application to protein sequences could be defined as:

$$AC(d) = \sum_{i=1}^{N-d} P_i P_{i+d} \quad d = 1, 2, \ldots, \text{nlag}$$

where $d$ is called the lag of the autocorrelation and $P_i$ and $P_{i+d}$ are the properties of the amino acids at position $i$ and $i + d$, respectively. nlag is the maximum value of the lag.

The normalized Moreau-Broto autocorrelation descriptors are defined as:

$$ATS(d) = \frac{AC(d)}{N - d} \quad d = 1, 2, \ldots, \text{nlag}$$

The corresponding function for this descriptor is `extrProtMoreauBroto()`. A typical call could be:

```
moreau = extrProtMoreauBroto(x)
head(moreau, n = 12L)
```

```
##  CIDH920105.lag1  CIDH920105.lag2  CIDH920105.lag3  CIDH920105.lag4
##      0.081573213     -0.016064817     -0.015982990     -0.025739038
##  CIDH920105.lag5  CIDH920105.lag6  CIDH920105.lag7  CIDH920105.lag8
##      0.079058632     -0.042771564     -0.036320847      0.024087298
##  CIDH920105.lag9 CIDH920105.lag10 CIDH920105.lag11 CIDH920105.lag12
##     -0.005273958      0.052274763      0.082170073      0.005419919
```

The 8 default properties used here are:

- **AccNo. CIDH920105** — Normalized Average Hydrophobicity Scales
- **AccNo. BHAR880101** — Average Flexibility Indices
- **AccNo. CHAM820101** — Polarizability Parameter
- **AccNo. CHAM820102** — Free Energy of Solution in Water, kcal/mole
- **AccNo. CHOC760101** — Residue Accessible Surface Area in Tripeptide
- **AccNo. BIGC670101** — Residue Volume
- **AccNo. CHAM810101** — Steric Parameter
- **AccNo. DAYM780201** — Relative Mutability

Users could change the property names of AAindex database with the argument `props`. The AAindex data shipped with **BioMedR** could be loaded by `data(AAindex)`, which has the detailed information of each property. With the argument `customprops` and `nlag`, users could specify their own properties and lag value to calculate with. For illustration, we could use:

```
# Define 3 custom properties
myprops = data.frame(AccNo = c("MyProp1", "MyProp2", "MyProp3"),
                     A = c(0.62,  -0.5, 15),  R = c(-2.53,   3, 101),
                     N = c(-0.78,  0.2, 58),  D = c(-0.9,    3, 59),
                     C = c(0.29,    -1, 47),  E = c(-0.74,   3, 73),
                     Q = c(-0.85,  0.2, 72),  G = c(0.48,    0, 1),
                     H = c(-0.4,  -0.5, 82),  I = c(1.38, -1.8, 57),
                     L = c(1.06,  -1.8, 57),  K = c(-1.5,    3, 73),
                     M = c(0.64,  -1.3, 75),  F = c(1.19, -2.5, 91),
                     P = c(0.12,     0, 42),  S = c(-0.18, 0.3, 31),
                     T = c(-0.05, -0.4, 45),  W = c(0.81, -3.4, 130),
                     Y = c(0.26,  -2.3, 107), V = c(1.08, -1.5, 43))

# Use 4 properties in the AAindex database, and 3 cutomized properties
moreau2 = extrProtMoreauBroto(x, customprops = myprops,
                              props = c('CIDH920105', 'BHAR880101',
                                        'CHAM820101', 'CHAM820102',
                                        'MyProp1', 'MyProp2', 'MyProp3'))
head(moreau2, n = 12L)
```

```
##  CIDH920105.lag1  CIDH920105.lag2  CIDH920105.lag3  CIDH920105.lag4
##      0.081573213     -0.016064817     -0.015982990     -0.025739038
##  CIDH920105.lag5  CIDH920105.lag6  CIDH920105.lag7  CIDH920105.lag8
##      0.079058632     -0.042771564     -0.036320847      0.024087298
##  CIDH920105.lag9 CIDH920105.lag10 CIDH920105.lag11 CIDH920105.lag12
##     -0.005273958      0.052274763      0.082170073      0.005419919
```

About the standard input format of `props` and `customprops`, see `?extrProtMoreauBroto` for details.

## *Moran Autocorrelation Descriptors*

Moran autocorrelation descriptors application to protein sequence may be defined as:

$$I(d) = \frac{\frac{1}{N-d}\sum_{i=1}^{N-d}(P_i - \bar{P}')(P_{i+d} - \bar{P}')}{\frac{1}{N}\sum_{i=1}^{N}(P_i - \bar{P}')^2} \quad d = 1, 2, \ldots, 30$$

where $d$ and $P_i$ and $P_{i+d}$ are defined in the same way as in the first place, and $\bar{P}'$ is the considered property $P$ along the sequence, i.e.,

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

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

With `extrProtMoran()`, which has exactly the same arguments with `extrProtMoreauBroto()`, we could compute the Moran autocorrelation descriptors (only output the first 36 elements of the result):

```
# Use the 3 custom properties defined before
# and 4 properties in the AAindex database
moran = extrProtMoran(x, customprops = myprops,
                      props = c('CIDH920105', 'BHAR880101',
                                'CHAM820101', 'CHAM820102',
                                'MyProp1', 'MyProp2', 'MyProp3'))
head(moran, n = 12L)
```

```
##  CIDH920105.lag1  CIDH920105.lag2  CIDH920105.lag3  CIDH920105.lag4
##      0.062895724     -0.044827681     -0.045065117     -0.055955678
##  CIDH920105.lag5  CIDH920105.lag6  CIDH920105.lag7  CIDH920105.lag8
##      0.060586377     -0.074128412     -0.067308852     -0.001293384
##  CIDH920105.lag9 CIDH920105.lag10 CIDH920105.lag11 CIDH920105.lag12
##     -0.033747588      0.029392193      0.061789800     -0.023368437
```

## *Geary Autocorrelation Descriptors*

Geary autocorrelation descriptors for protein sequence could be defined as:

$$C(d) = \frac{\frac{1}{2(N-d)}\sum_{i=1}^{N-d}(P_i - P_{i+d})^2}{\frac{1}{N-1}\sum_{i=1}^{N}(P_i - \bar{P}')^2} \qquad d = 1, 2, \ldots, 30$$

where $d$, $P$, $P_i$ and $P_{i+d}$, nlag have the same meaning as above.
For each amino acid index, there will be $3 \times$ nlag autocorrelation descriptors. The usage of `extrProtGeary()` is exactly the same with `extrProtMoreauBroto()` and `extrProtMoran()`:

```
# Use the 3 custom properties defined before
# and 4 properties in the AAindex database
geary = extrProtGeary(x, customprops = myprops,
                      props = c('CIDH920105', 'BHAR880101',
                                'CHAM820101', 'CHAM820102',
                                'MyProp1', 'MyProp2', 'MyProp3'))
head(geary, n = 12L)
```

```
##  CIDH920105.lag1  CIDH920105.lag2  CIDH920105.lag3  CIDH920105.lag4
##        0.9361830        1.0442920        1.0452843        1.0563467
##  CIDH920105.lag5  CIDH920105.lag6  CIDH920105.lag7  CIDH920105.lag8
##        0.9406031        1.0765517        1.0675786        0.9991363
##  CIDH920105.lag9 CIDH920105.lag10 CIDH920105.lag11 CIDH920105.lag12
##        1.0316555        0.9684585        0.9353130        1.0201990
```

## 4.5. Composition / Transition / Distribution

These descriptors are developed and described by Dubchak *et al.* (1995) and Dubchak *et al.* (1999).

| Sequence | M | T | E | I | T | A | S | M | V | K | E | L | R | E | A | T | G | T | G | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sequence Index | 1 | | | | 5 | | | | | 10 | | | | | 15 | | | | | 20 |
| Transformation | 3 | 2 | 1 | 3 | 2 | 2 | 2 | 3 | 3 | 1 | 1 | 3 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| Index for 1 | | | 1 | | | | | | | 2 | 3 | | 4 | 5 | | | | | | |
| Index for 2 | | 1 | | | 2 | 3 | 4 | | | | | | | | 5 | 6 | 7 | 8 | 9 | 10 |
| Index for 3 | 1 | | | 2 | | | | 3 | 4 | | | 5 | | | | | | | | |
| 1/2 Transitions | | | \| | | | | | | | | | | | | \| | | | | | |
| 1/3 Transitions | | | | \| | | | | | \| | | \| | \| | | | | | | |
| 2/3 Transitions | | \| | | | \| | | | \| | | | | | | | | | | | |

Figure 1: The sequence of a hypothetic protein indicating the construction of composition, transition and distribution descriptors of a protein. Sequence index indicates the position of an amino acid in the sequence. The index for each type of amino acids in the sequence (‘1’, ‘2’ or ‘3’) indicates the position of the first, second, third, ... of that type of amino acid. 1/2 transition indicates the position of ‘12’ or ‘21’ pairs in the sequence (1/3 and 2/3 are defined in the same way.).

**Step 1: Sequence Encoding**

The amino acids are divided in three classes according to its attribute and each amino acid is encoded by one of the indices 1, 2, 3 according to which class it belonged. The attributes used here include hydrophobicity, normalized van der Waals volume polarity, and polarizability, as in the references. The corresponding division is in the table 1.

Table 8: Amino acid attributes and the division of the amino acids into three groups for each attribute

| | **Group 1** | **Group 2** | **Group 3** |
|---|---|---|---|
| Hydrophobicity | Polar<br>R, K, E, D, Q, N | Neutral<br>G, A, S, T, P, H, Y | Hydrophobicity<br>C, L, V, I, M, F, W |
| Normalized van der Waals Volume | 0-2.78<br>G, A, S, T, P, D, C | 2.95-4.0<br>N, V, E, Q, I, L | 4.03-8.08<br>M, H, K, F, R, Y, W |
| Polarity | 4.9-6.2<br>L, I, F, W, C, M, V, Y | 8.0-9.2<br>P, A, T, G, S | 10.4-13.0<br>H, Q, R, K, N, E, D |
| Polarizability | 0-1.08<br>G, A, S, D, T | 0.128-0.186<br>C, P, N, V, E, Q, I, L | 0.219-0.409<br>K, M, H, F, R, Y, W |
| Charge | Positive<br>K, R | Neutral<br>A, N, C, Q, G, H, I, L, M, F, P, S, T, W, Y, V | Negative<br>D, E |
| Secondary Structure | Helix<br>E, A, L, M, Q, K, R, H | Strand<br>V, I, Y, C, W, F, T | Coil<br>G, N, P, S, D |
| Solvent Accessibility | Buried<br>A, L, F, C, G, I, V, W | Exposed<br>R, K, Q, E, N, D | Intermediate<br>M, S, P, T, H, Y |

For example, for a given sequence "MTEITAAMVKELRESTGAGA", it will be encoded as "32132223311311222222" according to its hydrophobicity division.

**Step 2: Compute Composition, Transition and Distribution Descriptors**

Three descriptors, *Composition* ($C$), *Transition* ($T$), and *Distribution* ($D$) were calculated for a given attribute as follows.

## *Composition*

It is the global percent for each encoded class in the sequence. In the above example using hydrophobicity division, the numbers for encoded classes "1", "2", "3" are 5, 10, 5 respectively, so the compositions for them are 5/20 = 25%, 10/20 = 10%, and 5/20 = 25% respectively, where 20 is the length of the protein sequence. Composition can be defined as

$$C_r = \frac{n_r}{n} \quad r = 1, 2, 3$$

where $n_r$ is the number of amino acid type $r$ in the encoded sequence and $N$ is the length of the sequence. An example for `extrProtCTDC()` could be:

```
ctdc = extrProtCTDC(x)
head(ctdc, 12L)
```

```
##  prop1.G1  prop1.G2  prop1.G3  prop2.G1  prop2.G2  prop2.G3  prop3.G1
## 0.2971530 0.4056940 0.2971530 0.4519573 0.2971530 0.2508897 0.3398577
##  prop3.G2  prop3.G3  prop4.G1  prop4.G2  prop4.G3
## 0.3327402 0.3274021 0.3309609 0.4181495 0.2508897
```

The result shows the elements whose names are `PropertyNumber.GroupNumber` in the returned vector.

## *Transition*

A transition from class 1 to 2 is the percent frequency with which 1 is followed by 2 or 2 is followed by 1 in the encoded sequence. Transition descriptor can be calculated as

$$T_{rs} = \frac{n_{rs} + n_{sr}}{N-1} \quad rs = \text{‘12’}, \text{‘13’}, \text{‘23’}$$

where $n_{rs}$, $n_{sr}$ is the numbers of dipeptide encoded as "rs" and "sr" respectively in the sequence and $N$ is the length of the sequence. An example for `extrProtCTDT()` could be:

```
ctdt = extrProtCTDT(x)
head(ctdt, 12L)
```

```
## prop1.Tr1221 prop1.Tr1331 prop1.Tr2332 prop2.Tr1221 prop2.Tr1331
##    0.2709447    0.1604278    0.2335116    0.2673797    0.2263815
## prop2.Tr2332 prop3.Tr1221 prop3.Tr1331 prop3.Tr2332 prop4.Tr1221
##    0.1711230    0.2103387    0.2049911    0.2370766    0.2727273
## prop4.Tr1331 prop4.Tr2332
##    0.1515152    0.2459893
```

## *Distribution*

The "distribution" descriptor describes the distribution of each attribute in the sequence.

There are five "distribution" descriptors for each attribute and they are the position percents in the whole sequence for the first residue, 25% residues, 50% residues, 75% residues and 100% residues, respectively, for a specified encoded class. For example, there are 10 residues encoded as "2" in the above example, the positions for the first residue "2", the 2th residue "2" (25%*10=2), the 5th "2" residue (50%*10=5), the 7th "2" (75%*10=7) and the 10th residue "2" (100%*10) in the encoded sequence are 2, 5, 15, 17, 20 respectively, so the distribution descriptors for "2" are: 10.0 (2/20*100), 25.0 (5/20*100), 75.0 (15/20*100), 85.0 (17/20*100) , 100.0 (20/20*100), respectively.

Finally, an example for `extrProtCTDD()` could be:

```
ctdd = extrProtCTDD(x)
head(ctdd, 12L)
```

```
##    prop1.G1.residue0  prop1.G1.residue25  prop1.G1.residue50
##            0.3558719          23.1316726          50.1779359
##  prop1.G1.residue75 prop1.G1.residue100   prop1.G2.residue0
##          73.8434164          99.8220641           0.5338078
##  prop1.G2.residue25  prop1.G2.residue50  prop1.G2.residue75
##          27.4021352          47.3309609          75.2669039
## prop1.G2.residue100   prop1.G3.residue0  prop1.G3.residue25
##         100.0000000           0.1779359          19.5729537
```

## 4.6. Conjoint Triad Descriptors

Conjoint triad descriptors are proposed by Shen *et al.* (2007). These conjoint triad descriptors abstracts the features of protein pairs based on the classification of amino acids. In this approach, each protein sequence is represented by a vector space consisting of descriptors of amino acids. To reduce the dimensions of vector space, the 20 amino acids were clustered into several classes according to their dipoles and volumes of the side chains. The conjoint triad descriptors are calculated as follows:

**Step 1: Classification of Amino Acids**

Electrostatic and hydrophobic interactions dominate protein-protein interactions. These two kinds of interactions may be reflected by the dipoles and volumes of the side chains of amino acids, respectively. Accordingly, these two parameters were calculated, respectively, by using the density-functional theory method B3LYP/6-31G and molecular modeling approach. Based on the dipoles and volumes of the side chains, the 20 amino acids could be clustered into seven classes (See Table 2). Amino acids within the same class likely involve synonymous mutations because of their similar characteristics.

Table 9: Classification of amino acids based on dipoles and volumes of the side chains

| No. | Dipole Scale[1] | Volume Scale[2] | Class |
|---|---|---|---|
| 1 | $-$ | $-$ | Ala, Gly, Val |
| 2 | $-$ | $+$ | Ile, Leu, Phe, Pro |
| 3 | $+$ | $+$ | Tyr, Met, Thr, Ser |
| 4 | $++$ | $+$ | His, Asn, Gln, Tpr |
| 5 | $+++$ | $+$ | Arg, Lys |
| 6 | $+'+'+'$ | $+$ | Asp, Glu |
| 7 | $+$[3] | $+$ | Cys |

**Step 2: Conjoint Triad Calculation**

The conjoint triad descriptors considered the properties of one amino acid and its vicinal amino acids and regarded any three continuous amino acids as a unit. Thus, the triads can be differentiated according to the classes of amino acids, i.e., triads composed by three amino acids belonging to the same classes, such as ART and VKS, could be treated identically. To conveniently represent a protein, we first use a binary space $(\mathbf{V}, \mathbf{F})$ to represent a protein

[1]Dipole Scale (Debye): $-$, Dipole $< 1.0$; $+$, $1.0 <$ Dipole $< 2.0$; $++$, $2.0 <$ Dipole $< 3.0$; $+++$, Dipole $>$ 3.0; $+'+'+'$, Dipole $> 3.0$ with opposite orientation.

[2]Volume Scale (Å$^3$): $-$, Volume $< 50$; $+$, Volume $> 50$.

[3]Cys is separated from class 3 because of its ability to form disulfide bonds.

sequence. Here, **V** is the vector space of the sequence features, and each feature $v_i$ represents a sort of triad type; **F** is the frequency vector corresponding to **V**, and the value of the $i$-th dimension of $\mathbf{F}(f_i)$ is the frequency of type $v_i$ appearing in the protein sequence. For the amino acids that have been catogorized into seven classes, the size of **V** should be $7 \times 7 \times 7$; thus $i = 1, 2, \ldots, 343$. The detailed description for $(\mathbf{V}, \mathbf{F})$ is illustrated in Figure 2.

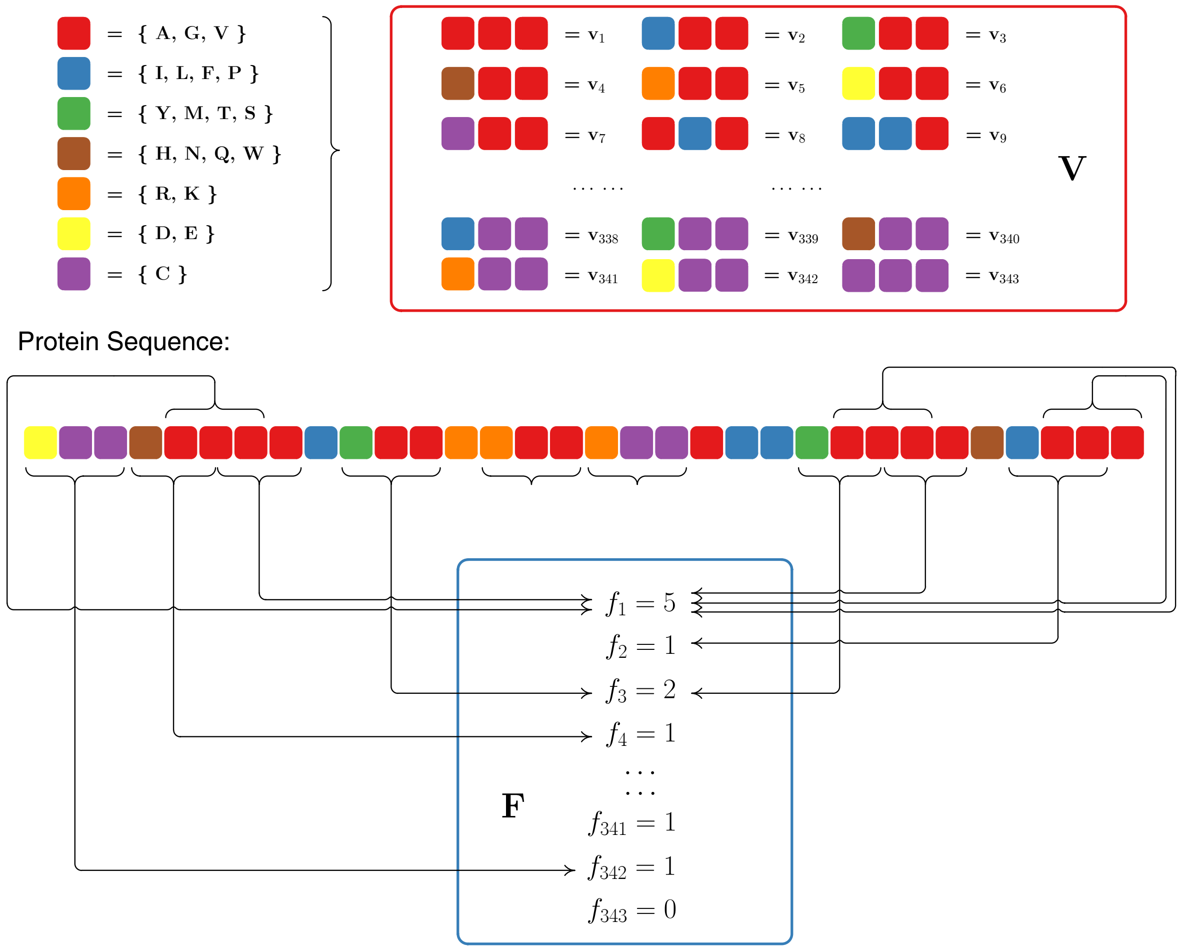

Figure 2: Schematic diagram for constructing the vector space $(\mathbf{V}, \mathbf{F})$ of protein sequence. **V** is the vector space of the sequence features; each feature $(v_i)$ represents a triad composed of three consecutive amino acids; **F** is the frequency vector corresponding to **V**, and the value of the $i$-th dimension of $\mathbf{F}(f_i)$ is the frequency that $v_i$ triad appeared in the protein sequence.

Clearly, each protein correlates to the length (number of amino acids) of protein. In general, a long protein would have a large value of $f_i$, which complicates the comparison between two heterogeneous proteins. Thus, we defined a new parameter, $d_i$, by normalizing $f_i$ with the following equation:

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

The numerical value of $d_i$ of each protein ranges from 0 to 1, which thereby enables the comparison between proteins. Accordingly, we obtain another vector space (designated **D**) consisting of $d_i$ to represent protein.

To compute conjoint triads of protein sequences, we could simply use:

```
ctriad = extrProtCTriad(x)
head(ctriad, n = 65L)
```

```
## VS111 VS211 VS311 VS411 VS511 VS611 VS711 VS121 VS221 VS321 VS421 VS521
##   0.1   0.3   0.6   0.2   0.4   0.0   0.3   1.0   0.6   0.5   0.0   0.2
## VS621 VS721 VS131 VS231 VS331 VS431 VS531 VS631 VS731 VS141 VS241 VS341
##   0.3   0.0   0.2   0.4   0.5   0.2   0.3   0.3   0.1   0.3   0.3   0.2
## VS441 VS541 VS641 VS741 VS151 VS251 VS351 VS451 VS551 VS651 VS751 VS161
##   0.2   0.0   0.1   0.2   0.2   0.2   0.5   0.1   0.2   0.0   0.0   0.1
## VS261 VS361 VS461 VS561 VS661 VS761 VS171 VS271 VS371 VS471 VS571 VS671
##   0.4   0.2   0.3   0.2   0.0   0.1   0.1   0.3   0.1   0.0   0.1   0.0
## VS771 VS112 VS212 VS312 VS412 VS512 VS612 VS712 VS122 VS222 VS322 VS422
##   0.1   0.8   0.4   0.4   0.6   0.1   0.5   0.2   0.8   0.5   0.2   0.3
## VS522 VS622 VS722 VS132 VS232
##   0.2   0.0   0.2   0.1   0.3
```

by which we only outputted the first 65 of total 343 dimension to save space.

## 4.7. Quasi-sequence-order Descriptors

The quasi-sequence-order descriptors are proposed by Chou (2000). They are derived from the distance matrix between the 20 amino acids.

### *Sequence-order-coupling Number*

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

$$\tau_d = \sum_{i=1}^{N-d} (d_{i,i+d})^2 \quad d = 1, 2, \ldots, \text{maxlag}$$

where $d_{i,i+d}$ is the distance between the two amino acids at position $i$ and $i + d$.

**Note**: maxlag is the maximum lag and the length of the protein must be not less than maxlag.

The function `extrProtSOCN(x)` is used for computing the sequence-order-coupling numbers:

```
socn = extrProtSOCN(x)
head(socn, 15L)
```

```
##  Schneider.lag1  Schneider.lag2  Schneider.lag3  Schneider.lag4
##        204.2036        199.8708        206.8102        197.4828
##  Schneider.lag5  Schneider.lag6  Schneider.lag7  Schneider.lag8
##        193.3366        208.1936        195.5476        200.9789
##  Schneider.lag9 Schneider.lag10 Schneider.lag11 Schneider.lag12
##          196.7110          193.9931          199.7031          204.9389
## Schneider.lag13 Schneider.lag14 Schneider.lag15
##          187.0140          198.4702          205.4526
```

Users could also specify the maximum lag value with the `nlag` argument.

**Note**: In addition to Schneider-Wrede physicochemical distance matrix (Schneider and Wrede 1994) used by Kuo-Chen Chou, another chemical distance matrix by Grantham (1974) is also used here. So the descriptors dimension will be `nlag * 2`. The quasi-sequence-order descriptors described next also utilized the two matrices.

## *Quasi-sequence-order Descriptors*

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

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

where $f_r$ is the normalized occurrence for amino acid type $i$ and $w$ is a weighting factor ($w = 0.1$). These are the first 20 quasi-sequence-order descriptors. The other 30 quasi-sequence-order are defined as:

$$X_d = \frac{w\tau_{d-20}}{\sum_{r=1}^{20} f_r + w \sum_{d=1}^{\text{maxlag}} \tau_d} \qquad d = 21, 22, \ldots, 20 + \text{maxlag}$$

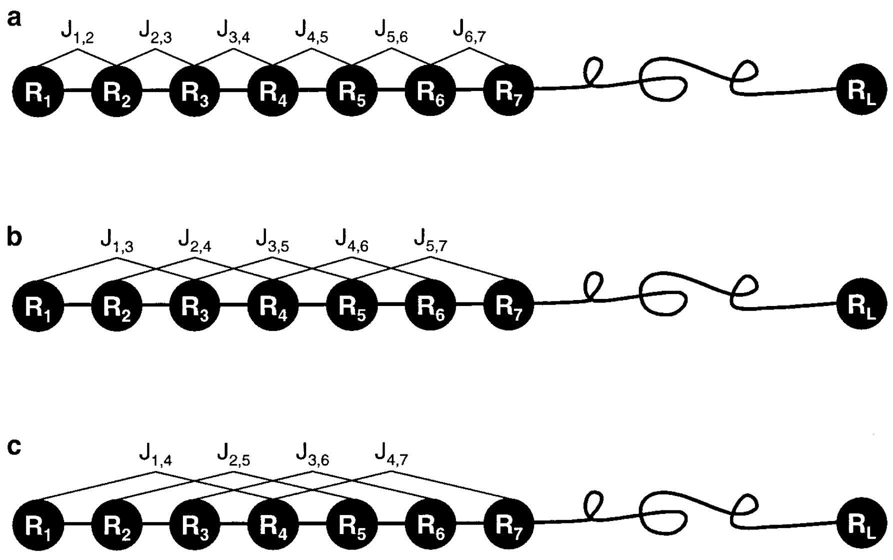

Figure 3: A schematic drawing to show (a) the 1st-rank, (b) the 2nd-rank, and (3) the 3rd-rank sequence-order-coupling mode along a protein sequence. (a) Reflects the coupling mode between all the most contiguous residues, (b) that between all the 2nd most contiguous residues, and (c) that between all the 3rd most contiguous residues. This figure is from Chou (2000).

An minimal example for `extrProtQSO()` could be:

```
qso = extrProtQSO(x)
head(qso, 15L)

## Schneider.Xr.A Schneider.Xr.R Schneider.Xr.N Schneider.Xr.D Schneider.Xr.C
##     0.06096218     0.06773576     0.03725467     0.04910842     0.06434897
## Schneider.Xr.E Schneider.Xr.Q Schneider.Xr.G Schneider.Xr.H Schneider.Xr.I
##     0.04572164     0.04572164     0.07789612     0.02878770     0.03386788
## Schneider.Xr.L Schneider.Xr.K Schneider.Xr.M Schneider.Xr.F Schneider.Xr.P
##     0.07281594     0.03725467     0.01185376     0.03048109     0.05080182
```

where users could also specify the maximum lag with argument `nlag` and the weighting factor with argument `w`.

## 4.8. Pseudo-Amino Acid Composition (PAAC)

This groups of descriptors are proposed in Chou (2001). PAAC descriptors are also called the *type 1 pseudo-amino acid composition.* Let $H_1^o(i)$ , $H_2^o(i)$, $M^o(i)$ ($i = 1, 2, 3, \ldots, 20$) be the original hydrophobicity values, the original hydrophilicity values and the original side chain masses of the 20 natural amino acids, respectively. They are converted to following qualities by a standard conversion:

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

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

Then, a correlation function could be defines as

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

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

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

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

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

For a set of n amino acid properties, it can be defined as: where $H_k(R_i)$ is the $k$-th property in the amino acid property set for amino acid $R_i$.

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

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

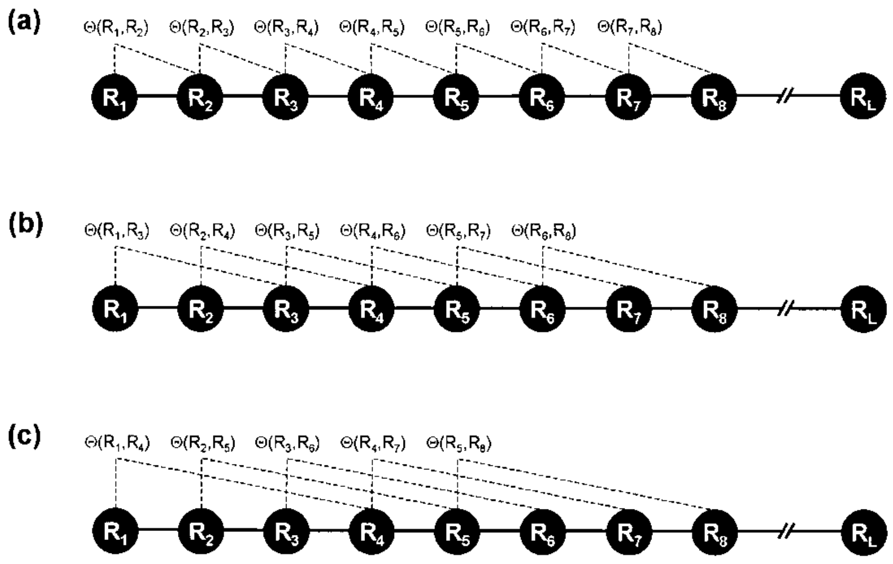
(a)
Θ(R1,R2) Θ(R2,R3) Θ(R3,R4) Θ(R4,R5) Θ(R5,R6) Θ(R6,R7) Θ(R7,R8)
(b)
Θ(R1,R3) Θ(R2,R4) Θ(R3,R5) Θ(R4,R6) Θ(R5,R7) Θ(R6,R8)
(c)
Θ(R1,R4) Θ(R2,R5) Θ(R3,R6) Θ(R4,R7) Θ(R5,R8)
R1 R2 R3 R4 R5 R6 R7 R8 RL

Figure 4: A schematic drawing to show (a) the first-tier, (b) the second-tier, and (3) the third-tiersequence order correlation mode along a protein sequence. Panel (a) reflects the correlation mode between all the most contiguous residues, panel (b) that between all the second-most contiguous residues, and panel (c) that between all the third-most contiguous residues. This figure is from Chou (2001).

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

$$\begin{aligned}
\theta_1 &= \frac{1}{N-1}\sum_{i=1}^{N-1}\Theta(R_i, R_{i+1}) \\
\theta_2 &= \frac{1}{N-2}\sum_{i=1}^{N-2}\Theta(R_i, R_{i+2}) \\
\theta_3 &= \frac{1}{N-3}\sum_{i=1}^{N-3}\Theta(R_i, R_{i+3}) \\
&\dots \\
\theta_\lambda &= \frac{1}{N-\lambda}\sum_{i=1}^{N-\lambda}\Theta(R_i, R_{i+\lambda})
\end{aligned}$$

$\lambda$ ($\lambda < L$) is a parameter to be chosen. Let $f_i$ be the normalized occurrence frequency of the 20 amino acids in the protein sequence, a set of $20 + \lambda$ descriptors called the pseudo-amino acid composition for a protein sequence can be defines as:

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

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

where $w$ is the weighting factor for the sequence-order effect and is set as $w = 0.05$ in **BioMedR** as suggested by Kuo-Chen Chou.

With `extrProtPAAC()`, we could compute the PAAC descriptors:

```
pacc = extrProtPAAC(x)
head(pacc, 25L)
```

```
##         Xc1.A         Xc1.R         Xc1.N         Xc1.D         Xc1.C
##    9.07025432   10.07806035    5.54293319    7.30659376    9.57415734
##         Xc1.E         Xc1.Q         Xc1.G         Xc1.H         Xc1.I
##    6.80269074    6.80269074   11.58976941    4.28317565    5.03903018
##         Xc1.L         Xc1.K         Xc1.M         Xc1.F         Xc1.P
##   10.83391488    5.54293319    1.76366056    4.53512716    7.55854527
##         Xc1.S         Xc1.T         Xc1.W         Xc1.Y         Xc1.V
##   12.59757544    6.29878772    3.27536961    6.04683621    7.05464225
## Xc2.lambda.1 Xc2.lambda.2 Xc2.lambda.3 Xc2.lambda.4 Xc2.lambda.5
##    0.02514092    0.02500357    0.02527773    0.02553159    0.02445265
```

The `extrProtPAAC()` fucntion also provides the `props` and `customprops` arguments, which is similar to the functions for Moreau-Broto/Moran/Geary autocorrelation descriptors. For minor differences, see `?extrPAAC`. Users could specify the lambda parameter and the weighting factor with arguments `lambda` and `w`.

**Note**: In the work of Kuo-Chen Chou, the definition for “normalized occurrence frequency” was not given. In this work, we define it as the occurrence frequency of amino acid in the sequence normalized to 100% and hence our calculated values are not the same as values by them.

## 4.9. Profile-based Descriptors

The profile-based descriptors for protein sequences are available in the **BioMedR** package. The feature vectors of profile-based methods were based on the PSSM by running PSI-BLAST, and often show good performance. See Ye *et al.* (2011) and Rangwala and Karypis (2005) for details. The functions `extrProtPSSM()`, `extrProtPSSMAcc()` and `extrProtPSSMFeature()` are used to generate these descriptors. Users need to install the NCBI-BLAST+ software package first to make the functions fully functional.

## 4.10. Descriptors for Proteochemometric Modeling

Proteochemometric (PCM) modeling utilizes statistical modeling techniques to model ligand-target interaction space. The below descriptors implemented in **BioMedR** are extensively used in Proteochemometric modeling.

- Scales-based descriptors derived by Principal Components Analysis
  - Scales-based descriptors derived by Amino Acid Properties from AAindex (Protein Fingerprint)

  - Scales-based descriptors derived by 20+ classes of 2D and 3D molecular descriptors (Topological, WHIM, VHSE, etc.)

- Scales-based descriptors derived by Factor Analysis
- Scales-based descriptors derived by Multidimensional Scaling
- BLOSUM and PAM matrix-derived descriptors

Note that each of the scales-based descriptor functions are freely to combine with the more than 20 classes of 2D and 3D molecular descriptors to construct highly customized scales-based descriptors. Of course, these functions are designed to be flexible enough that users could provide totally self-defined property matrices to construct scales-based descriptors.

For example, to compute the "topological scales" derived by PCA (using the first 5 principal components), one could use `extrProtDescScales()`:

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'BioMedR'))[[1]]
descscales = extrPCMDescScales(x, propmat = 'AATopo',
                               index = c(37:41, 43:47),
                               pc = 5, lag = 7, silent = FALSE)

## Summary of the first 5 principal components:
##                             PC1      PC2       PC3       PC4        PC5
## Standard deviation     2.581537 1.754133 0.4621854 0.1918666 0.08972087
## Proportion of Variance 0.666430 0.307700 0.0213600 0.0036800 0.00080000
## Cumulative Proportion  0.666430 0.974130 0.9954900 0.9991700 0.99998000
```

the argument `propmat` involkes the `AATopo` dataset shipped with **BioMedR** package, and the argument `index` selects the 37 to 41 and the 43 to 47 columns (molecular descriptors) in the `AATopo` dataset to use, the parameter `lag` was set for the Auto Cross Covariance (ACC) for generating scales-based descriptors of the same length. At last, we printed the summary of the first 5 principal components (standard deviation, proportion of variance, cumulative proportion of variance).

The result is a length 175 named vector, which is consistent with the descriptors before:

```
length(descscales)
## [1] 175
head(descscales, 15)

##     scl1.lag1     scl2.lag1     scl3.lag1     scl4.lag1     scl5.lag1
## -2.645644e-01 -1.717847e-02  1.975438e-02 -7.930659e-05 -3.710597e-05
##     scl1.lag2     scl2.lag2     scl3.lag2     scl4.lag2     scl5.lag2
##  3.548612e-01  1.343712e-01  5.699395e-03 -5.489472e-04 -6.364577e-05
##     scl1.lag3     scl2.lag3     scl3.lag3     scl4.lag3     scl5.lag3
##  2.011431e-02 -9.211136e-02 -1.461755e-03  6.747801e-04  2.386782e-04
```

For another example, to compute the descriptors derived by BLOSUM62 matrix and use the first 5 scales, one could use:

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'BioMedR'))[[1]]
blosum = extrPCMBLOSUM(x, submat = 'AABLOSUM62',
                       k = 5, lag = 7, scale = TRUE, silent = FALSE)

## Relative importance of all the possible 20 scales:
##  [1]  1.204960e+01  7.982007e+00  6.254364e+00  4.533706e+00  4.326286e+00
##  [6]  3.850579e+00  3.752197e+00  3.538207e+00  3.139155e+00  2.546405e+00
## [11]  2.373286e+00  1.666259e+00  1.553126e+00  1.263685e+00  1.024699e+00
## [16]  9.630187e-01  9.225759e-01  7.221636e-01  1.020085e-01 -4.714220e-16
```

The result is a length 175 named vector:

```
length(blosum)
## [1] 175
head(blosum, 15)

##     scl1.lag1     scl2.lag1     scl3.lag1     scl4.lag1     scl5.lag1
##  0.0042370555 -0.0021502057  0.0005993291  0.0006456375  0.0014849592
##     scl1.lag2     scl2.lag2     scl3.lag2     scl4.lag2     scl5.lag2
## -0.0014919096  0.0032873726  0.0011734162 -0.0021758536 -0.0018127568
##     scl1.lag3     scl2.lag3     scl3.lag3     scl4.lag3     scl5.lag3
## -0.0029413528  0.0001494193  0.0003298806 -0.0017877430 -0.0051044133
```

**Dealing with gaps.** In proteochemometrics, (sequence alignment) gaps can be very useful, since a gap in a certain position contains information. The **BioMedR** package has built-in support for such gaps. We deal with the gaps by using a dummy descriptor to code for the 21st type of amino acid. The function `extrPCMScalesGap()` and `extrPCMPropscaleGap()` could be used to deal with such gaps. See `?extrPCMScalesGap` and `?extrPCMPropscaleGap` for details.

## 4.11. Summary

The summary of the protein descriptors in the **BioMedR** package is listed in table 10.

The summary of the scales-based PCM descriptors in the **BioMedR** package is listed in table 11.

The summary of the amino acid descriptor sets used by scales-based descriptors provided in the **BioMedR** package is listed in table 11. Note that the non-informative descriptors (like the descriptors have only one value across all the 20 amino acids) in these datasets have already been filtered out.

[1]The number depends on the choice of the number of properties of amino acids and the choice of the maximum values of the `lag`. The default is use 8 types of properties and `lag` $= 30$.

[2]The number depends on the maximum value of `lag`. By default `lag` $= 30$. And two distance matrices were used, so the descriptor dimension is $30 \times 2 = 60$ and $(20 + 30) \times 2 = 100$.

[3]The number depends on the choice of the number of the set of amino acid properties and the choice of the $\lambda$ value. The default is use 3 types of properties proposed by Kuo-Chen Chou and $\lambda = 30$.

[4]The number depends on the choice of the $\lambda$ vlaue. The default is that $\lambda = 30$.

Table 10: List of commonly used descriptors in **BioMedR**

| Descriptor Group | Descriptor Name | Descriptor Dimension | Function Name |
|---|---|---|---|
| Amino Acid Composition | Amino Acid Composition | 20 | `extrProtAAC()` |
| | Dipeptide Composition | 400 | `extrProtDC()` |
| | Tripeptide Composition | 8000 | `extrProtTC()` |
| Autocorrelation | Normalized Moreau-Broto Autocorrelation | 240[1] | `extrProtMoreauBroto()` |
| | Moran Autocorrelation | 240[1] | `extrProtMoran()` |
| | Geary Autocorrelation | 240[1] | `extrProtGeary()` |
| CTD | Composition | 21 | `extrProtCTDC()`, `extrProtCTDCClass()` |
| | Transition | 21 | `extrProtCTDT()`, `extrProtCTDTClass()` |
| | Distribution | 105 | `extrProtCTDD()`, `extrProtCTDDClass()` |
| Conjoint Triad | Conjoint Triad | 343 | `extrProtCTriad()`, `extrProtCTriadClass()` |
| Quasi-Sequence-Order | Sequence-Order-Coupling Number | 60[2] | `extrProtSOCN()` |
| | Quasi-Sequence-Order Descriptors | 100[2] | `extrProtQSO()` |
| Pseudo-Amino Acid Composition | Pseudo-Amino Acid Composition | 50[3] | `extrProtPAAC()` |
| | Amphiphilic Pseudo-Amino Acid Composition | 80[4] | `extrProtAPAAC()` |

Table 11: List of PCM descriptors in **BioMedR**

| Derived by | Descriptor Class | Function Name |
|---|---|---|
| Principal Components Analysis | Scales-based descriptors derived by Principal Components Analysis | `extrPCMScales()`, `extrPCMScalesGap()` |
| | Scales-based descriptors derived by amino acid properties from AAindex (a.k.a. Protein Fingerprint) | `extrPCMPropScales()`, `extrProtFPGap()` |
| | Scales-based descriptors derived by 2D and 3D molecular descriptors (Topological, WHIM, VHSE, etc.) | `extrPCMDescScales()` |
| Factor Analysis | Scales-based descriptors derived by Factor Analysis | `extrPCMFAScales()` |
| Multidimensional Scaling | Scales-based descriptors derived by Multidimensional Scaling | `extrPCMMDSScales()` |
| Substitution Matrix | BLOSUM and PAM matrix-derived descriptors | `extrPCMBLOSUM()` |
| | Auto cross covariance (ACC) for generating scales-based descriptors of the same length | `acc()` |

Table 12: List of the pre-calculated descriptor sets of the 20 amino acids in **BioMedR**

| Dataset Name | Descriptor Set Name | Dimensionality | Calculated by |
|---|---|---|---|
| `AA2DACOR` | 2D Autocorrelations Descriptors | 92 | Dragon |
| `AA3DMoRSE` | 3D-MoRSE Descriptors | 160 | Dragon |
| `AAACF` | Atom-Centred Fragments Descriptors | 6 | Dragon |
| `AABurden` | Burden Eigenvalues Descriptors | 62 | Dragon |
| `AAConn` | Connectivity Indices Descriptors | 33 | Dragon |
| `AAConst` | Constitutional Descriptors | 23 | Dragon |
| `AAEdgeAdj` | Edge Adjacency Indices Descriptors | 97 | Dragon |
| `AAEigIdx` | Eigenvalue-Based Indices Descriptors | 44 | Dragon |
| `AAFGC` | Functional Group Counts Descriptors | 5 | Dragon |
| `AAGeom` | Geometrical Descriptors | 41 | Dragon |
| `AAGETAWAY` | GETAWAY Descriptors | 194 | Dragon |
| `AAInfo` | Information Indices Descriptors | 47 | Dragon |
| `AAMolProp` | Molecular Properties Descriptors | 12 | Dragon |
| `AARandic` | Randic Molecular Profiles Descriptors | 41 | Dragon |
| `AARDF` | RDF Descriptors | 82 | Dragon |
| `AATopo` | Topological Descriptors | 78 | Dragon |
| `AATopoChg` | Topological Charge Indices Descriptors | 15 | Dragon |
| `AAWalk` | Walk and Path Counts Descriptors | 40 | Dragon |
| `AAWHIM` | WHIM Descriptors | 99 | Dragon |
| `AACPSA` | CPSA Descriptors | 41 | Accelrys Discovery Studio |
| `AADescAll` | All the 2D Descriptors Calculated by Dragon | 1171 | Dragon |
| `AAMOE2D` | All the 2D Descriptors Calculated by MOE | 148 | MOE |
| `AAMOE3D` | All the 3D Descriptors Calculated by MOE | 143 | MOE |

# 5. Calculating DNA/RNA Commonly Used Descriptors

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

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

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

## 5.1. Kmer

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

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

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

```
require(BioMedR)
x = "GGAGTATGAGGCCGAATCTCATCCTCTAGTCCCAAGCCTCTCCACTACCAGGGCT"
extrDNAkmer(x)
```

```
## AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
##  2  2  5  3  4  7  1  7  3  3  4  2  3  7  1  0
```

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

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

```
## AA AC AG AT CA CC CG GA GC TA
##  2  4 12  3  5 11  1 10  3  3
```

## 5.2. Increment of diversity

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

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

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

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

```
## [[1]]
##  [1]    1.253961    3.582857 -175.436114 -175.249244 -261.305425
##  [6] -261.542600 -290.422865 -290.422865 -287.293692 -287.293692
## [11] -282.192809 -282.192809
```

### 5.3. Dinucleotide-based auto covariance

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

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

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

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

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

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

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

```
extrDNADAC(x)

## Twist.lag1 Twist.lag2  Tilt.lag1  Tilt.lag2
##     -0.223     -0.087     -0.123     -0.043

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

## Twist.lag1 Twist.lag2  Tilt.lag1  Tilt.lag2 cust1.lag1 cust1.lag2
##     -0.223     -0.087     -0.123     -0.043     -0.223     -0.061
## cust2.lag1 cust2.lag2
##     -0.247     -0.019
```

### 5.4. Dinucleotide-based cross covariance

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

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

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

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

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

```
extrDNADCC(x)

## Twist.Tilt.lag.1 Twist.Tilt.lag.2 Tilt.Twist.lag.1 Tilt.Twist.lag.2
##           -0.134           -0.107           -0.305           -0.036

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

##  Twist.Tilt.lag.1  Twist.Tilt.lag.2  Tilt.Twist.lag.1  Tilt.Twist.lag.2
##            -0.134            -0.107            -0.305            -0.036
## Twist.cust1.lag.1 Twist.cust1.lag.2 cust1.Twist.lag.1 cust1.Twist.lag.2
##            -0.289            -0.035            -0.140            -0.079
## Twist.cust2.lag.1 Twist.cust2.lag.2 cust2.Twist.lag.1 cust2.Twist.lag.2
##            -0.081            -0.007            -0.177            -0.086
##  Tilt.cust1.lag.1  Tilt.cust1.lag.2  cust1.Tilt.lag.1  cust1.Tilt.lag.2
##            -0.354             0.035            -0.035            -0.056
##  Tilt.cust2.lag.1  Tilt.cust2.lag.2  cust2.Tilt.lag.1  cust2.Tilt.lag.2
##            -0.206             0.075            -0.003            -0.117
## cust1.cust2.lag.1 cust1.cust2.lag.2 cust2.cust1.lag.1 cust2.cust1.lag.2
##            -0.045            -0.059            -0.338             0.055
```

### 5.5. Dinucleotide-based auto-cross covariance

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

```
extrDNADACC(x)
```

```
##        Twist.lag1        Twist.lag2         Tilt.lag1         Tilt.lag2
##            -0.223            -0.087            -0.123            -0.043
## Twist.Tilt.lag.1 Twist.Tilt.lag.2 Tilt.Twist.lag.1 Tilt.Twist.lag.2
##            -0.134            -0.107            -0.305            -0.036
```

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

```
##          Twist.lag1          Twist.lag2           Tilt.lag1           Tilt.lag2
##              -0.223              -0.087              -0.123              -0.043
##          cust1.lag1          cust1.lag2          cust2.lag1          cust2.lag2
##              -0.223              -0.061              -0.247              -0.019
##   Twist.Tilt.lag.1   Twist.Tilt.lag.2   Tilt.Twist.lag.1   Tilt.Twist.lag.2
##              -0.134              -0.107              -0.305              -0.036
## Twist.cust1.lag.1 Twist.cust1.lag.2 cust1.Twist.lag.1 cust1.Twist.lag.2
##              -0.289              -0.035              -0.140              -0.079
## Twist.cust2.lag.1 Twist.cust2.lag.2 cust2.Twist.lag.1 cust2.Twist.lag.2
##              -0.081              -0.007              -0.177              -0.086
##   Tilt.cust1.lag.1   Tilt.cust1.lag.2   cust1.Tilt.lag.1   cust1.Tilt.lag.2
##              -0.354               0.035              -0.035              -0.056
##   Tilt.cust2.lag.1   Tilt.cust2.lag.2   cust2.Tilt.lag.1   cust2.Tilt.lag.2
##              -0.206               0.075              -0.003              -0.117
## cust1.cust2.lag.1 cust1.cust2.lag.2 cust2.cust1.lag.1 cust2.cust1.lag.2
##              -0.045              -0.059              -0.338               0.055
```

## 5.6. Trinucleotide-based auto covariance

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

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

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

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

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

```
extrDNATAC(x)

##    Dnase I.lag1    Dnase I.lag2 Nucleosome.lag1 Nucleosome.lag2
##          -0.033          -0.023           0.276          -0.015

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

##    Dnase I.lag1    Dnase I.lag2 Nucleosome.lag1 Nucleosome.lag2
##          -0.033          -0.023           0.271          -0.015
##       cust.lag1       cust.lag2
##          -0.003          -0.004
```

## 5.7. Trinucleotide-based cross covariance

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

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

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

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

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

```
extrDNATCC(x)

## Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2 Nucleosome.Dnase I.lag.1
##                   -0.186                   -0.043                   -0.079
## Nucleosome.Dnase I.lag.2
##                   -0.082

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

## Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2 Nucleosome.Dnase I.lag.1
##                   -0.183                   -0.043                   -0.078
## Nucleosome.Dnase I.lag.2       Dnase I.cust.lag.1       Dnase I.cust.lag.2
##                   -0.081                   -0.004                    0.018
##       cust.Dnase I.lag.1       cust.Dnase I.lag.2    Nucleosome.cust.lag.1
##                   -0.062                   -0.117                   -0.061
##    Nucleosome.cust.lag.2    cust.Nucleosome.lag.1    cust.Nucleosome.lag.2
##                    0.055                   -0.123                   -0.102
```

## 5.8. Trinucleotide-based auto-cross covariance

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

```
extrDNATACC(x)

##             Dnase I.lag1             Dnase I.lag2          Nucleosome.lag1
##                   -0.033                   -0.023                    0.276
##          Nucleosome.lag2 Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2
##                   -0.015                   -0.186                   -0.043
## Nucleosome.Dnase I.lag.1 Nucleosome.Dnase I.lag.2
##                   -0.079                   -0.082
```

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

```
##             Dnase I.lag1             Dnase I.lag2          Nucleosome.lag1
##                   -0.033                   -0.023                    0.271
##          Nucleosome.lag2                cust.lag1                cust.lag2
##                   -0.015                   -0.003                   -0.004
## Dnase I.Nucleosome.lag.1 Dnase I.Nucleosome.lag.2 Nucleosome.Dnase I.lag.1
##                   -0.183                   -0.043                   -0.078
## Nucleosome.Dnase I.lag.2       Dnase I.cust.lag.1       Dnase I.cust.lag.2
##                   -0.081                   -0.004                    0.018
##       cust.Dnase I.lag.1       cust.Dnase I.lag.2    Nucleosome.cust.lag.1
##                   -0.062                   -0.117                   -0.061
##    Nucleosome.cust.lag.2    cust.Nucleosome.lag.1    cust.Nucleosome.lag.2
##                    0.055                   -0.123                   -0.102
```

## 5.9. Pseudo dinucleotide composition

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

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

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

where

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

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

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

where the correlation function is given by

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

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

```
extrDNAPseDNC(x)

##         Xc1.AA        Xc1.AC        Xc1.AG        Xc1.AT        Xc1.CA
##          0.030         0.030         0.076         0.046         0.061
##         Xc1.CC        Xc1.CG        Xc1.CT        Xc1.GA        Xc1.GC
##          0.106         0.015         0.106         0.046         0.046
##         Xc1.GG        Xc1.GT        Xc1.TA        Xc1.TC        Xc1.TG
##          0.061         0.030         0.046         0.106         0.015
##         Xc1.TT Xc2.lambda.1 Xc2.lambda.2 Xc2.lambda.3
##          0.000         0.067         0.059         0.053

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

##         Xc1.AA        Xc1.AC        Xc1.AG        Xc1.AT        Xc1.CA
##          0.027         0.027         0.067         0.040         0.054
##        Xc1.CC        Xc1.CG        Xc1.CT        Xc1.GA        Xc1.GC
##         0.094         0.013         0.094         0.040         0.040
##        Xc1.GG        Xc1.GT        Xc1.TA        Xc1.TC        Xc1.TG
##         0.054         0.027         0.040         0.094         0.013
##        Xc1.TT Xc2.lambda.1 Xc2.lambda.2
##         0.000         0.143         0.128
```

## 5.10. Pseudo k-tupler composition

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

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

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

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

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

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

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

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

```
pseknc = extrDNAPseKNC(x)
head(pseknc)
```

```
## Xc1.AAA Xc1.AAC Xc1.AAG Xc1.AAT Xc1.ACA Xc1.ACC
##    0.00    0.00    0.01    0.01    0.00    0.01
```

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

##       Xc1.AA       Xc1.AC       Xc1.AG       Xc1.AT       Xc1.CA
##        0.034        0.034        0.085        0.051        0.068
##       Xc1.CC       Xc1.CG       Xc1.CT       Xc1.GA       Xc1.GC
##        0.119        0.017        0.119        0.051        0.051
##       Xc1.GG       Xc1.GT       Xc1.TA       Xc1.TC       Xc1.TG
##        0.068        0.034        0.051        0.119        0.017
##       Xc1.TT Xc2.lambda.1
##        0.000        0.085
```

## 5.11. Summary

The summary of the DNA/RNA descriptors in the **BioMedR** package is listed in table 13.

Table 13: List of commonly used descriptors in **BioMedR**

| Descriptor Group | Descriptor Name | Descriptor Dimension | Function Name |
|---|---|---|---|
| Nucleic acid composition | Basic kmer | 16[1] | `extrDNAkmer()` |
| | Reverse compliment kmer | 10[1] | `extrDNAkmer()` |
| | Increment of diversity | 12[2] | `extrDNAIncDiv()` |
| Autocorrelation | Dinucleotide-based auto covariance | 4[3] | `extrDNADAC()` |
| | Dinucleotide-based cross covariance | 4[3] | `extrDNADCC()` |
| | Dinucleotide-based auto-cross covariance | 8[3] | `extrDNADACC()` |
| | Trinucleotide-based auto covariance | 4[3] | `extrDNATAC()` |
| | Trinucleotide-based cross covariance | 4[3] | `extrDNATCC()` |
| | Trinucleotide-based auto-cross covariance | 8[3] | `extrDNATACC()` |
| Pseudo nucleotide composition | Pseudo dinucleotide composition | 19[4] | `extrDNAPseDNC()` |
| | Pseudo k-tupler nucleotide composition | 65[5] | `extrDNAPseKNC()` |

The summary of the names of the 38 physicochemical indices for dinucleotides. in the **BioMedR** package is listed in table 14.

The summary of the names of the 12 physicochemical indices for trinucleotides. in the **BioMedR** package is listed in table 15.

The summary of the names of the 6 physicochemical indices for dinucleotides. in the **BioMedR** package is listed in table 16.

[1]The number depends on the choice of the k value of kmer. The default is $\boldsymbol{k} = 2$.

[2]The number depends on the choice of the k value of kmer. The default is $\boldsymbol{k} = 6$.

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

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

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

Table 14: The names of the 38 physicochemical indices for dinucleotides

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

Table 15: The names of the 12 physicochemical indices for trinucleotides.

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

Table 16: The names of the 6 physicochemical indices for dinucleotides.

| | | |
|---|---|---|
| Twist | Tilt | Roll |
| Shift | Slide | Rise |

## 6. Generating Interaction Descriptors between Drug, Protein and DNA/RNA

For chemogenomics modeling, **BioMedR** calculates compound-protein interaction (CPI) descriptors, compound-DNA descriptors (CDI), DNA-protein descriptors (DPI), DNA-DNA descriptors (DDI), compound-compound descriptors (CCI) and protein-protein interaction (PPI) descriptors.

### 6.1. Generating Drug-Target Interaction Descriptors

The prediction of novel interactions between drugs and target proteins is a key area in genomic drug discovery.

A drug-target interaction network can be naturally modeled as a bipartite graph, where the nodes are target proteins or drug molecules and edges (only drugs and proteins could be connected by edges) represent drug-target interactions. They are constructed as follows:

1. Separate the pairs in the above positive samples into single drugs and proteins;
2. Re-couple these singles into pairs in a way that none of them occurs in the corresponding positive dataset.

Ten generated negative sets were used in Cao *et al.* (2012a), here we only use one of them for a demonstration. The drug ID and target ID is stored in `GPCR.csv`. The first column is KEGG protein ID, and the second column is KEGG drug ID. The first 635 rows is the positive set, and the last 635 rows is the negative set.

```
require(BioMedR)

gpcr = read.table(system.file('vignettedata/GPCR.csv', package = 'BioMedR'),
                  header = FALSE, as.is = TRUE)
```

Get a glimpse of the data:

```
head(gpcr)

##          V1     V2
## 1 hsa:10161 D00528
## 2 hsa:10800 D00411
## 3 hsa:10800 D01828
## 4 hsa:10800 D05129
## 5 hsa:11255 D00234
## 6 hsa:11255 D00300
```

Next, we will download the target protein sequences (in FASTA format) and drug molecule (in SMILES format) from the KEGG database, in parallel:

```
gpcr   = read.table(system.file('vignettedata/GPCR.csv', package = 'BioMedR'),
                    header = FALSE, as.is = TRUE)
protid = unique(gpcr[, 1])
drugid = unique(gpcr[, 2])

protseq = BMgetProtSeqKEGG(protid, parallel = 5)
drugseq = BMgetDrugSmiKEGG(drugid, parallel = 5)
x0.prot = cbind(t(sapply(unlist(protseq), extrProtAPAAC)),
                t(sapply(unlist(protseq), extrProtCTriad)))

x0.drug = cbind(extrDrugEstateComplete(readMolFromSmi(textConnection(drugseq))),
                extrDrugMACCSComplete(readMolFromSmi(textConnection(drugseq))))
```

If the connection is slow or accidentally interrupts, just try more times until success.

Since the descriptors is only for the *uniqued* drug and target list, we need to generate the full descriptor matrix for the training data:

```
x.prot = matrix(NA, nrow = nrow(gpcr), ncol = ncol(x0.prot))
x.drug = matrix(NA, nrow = nrow(gpcr), ncol = ncol(x0.drug))
for (i in 1:nrow(gpcr)) x.prot[i, ] = x0.prot[which(gpcr[, 1][i] == protid), ]
for (i in 1:nrow(gpcr)) x.drug[i, ] = x0.drug[which(gpcr[, 2][i] == drugid), ]
```

Generate drug-target interaction descriptors using `getCPI()`.

```
x = getCPI(x.prot, x.drug, type = 'combine')
head(x[, 1:5])

##          [,1]     [,2]      [,3]     [,4]     [,5]
## [1,] 13.37680 11.59323 19.619312 7.134295 13.37680
## [2,] 13.68312 10.03428 13.683115 9.122077 11.85870
## [3,] 13.68312 10.03428 13.683115 9.122077 11.85870
## [4,] 13.68312 10.03428 13.683115 9.122077 11.85870
## [5,] 43.48494 25.22126  9.566686 7.827289 10.43638
## [6,] 43.48494 25.22126  9.566686 7.827289 10.43638
```

The pairwise interaction is another useful type of representation in drug-target , drug-DNA/RNA, DNA/RNA-protein, DNA/RNA-DNA/RNA, drug-drug and protein-protein interaction prediction and related research. **BioMedR** also provides `getPPI()` to generate protein-protein interaction descriptors. `getPPI()`, `getDDI()` and `getCCI()` provides three types of interactions while `getCPI()`, `getCDI()` and `getDPI()` provides two types. The argument `type` is used to control this:

- Compound-Protein Interaction (CPI) Descriptors

  For compound descriptor vector $\boldsymbol{d_1^{1\times p_1}}$ and the protein descriptor vector $\boldsymbol{d_2^{1\times p_2}}$, there are two methods for construction of descriptor vector $\boldsymbol{d}$ for compound-protein interaction:

  1. `'combine'` - combine the two feature matrix, $\boldsymbol{d}$ has $\boldsymbol{p_1 + p_2}$ columns;
  2. `'tensorprod'` - column-by-column (pseudo)-tensor product type interactions, $\boldsymbol{d}$ has $\boldsymbol{p_1 \times p_2}$ columns.

- Protein-Protein Interaction (PPI) Descriptors

  For interaction protein A and protein B, let $\boldsymbol{d_1^{1\times p}}$ and $\boldsymbol{d_2^{1\times p}}$ be the descriptor vectors. There are three methods to construct the protein-protein interaction descriptor $\boldsymbol{d}$:

  1. `'combine'` - combine the two descriptor matrix, $\boldsymbol{d}$ has $\boldsymbol{p + p}$ columns;
  2. `'tensorprod'` - column-by-column (pseudo)-tensor product type interactions, $\boldsymbol{d}$ has $\boldsymbol{p \times p}$ columns;
  3. `'entrywise'` - entrywise product and entrywise sum of the two matrices, then combine them, $\boldsymbol{d}$ has $\boldsymbol{p + p}$ columns.

### 6.2. Summary

The summary of the compound protein DNA/RNA interation descriptors in the **BioMedR** package is listed in table 17.

# 7. Clustering

Table 17: Compound protein DNA/RNA interation descriptors

| Function name | Function description |
|---|---|
| `getPPI()` | Generating protein-protein interaction descriptors |
| `getCCI()` | Generating compound-compound interaction descriptors |
| `getDDI()` | Generating DNA-DNA interaction descriptors |
| `getCPI()` | Generating compound-protein interaction descriptors |
| `getCDI()` | Generating compound-DNA interaction descriptors |
| `getDPI()` | Generating DNA-protein interaction descriptors |

Apart from supervised methods (classification and regression), unsupervised approaches, like clustering, is also widely applied in the quantitative research of drugs.

In reality, there are usually too many chemical compounds available for identifying drug-like molecules. Thus it would be attractive using clustering methods to aid the selection of a representative subset of all available compounds. For a clustering approach that groups compounds together by their structural similarity, applying the principle *similar compounds have similar properties* means that we only need to test the representative compounds from each individual cluster, rather than do the time-consuming complete set of experiments, and this should be sufficient to understand the structure-activity relationships of the whole compound set.

## 7.1. Binning Clustering

Compound libraries can be clustered into discrete similarity groups with the binning clustering function `clusterCMP`. The function accepts as input an atom pair (`APset`) or a fingerprint (`FPset`) descriptor database as well as a similarity threshold. The binning clustering result is returned in form of a data frame. Single linkage is used for cluster joining. The function calculates the required compound-to-compound distance information on the fly, while a memory-intensive distance matrix is only created upon user request via the `save.distances` argument (see below).

Because an optimum similarity threshold is often not known, the `clusterCMP` function can calculate cluster results for multiple cutoffs in one step with almost the same speed as for a single cutoff. This can be achieved by providing several cutoffs under the cutoff argument. The clustering results for the different cutoffs will be stored in one data frame.

One may force the `clusterCMP` function to calculate and store the distance matrix by supplying a file name to the `save.distances` argument. The generated distance matrix can be loaded and passed on to many other clustering methods available in R, such as the hierarchical clustering function `hclust` (see below).

If a distance matrix is available, it may also be supplied to `clusterCMP` via the `save.distances` argument. This is useful when one has a pre-computed distance matrix either from a previous call to `clusterCMP` or from other distance calculation subroutines.

Single-linkage binning clustering with one or multiple cutoffs:

```
data(sdfbcl)
apbcl = convSDFtoAP(sdfbcl)
```

```
clusters <- clusterCMP(db = apbcl, cutoff = c(0.7, 0.8, 0.9),
                                                   quiet = TRUE)
head(clusters)
```

```
##       ids CLSZ_0.7 CLID_0.7 CLSZ_0.8 CLID_0.8 CLSZ_0.9 CLID_0.9
## 38 CMP38       10       38        5       38        3       38
## 39 CMP39       10       38        5       38        1       39
## 40 CMP40       10       38        5       38        1       40
## 41 CMP41       10       38        5       38        3       38
## 44 CMP44       10       38        5       38        3       38
## 42 CMP42       10       38        2       42        1       42
```

Clustering of `FPset` objects with multiple cutoffs:

```
fpbcl = convAPtoFP(apbcl)
clusters2 <- clusterCMP(fpbcl, cutoff=c(0.5, 0.7, 0.9),
                           method = "Tanimoto", quiet = TRUE)
head(clusters2)
```

```
##    ids CLSZ_0.5 CLID_0.5 CLSZ_0.7 CLID_0.7 CLSZ_0.9 CLID_0.9
## 1 CMP1       50        1       47        1       18        1
## 2 CMP2       50        1       47        1       18        1
## 3 CMP3       50        1       47        1       18        1
## 4 CMP4       50        1       47        1        1        4
## 5 CMP5       50        1       47        1       18        1
## 6 CMP6       50        1       47        1       18        1
```

Sames as above, but using Tversky similarity measure:

```
clusters3 <- clusterCMP(fpbcl, cutoff = c(0.5, 0.7, 0.9),
                        method = "Tversky", alpha = 0.3, beta = 0.7, quiet = TRUE)

head(clusters3)
```

```
##    ids CLSZ_0.5 CLID_0.5 CLSZ_0.7 CLID_0.7 CLSZ_0.9 CLID_0.9
## 1 CMP1       50        1       50        1       50        1
## 2 CMP2       50        1       50        1       50        1
## 3 CMP3       50        1       50        1       50        1
## 4 CMP4       50        1       50        1       50        1
## 5 CMP5       50        1       50        1       50        1
## 6 CMP6       50        1       50        1       50        1
```

Return cluster size distributions for each cutoff:

```
clusterStat(clusters, cluster.result=1)
```

```
##   cluster size count
## 1            1     8
## 2            2     3
## 3            3     2
## 4            4     2
## 5            6     2
## 6           10     1
```

## 7.2. Jarvis-Patrick Clustering

The Jarvis-Patrick clustering algorithm is widely used in cheminformatics (Jarvis and Patrick 1973). It requires a nearest neighbor table, which consists of $j$ nearest neighbors for each item. The nearest neighbor table is then used to join items into clusters when they meet the following requirements: (a) they are contained in each other's neighbor list and (b) they share at least $k$ nearest neighbors. The values for $j$ and $k$ are user-defined parameters. The `clusterJP` function implemented in **BioMedR** takes a nearest neighbor table generated by `NNeighbor`, which works for `APset` and `FPset` objects. This function takes either the standard Jarvis-Patrick $j$ parameter (as the `numNbrs` parameter), or else a `cutoff` value, which is an extension to the basic algorithm that we have added. Given a cutoff value, the nearest neighbor table returned contains every neighbor with a similarity greater than the cutoff value, for each item. This allows one to generate tighter clusters and to minimize certain limitations of this method, such as false joins of completely unrelated items when operating on small data sets.

The `clusterJP` function also allows one to relaxsome of the requirements of the algorithm through the `mode` parameter. When set to "a1a2b", then all requirements are used. If set to "a1b", then (a) is relaxed to a unidirectional requirement. Lastly, if `mode` is set to "b", then only requirement (b) is used, which means that all pairs of items will be checked to see if (b) is satisfied between them. The size of the clusters generated by the different methods increases in this order: "a1a2b" < "a1b" < "b". The run time of method "a1a2b" follows a close to linear relationship, while it is nearly quadratic for the much more exhaustive method "b". Only methods "a1a2b" and "a1b" are suitable for clustering very large data sets (e.g. > 50,000 items) in a reasonable amount of time.

An additional extension to the algorithm is the ability to set the linkage mode. The `linkage` parameter can be one of "single", "average", or "complete", for single linkage, average linkage and complete linkage merge requirements, respectively. In the context of Jarvis-Patrick, average linkage means that at least half of the pairs between the clusters under consideration must meet requirement (b). Similarly, for complete linkage, all pairs must requirement (b). Single linkage is the normal case for Jarvis-Patrick and just means that at least one pair must meet requirement (b).

The output is a cluster `vector` with the item labels in the name slot and the cluster IDs in the data slot. There is a utility function called `byCluster`, which takes out cluster vector output by `clusterJP` and transforms it into a list of vectors. Each slot of the list is named with a cluster id and the vector contains the cluster members. By default the function excludes singletons from the output, but they can be included by setting `excludeSingletons` is `FALSE`.

Standard Jarvis-Patrick clustering on `APset` and `FPset` objects:

```
jpap = clusterJP(NNeighbors(apbcl, numNbrs = 6), k = 5, mode = "a1a2b")
head(jpap)

## CMP1 CMP2 CMP3 CMP4 CMP5 CMP6
##    1    1    1    2    2    2

jpfp = clusterJP(NNeighbors(fpbcl, numNbrs = 6), k = 5, mode = "a1a2b")
head(jpfp)

## CMP1 CMP2 CMP3 CMP4 CMP5 CMP6
##    1    2    2    3    4    5
```

Jarvis-Patrick clustering with a minimum similarity cutoff value (here Tanimoto coefficient). In addition, it uses the much more exhaustive "b" method that generates larger cluster sizes, but significantly increased the run time. For more details, consult the corresponding help file with ?`clusterJP`.

```
require(ChemmineR)
cl <- clusterJP(NNeighbors(fpbcl, cutoff = 0.6,
                          method = "Tanimoto"), k = 2 ,mode = "b")
byCluster(cl)

## $`1`
##  [1] "CMP1"  "CMP2"  "CMP3"  "CMP4"  "CMP5"  "CMP6"  "CMP7"  "CMP8"
##  [9] "CMP9"  "CMP10" "CMP13" "CMP14" "CMP15" "CMP16" "CMP17" "CMP18"
## [17] "CMP19" "CMP20" "CMP21" "CMP22" "CMP23" "CMP24" "CMP25" "CMP26"
## [25] "CMP27" "CMP28" "CMP30" "CMP31" "CMP32" "CMP33" "CMP34" "CMP35"
## [33] "CMP36" "CMP37" "CMP38" "CMP39" "CMP40" "CMP41" "CMP42" "CMP43"
## [41] "CMP44" "CMP45" "CMP46" "CMP47" "CMP48" "CMP49" "CMP50"
```

## 7.3. Multi-Dimensional Scaling (MDS)

To visualize and compare clustering results, the `clusterPlot` function can be used. The function performs Multi-Dimensional Scaling (MDS) and visualizes the results in form of a scatter plot. It requires as input an `APset`, a clustering result from `clusterCMP`, and a cutoff for the minimum cluster size to consider in the plot. To help determining a proper cutoff size, the `clusterStat` function is provided to generate cluster size statistics.

MDS clustering and scatter plot:

```
mds = clusterMDS(apbcl, clusters, size.cutoff = 2, quiet = TRUE)
head(mds)

##                V1         V2 CLID_0.7 Clicked
## CMP38 -0.08110183 -0.3937917       38       0
## CMP39 -0.07078328 -0.4024284       38       0
```

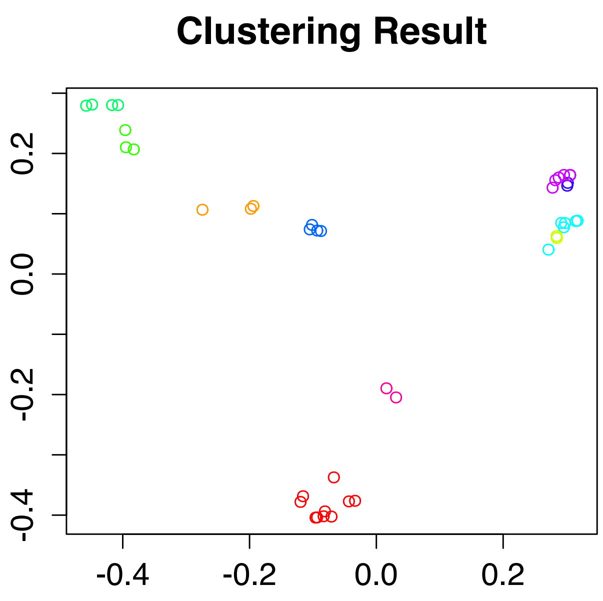

Figure 5: Multi-Dimensional Scaling (MDS)

```
## CMP40 -0.08319071 -0.4019104        38         0
## CMP41 -0.09350389 -0.4038405        38         0
## CMP44 -0.09595493 -0.4041470        38         0
## CMP42 -0.11566290 -0.3686453        38         0
```

### 7.4. Kohonen's self-organising map (SOM)

Self-organizing maps (Kohonen 2001) tackle the problem in a way similar to MDS, but instead of trying to reproduce distances they aim at reproducing topology, or in other words, they try to keep the same neighbours. So if two high-dimensional objects are very similar, then their position in a two-dimensional plane should be very similar as well. Rather than mapping objects in a continuous space, SOMs use a regular grid of units onto which objects are mapped. The differences with MDS can be seen as both strengths and weaknesses: where in a 2D MDS plot a distance – also a large distance – can be directly interpreted as an estimate of the true distance, in a SOM plot this is not the case: one can only say that objects mapped to the same, or neighbouring, units are very similar. In other words, SOMs concentrate on the largest similarities, whereas MDS concentrates on the largest dissimilarities. Which of these is more useful depends on the application.

There is an example: we collected 80 drugs binding to bcl2 protein and 300 compounds unbinding to bcl2 protein. Firstly, we analyze the global character for total 380 compounds and maped them to 5 $\times$ 5 lattices. Secondly, we analyzed the structral characters for 80 drugs and maped them to these 5 $\times$ 5 lattices and counted the drugs in each lattice. From the result, we can choose some negative compounds for bcl2 protein with chemical structures similar to its positive drugs. The R code to generate the plot is as follows:

```
data(som.bcl)

idx = unique(som.bcl$unit.classif[1:80])
bgcols <- c("gray", "lightgreen")
index = rep(1, 5 * 5)
index[idx]  = 2
bgcol = bgcols[as.integer(index)]
label = c()
for (i in 1:length(idx)) {
  label[i] = length(which(som.bcl$unit.classif[1:80] == idx[i]))
  }
opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
clusterPlotSOMmap(som.bcl, pchs = 1)
clusterPlotSOMmap(som.bcl, index = idx, label = label, bgcol = bgcol)
par(opar)
```

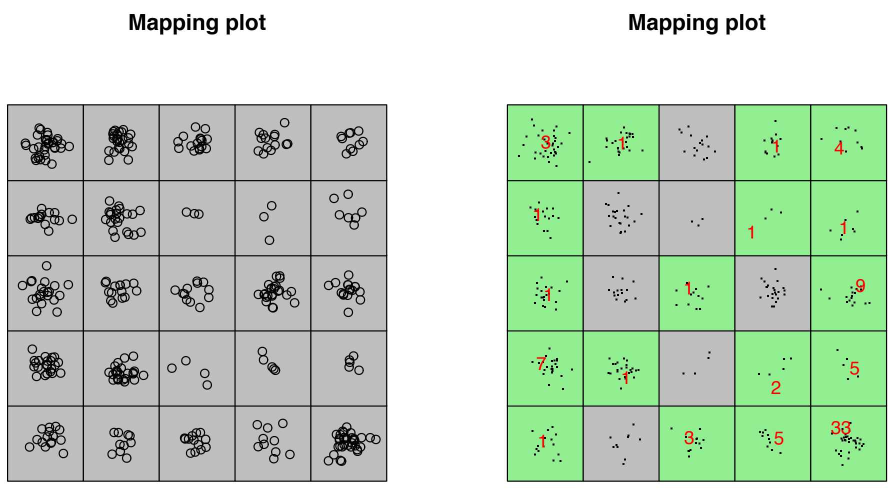

Figure 6: Kohonen's self-organising map (SOM)

## 7.5. Summary

The summary of the cluster methods in the **BioMedR** package is listed in table 18.

# 8. Similarity

## 8.1. Structure-Based Chemical Similarity Searching

Table 18: Clustering

| Function name | Function description |
|---|---|
| `clusterCMP()` | Binning Clustering |
| `clusterMDS()` | Multi-Dimensional Scaling (MDS) |
| `clusterJP()` | Jarvis-PatrickClustering |
| `clusterSOMPlot()` | Kohonen'sself-organisingmap(SOM) |

Structure-based chemical similarity searching ranks molecules in a database by their similarity degree to one query molecule structure. The numerical similarity value is usually computed based on the molecular fingerprints with selected metrics or by maximum common structure search. It is one of the core techniques for ligand-based virtual screening in drug discovery.

```
mol   = system.file('compseq/example.sdf', package = 'BioMedR')
moldb = system.file('compseq/bcl.sdf', package = 'BioMedR')
```

We could do parallelized drug molecular similarity search with the `searchDrug()` function in **BioMedR**. Here we choose the search criterion to be MACCS keys with cosine similarity, FP2 fingerprints with tanimoto similarity, and maximum common substructure search with tanimoto similarity.

```
rank1 = searchDrug(mol, moldb, cores = 4, method = 'fp',
                   fptype = 'maccs', fpsim = 'tanimoto')
rank2 = searchDrug(mol, moldb, cores = 4, method = 'fp',
                   fptype = 'fp2', fpsim = 'cosine')
```

The returned search result is stored as a numerical vector, each element's name is the molecule number in the database, and the value is the similarity value between the query molecule and this molecule. We shall print the top search results here:

The **BioMedR** package also integrated the functionality of converting molecular file formats. For example, we could convert the SDF files to SMILES files using `convMolFormat()`. Since the No. 50 molecule ranks the highest in the three searches performed, we will calculate the similarity derived by maximum common substructure search between the query molecule and the No. 50 molecule using `calcDrugMCSSim()`:

The MCS search result is stored in a list, which contains the original MCS result provided by the **fmcsR** package (Wang *et al.* 2013), the Tanimoto coefficient and the overlap coefficient.

## 8.2. Similarity Calculation by Sequence Alignment

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

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

```
> s1 = readFASTA(system.file('protseq/P00750.fasta', package = 'BioMedR'))[[1]]
> s2 = readFASTA(system.file('protseq/P08218.fasta', package = 'BioMedR'))[[1]]
> s3 = readFASTA(system.file('protseq/P10323.fasta', package = 'BioMedR'))[[1]]
> s4 = readFASTA(system.file('protseq/P20160.fasta', package = 'BioMedR'))[[1]]
> s5 = readFASTA(system.file('protseq/Q9NZP8.fasta', package = 'BioMedR'))[[1]]
> plist   = list(s1, s2, s3, s4, s5)
> psimmat = parSeqSim(plist, cores = 4, type = 'local', submat = 'BLOSUM62')
> print(psimmat)
```

```
           [,1]       [,2]       [,3]       [,4]       [,5]
[1,] 1.00000000 0.11825938 0.10236985 0.04921696 0.03943488
[2,] 0.11825938 1.00000000 0.18858241 0.12124217 0.06391103
[3,] 0.10236985 0.18858241 1.00000000 0.05819984 0.06175942
[4,] 0.04921696 0.12124217 0.05819984 1.00000000 0.05714638
[5,] 0.03943488 0.06391103 0.06175942 0.05714638 1.00000000
```

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

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

## 8.3. Similarity Calculation by GO Semantic Similarity Measures

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

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

```
# by GO Terms
> go1 = c('GO:0005215', 'GO:0005488', 'GO:0005515',
+         'GO:0005625', 'GO:0005802', 'GO:0005905')  # AP4B1
> go2 = c('GO:0005515', 'GO:0005634', 'GO:0005681',
+         'GO:0008380', 'GO:0031202')                # BCAS2
> go3 = c('GO:0003735', 'GO:0005622', 'GO:0005840',
+         'GO:0006412')                              # PDE4DIP
```

```
> glist = list(go1, go2, go3)
> gsimmat1 = parGOSim(glist, type = 'go', ont = 'CC')
> print(gsimmat1)

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

# by Entrez gene id
> genelist = list(c('150', '151', '152', '1814', '1815', '1816'))
> gsimmat2 = parGOSim(genelist, type = 'gene')
> print(gsimmat2)

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

## 8.4. Summary

The summary of the similarity in the **BioMedR** package is listed in table 19.

Table 19: Similarity and similarity searching

| Function name | Function description |
|---|---|
| `calcDrugFPSim()` | Calculate drug molecule similarity derived by molecular fingerprints |
| `calcDrugMCSSim()` | Calculate drug molecule similarity derived by maximum common substructure search |
| `searchDrug()` | Parallelized drug molecule similarity search by molecular fingerprints similarity or maximum common substructure search |
| `calcTwoProtSeqSim()` | Similarity calculation based on sequence alignment for a pair of protein sequences |
| `calcParProtSeqSim()` | Parallellized protein sequence similarity calculation based on sequence alignment |
| `calcTwoProtGOSim()` | Similarity calculation based on Gene Ontology (GO) similarity between two proteins |
| `calcParProtGOSim()` | Protein similarity calculation based on Gene Ontology (GO) similarity |

# 9. Applications

## 9.1. Regression Modeling in QSRR Study of logD

In 2015, 1130 compounds with LogD7.4 values were collected from different literatures by our group (Wang *et al.* 2015). And then, partial least squares (PLS) and support vector machine (SVM) regressions were employed to build prediction models with 30 molecular descriptors selected by genetic algorithm. Here we choose the molecules and logD as our benchmark dataset.

Just like the last section, we load the **BioMedR** package, and read the molecules stored in a SMILES file:

```
x.mol = readMolFromSmi(system.file('vignettedata/logD.smi',
package = 'BioMedR'), type = 'mol')
x.tab = read.csv(system.file('vignettedata/logD.csv',
package = 'BioMedR'), sep = ';', header = TRUE)
y = x.tab$logD
```

The `readMolFromSmi()` function is used for reading molecules from SMILES files, for molecules stored in SDF files, use `readMolFromSDF()` instead.

The CSV file `logD.csv` contains the smiles structures and experimental values of 1310 compounds, Here we only extracted the logD values by calling `x.tab$logD`.

After the molecules were properly loaded, we calculate several selected molecular descriptors.

```
# calculate selected molecular descriptors
x = suppressWarnings(cbind(
    extrDrugBCUT(x.mol), extrDrugTPSA(x.mol),
    extrDrugKierHallSmarts(x.mol),
    extrDrugALOGP(x.mol),
    extrDrugLogP(x.mol),
    extrDrugMannholdLogP(x.mol)
    ))
```

After the descriptors were calculated, the result `x` would be an `R` data frame, each row represents one molecule, and each column is one descriptor (predictor).

Next, a partial least squares(PLS) and random forest(RF) model will be

```
# regression on training set
require(pls)
require(randomForest)

# feature selection
# this step may take several minutes, please wait!
result <- rf.fs(x, y, scale = FALSE, step = -1)
with(result, plot(n.var, error.cv, type = "l", lwd = 2))
```

We see that the error of the randomForest regression model was decreasing when the n.var was increasing. In general, we select the features with which the error value decrease at the fastest rate, here we selected 17 features for further study.

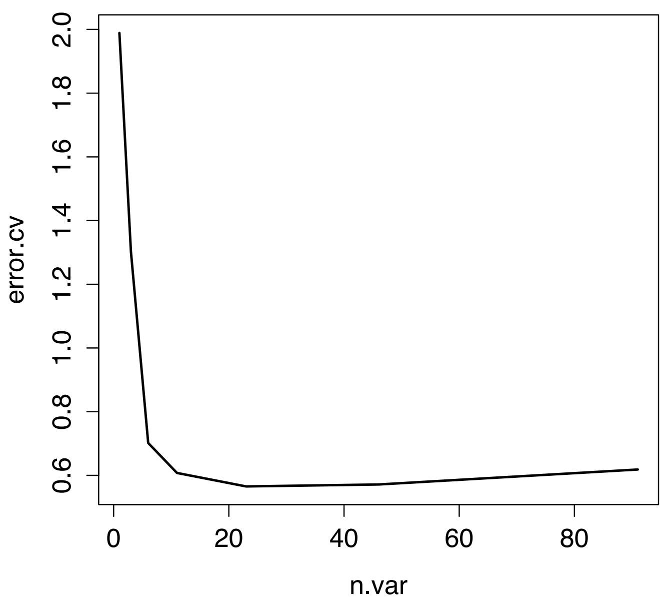

Figure 7: Random Forest Cross-Valdidation for feature selection

```
xtr = x[, order(result$res[[75]], decreasing = T)[1:17]]

# split training and test set
set.seed(1002)
index = sample(1:1096, 872)
trainX = xtr[index, ]
trainy = y[index]
testX = xtr[-index, ]
testy = y[-index]

# cross-validation of RF model
rf = rf.cv(trainX, trainy, mtrysize = 4)

# cross-validation of pls model
pls = pls.cv(trainX, trainy, maxcomp = 5)
```

The cross-validation result is:

```
rf$Q2
```

```
## [1,] 0.8130789
```

```
pls$Q2
```

```
## [1] 0.6363464

# predict test set
rf.out = randomForest(trainy ~ ., data = data.frame(trainy, trainX), mtry = 4)
rf.pred = predict(rf.out, testX)

xcal <- scale(trainX, center = TRUE, scale = TRUE)
xcen <- attributes(xcal)$'scaled:center'
xsca <- attributes(xcal)$'scaled:scale'
xtest <- scale(testX, xcen, xsca)
ycal <- scale(trainy, center = TRUE, scale = FALSE)
ycen <- attributes(ycal)$'scaled:center'

mvrout <- plsr(ycal ~ ., ncomps = 5, data = data.frame(xcal, ycal),
                 scale = FALSE, method = 'simpls')
pls.pred <- predict(mvrout, comps = 1:5, xtest) + ycen

# plot experimental logD vs predicted logD
require(ggplot2)
predict <- c(rf$RFpred, rf.pred, pls$plspred, pls.pred)
ytrain <- rep(c(trainy, testy), 2)
methods <- c(rep("RF", 1096), rep("PLS", 1096))
Ind <- rep(c(rep("training", 872), rep("test", 224)), 2)
Expre <- data.frame(Predicted = predict, ytrain = ytrain,
                         Methods = methods, Ind = Ind)
p <- ggplot(Expre, aes(x = ytrain, y = Predicted, shape = Ind,
                       colour = Ind)) + geom_point() +
  geom_abline(intercept = 0, slope = 1, colour = "#CCCCCC") +
  scale_shape_manual(values = c(17,16)) +
  xlim(range(y)) + ylim(range(y))

p+facet_wrap(~ Methods, nrow = 1)+
  facet_wrap(~ Methods, nrow = 1)+
  xlab("Experimental logD") + ylab("Predicted logD")+
  labs(colour = "Data", shape = "Data")
```

## 9.2. Classification Modeling in QSRR Study of hERG

In the perspective of quantitative pharmacology, the successful discovery of novel drugs depends on the pharmacokinetics properties, like **a**bsorption, **d**istribution, **m**etabolism, and **e**xcretion. In addition, the potential **t**oxicity of chemical compounds is taken into account. QSAR or QSPR methods are usually employed to predict the ADME/T qualities of potential drug candidates.

During cardiac depolarization and repolarization, a voltage-gated potassium channel encoded by the human ether-a-go-go related gene (hERG) plays a major role in the regulation of the exchange of cardiac action potential and resting potential. The hERG blockade may cause long

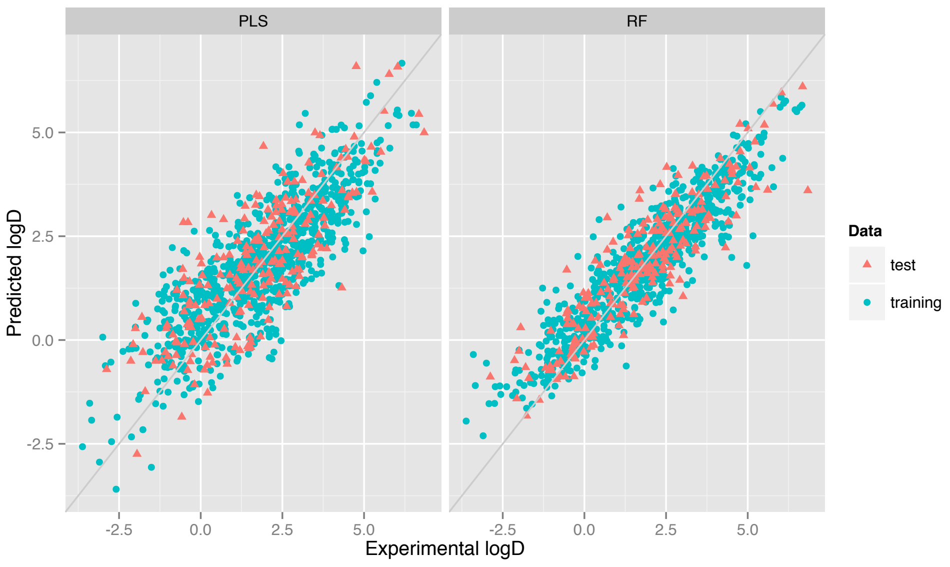

Figure 8: Experimental logD vs. Predicted logD

QT syndrome (LQTS), arrhythmia, and Torsade de Pointes (TdP), which lead to palpitations, fainting, or even sudden death. Therefore, assessment of hERG-related cardiotoxicity has become an important step in the drug design/discovery pipeline. In this study, we collected 655 hERG blocker from Hou's study published in 2016.

First, load the drug molecules stored in a SMILES file into R:

```
require(BioMedR)

cpdbas.smi = system.file('vignettedata/CPDBAS.smi', package = 'BioMedR')
cpdbas.csv = read.csv(system.file('vignettedata/CPDBAS.csv', package =
                      'BioMedR'), header = TRUE, sep = ';')

x.mol = readMolFromSmi(cpdbas.smi, type = 'mol')
x.smi = readMolFromSmi(cpdbas.smi, type = 'text')
y     = as.factor(cpdbas.csv[, 3])
```

The object `x.mol` is used for computing the MACCS and E-state fingerprints, the object `x.smi` is used for computing the FP4 fingerprints. The 0-1 class labels stored in `FDAMDD.csv` indicates whether the drug molecule has high toxicity or not.

Then we calculate three different types of molecular fingerprints (E-state and MACCS) for the drug molecules:

```
# calculate molecular fingerprints
x1 = extrDrugEstateComplete(x.mol)
x2 = extrDrugMACCSComplete(x.mol)
```

As the nature of fingerprint-based structure representation, the calculated 0-1 matrix `x1`, `x2`, and `x3` will be very sparse. Since there are several columns have nearly exactly the same value for all the molecules, we should remove them with `nearZeroVar()` in **caret** before modeling, and split our training set and test set:

```
name1 = c()
for (i in 1:dim(x1)[2]) {
  name1 = c(name1, paste('X', i, sep = ''))
}
colnames(x1) = name1

name2 = c()
for (i in 1:dim(x2)[2]) {
  name2 = c(name2, paste('X', i, sep = ''))
}
colnames(x2) = name2


# Remove near zero variance variables
require(caret)
x1 = x1[, -nearZeroVar(x1)]
x2 = x2[, -nearZeroVar(x2)]

# split training and test set
set.seed(1003)
tr.idx = sample(1:nrow(x1), round(nrow(x1) * 0.8))
te.idx = setdiff(1:nrow(x1), tr.idx)
x1.tr  = x1[tr.idx, ]
x1.te  = x1[te.idx, ]
x2.tr  = x2[tr.idx, ]
x2.te  = x2[te.idx, ]
y.tr   = y[tr.idx]
y.te   = y[te.idx]
```

On the training sets, we will train three classification models separately using PLS and randomForest. The cross-validation setting is 5-fold repeated CV. Then print the cross-validation result.

```
rf.x1 = rf.cv(x1.tr, y.tr, type = 'classification', mtrysize = 5)

rf.x2 = rf.cv(x2.tr, y.tr, type = 'classification', mtrysize = 11)
```

The training result when using E-state fingerprints:

```
# print cross-validation result
# accuracy
rf.x1$ACC

## [1] 0.7368421
```

```
# sensitivity
rf.x1$SE

## [1] 0.7472924

# specificity
rf.x1$SP

## [1] 0.7254902
```

We could see that after removing the near zero variance predictors, there are only 21 predictors left for the original length 79 E-state fingerprints.

The training result when using MACCS keys:

```
# accuracy
rf.x2$ACC

## [1] 0.7706767

# sensitivity
rf.x2$SE

## [1] 0.8050542

# specificity
rf.x2$SP

## [1] 0.7333333
```

There are 111 predictors left for the original length 166 MACCS keys after removing the near zero variance predictors. The model performance by ACC values is slightly better than using the E-state fingerprints.

We predict on the test sets with the established models, and plot the ROC curves in one figure, as is shown in figure 9.

```
# predict on test set
rf.model1 = randomForest(y.tr ~ ., data = data.frame(x1.tr, y.tr), mtry = 5,
                         ntree = 500)
rf.pred1 = predict(rf.model1, x1.te, type = "prob")[, 2]

rf.model2 = randomForest(y.tr ~ ., data = data.frame(x2.tr, y.tr), mtry = 11,
                         ntree = 500)
rf.pred2 = predict(rf.model2, x2.te, type = "prob")[, 2]

# generate colors
require(RColorBrewer)
pal = brewer.pal(3, 'Set1')

# ROC curves of different fingerprints
```

```
require(pROC)

# cross-validation
opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
plot(smooth(roc(y.tr, rf.x1$prob[, 2])), col = pal[1], grid = TRUE,
                                   main = 'Cross Validation')
plot(smooth(roc(y.tr, rf.x2$prob[, 2])), col = pal[2], grid = TRUE, add = TRUE)

# prediction
plot(smooth(roc(y.te, rf.pred1)), col = pal[1], grid = TRUE,
                                   main = 'Prediction')
plot(smooth(roc(y.te, rf.pred2)), col = pal[2], grid = TRUE, add = TRUE)
par(opar)
```

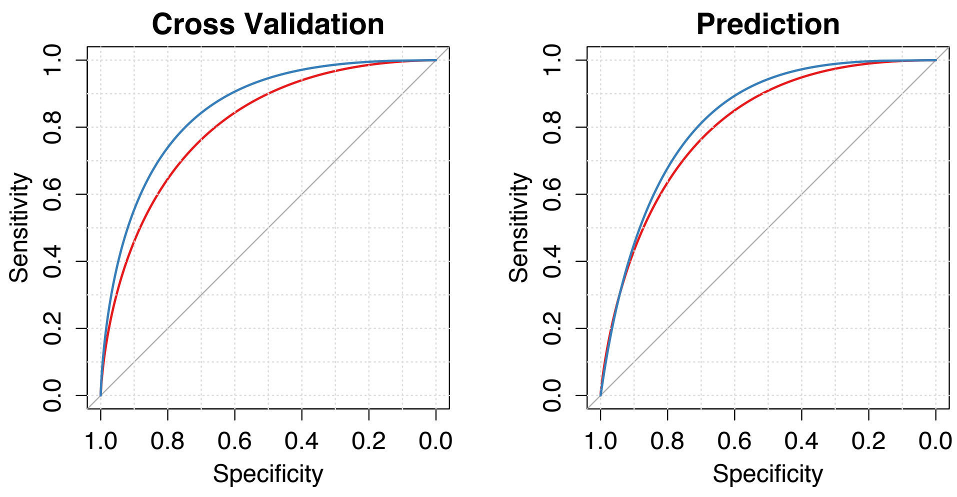

Figure 9: Smoothed ROC curves for different fingerprint types

## 9.3. Chemical Similarity Searching

Structure-based chemical similarity searching ranks molecules in a database by their similarity degree to one query molecule structure. The numerical similarity value is usually computed based on the molecular fingerprints with selected metrics or by maximum common structure search. It is one of the core techniques for ligand-based virtual screening in drug discovery.

```
mol   = system.file('compseq/example.sdf', package = 'BioMedR')
moldb = system.file('compseq/bcl.sdf', package = 'BioMedR')
```

We could do parallelized drug molecular similarity search with the `searchDrug()` function in **BioMedR**. Here we choose the search criterion to be MACCS keys with cosine similarity, FP2 fingerprints with tanimoto similarity, and maximum common substructure search with tanimoto similarity.

```
rank1 = searchDrug(mol, moldb, cores = 4, method = 'fp',
                     fptype = 'maccs', fpsim = 'tanimoto')
rank2 = searchDrug(mol, moldb, cores = 4, method = 'fp',
                     fptype = 'standard', fpsim = 'cosine')
```

The returned search result is stored as a numerical vector, each element's name is the molecule number in the database, and the value is the similarity value between the query molecule and this molecule. We shall print the top search results here:

```
head(rank1)
##        50        15        13        19        16        17
## 0.9532710 0.9439252 0.9351852 0.9351852 0.9266055 0.9266055

head(rank2)
##        50        20        14         1         6         8
## 0.9706932 0.9233684 0.9130435 0.9071174 0.9025330 0.9014839
```

```
require(ChemmineR)
sdf1 = read.SDFset(system.file('compseq/example.sdf', package = 'BioMedR'))
sdf2 = read.SDFset(system.file('compseq/bcl.sdf', package = 'BioMedR'))

opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
plotStructure(sdf2[[50]])
plotStructure(sdf1[[1]])
par(opar)
```

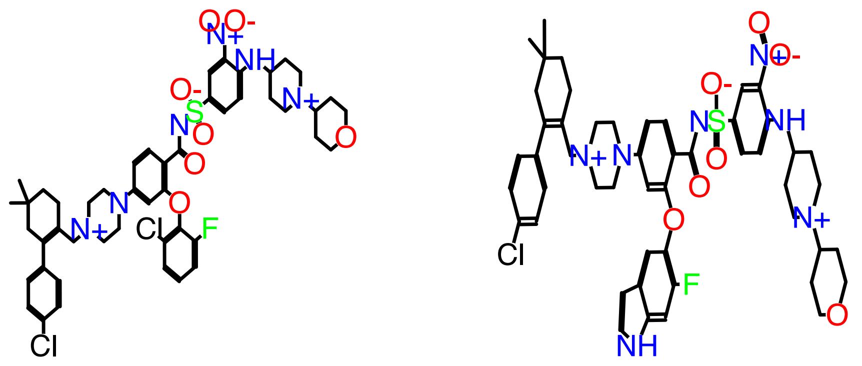

Figure 10: Maximum common structure of the query molecule and No.50 molecule in the drug database (SDF file)

## 9.4. Clustering of Molecules Based on Structural Similarities

Apart from supervised methods (classification and regression), unsupervised approaches, like clustering, is also widely applied in the quantitative research of drugs.

In reality, there are usually too many chemical compounds available for identifying drug-like molecules. Thus it would be attractive using clustering methods to aid the selection of a representative subset of all available compounds. For a clustering approach that groups compounds together by their structural similarity, applying the principle *similar compounds have similar properties* means that we only need to test the representative compounds from each individual cluster, rather than do the time-consuming complete set of experiments, and this should be sufficient to understand the structure-activity relationships of the whole compound set.

The **BioMedR** package provides easy-to-use functions for computing the similarity between small molecules derived by molecular fingerprints and maximum common substructure search.

As a example, the SDF file `bcl.sdf` below is a database composed by searching UniProt *P10415* in ChEMBL. We load this SDF file into R using `readMolFromSDF()`:

```
require(BioMedR)
mols = readMolFromSDF(system.file('compseq/bcl.sdf', package = 'BioMedR'))
```

Then compute the E-state fingerprints for all the molecules using `extrDrugEstate()`, and calculate their pairwise similarity matrix with `calcDrugFPSim()`:

```
simmat = diag(length(mols))

for (i in 1:length(mols)) {
 for (j in i:length(mols)) {
   fp1 = extrDrugEstate(mols[[i]])
   fp2 = extrDrugEstate(mols[[j]])
   tmp = calcDrugFPSim(fp1, fp2, fptype = 'compact', metric = 'tanimoto')
   simmat[i, j] = tmp
   simmat[j, i] = tmp
 }
}
```

For the computed similarity matrix `simmat`, we will try to do hierarchical clustering with it, then visualize the clustering result:

```
mol.hc = hclust(as.dist(1 - simmat), method = 'ward.D')

require(ape)  # for tree-like visualization, if not please install.
clus5 = cutree(mol.hc, 5)  # cut dendrogram into 5 clusters

# generate colors
require(RColorBrewer)
pal5 = brewer.pal(5, 'Set1')
plot(as.phylo(mol.hc), type = 'fan', tip.color = pal5[clus5],
     label.offset = 0.1, cex = 0.7)
```

The clustering result for these molecules is shown in figure 11.

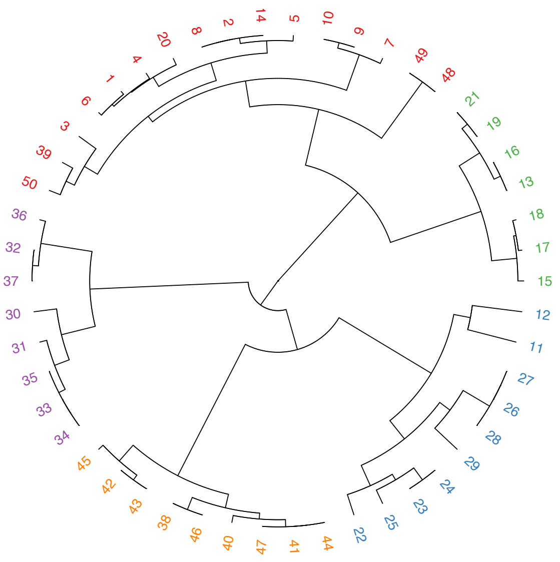

Figure 11: Tree visualization of molecular clustering result

**BioMedR** allows the user to take advantage of the wide spectrum of clustering utilities available in R. An example on how to perform hierarchical clustering with the hclust function is given below.

```
# Clustering with other algorithms
require(gplots) # if not please install.
data(sdfbcl)
apbcl = convSDFtoAP(sdfbcl)
dummy = clusterCMP(db = apbcl, cutoff = 0, save.distances = 'distmat.rda',
                        quiet = TRUE)
load('distmat.rda')
# Hierarchical clustering

hc <- hclust(as.dist(distmat), method = 'single')

heatmap.2(1-distmat, Rowv = as.dendrogram(hc), Colv = as.dendrogram(hc),
                    col = colorpanel(40, 'darkblue', 'yellow', 'white'),
                          density.info = 'none', trace = 'none')
```

## 9.5. Predicting Protein Subcellular Localization

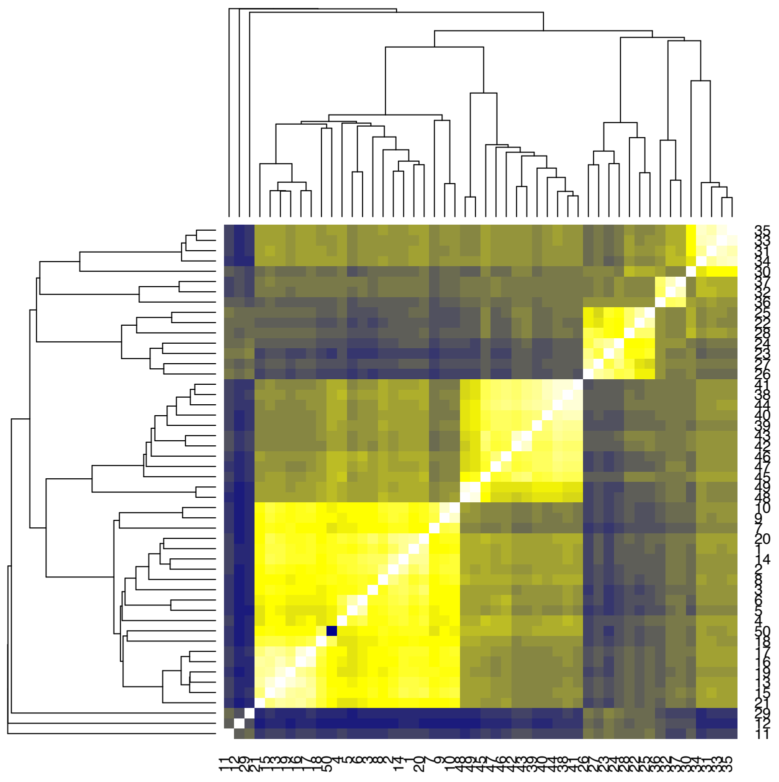

Figure 12: heatmap visualization of molecular clustering result

Protein subcellular localization prediction involves the computational prediction of where a protein resides in a cell. It is an important component of bioinformatics-based prediction of protein function and genome annotation, and could also aid us to identify novel drug targets.

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

```
# load FASTA files
Cytoplasm = readFASTA(system.file('vignettedata/Cytoplasm.fasta',
                                  package = 'BioMedR'))
Nuclear = readFASTA(system.file('vignettedata/Nuclear.fasta',
                                package = 'BioMedR'))
```

To read protein sequences stored in PDB format files, use `readPDB()` instead. The loaded sequences will be stored as two lists in R, and each component in the list is a character string representing one protein sequence. In this case, there are 300 *Cytoplasm* protein sequences and 300 *Nuclear* protein sequences:

To assure that the protein sequences only have the twenty standard amino acid types which is required for the descriptor computation, we use the `checkProt()` function in **BioMedR** to do the amino acid type sanity checking and remove the *non-standard* sequences:

```
Cytoplasm = Cytoplasm[(sapply(Cytoplasm, checkProt))]
Nuclear = Nuclear[(sapply(Nuclear, checkProt))]
```

Two protein sequences were removed from each class. For the remaining sequences, we calculate the amphiphilic pseudo amino acid composition (APAAC) descriptor (Chou 2005) and make class labels for classification modeling.

```
# calculate APAAC descriptors
x1 = t(sapply(Cytoplasm, function(x) {c(extrProtCTDC(x),
                                        extrProtCTDD(x),
                                        extrProtCTDT(x))}))
x2 = t(sapply(Nuclear, function(x) {c(extrProtCTDC(x),
                                      extrProtCTDD(x),
                                      extrProtCTDT(x))}))

x  = rbind(x1, x2)
na.idx = which(is.na(x), arr.ind = TRUE)
x = x[, -unique(na.idx[, 2])]

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

In **BioMedR**, the functions of commonly used descriptors for protein sequences and proteochemometric (PCM) modeling descriptors are named after `extrProt...()` and `extrPCM...()`.

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

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

We will train a random forest classification model on the training set with 5-fold cross-validation.

```
rf.prot = rf.cv(x.tr, y.tr, type = 'classification', cv.fold = 5)
```

The training result is:

```
# print cross-validation result
# accuracy
rf.prot$ACC

## [1] 0.8066667

# sensitivity
rf.prot$SE

## [1] 0.82

# specificity
rf.prot$SP

## [1] 0.7933333
```

With the model trained on the training set, we predict on the test set and plot the ROC curve with the **pROC** package, as is shown in figure 13.

```
# predict on test set
require(randomForest)
rf.p = randomForest(y.tr ~ ., data = data.frame(y.tr, x.tr))
rf.pred = predict(rf.p, newdata = x.te, type = 'prob')[, 1]

# plot ROC curve
require(RColorBrewer)
pal = brewer.pal(3, 'Set1')
require(pROC)
opar <- par(no.readonly = TRUE)
```

```
par(mfrow=c(1,2))
plot.roc(y.tr, rf.prot$prob[, 1], col = pal[2], grid = TRUE,
          print.auc = TRUE, main = 'Cross Validation')
plot.roc(y.te, rf.pred, col = pal[1], grid = TRUE, print.auc = TRUE,
          main = 'prediction')
par(opar)
```

The area under the ROC curve (AUC) is:

```
# cross validation
## Call:
## plot.roc.default(x = y.tr, predictor = rf.prot$prob[, 1], col = pal[2], grid
##                      = TRUE, print.auc = TRUE, main = "prediction")
##
## Data: rf.prot$prob[, 1] in 150 controls (y.tr 0) > 150 cases (y.tr 1).
## Area under the curve: 0.881

# predict
## Call:
## plot.roc.default(x = y.te, predictor = rf.pred, col = pal[1], grid = TRUE,
##                    print.auc = TRUE, main = "Cross Validation")
##
## Data: rf.pred in 50 controls (y.te 0) > 50 cases (y.te 1).
## Area under the curve: 0.823
```

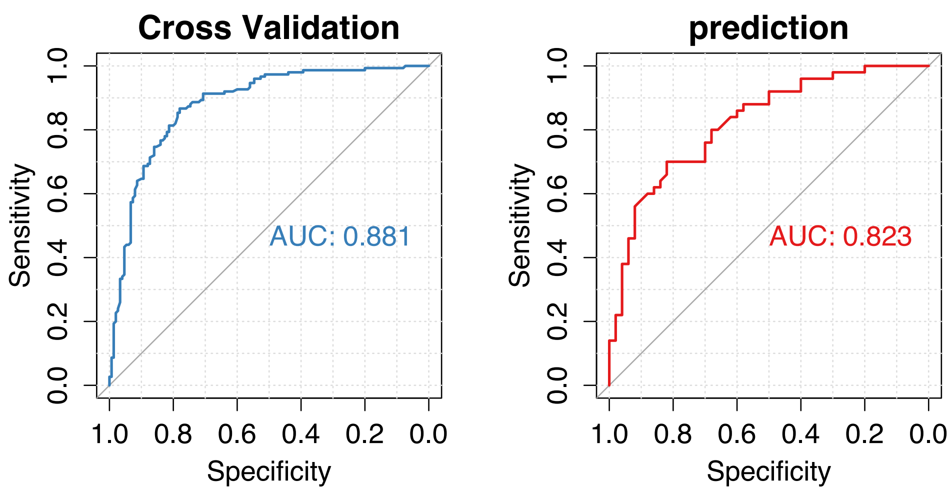

Figure 13: ROC curve for protein subcellular localization data

### 9.6. Predicting nucleosome positioning in genomes

In the living cell nucleus, genomic DNA is packaged into chromatin. DNA sequences that regulate transcription and other chromosomal processes are associated with local disruptions, or openings, in chromatin structure caused by the cooperative action of regulatory proteins (Noble *et al.* 2005),

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 `randomForest`, and its performance was evaluated by 5-fold cross-validation.

```
# load FASTA files
neg = readFASTA(system.file('vignettedata/H_sapiens_neg.fasta',
                            package = 'BioMedR'))
pos = readFASTA(system.file('vignettedata/H_sapiens_pos.fasta',
                            package = 'BioMedR'))
```

each component in the list is a character string representing one protein sequence. In this case, there are 300 *neg* DNA sequences and 300 *pos* DNA sequences:

To assure that the protein sequences only have the twenty standard amino acid types which is required for the descriptor computation, we use the `checkDNA()` function in **BioMedR** to do the amino acid type sanity checking and remove the *non-standard* sequences:

```
neg = neg[(sapply(neg, checkDNA))]
pos = pos[(sapply(pos, checkDNA))]
```

Two protein sequences were removed from each class. For the remaining sequences, we calculate the amphiphilic pseudo amino acid composition (APAAC) descriptor (Chou 2005) and make class labels for classification modeling.

```
# calculate APAAC descriptors
x1 = t(sapply(neg, function(x) {c(extrDNAkmer(x),
                                  extrDNADAC(x),
                                  extrDNADCC(x))}))
x2 = t(sapply(pos, function(x) {c(extrDNAkmer(x),
                                  extrDNADAC(x),
                                  extrDNADCC(x))}))

x  = rbind(x1, x2)

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

In **BioMedR**, the functions of commonly used descriptors for protein sequences and proteochemometric (PCM) modeling descriptors are named after `extrProt...()` and `extrPCM...()`.

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

```
# split training and test set
set.seed(1001)
```

```
tr.idx = c(sample(1:nrow(x1), round(nrow(x1) * 0.75)),
           sample(nrow(x1) + 1:nrow(x2), round(nrow(x2) * 0.75)))
te.idx = setdiff(1:nrow(x), tr.idx)
x.tr   = x[tr.idx, ]
x.te   = x[te.idx, ]
y.tr   = labels[tr.idx]
y.te   = labels[te.idx]
rownames(x.tr) = NULL
rownames(x.te) = NULL
```

We will train a random forest classification model on the training set with 5-fold cross-validation.

```
rf.dna = rf.cv(x.tr, y.tr, type = 'classification', cv.fold = 5)
```

With the model trained on the training set, we predict on the test set and plot the ROC curve with the **pROC** package, as is shown in figure 14.

```
# predict on test set
rf.d = randomForest(y.tr ~ ., data = data.frame(y.tr, x.tr))
rf.pred = predict(rf.d, newdata = x.te, type = 'prob')[, 1]

# plot ROC curve
require(RColorBrewer)
pal = brewer.pal(3, 'Set1')
require(pROC)
opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
plot.roc(y.tr, rf.dna$prob[, 1], col = pal[2], grid = TRUE,
          print.auc = TRUE, main = 'Cross Validation')
plot.roc(y.te, rf.pred, col = pal[1], grid = TRUE, print.auc = TRUE,
          main = 'prediction')
par(opar)
```

The area under the ROC curve (AUC) is:

```
# cross validation
## Call:
## plot.roc.default(x = y.tr, predictor = rf.dna$prob[, 1], col = pal[2], grid
##                        = TRUE, print.auc = TRUE, main = "Cross Validation")
##
## Data: rf.dna$prob[, 1] in 225 controls (y.tr 0) > 225 cases (y.tr 1).
## Area under the curve: 0.81

# predict
## Call:
```

```
## plot.roc.default(x = y.te, predictor = rf.pred, col = pal[1], grid = TRUE,
##                  print.auc = TRUE, main = "prediction")
##
## Data: rf.pred in 75 controls (y.te 0) > 75 cases (y.te 1).
## Area under the curve: 0.8047
```

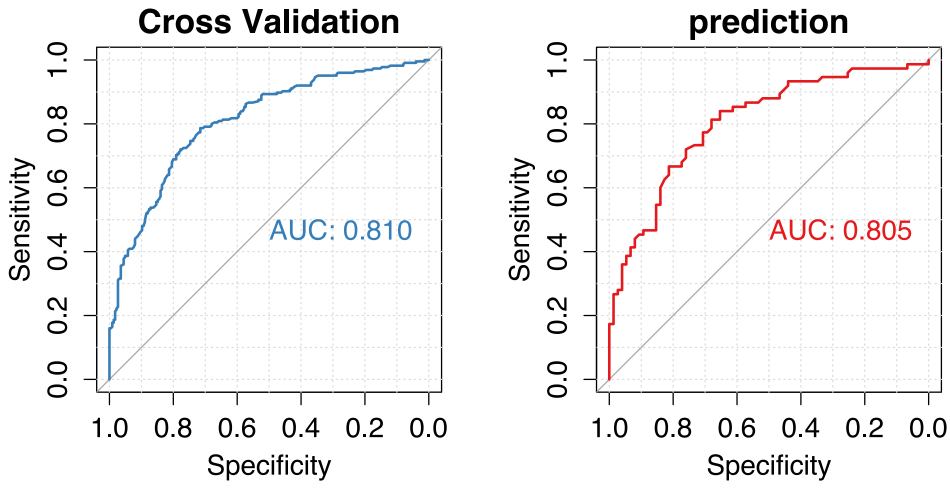

Figure 14: ROC curve for nucleosome positioning data

## 9.7. Predicting Drug-Target Interaction by Integrating Chemical and Genomic Spaces

In this example, we use the G protein-coupled receptor (GPCR) dataset provided by Yamanishi *et al.* (2008) as our benchmark dataset.

A drug-target interaction network can be naturally modeled as a bipartite graph, where the nodes are target proteins or drug molecules and edges (only drugs and proteins could be connected by edges) represent drug-target interactions. Initially, the graph only contains edges describing the *real* drug-target interactions determined by experiments or other ways. In this example, all real drug-target interaction pairs (i.e., 635 drug-target interactions) are used as the positive samples. For negative samples we select random, non-interacting pairs from these drugs and proteins.

Ten generated negative sets were used in Cao *et al.* (2012a), here we only use one of them for a demonstration. The drug ID and target ID is stored in `GPCR.csv`. The first column is KEGG protein ID, and the second column is KEGG drug ID. The first 635 rows is the positive set, and the last 635 rows is the negative set.

we will download the target protein sequences (in FASTA format) and drug molecule (in SMILES format) from the KEGG database, in parallel:

```
require(BioMedR)

gpcr = read.table(system.file('vignettedata/GPCR.csv', package = 'BioMedR'),
                    header = FALSE, as.is = TRUE)

protid = unique(gpcr[, 1])
drugid = unique(gpcr[, 2])
protseq = BMgetProtSeqKEGG(protid, parallel = 5)

drugseq = c()
for (id in 1:length(drugid)) {
  drugseq[id] = BMgetDrugSmiKEGG(drugid[id])
}

# if the network environment is not good, use the following code instead.
# protseq = readFASTA(system.file('vignettedata/GPCR_seq.fasta', package = 'BioMedR'))
# drugseq = as.vector(read.table(system.file('vignettedata/GPCR_smi.txt',
# package = 'BioMedR'), col.names = 'SMILES'))

x0.prot = cbind(t(sapply(unlist(protseq), extrProtMoreauBroto)),
                t(sapply(unlist(protseq), extrProtCTDC)))

x0.drug = cbind(extrDrugGraphComplete(readMolFromSmi(textConnection(drugseq))),
                extrDrugPubChemComplete(readMolFromSmi(textConnection(drugseq))))
```

If the connection is slow or accidentally interrupts, just try more times until success.

Since the descriptors is only for the *uniqued* drug and target list, we need to generate the full descriptor matrix for the training data:

```
x.prot = matrix(NA, nrow = nrow(gpcr), ncol = ncol(x0.prot))
x.drug = matrix(NA, nrow = nrow(gpcr), ncol = ncol(x0.drug))
for (i in 1:nrow(gpcr)) x.prot[i, ] = x0.prot[which(gpcr[, 1][i] == protid), ]
for (i in 1:nrow(gpcr)) x.drug[i, ] = x0.drug[which(gpcr[, 2][i] == drugid), ]

y = as.factor(c(rep('1', nrow(gpcr)/2), rep('0', nrow(gpcr)/2)))
```

Generate drug-target interaction descriptors using `getCPI()`.

```
x = getCPI(x.prot, x.drug, type = 'combine')
colnames(x) = paste('CCI', 1:dim(x)[2], sep = '_')
```

Train a random forest classification model with 5-fold repeated CV:

```
require(caret)
x = x[, -nearZeroVar(x)]

# training set split
set.seed(20180808)
split_index = createDataPartition(y, p = 0.75, list = FALSE)
```

```
train_x = x[split_index, ]
train_y = y[split_index]
test_x = x[-split_index, ]
test_y = y[-split_index]

# cross-validation
require(randomForest)
cv_result = rf.cv(train_x, train_y, cv.fold = 5, type = 'classification',
                  trees = 500, mtry = 30)

# train a random forest classifier
rf.fit = randomForest(x = train_x, y = train_y, ntree = 500,
                      mtry = 30, importance = TRUE)
```

Predict on the training set (for demonstration purpose only) and plot ROC curve.

```
# Predict on the test set
pre_res = predict(rf.fit, newdata = test_x, type = 'prob')[, 2]

require(pROC)
# plot the CV result and test result
require(RColorBrewer)
pal = brewer.pal(3, 'Set1')
opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
plot.roc(train_y, cv_result$prob[,1], col = pal[2], grid = TRUE,
         print.auc = TRUE, main = 'Cross Validation')
plot.roc(test_y, pre_res, col = pal[1], grid = TRUE, print.auc = TRUE,
         main = 'prediction')
par(opar)
```

The ROC curve is shown in figure 15.

## Acknowledgments

The authors thank all members of the Computational Biology and Drug Design (CBDD) Group (http://www.scbdd.com/) of Central South University for their support.

This work is financially supported by the National Natural Science Foundation of China (Grants No. 81402853), the Central South University Innovation Foundation for Postgraduate (2016zzts498), the Project of Innovation-driven Plan in Central South University, and the Postdoctoral Science Foundation of Central South University, the Chinese Postdoctoral Science Foundation (2014T70794, 2014M562142). The studies meet with the approval of the university's review board.

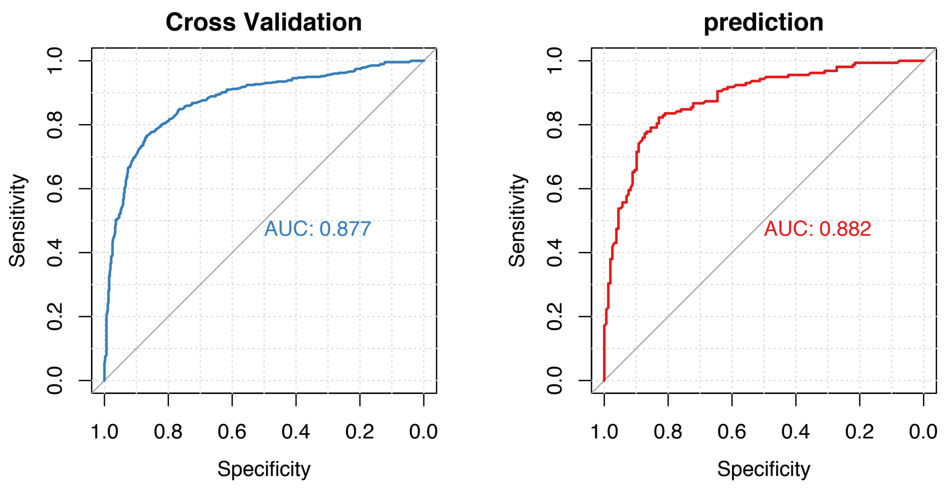

Figure 15: ROC curve for predicting on the training set of the GPCR drug-target interaction dataset using random forest

## References

Atchley WR, Zhao J, Fernandes AD, Drüke T (2005). "Solving the protein sequence metric problem." *Proceedings of the National Academy of Sciences of the United States of America*, **102**(18), 6395–6400.

Bhasin M, Raghava GPS (2004). "Classification of Nuclear Receptors Based on Amino Acid Composition and Dipeptide Composition." *Journal of Biological Chemistry*, **279**(22), 23262–6.

Cao DS, Liang YZ, Deng Z, Hu QN, He M, Xu QS, Zhou GH, Zhang LX, Deng Zx, Liu S (2013a). "Genome-Scale Screening of Drug-Target Associations Relevant to Ki Using a Chemogenomics Approach." *PloS one*, **8**(4), e57680.

Cao DS, Liang YZ, Yan J, Tan GS, Xu QS, Liu S (2013b). "PyDPI: Freely Available Python Package for Chemoinformatics, Bioinformatics, and Chemogenomics Studies." *Journal of chemical information and modeling*.

Cao DS, Liu S, Xu QS, Lu HM, Huang JH, Hu QN, Liang YZ (2012a). "Large-scale prediction of drug-target interactions using protein sequences and drug topological structures." *Analytica chimica acta*, **752**, 1–10.

Cao DS, Xu QS, Hu QN, Liang YZ (2013c). "ChemoPy: freely available python package for computational biology and chemoinformatics." *Bioinformatics*, **29**(8), 1092–1094.

Cao DS, Xu QS, Liang YZ (2013d). "propy: a tool to generate various modes of Chou's PseAAC." *Bioinformatics*.

Cao DS, Zhao JC, Yang YN, Zhao CX, Yan J, Liu S, Hu QN, Xu QS, Liang YZ (2012b). "In silico toxicity prediction by support vector machine and SMILES representation-based string kernel." *SAR and QSAR in Environmental Research*, **23**(1-2), 141–153.

Cao Y, Charisi A, Cheng LC, Jiang T, Girke T (2008). "ChemmineR: a compound mining framework for R." *Bioinformatics*, **24**(15), 1733–1734.

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

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

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

Chou KC (2000). "Prediction of Protein Subcellar Locations by Incorporating Quasi-Sequence-Order Effect." *Biochemical and Biophysical Research Communications*, **278**, 477–483.

Chou KC (2001). "Prediction of Protein Cellular Attributes Using Pseudo-Amino Acid Composition." *PROTEINS: Structure, Function, and Genetics*, **43**, 246–255.

Chou KC (2005). "Using Amphiphilic Pseudo Amino Acid Composition to Predict Enzyme Subfamily Classes." *Bioinformatics*, **21**, 10–19.

Chou KC, Cai YD (2004). "Prediction of Protein Sub-cellular Locations by GO-FunD-PseAA Predictor." *Biochemical and Biophysical Research Communications*, **320**, 1236–1239.

Damborsky J (1998). "Quantitative Structure-function and Structure-stability Relationships of Purposely Modified Proteins." *Protein Engineering*, **11**, 21–30.

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

Dubchak I, Muchink I, Holbrook SR, Kim SH (1995). "Prediction of Protein Folding Class Using Global Description of Amino Acid Sequence." *Proceedings of the National Academy of Sciences*, **92**, 8700–8704.

Dubchak I, Muchink I, Mayor C, Dralyuk I, Kim SH (1999). "Recognition of a Protein Fold in the Context of the SCOP Classification." *Proteins: Structure, Function and Genetics*, **35**, 401–407.

Georgiev AG (2009). "Interpretable numerical descriptors of amino acid space." *Journal of Computational Biology*, **16**(5), 703–723.

Ghose AK, Crippen GM (1986). "Atomic physicochemical parameters for three-dimensional structure-directed quantitative structure-activity relationships I. Partition coefficients as a measure of hydrophobicity." *Journal of Computational Chemistry*, **7**(4), 565–577.

Ghose AK, Crippen GM (1987). "Atomic physicochemical parameters for three-dimensional-structure-directed quantitative structure-activity relationships. 2. Modeling dispersive and hydrophobic interactions." *Journal of chemical information and computer sciences*, **27**(1), 21–35.

Grantham R (1974). "Amino Acid Difference Formula to Help Explain Protein Evolution." *Science*, **185**, 862–864.

Guha R, Jurs P (2005). "Integrating R with the CDK for QSAR modeling." In *230th American Chemical Society Meeting & Conference, Washington DC*, volume 32.

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

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

Hellberg S, Sjoestroem M, Skagerberg B, Wold S (1987). "Peptide quantitative structure-activity relationships, a multivariate approach." *Journal of medicinal chemistry*, **30**(7), 1126–1135.

Hopp-Woods (1981). "Prediction of Protein Antigenic Determinants from Amino Acid Sequences." *Proceedings of the National Academy of Sciences*, **78**, 3824–3828.

Horan K, Girke T (2013). *ChemmineOB: R interface to a subset of OpenBabel functionalities*. R package version 1.0.1, URL `http://manuals.bioinformatics.ucr.edu/home/chemminer`.

Jarvis RA, Patrick EA (1973). "Clustering using a similarity measure based on shared near neighbors." *IEEE Transactions on Computers*, **100**(11), 1025–1034.

Kawashima S, Kanehisa M (2000). "AAindex: Amino Acid Index Database." *Nucleic Acids Research*, **28**, 374.

Kawashima S, Ogata H, Kanehisa M (1999). "AAindex: Amino Acid Index Database." *Nucleic Acids Research*, **27**, 368–369.

Kawashima S, Pokarowski P, Pokarowska M, Kolinski A, Katayama T, Kanehisa M (2008). "AAindex: Amino Acid Index Database (Progress Report)." *Nucleic Acids Research*, **36**, D202–D205.

Kohonen T (2001). "Self-organizing maps, vol. 30 of Springer Series in Information Sciences." *ed: Springer Berlin.*

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

Li Z, Lin H, Han Y, Jiang L, Chen X, Chen Y (2006). "PROFEAT: A Web Server for Computing Structural and Physicochemical Features of Proteins and Peptides from Amino Acid Sequence." *Nucleic Acids Research*, **34**, 32–37.

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

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

Mei H, Liao ZH, Zhou Y, Li SZ (2005). "A new set of amino acid descriptors and its application in peptide QSARs." *Peptide Science*, **80**(6), 775–786.

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

Pages H, Aboyoun P, Gentleman R, DebRoy S (2013). *Biostrings: String objects representing biological sequences, and matching algorithms.* R package version 2.30.1.

Pearlman RS, Smith KM (1999). "Metric validation and the receptor-relevant subspace concept." *Journal of Chemical Information and Computer Sciences*, **39**(1), 28–35.

Rangwala H, Karypis G (2005). "Profile-based direct kernels for remote homology detection and fold recognition." *Bioinformatics*, **21**(23), 4239–4247.

Rao H, Zhu F, Yang G, Li Z, Chen Y (2011). "Update of PROFEAT: A Web Server for Computing Structural and Physicochemical Features of Proteins and Peptides from Amino Acid Sequence." *Nucleic Acids Research*, **39**, 385–390.

Sandberg M, Eriksson L, Jonsson J, Sjöström M, Wold S (1998). "New chemical descriptors relevant for the design of biologically active peptides. A multivariate characterization of 87 amino acids." *Journal of medicinal chemistry*, **41**(14), 2481–2491.

Schneider G, Wrede P (1994). "The Rational Design of Amino Acid Sequences by Artificial Neural Networks and Simulated Molecular Evolution: Do Novo Design of an Idealized Leader Cleavage Site." *Biophysical Journal*, **66**, 335–344.

Shen J, Zhang J, Luo X, Zhu W, Yu K, Chen K, Li Y, Jiang H (2007). "Predicting Protein-protein Interactions Based Only on Sequences Information." *Proceedings of the National Academy of Sciences*, **104**, 4337–4341.

Sjöström M, Rännar S, Wieslander Å (1995). "Polypeptide sequence property relationships in Escherichia coli based on auto cross covariances." *Chemometrics and intelligent laboratory systems*, **29**(2), 295–305.

Steinbeck C, Han Y, Kuhn S, Horlacher O, Luttmann E, Willighagen E (2003). "The Chemistry Development Kit (CDK): An open-source Java library for chemo-and bioinformatics." *Journal of chemical information and computer sciences*, **43**(2), 493–500.

Tian F, Zhou P, Li Z (2007). "T-scale as a novel vector of topological descriptors for amino acids and its application in QSARs of peptides." *Journal of molecular structure*, **830**(1), 106–115.

van Westen GJ, Swier RF, Cortes-Ciriano I, Wegner JK, Overington JP, IJzerman AP, van Vlijmen HW, Bender A (2013a). "Benchmarking of protein descriptor sets in proteochemometric modeling (part 2): modeling performance of 13 amino acid descriptor sets." *Journal of cheminformatics*, **5**(1), 42.

van Westen GJ, Swier RF, Wegner JK, IJzerman AP, van Vlijmen HW, Bender A (2013b). "Benchmarking of protein descriptor sets in proteochemometric modeling (part 1): comparative study of 13 amino acid descriptor sets." *Journal of cheminformatics*, **5**(1), 41.

van Westen GJ, van den Hoven OO, van der Pijl R, Mulder-Krieger T, de Vries H, Wegner JK, IJzerman AP, van Vlijmen HW, Bender A (2012). "Identifying novel adenosine receptor ligands by simultaneous proteochemometric modeling of rat and human bioactivity data." *Journal of Medicinal Chemistry*, **55**(16), 7010–7020.

van Westen GJ, Wegner JK, Geluykens P, Kwanten L, Vereycken I, Peeters A, IJzerman AP, van Vlijmen HW, Bender A (2011). "Which compound to select in lead optimization? Prospectively validated proteochemometric models guide preclinical development." *PloS one*, **6**(11), e27518.

Venkatarajan MS, Braun W (2001). "New quantitative descriptors of amino acids based on multidimensional scaling of a large number of physical–chemical properties." *Molecular modeling annual*, **7**(12), 445–453.

Wang JB, Cao DS, Zhu MF, Yun YH, Xiao N, Liang YZ (2015). "In silico evaluation of logD7. 4 and comparison with other prediction methods." *Journal of Chemometrics*, **29**(7), 389–398.

Wang Y, Backman TW, Horan K, Girke T (2013). "fmcsR: mismatch tolerant maximum common substructure searching in R." *Bioinformatics*, **29**(21), 2792–2794.

Wiener H (1947). "Structural determination of paraffin boiling points." *Journal of the American Chemical Society*, **69**(1), 17–20.

Wikberg JE, Lapinsh M, Prusis P (2004). "Proteochemometrics: a tool for modeling the molecular interaction space." *Chemogenomics in drug discovery*, pp. 289–309.

Xiao N, Cao D, Xu Q (2014a). *Rcpi: Toolkit for Compound-Protein Interaction in Drug Discovery.* R package version 1.0.0, URL http://www.bioconductor.org/packages/release/bioc/html/Rcpi.html.

Xiao N, Xu Q, Cao D (2014b). *protr: Protein Sequence Descriptor Calculation and Similarity Computation with R.* R package version 0.2-1, URL http://CRAN.R-project.org/package=protr.

Yamanishi Y, Araki M, Gutteridge A, Honda W, Kanehisa M (2008). "Prediction of drug–target interaction networks from the integration of chemical and genomic spaces." *Bioinformatics*, **24**(13), i232–i240.

Ye X, Wang G, Altschul SF (2011). "An assessment of substitution scores for protein profile–profile comparison." *Bioinformatics*, **27**(24), 3356–3363.

Yu G, Li F, Qin Y, Bo X, Wu Y, Wang S (2010). "GOSemSim: an R package for measuring semantic similarity among GO terms and gene products." *Bioinformatics*, **26**(7), 976–978.

Zaliani A, Gancia E (1999). "MS-WHIM scores for amino acids: a new 3D-description for peptide QSAR and QSPR studies." *Journal of chemical information and computer sciences*, **39**(3), 525–533.

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

Zhang QC, Petrey D, Deng L, Qiang L, Shi Y, Thu CA, Bisikirska B, Lefebvre C, Accili D, Hunter T, *et al.* (2012). "Structure-based prediction of protein-protein interactions on a genome-wide scale." *Nature*, **490**(7421), 556–560.

Zhu M, Jie D, Cao D (2016a). *BioMedR: Toolkit for Compound, Protein and DNA/RNA Interaction.* R package version 1.0.0, URL http://www.bioconductor.org/packages/release/bioc/html/BioMedR.html.

Zhu M, Jie D, Cao D (2016b). *redness: DNA/RNA Sequence Descriptor Calculation and Similarity Computation with R.* R package version 0.1-1, URL http://CRAN.R-project.org/package=rDNAse.

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

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

Dongsheng Cao
School of Pharmaceutical Sciences
Central South University
Changsha, Hunan, P. R. China
E-mail: `oriental-cds@163.com`
URL: `http://cbdd.csu.edu.cn`