**Rdkit**

RDKit 是一个常用的生物化学信息python工具包。它提供了大量对化学分子2D或3D的计算操作，可生成用于机器学习的分子描述符，以及提供其他更强大的功能。

**分子的表示：**

1. 分子指纹Fingerpoint，将分子编码为二进制向量不是一个可逆的过程（这是有损的转化）；
2. 简化分子线性输入规范（SMILES）是一种用字符串明确描述分子结构的规范；
3. InChIKey 是对 InChI 运用 SHA-256 算法处理后得到的哈希值，它的出现是为了解决 InChI 长度不定的问题；
4. Graph，直接使用图结构表示分子，这是目前常用于GNN处理的数据格式，它能够有效保留分子的结构信息。

**读写分子：**

RDKit 支持从Smiles、mol、sdf 文件中读入分子获得分子对象。 Smiles、mol 是通常用于保存单个分子；而sdf格式当初是作为分子库形式设计的。 因此读入sdf得到的是分子迭代器，读入Smiles 和mol 文件是分子对象。

**1.读入SMILES/SMARTS：**

**1.1 基本读取**

|  |
| --- |
| >>smi='CC(C)OC(=O)C(C)NP(=O)(OCC1C(C(C(O1)N2C=CC(=O)NC2=O)(C)F)O)OC3=CC=CC=C3'  >>from rdkit import Chem  >>from rdkit.Chem import AllChem  >>mol = Chem.MolFromSmiles(smi) # 读smiles  >>print(mol) # <rdkit.Chem.rdchem.Mol object at 0x0000016D1CC557B0>  >>print(type(mol)) # <class 'rdkit.Chem.rdchem.Mol'> |

**1.2 从.smi文件批量读取：**

SmilesMolSupplier(data, delimiter, smilesColumn, nameColumn, titleLine, sanitize)

data：数据文件

delimiter：分隔符，默认为' '

smilesColumn：SMILES所在列，默认为0

nameColumn：SMILES名称所在列，默认为1

titleLine：是否含有标题行，默认True

sanitize：是否检查正确性，默认True

|  |
| --- |
| >>> suppl = Chem.SmilesMolSupplier('data/batch\_smiles.smi', delimiter='\t')  >>> mols = [Chem.MolToSmiles(mol) for mol in suppl]  >>> print(mols)  ['C1=CC=CC=CC=C1', 'c1ccccc1', 'c1ccoc1'] |

**1.3 文本批量读取：**

从文本批量读取：SmilesMolSupplierFromText()，参数基本同上

|  |
| --- |
| >>> with open('data/batch\_smiles.smi', 'r') as f:  >>> mols\_text = f.read()  >>> suppl = Chem.SmilesMolSupplierFromText(mols\_text, delimiter='\t')  >>> mols = [Chem.MolToSmiles(mol) for mol in suppl]  >>> print(mols)  ['C1=CC=CC=CC=C1', 'c1ccccc1', 'c1ccoc1'] |

**1.4 DataFrame批量读取：**

读取DataFrame中的SMILES：AddMoleculeColumnToFrame(frame, smilesCol, molCol, includeFingerprints)

frame：DataFrame对象

smilesCol：SMILES所在列

molCol：新列名，将存放产生的rdkit mol对象

includeFingerprints：是否生成指纹

顺便计算下分子量：Descriptors.MolWt()

|  |
| --- |
| >>> import pandas as pd  >>> from rdkit.Chem import Descriptors  >>> from rdkit.Chem import PandasTools  >>> df = pd.read\_csv('data/smiles\_df.csv')  >>> PandasTools.AddMoleculeColumnToFrame(df,'SMILES','mol',includeFingerprints=True)  >>> df['MW'] = df['mol'].apply(Descriptors.MolWt)  >>> df.head(2) |

**2.读入mol文件：**

|  |
| --- |
| >>mol = Chem.MolFromMolFile('rd.mol')  >>> print(Chem.MolToSmiles(mol)) # c1cocc1  >>print(type(mol)) # <class 'rdkit.Chem.rdchem.Mol'> |

**3.读入sdf文件：**

**3.1 文件批量读取**

从.sdf里批量读取：SDMolSupplier(fileName, sanitize, removeHs, strictParsing)

fileName：文件名

sanitize：检查化合价，计算芳香性、共轭、杂化、kekule，默认True

removeHs：是否隐藏氢原子，默认True

strictParsing：是否使用严格模式进行解析，默认True

|  |
| --- |
| >>> suppl = Chem.SDMolSupplier('data/batch.sdf')  >>> mols = [Chem.MolToSmiles(mol) for mol in suppl if mol]  >>> print(mols) # ['C1=C\\C=C/C=C\\C=C/1', 'c1ccccc1', 'c1ccoc1']  >>>print(type(mols\_suppl)) # <class 'rdkit.Chem.rdmolfiles.SDMolSupplier'> |

RDKit能读取各种各样的化学结构文件，类和方法主要在rdkit.Chem.rdmolfiles这个模块下，因此需要先导入包

|  |
| --- |
| >>from rdkit import Chem |

以sdf文件为例，官方给出了4种等价的读取方法，这些方法返回一个或多个rdkit.Chem.rdchem.Mol对象

|  |
| --- |
| >>suppl = Chem.SDMolSupplier('in.sdf') # 不省略H原子removeHs=False  >>mol = suppl[0] # 显示变量mols\_suppl的类型：<class ‘rdkit.Chem.rdmolfiles.SDMolSupplier’>。mols\_suppl可以看成是mol的列表，支持索引操作和迭代操作  >>print(mol.GetNumAtoms()) # 9 |

**3.2 压缩包批量读取**

从file object/.gz里读取

|  |
| --- |
| >>> import gzip  >>> gz\_file = gzip.open('data/batch.sdf.gz', 'r')  >>> suppl = Chem.ForwardSDMolSupplier(gz\_file)  >>> mols = [Chem.MolToSmiles(mol) for mol in suppl if mol]  >>> print(mols)  >>> f.close()  ['C1=C\\C=C/C=C\\C=C/1', 'c1ccccc1', 'c1ccoc1'] |

**输出分子：**

**1.输出SMILES/SMARTS**

**1.1 输出默认式**

输出SMILES：MolToSmiles(mol, isomericSmiles, kekuleSmiles, canonical, ...)

kekuleSmiles：默认False，不使用kekule时：脂肪族碳用"C"表示（大写），芳香族用"c"表示

isomericSmiles：默认True，区分同分异构体（"@"表示手性，"\"和"/"表示顺反异构）

canonical：默认True，输出标准SMILES

|  |
| --- |
| >>> m1 = Chem.MolFromSmiles('C1=CC=CC=CC=C1')  >>> m2 = Chem.MolFromSmiles('C1=CC=CC=C1')  >>> m3 = Chem.MolFromSmiles('C1=COC=C1')  >>> mols = [m1, m2, m3]  >>> print([Chem.MolToSmiles(mol) for mol in mols])  ['C1=CC=CC=CC=C1', 'c1ccccc1', 'c1ccoc1'] |

**1.2. 输出kekule式**

kekule形式：在符合4N+2规则的芳香体系中，通过使用双键代替小写的碳原子来表示芳香性

4N+2规则：也叫Hueckel规则，在闭环共轭体系中，当π电子数为4n+2时，才具有芳香性

|  |
| --- |
| >>> for mol in mols:  >>> Chem.Kekulize(mol)  >>> print([Chem.MolToSmiles(mol, kekuleSmiles=True) for mol in mols])  ['C1=CC=CC=CC=C1', 'C1=CC=CC=C1', 'C1=COC=C1']  # m1有共轭结构，但不属于芳香系统。m3中氧提供了2个π电子，碳各提供1个，总数为6，属于芳香系统 |

**1.3. 设置立体参数**

不区分同分异构体，通过isomericSmiles控制

|  |
| --- |
| >>> m4 = Chem.MolFromSmiles('C[C@H](O)c1ccccc1')  >>> print(Chem.MolToSmiles(m4))  C[C@H](O)c1ccccc1  >>> print(Chem.MolToSmiles(m4, isomericSmiles=False))  CC(O)c1ccccc1 |

**1.4. 批量输出SMILES**

批量输出SMILES：SmilesWriter(fileName, delimiter, includeHeader, nameHeader, isomericSmiles, kekuleSmiles)

fileName：输出文件名

delimiter：分隔符，默认为空格' '

includeHeader：是否写入表头，默认True

nameHeader：分子名一列的列名，默认'Name'

isomericSmiles：立体信息，默认True

kekuleSmiles：kekule形式，默认False

|  |
| --- |
| >>> writer = Chem.SmilesWriter('data/batch.smi', delimiter='\t')  >>> for i, mol in enumerate(mols):  >>> writer.write(mol)  >>> writer.close()  输出： |

**1.5. 批量输出SMILES和属性**

批量输出SMILES及属性，通过以下函数进行操作：

mol.GetPropNames()，查看分子属性列表

mol.GetProp()，获取相应属性

mol.SetProp(key, val)，新增属性名key、对应属性值val

writer.SetProps()，设置哪些属性要输出

以输出分子量和LogP为例，使用Descriptors计算属性，并添加

|  |
| --- |
| >>> writer = Chem.SmilesWriter('data/batch\_smiles.smi', delimiter='\t', nameHeader='mol\_id')  >>> writer.SetProps(['LOGP', 'MW'])  >>> for i, mol in enumerate(mols):  >>> mw = Descriptors.ExactMolWt(mol)  >>> logp = Descriptors.MolLogP(mol)  >>> mol.SetProp('MW', '%.2f' %(mw))  >>> mol.SetProp('LOGP', '%.2f' %(logp))  >>> mol.SetProp('\_Name', 'No\_%s' %(i))  >>> writer.write(mol)  >>> writer.close()  >>> print('number of mols:', writer.NumMols())  number of mols: 3  >>> print('mol properties:', [i for i in mol.GetPropNames()])  mol properties: ['MW', 'LOGP']  前面表格多了“MW”和“LOGP“两列 |

**1.6. 输出SMARTS**

|  |
| --- |
| >>> Chem.MolToSmarts(m3, isomericSmiles=True)  '[#6]1:[#6]:[#8]:[#6]:[#6]:1' |

**2.输出.sdf**

**2.1. 批量输出到.sdf**

批量输出到文件：SDWriter()

使用方法类似于SMILES的批量输出

可以像1.5.一样自定义属性信息，并记录在.sdf文件中

|  |
| --- |
| >>> writer = Chem.SDWriter('data/batch.sdf')  >>> writer.SetProps(['LOGP', 'MW'])  >>> for i, mol in enumerate(mols):  >>> mw = Descriptors.ExactMolWt(mol)  >>> logp = Descriptors.MolLogP(mol)  >>> mol.SetProp('MW', '%.2f' %(mw))  >>> mol.SetProp('LOGP', '%.2f' %(logp))  >>> mol.SetProp('\_Name', 'No\_%s' %(i))  >>> writer.write(mol)  >>> writer.close() |

**2.2. 批量输出到.gz**

|  |
| --- |
| >>> outf = gzip.open('data/batch.sdf.gz','wt+')  >>> writer = Chem.SDWriter(outf)  >>> for mol in mols:  >>> writer.write(mol)  >>> writer.close()  >>> outf.close() |

**3.输出.mol**

输出到文件：MolToMolFile(mol, filename, includeStereo, ...)

mol：mol对象

filename：文件名

includeStereo：立体信息，默认True

|  |
| --- |
| >>> m1.SetProp('\_Name', 'cyclobutane')  >>> Chem.MolToMolFile(m1, 'data/output.mol') |

**分子可视化：**

**1.单个展示：**

从mol对象到图片：MolToImage(mol, size, kekulize, wedgeBonds, fitImage, ...)

mol：mol对象

size：图片尺寸，默认(300, 300)

kekulize：是否展示kekule形式，默认True（True：芳香系统用实线表示，False：虚线表示）

wedgeBonds：是否展示楔形键，即立体构型，默认True

|  |
| --- |
| >>> from rdkit.Chem import Draw  >>> mol = Chem.MolFromSmiles('C[C@H](O)c1ccccc1')  >>> Draw.MolToImage(mol, size=(150,150), kekulize=True) |

在新窗口中展示图片：ShowMol()

参数基本同上

|  |
| --- |
| >>> Draw.ShowMol(mol, size=(150,150), kekulize=False) |

保存图片MolToFile(mol, filename, size, kekulize, wedgeBonds, ...)

参数基本同上

|  |
| --- |
| >>> Draw.MolToFile(mol, 'data/output.png', size=(150, 150)) |

**2.批量展示：**

**2.1. 从DataFrame中展示**

从df中展示：FrameToGridImage(frame, column, molsPerRow, subImgSize, legendsCol, ...)

frame：DataFrame对象

column：rdkit mol对象所在列

molsPerRow,：每行显示的分子数

subImgSize：图片大小

legendsCol：标题所在列

|  |
| --- |
| >>>df = pd.read\_csv('data/smiles\_df.csv')  >>>PandasTools.AddMoleculeColumnToFrame(df,'SMILES','mol',includeFingerprints=True)  >>>PandasTools.FrameToGridImage(df, column='mol', molsPerRow=5, subImgSize=(200,200), legendsCol='Name') |

**2.2. 从mol列表中展示**

从列表生成分子结构：MolsToGridImage(mols, maxMols, molsPerRow, subImgSize, legends, ...)

部分参数和上面的一致

mols：mol对象列表

maxMols：最多显示的分子数

molsPerRow,：每行显示的分子数

subImgSize：图片大小

legends：图题

|  |
| --- |
| >>> mols = df.mol.tolist()  >>> legends = df.Name.tolist()  >>>Draw.MolsToGridImage(mols, maxMols=2, molsPerRow=2, subImgSize=(300,300), legends=legends) |

**3. 3D展示;**

转换3D时，为了得到靠谱的三维构象，一般先加氢：AddHs(mol)

通过距离几何算法计算3D坐标：EmbedMolecule(mol, randomSeed, ...)

mol：mol对象

randomSeed：随机种子

转换完后再进行一步力场优化，比如MMFF94：MMFFOptimizeMolecule(mol)

|  |
| --- |
| >>> m3d = Chem.MolFromSmiles('CNC(=O)N(N(CCCl)S(C)(=O)=O)S(C)(=O)=O')  >>> m3d = Chem.AddHs(m3d)  >>> AllChem.EmbedMolecule(m3d, randomSeed=3)  >>> AllChem.MMFFOptimizeMolecule(m3d)  >>> Draw.MolToImage(m3d, size=(250,250)) |

**化学特征提取：**

|  |
| --- |
| from rdkit import Chem #rdkit.Chem.Lipinski模块进行计算描述符  from rdkit.Chem import ChemicalFeatures  from rdkit import RDConfig  import os  fdefName = os.path.join(RDConfig.RDDataDir,'BaseFeatures.fdef')  factory = ChemicalFeatures.BuildFeatureFactory(fdefName) |

默认情况下对任何化合物都不需要改动上面的代码。利用MolFromSmiles方法读取分子。

|  |
| --- |
| m = Chem.MolFromSmiles('OCc1ccccc1CN')  feats = factory.GetFeaturesForMol(m)  len(feats) # 8 |

这里分子为OCc1ccccc1CN，我们最终生成了8个特征，每个特征的提取方法如下所示

|  |
| --- |
| feats[0].GetFamily() # 'Donor'  feats[0].GetType() # 'SingleAtomDonor'  feats[0].GetAtomIds() # (0,)  feats[4].GetFamily() # 'Aromatic'  feats[4].GetAtomIds() # (2, 3, 4, 5, 6, 7) |

**一、描述符计算模块**

**1. rdkit.Chem.Lipinski模块**

rdkit中提供了许多描述符的计算方法，可用于分子筛选、成药性评估等。以lipinski类药的相关规则为例，可以通过rdkit.Chem.Lipinski模块进行计算。常用的一些性质举例：

氢键受体数NumHAcceptors

氢键供体数NumHDonors

可旋转键数NumRotatableBonds

脂肪环数量NumAliphaticRings

芳香环数量NumAromaticRings

SP3杂化碳原子比例FractionCSP3

|  |
| --- |
| >>> from rdkit import Chem  >>> from rdkit.Chem import Lipinski  >>> mol = Chem.MolFromSmiles('c1ccccc1C(=O)O')  >>> Lipinski.NumHAcceptors(mol) # 获取氢键受体数NumHAcceptors  1 |

将描述符名称(key)和值(val)添加到分子属性中：m.SetProp(key, val)

获取分子属性：m.GetProp(key)

|  |
| --- |
| >>> Ha = Lipinski.NumHAcceptors(mol)  >>> mol.SetProp('Ha', '%s'%Ha)  >>> mol.GetProp('Ha')  '1' |

**2.rdkit.Chem.Descriptors模块**

大部分的描述符都可以通过rdkit.Chem.Descriptors模块进行计算。该模块也包含了Lipinski的描述符。常用的一些描述符举例：

分子量MolWt

脂水分配系数MolLogP

拓扑极表面积TPSA

以计算TPSA为例：Descriptors.TPSA()

|  |
| --- |
| >>> from rdkit.Chem import Descriptors  >>> print(Descriptors.TPSA(mol), Descriptors.MolLogP(mol), Descriptors.MolWt(mol))  37.3 1.3848 122.12299999999998 |

**3.rdkit.ML.Descriptors.MoleculeDescriptors模块**

该模块可以批量计算描述符。

先指定一个列表des\_list，包含所要计算的描述符名称，

使用MolecularDescriptorCalculator创建一个计算描述符的对象，传入要计算的des\_list

调用对象的CalcDescriptors方法，传入要计算的mol对象，得到所需的描述符

|  |
| --- |
| >>> from rdkit.ML.Descriptors import MoleculeDescriptors  >>> des\_list = ['MolWt', 'NumHAcceptors', 'NumHDonors', 'MolLogP', 'NumRotatableBonds']  >>> calculator = MoleculeDescriptors.MolecularDescriptorCalculator(des\_list)  >>> calculator.CalcDescriptors(mol)  (122.12299999999998, 1, 1, 1.3848, 1) |

查看各种描述符的含义：GetDescriptorSummaries()

|  |
| --- |
| >>> calculator.GetDescriptorSummaries()  ['The average molecular weight of the molecule', # 该分子的平均分子量  'Number of Hydrogen Bond Acceptors', # 氢键受体数  'Number of Hydrogen Bond Donors', # 氢键供体数  'Wildman-Crippen LogP value',  'Number of Rotatable Bonds'] |

获取所有描述符：Descriptors.\_descList

|  |
| --- |
| >>> des\_list = [x[0] for x in Descriptors.\_descList]  >>> len(des\_list)  200 |

将calculator保存起来：SaveState(filename)

再次调用该calculator：pickle.load()

|  |
| --- |
| >>> import pickle  >>> calculator.SaveState('data/descriptor\_calculator')  >>> with open('data/descriptor\_calculator', 'rb') as f:  >>> calc = pickle.load(f)  >>> calc.CalcDescriptors(mol)  (122.12299999999998, 1, 1, 1.3848, 1) |

**二、原子描述符可视化**

可以用相似性地图来查看每个原子对描述符的贡献，更多相似性地图的应用可以查看这篇文章的第二部分。

**1.原子partial charge可视化**

计算partial charge要复杂一点。计算出的partial charge存储在每个原子的属性中，可以通过GetDoubleProp（浮点数）或GetProp（字符串）来获取。

partial charge可以表示电子的分布。分子中的化学键是由分布在相连原子周围的电子对组成的。但由于原子的电负性不同，成键电子并不是均匀分布。电负性大的原子吸电子能力强，成键的电子对会更偏向该原子，导致该原子带有负电的性质。相对应的，电负性小的原子则带有正电性质。Partial charge衡量了电子偏向的程度。

先计算partial charge：AllChem.ComputeGasteigerCharges()

获取某一个原子：GetAtomWithIdx()

获取原子的partial charge：GetDoubleProp(’\_GasteigerCharge’)

|  |
| --- |
| >>> from rdkit.Chem import AllChem  >>> mol=Chem.MolFromSmiles('COc1cccc2cc(C(=O)NCCCCN3CCN(c4cccc5nccnc54)CC3)oc21')  >>> AllChem.ComputeGasteigerCharges(mol)  >>> atom = mol.GetAtomWithIdx(0)  >>> atom.GetDoubleProp('\_GasteigerCharge')  0.07771844728655561 |

获取每个原子的partial charge，放到contribs中

|  |
| --- |
| >>> contribs = [round(mol.GetAtomWithIdx(i).GetDoubleProp('\_GasteigerCharge'), 2) for i in range(mol.GetNumAtoms())]  >>> print(contribs)  [0.08, -0.49, 0.16, -0.02, -0.06, ...] |

根据给定的权重，生成分子权重图：GetSimilarityMapFromWeights(mol, weights, colorMap, contourLines, …)

mol：要绘制的mol对象

weights：权重

colorMap：matplotlib中的色系

contourLines：等高线数量

|  |
| --- |
| >>> from rdkit.Chem.Draw import SimilarityMaps  >>> fig = SimilarityMaps.GetSimilarityMapFromWeights(mol, contribs, contourLines=10) |

**2.原子logP可视化**

LogP表示脂水分配系数，该值认为与细胞通透性有一定相关性。rdkit中提供的Descriptors.MolLogP()方法可以粗略计算logP值，该方法首先做了一个原子分类系统，根据原子及其相连原子的不同而进行分类，再对化学性质相似、logP贡献相似的类别做合并，最终得到了68种精确的原子类别和4种通配类别，并用SMARTS表示。计算时，对一个分子中所有原子进行分类，再乘以每一类的权重并加和，最终得到LogP值。该方法在9920个分子的训练集上的r2为0.918，标准差为0.667。此外摩尔折射率（molar refractivity，MR）也可以通过这种方法计算得到。

计算每个原子的logP和MR值：rdMolDescriptors.\_CalcCrippenContribs

返回结果是每个原子logP和MR元组的列表

|  |
| --- |
| >>> from rdkit.Chem import rdMolDescriptors  >>> contribs = rdMolDescriptors.\_CalcCrippenContribs(mol)  >>> contribs[:3]  [(-0.2035, 2.753), (-0.4195, 1.182), (0.5437, 3.853)] |

生成分子权重图：GetSimilarityMapFromWeights()

|  |
| --- |
| >>> fig = SimilarityMaps.GetSimilarityMapFromWeights(mol, [x for x,y in contribs]) |