

RDKit + scikit-learn + SDF

by takigawa

すこしづつ拡充予定... というか拡充して！

このJupyter Notebookのスク립トは上のメニューからFile -> Download As -> Python(.py)とやれば普通に実行できるpythonスク립トとしてexportできます。実際はコード部分以外がコメントアウトされるだけですけど。

関連パッケージのimport

cf) pyenv + anacondaでのインストール

```
$ pyenv install --list | grep anaconda
$ pyenv install anaconda2-2.5.0
$ pyenv global anaconda2-2.5.0
$ conda update conda
$ conda install -c https://conda.anaconda.org/rdkit rdkit
```

```
In [5]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import collections
from numpy import vectorize as vec

from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem
from rdkit.Chem import Draw
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Descriptors, PandasTools
from rdkit.ML.Descriptors import MoleculeDescriptors
```

```
/Users/takigawa/.pyenv/versions/anaconda2-2.5.0/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserWarning: Matplotlib is building the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list
. This may take a moment.')
```

テスト: PubChem BioAssay AID 1851

データは下記のページ右上のDownloadのtested substancesから「SDF」、Data Tableの右上のDownloadのCSV Saveから「Data Table(All)」の二つをダウンロード。

<https://pubchem.ncbi.nlm.nih.gov/bioassay/1851>

(<https://pubchem.ncbi.nlm.nih.gov/bioassay/1851>)

Molオブジェクト

<http://www.rdkit.org/docs/api/rdkit.Chem.rdchem.Mol-class.html>
(<http://www.rdkit.org/docs/api/rdkit.Chem.rdchem.Mol-class.html>)

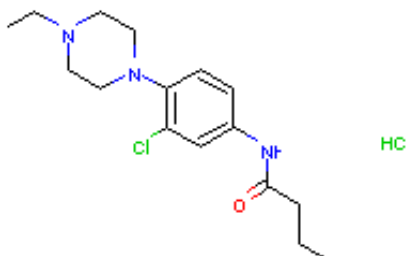
SDMolSupplierを使うとmolオブジェクトのイテレータが得られる。

```
In [6]: sdfFile = './aid_1851/AID_1851_tested_substances.sdf'
        suppl = Chem.SDMolSupplier(sdfFile)
```

一つ目のmolオブジェクトを取りだしてmolとする。そのままmolを入力すると構造式で表示される。

```
In [7]: mol = suppl[0]
        mol
```

Out[7]:



molオブジェクトにはいろいろなメソッドが用意されている。

```
In [8]: mol.GetNumAtoms(), mol.GetNumBonds(), mol.GetNumHeavyAtoms()
```

Out[8]: (22, 22, 22)

SDFファイルに分子に関するメタ情報がついていた場合、それを抽出できる。

```
In [9]: list(mol.GetPropNames())
```

```
Out[9]: ['PUBCHEM_CID_ASSOCIATIONS',
         'PUBCHEM_COMPOUND_ID_TYPE',
         'PUBCHEM_COORDINATE_TYPE',
         'PUBCHEM_EXT_DATASOURCE_NAME',
         'PUBCHEM_EXT_DATASOURCE_REGID',
         'PUBCHEM_EXT_DATASOURCE_URL',
         'PUBCHEM_SUBSTANCE_COMMENT',
         'PUBCHEM_SUBSTANCE_ID',
         'PUBCHEM_SUBSTANCE_SYNONYM',
         'PUBCHEM_SUBSTANCE_VERSION',
         'PUBCHEM_TOTAL_CHARGE',
         'PUBCHEM_XREF_EXT_ID']
```

```
In [10]: mol.GetProp('PUBCHEM_SUBSTANCE_SYNONYM')
```

```
Out[10]: 'MLS000034554\nN-[3-Chloro-4-(4-ethyl-piperazin-1-yl)-phenyl]-butyramide\nSMR000014974'
```

化学構造式データ(SDF)のロード

下記がSDFファイルをPandasのデータテーブルとしてロードする。MolSupplierでも良いがPandasのテーブルとして直で読んだほうがデータ操作においては便利。ロード時にSDFのmolパートが化学的に正しいかをチェックする。

一部pubchemからダウンロードしたSDFに、Valenceが変な分子が混じっていてRDKitがErrorを吐くが、とりあえず気にしないですすむ。3つの化合物でエラーが出てデータに入らない(これが原因で構造が3個少なくなるが気にしない)。

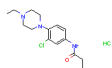
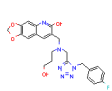
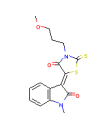
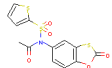
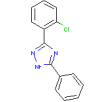
```
In [11]: sdfFile = './aid_1851/AID_1851_tested_substances.sdf'
df = PandasTools.LoadSDF(sdfFile, smilesName='SMILES', molColName='Molecu
```

```
RDKit ERROR: [01:28:28] Explicit valence for atom # 0 Sb, 7, is greater than permitted
RDKit ERROR: [01:28:28] ERROR: Could not sanitize molecule ending on line 510448
RDKit ERROR: [01:28:28] ERROR: Explicit valence for atom # 0 Sb, 7, is greater than permitted
RDKit ERROR: [01:28:28] Explicit valence for atom # 24 C, 5, is greater than permitted
RDKit ERROR: [01:28:28] ERROR: Could not sanitize molecule ending on line 541463
RDKit ERROR: [01:28:28] ERROR: Explicit valence for atom # 24 C, 5, is greater than permitted
RDKit ERROR: [01:28:28] Explicit valence for atom # 2 Cl, 3, is greater than permitted
RDKit ERROR: [01:28:28] ERROR: Could not sanitize molecule ending on line 594137
```

pandasと同じように読み込まれたデータテーブルdfに対してhead(n)とすると最初のn行を表示する。nを省略するとn=5。

```
In [12]: df.head()
```

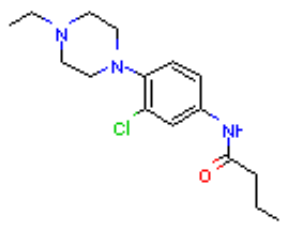
```
Out[12]:
```

	ID	Molecule	PUBCHEM_BONDANNOTATIONS	PUBCHEM_CID_ASSOCIATION
0	842238		NaN	6602638 1\n313 2\n644499 2
1	842250		NaN	644510 1
2	842319		NaN	1960010 1
3	842408		NaN	644675 1
4	842584		NaN	644851 1

下記のような感じでRのようにsubsettingしたりできる。詳しくはpandasの機能を調べること。

```
In [13]: df[df['PUBCHEM_SUBSTANCE_ID'] == '842238'][['ID', 'Molecule', 'PUBCHEM_
```

```
Out[13]:
```

	ID	Molecule	PUBCHEM_SUBSTANCE_ID
0	842238		842238

活性値の計測結果(CSV)のロード

こちらはふつうにCSVファイルをpandasの機能で読む。まず、テーブルの列名用にヘッダだけ先読み。

```
In [14]: csvHeader = pd.read_csv('./aid_1851/AID_1851_datatable_all.csv', nrows=
colnames = list(csvHeader.columns)
```

最初の9行は様々なヘッダ情報なのでとばして、次にデータ部分を直接よむ。併せて、さきほど先読みしておいた列名を付与。型が混合していて分からないエラーがでるが使用する列には関係ないためとりあえずそのまま進む。

```
In [15]: csvFile = pd.read_csv('./aid_1851/AID_1851_datatable_all.csv', skiprows=
csvFile.columns = colnames
```

```
/Users/takigawa/.pyenv/versions/anaconda2-2.5.0/lib/python2.7/site-p
ackages/IPython/core/interactiveshell.py:2902: DtypeWarning: Columns
(9) have mixed types. Specify dtype option on import or set low_memo
ry=False.
    interactivity=interactivity, compiler=compiler, result=result)
```

データ表のサイズを確認しておく。

```
In [16]: csvFile.shape
```

```
Out[16]: (17143, 147)
```

読み込んだデータ表の最初5行を表示。

```
In [17]: csvFile.head()
```

```
Out[17]:
```

		PUBCHEM_RESULT_TAG	PUBCHEM_SID	PUBCHEM_CID	PUBCHEM_ACTIVITY_C
0	1		842238	6602638	NaN
1	2		842250	644510	NaN
2	3		842319	1960010	NaN
3	4		842408	644675	NaN
4	5		842584	644851	NaN

活性値情報の抽出 by pandas

活性値情報を含む列名には「Activity Outcome」という文字列が含まれていることがわかるとおもうのでこの列のindexを抜いておく。また、各化合物のSIDとCIDが入っている1-3列目もTrueにしておく。

```
In [18]: out_idx = csvFile.columns.str.contains('Activity Outcome')
out_idx[0:3]= True
```

上でTrueで選択した列のみで、データ表を抽出。

```
In [19]: mat = csvFile.ix[:, out_idx]
mat.head()
```

```
Out[19]:
```

	PUBCHEM_RESULT_TAG	PUBCHEM_SID	PUBCHEM_CID	Activity Outcome	Activity Outcome.1
0	1	842238	6602638	Inactive	Inconclusiv
1	2	842250	644510	Inconclusive	Inconclusiv
2	3	842319	1960010	Inconclusive	Inactive
3	4	842408	644675	Active	Inactive
4	5	842584	644851	Active	Inconclusiv

各々の活性値でActiveとInactiveの数を調べて見る。各々、下記論文の1851(2c19)、1851(2d6)、1851(3a4)、1851(1a2)、1851(2c9)に相当。論文のTable 1と同じActiveとInactiveの個数になっていることを確認する。論文では構造式からの特徴ベクトル計算ソフトウェアであるDragonの特徴(記述子)を使っている。

Multi-task Neural Networks for QSAR Predictions <https://arxiv.org/abs/1406.1231>
(<https://arxiv.org/abs/1406.1231>)

分子記述子計算ソフトウェア Dragon <http://affinity-science.com/dragon/index.html>
(<http://affinity-science.com/dragon/index.html>)

なおこの論文は下記の予測コンペで優勝したディープラーニングによるモデル。ディープラーニングが流行するキッカケにもなった成果の一つ。優勝賞金は\$22,000 (約220万円)だった。

Merck Molecular Activity Challenge <https://www.kaggle.com/c/MerckActivity>
(<https://www.kaggle.com/c/MerckActivity>)

```
In [20]: collections.Counter(mat.ix[:, 3])
```

```
Out[20]: Counter({'Active': 5913, 'Inactive': 7532, 'Inconclusive': 3698})
```

```
In [21]: collections.Counter(mat.ix[:, 4])
```

```
Out[21]: Counter({'Active': 2771, 'Inactive': 11139, 'Inconclusive': 3233})
```

```
In [22]: collections.Counter(mat.ix[:, 5])
```

```
Out[22]: Counter({'Active': 5266, 'Inactive': 7751, 'Inconclusive': 4126})
```

```
In [23]: collections.Counter(mat.ix[:, 6])
```

```
Out[23]: Counter({'Active': 6000, 'Inactive': 7256, 'Inconclusive': 3887})
```

```
In [24]: collections.Counter(mat.ix[:, 7])
```

```
Out[24]: Counter({'Active': 4119, 'Inactive': 8782, 'Inconclusive': 4242})
```

上のデータで機械学習してみる

やること：

1. 活性値のデータを取り出しPUBCHEM_SIDと活性値の表を作成する。
2. 活性値のデータにあるPUBCHEM_SIDのところだけの構造式データを対応づける。
3. 構造式データの部分を特徴ベクトルに変換する。

活性値のデータを取り出す

1851(2c19)の活性値の予測問題のデータを作成

```
In [25]: collections.Counter(mat.ix[:, 3])
```

```
Out[25]: Counter({'Active': 5913, 'Inactive': 7532, 'Inconclusive': 3698})
```

```
In [26]: 5913 + 7532
```

```
Out[26]: 13445
```

```
In [27]: idx = mat.ix[:, 3].isin(['Active', 'Inactive'])  
label = mat.ix[idx, [1, 3]]  
label.shape
```

```
Out[27]: (13445, 2)
```

```
In [28]: label.head()
```

```
Out[28]:
```

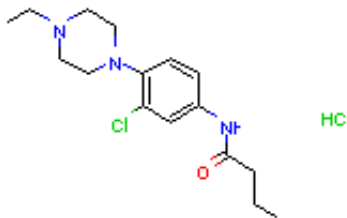
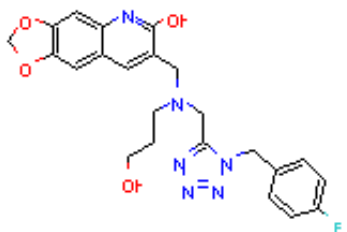
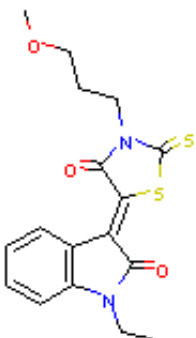
	PUBCHEM_SID	Activity Outcome
0	842238	Inactive
3	842408	Active
4	842584	Active
5	842618	Active
6	842697	Active

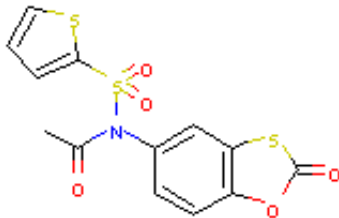
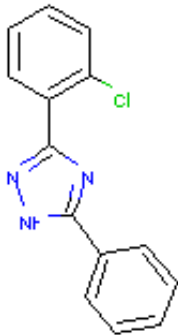
構造式データをSIDに対応づける

構造データをこのデータに含まれるSIDの部分セットにして対応づける

```
In [82]: struct = pd.DataFrame()  
struct['PUBCHEM_SID'] = df['PUBCHEM_SUBSTANCE_ID'].astype('int')  
struct['Molecule'] = df['Molecule']  
struct.head()
```

Out[82]:

	PUBCHEM_SID	Molecule
0	842238	 <chem>CCN1CCN(CC1)C(=O)c2ccc(Cl)cc2</chem>
1	842250	 <chem>COc1ccc(cc1)C2=CN(C2)C3=CC(=CC=C3)N4=CN(C4)C5=CC(=CC=C5)N6=CN(C6)C7=CC(=CC=C7)F</chem>
2	842319	 <chem>COc1ccc(cc1)C2=CN(C2)C3=CC(=CC=C3)N4=CN(C4)C5=CC(=CC=C5)N6=CN(C6)C7=CC(=CC=C7)N8=CN(C8)C9=CC(=CC=C9)OC</chem>

3	842408	
4	842584	

2つのデータフレームlabelとstructをキーPUBCHEM_SIDで結合。ここではとりあえずleft(label)のほうへ結合してみる。

```
In [85]: descr = pd.merge(label, struct, how='left', on='PUBCHEM_SID')
descr.shape
```

```
Out[85]: (13445, 3)
```

総数があうが、最初にSDFをロードしたときエラーになった下記3つが読み込めていない。

```
In [87]: descr[descr['Molecule'].isnull()].head()
```

```
Out[87]:
```

	PUBCHEM_SID	Activity Outcome	Molecule
3887	4252620	Inactive	NaN
4177	4252974	Inactive	NaN
4648	4253551	Active	NaN

とりあえず、結合するときに両方ともにあるものだけにinner結合を行う。上の3つは除外される。

```
In [88]: descr = pd.merge(label, struct, how='inner', on='PUBCHEM_SID')
descr.shape
```

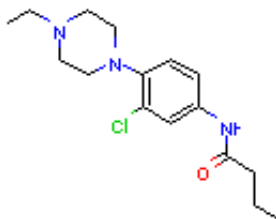
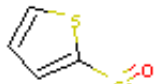
```
Out[88]: (13442, 3)
```

機械学習に使えるデータ「構造式」「活性値」のペアデータが完成。特に意味はないが見や

すいように列順を入れ替えておく。

```
In [89]: mydata = descr[['PUBCHEM_SID', 'Molecule', 'Activity Outcome']]  
mydata.head()
```

Out[89]:

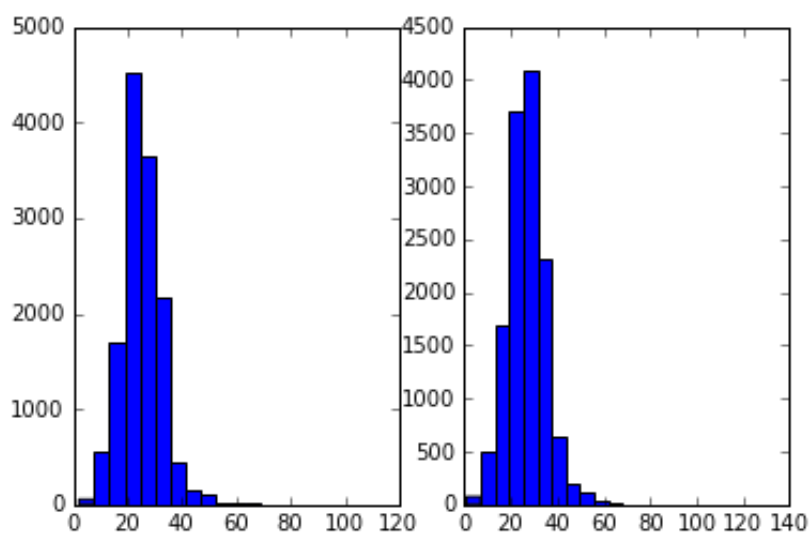
	PUBCHEM_SID	Molecule	Activity Outcome
0	842238		Inactive
			

```
In [90]: atom_num = [m.GetNumAtoms() for m in descr['Molecule']]  
bond_num = [m.GetNumBonds() for m in descr['Molecule']]
```

```
In [91]: max(atom_num), min(atom_num), max(bond_num), min(bond_num)
```

Out[91]: (114, 2, 123, 1)

```
In [92]: plt.subplot(1, 2, 1)  
p1 = plt.hist(atom_num, 20)  
plt.subplot(1, 2, 2)  
p2 = plt.hist(bond_num, 20)
```



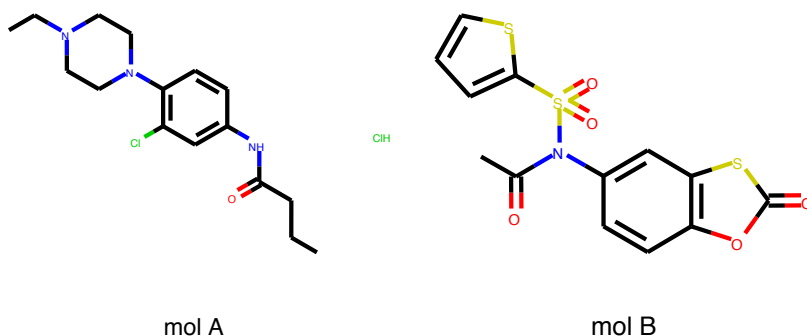
フィンガープリントについて

molオブジェクトからフィンガープリントを得る機能が用意されている。基本的にはビットベクトルオブジェクトになる(バイナリ値)が、類似度比較や検索などに使うことができる。

```
In [93]: mols = mydata['Molecule']
```

```
In [94]: mol_a = mols[0]
mol_b = mols[1]
Draw.MolsToGridImage([mol_a, mol_b], legends=['mol A', 'mol B'], molsPerRow=2)
```

Out[94]:



フィンガープリント間の距離はよく Tanimoto 係数(Jaccard Indexと等価)という指標で測る。

```
In [95]: bitvec1 = AllChem.GetHashedMorganFingerprint(mol_a, radius=2, nBits = 1024)
bitvec2 = AllChem.GetHashedMorganFingerprint(mol_b, radius=2, nBits = 1024)
DataStructs.TanimotoSimilarity(bitvec1, bitvec2)
```

Out[95]: 0.14150943396226415

UIntSparseIntVect という型で保持されているので、非ゼロの添え字とそこにハッシュされる構造がいくつあるかの辞書になっている。ほとんどのビットが0の場合、効率が良いスパース表現での保持。

```
In [96]: type(bitvec1)
```

Out[96]: rdkit.DataStructs.cDataStructs.UIntSparseIntVect

参考情報 : *Morgan Fingerprint*

MorganFingerprintAsBitVect, HashedMorganFingerprint, GetMorganFingerprintの違い

<http://cheminformist.itmol.com/TEST/?p=423> (<http://cheminformist.itmol.com/TEST/?p=423>)

```
In [97]: m = mol_b
mgfp2 = AllChem.GetMorganFingerprintAsBitVect(m, 2, nBits=1024)
print mgfp2
for i in range(mgfp2.GetNumBits()):
    if mgfp2.GetBit(i) == True:
        print i, ", ",

mghfc2 = AllChem.GetHashedMorganFingerprint(m, 2, nBits=1024)
print mghfc2.GetLength()
dic = mghfc2.GetNonzeroElements()
print sorted(dic.items(), key=lambda x: x[0])
```

```
<rdkit.DataStructs.cDataStructs.ExplicitBitVect object at 0x12caf5de0>
14 , 33 , 64 , 72 , 101 , 162 , 171 , 241 , 249 , 314 , 350 , 356 ,
383 , 452 , 497 , 507 , 543 , 564 , 580 , 617 , 635 , 650 , 656 , 65
8 , 675 , 726 , 758 , 762 , 784 , 786 , 798 , 807 , 815 , 818 , 841
, 849 , 875 , 881 , 885 , 893 , 901 , 955 , 960 , 1017 , 1024
[(14, 1), (33, 1), (64, 1), (72, 1), (101, 1), (162, 1), (171, 1), (
241, 1), (249, 1), (314, 1), (350, 1), (356, 5), (383, 1), (452, 2),
(497, 1), (507, 1), (543, 1), (564, 1), (580, 1), (617, 1), (635, 1)
, (650, 4), (656, 1), (658, 1), (675, 2), (726, 3), (758, 1), (762,
1), (784, 1), (786, 2), (798, 1), (807, 1), (815, 1), (818, 1), (841
, 1), (849, 6), (875, 1), (881, 1), (885, 1), (893, 1), (901, 1), (9
55, 1), (960, 1), (1017, 1)]
```

ExplicitBitVectorのほうはnumpy.arrayに変換できる。

```
In [98]: type(mgfp2)
```

```
Out[98]: rdkit.DataStructs.cDataStructs.ExplicitBitVect
```

```
In [99]: vv1 = np.array(mgfp2)
```

SparseIntVectのほうはExplicitBitVectorに何らかのやり方でなおしてから使う。

```
In [100]: type(mghfc2)
```

```
Out[100]: rdkit.DataStructs.cDataStructs.UIntSparseIntVect
```

```
In [101]: bv = DataStructs.ExplicitBitVect(1024)
bv.SetBitsFromList(dic.keys())
vv2 = np.array(bv)
```



```
In [103]: fp = Chem.RDKFingerprint(mol_a)
          fp.ToBitString(), fp.GetNumOnBits(), fp.GetNumBits(), fp.GetNumOffBits(

Out[103]: ('000001000001000000001000000000000000100100000101001000000100000010
10001000001100000100000000000010000001000000001000100000000000100000
001000001001000010000001000100001010100100000000000000010010011010000
10001010010000101110011000000000010000000000000010110010000011011010
0000001000000000000001001000000100100010000100100100000010010001110
00100000100001100000000101000000000000010001010011010101000010000000
00000111100000100100000000000000001011000001000000000001110000011000
00000000000000011000000010010010010010010100001000000010100000000001
00001010001100100000010000000001010001100010010001000001000010000100
00000010011001000010000100000100100101110100000000100100100000001001
000100010000000000001110000000001000001000010010000001100110110000001
01000000010001100001100000100100000101001000101000101101110100010010
000000000111000000010010000010000100000001100001000000000000001010100
00101010000110000000000001010000000010001101000000100010000000000000
10000000000000110000010001010000100110000101000101000010000010010000
1100000010000000010100100000000010000000000000000000000100001100100000
00100010000100000001011011000000010000000010000111101010000000111000
00000100100110010000000100100000000000110000010000100100000101000001
1010000000010000001000000100000000000010010000010000010010000000100
00000000001001001011000000000000100000001000000010000100001000010000
10001000101100101000110000100010000001000101000000100110010001000100
00000000000000000000000011000100001000000000100001000100000010000000
001000010010010000000001000100100100111000000100001000000101100010001
001000001110100111000010000000010000110001010000000000000000000010001
00010000000001000100100100001100000001001000001100000000000000000010
00000000101110000011001010010001000000000011000000010110000001000000
00000010100000000010100001000000001000000010100110010110100000000001
0000100010010000000000000000000001010000010001000000000001001001110100
010101001000000000010011001000000000000100000000000000010010001000000
00000010000010000000000010000100000000000000000000000000000000000011
0000000001',
          431,
          2048,
          1617)
```

構造データをフィンガープリントに変換

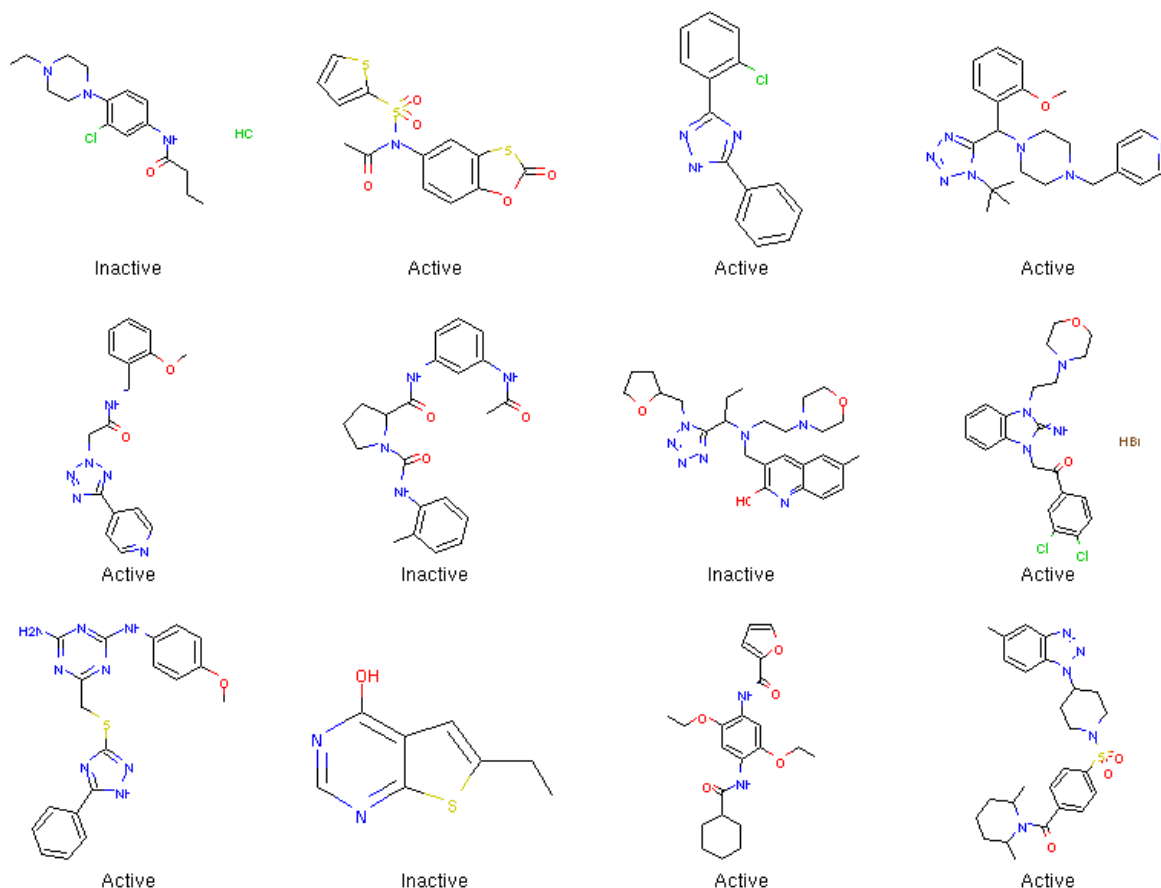
作成したデータの構造式部分を特徴ベクトルに変換。ここではMorgan Fingerprintのビットベクトルに変換する。明示的にhydrogen(水素)を付与したものもつっておく。

```
In [104]: mols = mydata['Molecule']
          molsWithHs = [Chem.AddHs(m) for m in mols]
```

下記のようにすると画像グリッドでみることができる。

```
In [105]: Draw.MolsToGridImage(mols[0:12], legends=mydata['Activity Outcome'])[0:12]
```

Out[105]:



RDKitで利用できるFingerprint <http://cheminformist.itmol.com/TEST/?p=438>
(<http://cheminformist.itmol.com/TEST/?p=438>)

Morganフィンガープリントの半径パラメタを変えてみたり、下記の他のフィンガープリントあたりも試してみると良い。

- GetMorganFingerprintAsBitVect
- GetHashedMorganFingerprint
- GetMorganFingerprint
- GetAtomPairFingerprint
- GetHashedAtomPairFingerprintAsBitVect
- GetTopologicalTorsionFingerprint
- GetHashedTopologicalTorsionFingerprintAsBitVect
- GetMACCSKeysFingerprint
- GetAvalonFP

下記がRDKitのページにのっているやり方。やり方はいろいろある。

```
X = []
for fp in fps:
    arr = np.zeros((1,))
    DataStructs.ConvertToNumpyArray(fp, arr)
    X.append(arr)
```



```
In [112]: print(rf.predict([X[0]]))
          print(rf.predict_proba([X[0]]))

          ['Inactive']
          [[ 0.12  0.88]]
```

なお、AUCを計算するためにはyyを0/1にしておく必要があるみたいなので、変換してみる。

```
In [113]: XX = np.array(X)
          encode = {'Active':1, 'Inactive':0}
          yy = np.array([ encode[l] for l in y])
```

```
In [114]: from sklearn.model_selection import cross_val_score
          from sklearn.metrics import roc_auc_score
```

5-fold cross validationして、まず正答率を表示してみる。

```
In [115]: scores = cross_val_score(rf, X, y, cv=5, scoring='accuracy')
          print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

          Accuracy: 0.73 (+/- 0.12)
```

5-fold cross validationでのAUC値を推定してみる。

```
In [116]: scores = cross_val_score(rf, XX, yy, cv=5, scoring='roc_auc')
          print("AUC: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

          AUC: 0.81 (+/- 0.12)
```

すこし細かくみてる場合：全データを適当に訓練セット(75%)とテストセット(25%)に分ける。

```
In [117]: from sklearn.model_selection import ShuffleSplit
          shufsp = ShuffleSplit(n_splits=100, test_size=0.25, random_state=0)
```

```
In [118]: train_index, test_index = next(iter(shufsp.split(XX)))
          X_train, X_test = np.array(XX)[train_index], np.array(XX)[test_index]
          y_train, y_test = np.array(yy)[train_index], np.array(yy)[test_index]
```

```
In [119]: rf = RandomForestClassifier(n_estimators=100, random_state=0)
rf.fit(X_train, y_train)
```

```
Out[119]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
'gini',
                                max_depth=None, max_features='auto', max_leaf_nodes=None
,
                                min_impurity_split=1e-07, min_samples_leaf=1,
                                min_samples_split=2, min_weight_fraction_leaf=0.0,
                                n_estimators=100, n_jobs=1, oob_score=False, random_stat
e=0,
                                verbose=0, warm_start=False)
```

```
In [120]: from sklearn.metrics import confusion_matrix
y_pred = rf.predict(X_test)
confusion_matrix(y_test, y_pred)
```

```
Out[120]: array([[1506,  349],
                 [ 274, 1232]])
```