NAME

RDKitCalculateMolecularDescriptors.py - Calculate 2D/3D molecular descriptors

SYNOPSIS

RDKitCalculateMolecularDescriptors.py [--autocorr2DExclude <yes or no>] [--fragmentCount <yes or no>] [--descriptorNames <Name1,Name2,...>] [--infileParams <Name,Value,...>] [--mode <2D, 3D, All...>] [--mp <yes or no>] [--mpParams <Name.Value,...>] [--outfileParams <Name,Value,...>] [--overwrite] [--precision <number>] [--smilesOut <yes or no>] [-w <dir>] -i <infile> -o <outfile>

RDKitCalculateMolecularDescriptors.py -l | --list

RDKitCalculateMolecularDescriptors.py -h | --help | -e | --examples

DESCRIPTION

Calculate 2D/3D molecular descriptors for molecules and write them out to a SD or CSV/TSV text file.

The complete list of currently available molecular descriptors may be obtained by using '-l, --list' option. The names of valid 2D, fragment count, and 3D molecular descriptors are shown below:

2D descriptors: Autocorr2D, BalabanJ, BertzCT, Chi0, Chi1, Chi0n - Chi4n, Chi0v - Chi4v, EState_VSA1 - EState_VSA11, ExactMolWt, FpDensityMorgan1, FpDensityMorgan2, FpDensityMorgan3, FractionCSP3, HallKierAlpha, HeavyAtomCount, HeavyAtomMolWt, Ipc, Kappa1 - Kappa3, LabuteASA, MaxAbsEStateIndex, MaxAbsPartialCharge, MaxEStateIndex, MaxPartialCharge, MinAbsEStateIndex, MinAbsPartialCharge, MinEStateIndex, MinPartialCharge, MolLogP, MolMR, MolWt, NHOHCount, NOCount, NumAliphaticCarbocycles, NumAliphaticHeterocycles, NumAliphaticRings, NumAromaticCarbocycles, NumAromaticHeterocycles, NumAromaticRings, NumHAcceptors, NumHDonors, NumHeteroatoms, NumRadicalElectrons, NumRotatableBonds, NumSaturatedCarbocycles, NumSaturatedHeterocycles, NumSaturatedRings, NumValenceElectrons, PEOE_VSA1 - PEOE_VSA14, RingCount, SMR_VSA1 - SMR_VSA10, SlogP_VSA1 - SlogP_VSA12, TPSA, VSA_EState1 - VSA_EState10, qed

FragmentCount 2D descriptors: fr_Al_COO, fr_Al_OH, fr_Al_OH_noTert, fr_ArN, fr_Ar_COO, fr_Ar_N, fr_Ar_NH, fr_Ar_OH, fr_COO, fr_COO2, fr_C_O, fr_C_O_noCOO, fr_C_S, fr_HOCCN, fr_Imine, fr_NH0, fr_NH1, fr_NH2, fr_N_O, fr_Ndealkylation1, fr_Ndealkylation2, fr_Nhpyrrole, fr_SH, fr_aldehyde, fr_alkyl_carbamate, fr_alkyl_halide, fr_allylic_oxid, fr_amide, fr_amidine, fr_aniline, fr_aryl_methyl, fr_azide, fr_azo, fr_barbitur, fr_benzene, fr_benzodiazepine, fr_bicyclic, fr_diazo, fr_dihydropyridine, fr_epoxide, fr_ester, fr_ether, fr_furan, fr_guanido, fr_halogen, fr_hdrzine, fr_hdrzone, fr_imidazole, fr_imide, fr_isocyan, fr_isothiocyan, fr_ketone, fr_ketone_Topliss, fr_lactam, fr_lactone, fr_methoxy, fr_morpholine, fr_nitrile, fr_nitro, fr_nitro_arom, fr_nitro_arom_nonortho, fr_nitroso, fr_oxazole, fr_oxime, fr_para_hydroxylation, fr_phenol, fr_phenol_noOrthoHbond, fr_phos_acid, fr_phos_ester, fr_piperdine, fr_piperzine, fr_priamide, fr_prisulfonamd, fr_pyridine, fr_quatN, fr_sulfide, fr_sulfonamd, fr_sulfone, fr_term_acetylene, fr_tetrazole, fr_thiazole, fr_thiocyan, fr_thiophene, fr_unbrch_alkane, fr_urea

3D descriptors: Asphericity, Autocorr3D, Eccentricity, GETAWAY, InertialShapeFactor, MORSE, NPR1, NPR2, PMI1, PMI2, PMI3, RDF, RadiusOfGyration, SpherocityIndex, WHIM

The supported input file formats are: Mol (.mol), SD (.sdf, .sd), SMILES (.smi, .txt, .csv, .tsv)

The supported output file formats are: SD File (.sdf, .sd), CSV/TSV (.csv, .tsv, .txt)

OPTIONS

-a, --autocorr2DExclude <yes or no> [default: yes]

Exclude Autocorr2D descriptor from the calculation of 2D descriptors.

-f, --fragmentCount <yes or no> [default: yes]

Include 2D fragment count descriptors during the calculation. These descriptors are counted using SMARTS patterns specified in FragmentDescriptors.csv file distributed with RDKit. This option is only used during '2D' or 'All' value of '-m, --mode' option.

-d, --descriptorNames <Name1,Name2,...> [default: none]

A comma delimited list of supported molecular descriptor names to calculate. This option is only used during 'Specify' value of '-m, --mode' option.

-e, --examples

Print examples.

-h, --help

Print this help message.

-i, --infile <infile>

Input file name.

--infileParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs for reading molecules from files. The supported parameter names for different file formats, along with their default values, are shown below:

```
SD, MOL: removeHydrogens,yes,sanitize,yes,strictParsing,yes
SMILES: smilesColumn,1,smilesNameColumn,2,smilesDelimiter,space,
    smilesTitleLine,auto,sanitize,yes
```

Possible values for smilesDelimiter: space, comma or tab.

-l, --list

List molecular descriptors without performing any calculations.

-m, --mode <2D, 3D, All, FragmentCountOnly, or Specify> [default: 2D]

Type of molecular descriptors to calculate. Possible values: 2D, 3D, All or Specify. The name of molecular descriptors must be specified using '-d, --descriptorNames' for 'Specify'. 2D descriptors also include 1D descriptors. The structure of molecules must contain 3D coordinates for the calculation of 3D descriptors.

--mp <yes or no> [default: no]

Use multiprocessing.

By default, input data is retrieved in a lazy manner via mp.Pool.imap() function employing lazy RDKit data iterable. This allows processing of arbitrary large data sets without any additional requirements memory.

All input data may be optionally loaded into memory by mp.Pool.map() before starting worker processes in a process pool by setting the value of 'inputDataMode' to 'InMemory' in '--mpParams' option.

A word to the wise: The default 'chunkSize' value of 1 during 'Lazy' input data mode may adversely impact the performance. The '--mpParams' section provides additional information to tune the value of 'chunkSize'.

--mpParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs for to configure multiprocessing.

The supported parameter names along with their default and possible values are shown below:

```
chunkSize, auto
inputDataMode, Lazy   [ Possible values: InMemory or Lazy ]
numProcesses, auto   [ Default: mp.cpu_count() ]
```

These parameters are used by the following functions to configure and control the behavior of multiprocessing: mp.Pool(), mp.Pool.map(), and mp.Pool.imap().

The chunkSize determines chunks of input data passed to each worker process in a process pool by mp.Pool.map() and mp.Pool.imap() functions. The default value of chunkSize is dependent on the value of 'inputDataMode'.

The mp.Pool.map() function, invoked during 'InMemory' input data mode, automatically converts RDKit data iterable into a list, loads all data into memory, and calculates the default chunkSize using the following method as shown in its code:

```
chunkSize, extra = divmod(len(dataIterable), len(numProcesses) * 4)
if extra: chunkSize += 1
```

For example, the default chunkSize will be 7 for a pool of 4 worker processes and 100 data items.

The mp.Pool.imap() function, invoked during 'Lazy' input data mode, employs 'lazy' RDKit data iterable to retrieve data as needed, without loading all the data into memory. Consequently, the size of input data is not known a priori. It's not possible to estimate an optimal value for the chunkSize. The default chunkSize is set to 1.

The default value for the chunkSize during 'Lazy' data mode may adversely impact the performance due to the overhead associated with exchanging small chunks of data. It is generally a good idea to explicitly set chunkSize to a larger value during 'Lazy' input data mode, based on the size of your input data and number of processes in the process pool.

The mp.Pool.map() function waits for all worker processes to process all the data and return the results. The mp.Pool.imap() function, however, returns the the results obtained from worker processes as soon as the results become available for specified chunks of data.

The order of data in the results returned by both mp.Pool.map() and mp.Pool.imap() functions always corresponds to the input data.

-o, --outfile <outfile>

Output file name.

--outfileParams <Name,Value,...> [default: auto]

A comma delimited list of parameter name and value pairs for writing molecules to files. The supported parameter names for different file formats, along with their default values, are shown below:

```
SD: compute2DCoords,auto,kekulize,no
```

Default value for compute2DCoords: yes for SMILES input file; no for all other file types.

-p, --precision <number> [default: 3]

Floating point precision for writing the calculated descriptor values.

-s, --smilesOut <yes or no> [default: no]

Write out SMILES string to CSV/TSV text output file.

--overwrite

Overwrite existing files.

-w, --workingdir <dir>

Location of working directory which defaults to the current directory.

EXAMPLES

To compute all available 2D descriptors except Autocorr2D descriptor and write out a CSV file, type:

```
% RDKitCalculateMolecularDescriptors.py  -i Sample.smi -o SampleOut.csv
```

To compute all available 2D descriptors except Autocorr2D descriptor in multiprocessing mode on all available CPUs without loading all data into memory, and write out a CSV file, type:

```
% RDKitCalculateMolecularDescriptors.py  --mp yes -i Sample.smi
  -o SampleOut.csv
```

To compute all available 2D descriptors except Autocorr2D descriptor in multiprocessing mode on all available CPUs by loading all data into memory, and write out a CSV file, type:

```
% RDKitCalculateMolecularDescriptors.py  --mp yes --mpParams
  "inputDataMode,InMemory" -i Sample.smi -o SampleOut.csv
```

To compute all available 2D descriptors except Autocorr2D descriptor in multiprocessing mode on specific number of CPUs and chunk size without loading all data into memory, and write out a SDF file, type:

```
% RDKitCalculateMolecularDescriptors.py  --mp yes --mpParams
  "inputDataMode,Lazy,numProcesses,4,chunkSize,8" -i Sample.smi
  -o SampleOut.sdf
```

To compute all available 2D descriptors including Autocorr2D descriptor and excluding fragment count descriptors, and write out a TSV file, type:

```
% RDKitCalculateMolecularDescriptors.py  -m 2D -a no -f no
  -i Sample.smi -o SampleOut.tsv
```

To compute all available 3D descriptors and write out a SD file, type:

```
    % RDKitCalculateMolecularDescriptors.py  -m 3D -i Sample3D.sdf
      -o Sample3DOut.sdf
```

To compute only fragment count 2D descriptors and write out a SD file file, type:

```
    % RDKitCalculateMolecularDescriptors.py  -m FragmentCountOnly
      -i Sample.sdf -o SampleOut.sdf
```

To compute all available 2D and 3D descriptors including fragment count and Autocorr2D and write out a CSV file, type:

```
    % RDKitCalculateMolecularDescriptors.py  -m All -a no -i Sample.sdf
      -o SampleOut.csv
```

To compute a specific set of 2D and 3D descriptors and write out a write out a TSV file, type:

```
    % RDKitCalculateMolecularDescriptors.py  -m specify
      -d 'MolWt,MolLogP,NHOHCount, NOCount,RadiusOfGyration'
      -i Sample3D.sdf -o SampleOut.csv
```

To compute all available 2D descriptors except Autocorr2D descriptor for molecules in a CSV SMILES file, SMILES strings in column 1, name in column 2, and write out a SD file without calculation of 2D coordinates, type:

```
    % RDKitCalculateMolecularDescriptors.py --infileParams
      "smilesDelimiter,comma,smilesTitleLine,yes,smilesColumn,1,
      smilesNameColumn,2" --outfileParams "compute2DCoords,no"
      -i SampleSMILES.csv -o SampleOut.sdf
```

AUTHOR

Manish Sud(msud@san.rr.com)

SEE ALSO

RDKitCalculateRMSD.py, RDKitCompareMoleculeShapes.py, RDKitConvertFileFormat.py, RDKitGenerateConformers.py, RDKitPerformMinimization.py

COPYRIGHT