Library selection demo

June 20, 2019

In [1]:
```
from os import listdir
from os.path import join
from fnmatch import fnmatch
import pickle
import pandas as pd

import numpy as np
from scipy.stats import gaussian_kde
from sklearn.decomposition import IncrementalPCA
from sklearn.manifold import TSNE

from rdkit import DataStructs
from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem import Descriptors
from rdkit.Chem import Draw
from rdkit.Chem.Fingerprints import FingerprintMols

from matplotlib import pyplot as plt
import py3Dmol

from tqdm import tqdm_notebook as tqdm
```

```
        import warnings
        warnings.filterwarnings('ignore')
```

In [2]:
```
        def draw3D(m,p=None,confId=-1):
            '''
            draw chemical structures using 3D stick depictions
            adapted from http://rdkit.blogspot.com/2016/07/a-recent-post-on-in-pipeline-talked.html
            '''
            mb = Chem.MolToMolBlock(m,confId=confId)
            if p is None:
                p = py3Dmol.view(width=400,height=400)
            p.removeAllModels()
            p.addModel(mb,'sdf')
            p.setStyle({'stick':{}})
            p.setBackgroundColor('0xeeeeee')
            p.zoomTo()
            return p.show()

        def gaussianKDE(x, y):
            '''
            kernel density estimation for coloring scatter plots
            see https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html
            '''
            xy = np.vstack([x,y])
            z = gaussian_kde(xy)(xy)
            return z

        def load_library(library):
            '''
            read structure files with .sdf and .smi extensions
            '''
            if library.split('.')[-1] == 'sdf':
                mols = Chem.SDMolSupplier(join('data', library.split('/')[-1]))
            elif library.split('.')[-1] == 'smi':
```

```
        mols = Chem.SDMolSupplier(join('data', library.split('/')[-1]))
    else:
        print('ERROR: please provide a path to a molecule data file in valid .smi or .sdf format')
    return mols
```

1 Simplified Molecular Input Line Entry System (SMILES)

1.1 Advantages:

- Memory efficient: relatively small data files

1.2 Disadvantages:

- Only capable of encoding 2D structure information
- Can only include one identifier for each compound (i.e. no other molecule attributes)

```
In [3]: mols = Chem.SmilesMolSupplier(join('data', 'Maybridge_Ro3_Diversity_Set1.smi')) # read SMILES file
        m = mols[1] # second molecule
        print(Chem.MolToSmiles(m)) # SMILES pattern for second molecule
        Draw.MolToImage(m) # draw 2D structure
```

```
CC(=O)NC1CCSC1=O
```

```
Out[3]:
```

2 Structure Data File (SDF)

2.1 Advantages:

- Capable of encoding 3D coordinates
- Can include as many properties for each molecule as desired

2.2 Disadvantages:

- Larger files (require $10 - 100 \times$ more memory)

```
In [4]: mols = Chem.SDMolSupplier(join('data', 'Maybridge_Ro3_Diversity_Set1.sdf')) # read SD file (coordinates are 2D)
        m = mols[1] # second molecule
        print(Chem.MolToMolBlock(m)) # 2D block for second molecule
```

```
     RDKit          2D

 10 10  0  0  0  0  0  0  0  0999 V2000
    4.6250   -5.5000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    5.3417   -4.2542    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
    4.6292   -4.6750    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.9167   -5.9167    0.0000 S   0  0  0  0  0  0  0  0  0  0  0  0
    6.0542   -4.6625    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    5.3417   -5.9125    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    6.0667   -5.4875    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.2042   -4.6667    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.2042   -5.4917    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    6.7667   -4.2417    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  2  3  1  0
  3  1  1  0
  4  1  1  0
  5  2  1  0
  6  1  2  0
  7  5  2  0
  8  3  1  0
  9  4  1  0
 10  5  1  0
  8  9  1  0
M  END
```

```
In [5]: draw3D(m) # show 3D depiction
```

2.3 2D molecules can be converted to 3D using RDKit

- 3D structure coordinates are needed to calculate 3D molecular descriptors (see normalized principle moment ratio and plane of best fit later on)
- After 3D coordinates are generated, they can be saved in SDF format

```
In [6]: m2 = Chem.AddHs(m) # hydrogens need to be added for accurate 3D structure prediction
        AllChem.EmbedMolecule(m2) # add 3d coordinates
        AllChem.MMFFOptimizeMolecule(m2) # forcefield minimization
        print(Chem.MolToMolBlock(m2)) # 3D block for second molecule
```

```
     RDKit          3D

 19 19  0  0  0  0  0  0  0  0999 V2000
   -1.3912   -0.7729    1.3471 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.8149   -0.2635    0.4168 N   0  0  0  0  0  0  0  0  0  0  0  0
   -0.5829    0.1418    0.4393 C   0  0  0  0  0  0  0  0  0  0  0  0
   -3.0360   -0.9679    0.7518 S   0  0  0  0  0  0  0  0  0  0  0  0
    1.7695    0.5369   -0.1621 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.9751   -1.3091    2.3644 O   0  0  0  0  0  0  0  0  0  0  0  0
    1.5013    1.5607   -0.7817 O   0  0  0  0  0  0  0  0  0  0  0  0
   -1.2598    0.0308   -0.9249 C   0  0  0  0  0  0  0  0  0  0  0  0
   -2.7563    0.1070   -0.6829 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.1829    0.0599    0.0334 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.1099   -0.9218    1.1331 H   0  0  0  0  0  0  0  0  0  0  0  0
   -0.6496    1.1641    0.8313 H   0  0  0  0  0  0  0  0  0  0  0  0
   -0.9370    0.8272   -1.6038 H   0  0  0  0  0  0  0  0  0  0  0  0
   -1.0022   -0.9206   -1.4107 H   0  0  0  0  0  0  0  0  0  0  0  0
   -3.0643    1.1273   -0.4334 H   0  0  0  0  0  0  0  0  0  0  0  0
   -3.3289   -0.2370   -1.5482 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.6838    0.7108    0.7545 H   0  0  0  0  0  0  0  0  0  0  0  0
```

```
    3.2128   -0.9696    0.4012 H   0  0  0  0  0  0  0  0  0  0  0  0
    3.7081    0.0959   -0.9251 H   0  0  0  0  0  0  0  0  0  0  0  0
  2  3  1  0
  3  1  1  0
  4  1  1  0
  5  2  1  0
  6  1  2  0
  7  5  2  0
  8  3  1  0
  9  4  1  0
 10  5  1  0
  8  9  1  0
  2 11  1  0
  3 12  1  0
  8 13  1  0
  8 14  1  0
  9 15  1  0
  9 16  1  0
 10 17  1  0
 10 18  1  0
 10 19  1  0
M  END
```

```
In [7]: draw3D(m2) # show 3D representation
```

2.4 Additional molecular properties are stored for each molecule after the end of the structure coordinates (M END) and before the compound delimiter ($$$$)

```
In [8]: [str(x) for x in m.GetPropNames()] # list properties stored for compounds in current SD file
```

```
Out[8]: ['Code',
         'Structure_smiles',
```

```
'Appearance',
'casno',
'product_name',
'c_log_p',
'acd_code',
'Flexibility',
'PSA',
'h_bond_donors',
'h_bond_acceptors',
'parent_mw',
'Heavy_Atom_Count',
'PAINS_Free',
'Set']
```

3 2D descriptors:

3.1 A popular school of thought holds that fragment libraries should contain molecules with the following attributes:

- Molecular Weight < 300 g/mol
- ClogP ≤ 3
- Number of hydrogen bond donors ≤ 3
- Number of hydrogen bond acceptors ≤ 3
- Number of rotatable bonds ≤ 3
- Polar surface area ≤ 60 Å

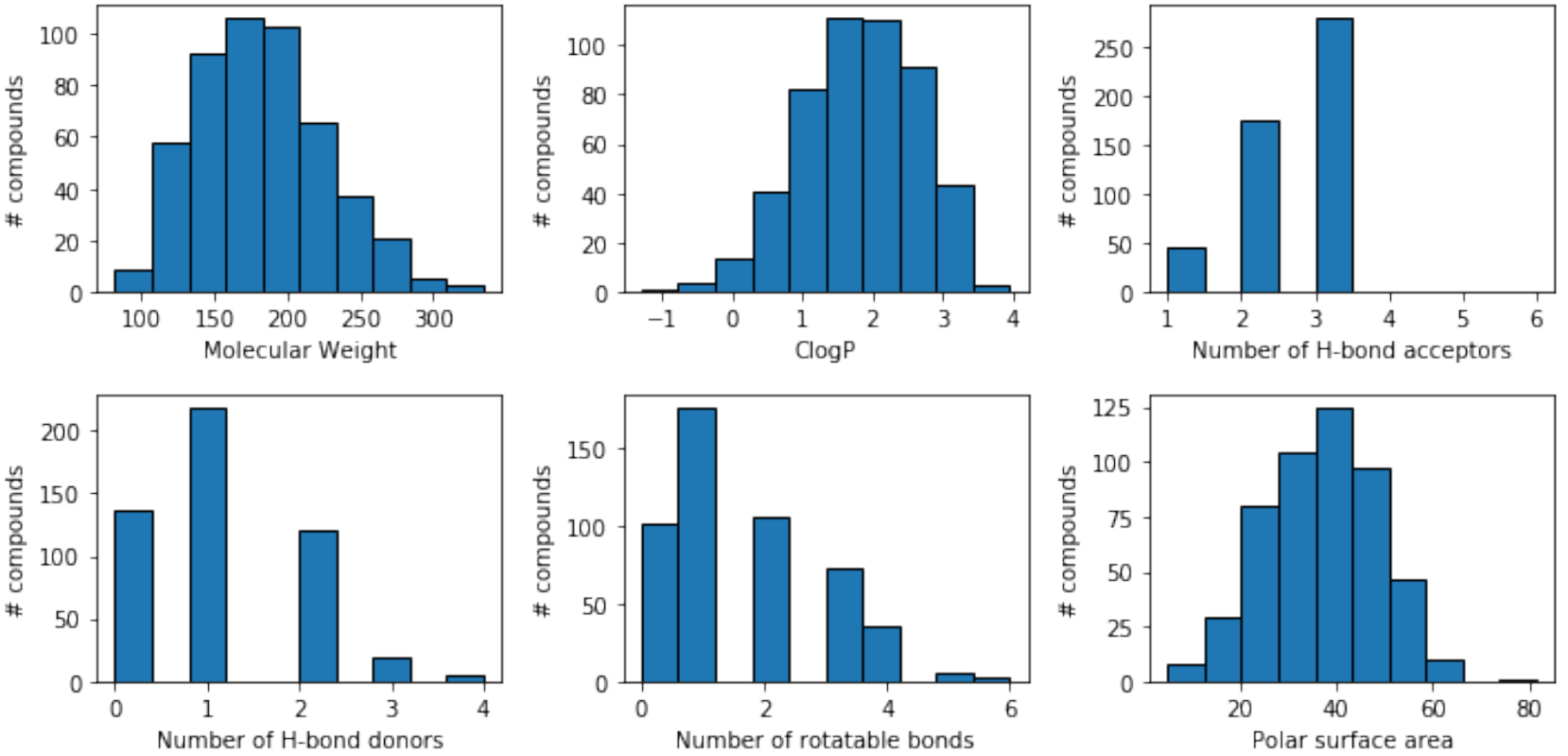
```
In [9]: def calc_descriptors(library):
            '''
            calculate 2D descriptors for a collection of molecules
            '''
            mols = load_library(library)
            xrange = range(len(mols))
            zeros = np.zeros(len(mols))
            df = pd.DataFrame({'Molecular Weight' : zeros, 'ClogP' : zeros, 'Number of H-bond acceptors' : zeros,
```

```
                        'Number of H-bond donors' : zeros, 'Number of rotatable bonds' : zeros,
                        'Polar surface area' : zeros})
        df.name = library
        for x, tq in zip(xrange, tqdm(xrange, desc='calculating...')):
            try:
                df['Molecular Weight'][x] = Descriptors.MolWt(mols[x])
                df['ClogP'][x] = Descriptors.MolLogP(mols[x])
                df['Number of H-bond acceptors'][x] = AllChem.CalcNumLipinskiHBA(mols[x])
                df['Number of H-bond donors'][x] = AllChem.CalcNumLipinskiHBD(mols[x])
                df['Number of rotatable bonds'][x] = Descriptors.NumRotatableBonds(mols[x])
                df['Polar surface area'][x] = Descriptors.TPSA(mols[x])
            except:
                pass
        return df

    def plot_descriptors(library):
        '''
        plot histograms for Ro3 descriptors from a collection of molecules
        '''
        df = calc_descriptors(library)
        fig, ax = plt.subplots(2, 3, figsize=(10, 5))
        c = 0
        for i in range(len(ax)):
            for j in range(len(ax[0])):
                desc = list(df)[c]
                ax[i][j].hist(df[desc], edgecolor='k', label=df.name)
                ax[i][j].set_xlabel(desc)
                ax[i][j].set_ylabel("# compounds")
                c += 1
        plt.tight_layout()
        plt.show()
        plt.close()
```

In [10]:
```
library = 'data/Maybridge_Ro3_Diversity_Set1.sdf' # Note: can change this to any other  structure file
```

```
plot_descriptors(library)
```



4 3-dimensionality (vs. flat structures) as a strategy to increase library diversity

4.1 Plots quantifying molecule 3D character depicted below

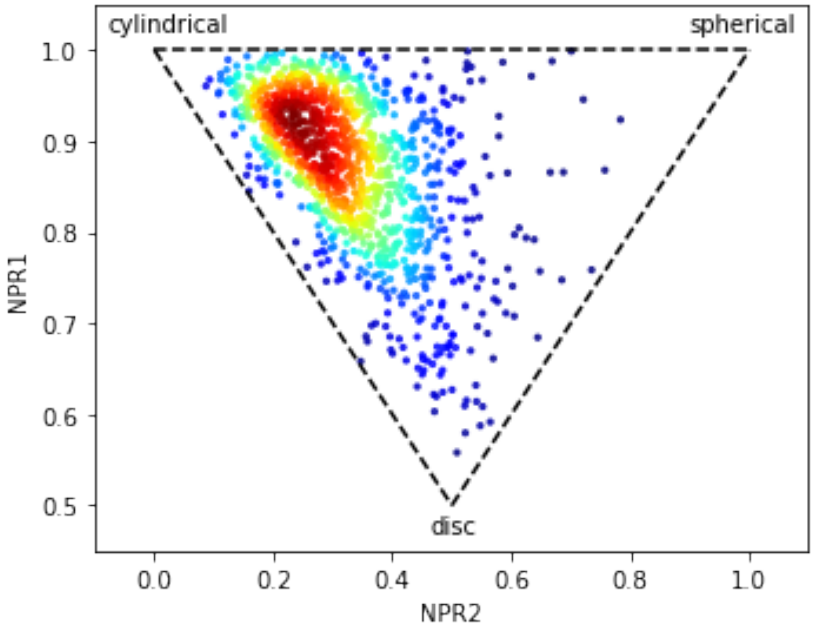
- 2-dimensional normalized ratios of principle moments of inertia
- Plane of best fit (PBF) score versus sum of NPRs

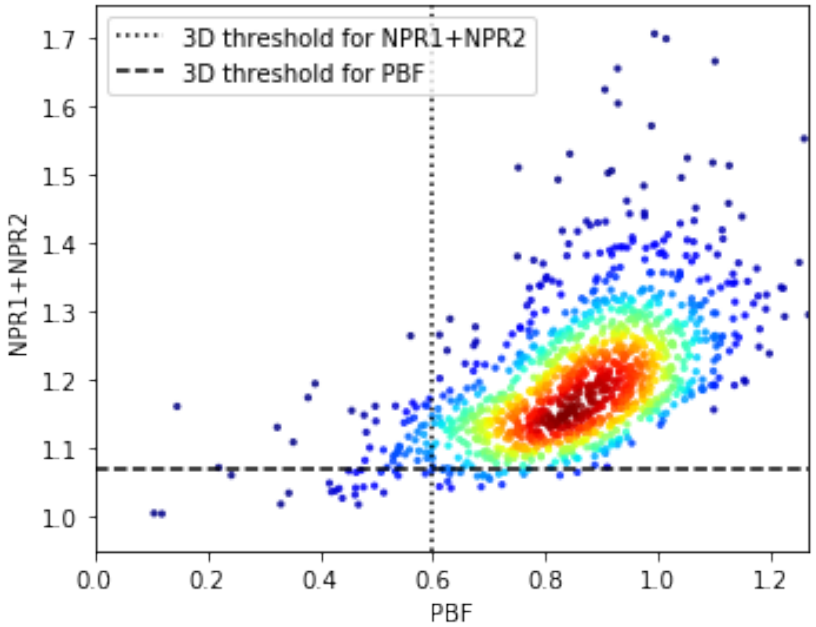
```
In [11]: def calc_3D_descriptors(library):
             '''
             Calculate 3D descriptors from collection of molecules
             '''
             mols = load_library(library)
             xrange = range(len(mols))
             zeros = np.zeros(len(mols))
             df = pd.DataFrame({'NPR1' : zeros, 'NPR2' : zeros, 'PBF' : zeros})
             df.name = library
             for x, tq in zip(xrange, tqdm(xrange, desc='calculating...')):
                 try:
                     m2 = Chem.AddHs(mols[x])
                     AllChem.EmbedMolecule(m2)
                     AllChem.MMFFOptimizeMolecule(m2)
                     df['NPR1'][x] = AllChem.CalcNPR1(m2)
                     df['NPR2'][x] = AllChem.CalcNPR2(m2)
                     df['PBF'][x] = AllChem.CalcPBF(m2)
                 except:
                     pass
             return df

         def plot_3D_descriptors(library):
             '''
             Create plots to assess molecule 3D character
             '''
             df = calc_3D_descriptors(library)
             fig, ax = plt.subplots(1, 2, figsize=(10, 4))
```

```
    ax[0].scatter(df.NPR1, df.NPR2, c=gaussianKDE(df.NPR1, df.NPR2), cmap='jet', s=5, alpha=0.8)
    ax[0].set_ylim([0.45, 1.05])
    ax[0].set_xlim([-.1, 1.1])
    ax[0].plot(np.linspace(0, 1, 100), np.repeat(1, 100), c='k', linestyle='--')
    ax[0].plot(np.linspace(0, 0.5, 100), np.linspace(0, 0.5, 100)*-1+1, c='k', linestyle='--')
    ax[0].plot(np.linspace(0.5, 1, 100), np.linspace(0.5, 1, 100)*1, c='k', linestyle='--')
    ax[0].text(-.08, 1.02, 'cylindrical')
    ax[0].text(0.90, 1.02, 'spherical')
    ax[0].text(0.465, 0.47, 'disc')
    ax[0].set_xlabel('NPR2')
    ax[0].set_ylabel('NPR1')
    ax[1].scatter(df.PBF, df.NPR1 + df.NPR2, c=gaussianKDE(df.PBF, df.NPR1 + df.NPR2),
                  cmap='jet', s=5, alpha=0.8)
    ax[1].set_ylim([0.95, 1.75])
    ax[1].set_xlim([0, np.max(df.PBF)])
    ax[1].plot(np.repeat(0.6, 100), np.linspace(0.95, 1.75, 100), c='k', linestyle=':',
              label='3D threshold for NPR1+NPR2')
    ax[1].plot(np.linspace(0, np.max(df.PBF), 100), np.repeat(1.07, 100), c='k', linestyle='--',
              label='3D threshold for PBF')
    ax[1].set_xlabel('PBF')
    ax[1].set_ylabel('NPR1+NPR2')
    ax[1].legend()
    plt.tight_layout()
    plt.show()
    plt.close()
```

In [12]:
```
library = 'data/Enamine_3D_Shaped_Diverse_Fragments.sdf' # Note: can change this to any other structure file
plot_3D_descriptors(library)
```





5 Fingerprinting and molecular similarity

5.1 click here for a more detailed explanation of the theory behind fingerprints

- In general, molecular substructures are converted into bitmaps that can be compared mathematically for similarity

5.2 Comparing nearest-neighbor similarity between two libraries

- Example use case: would combining two different compound libraries lead to substantial redundancy?
- In the example shown below, each compound fingerprint in one library is compared to each compound fingerprint in another and the highest pair-wise similarity score is recorded for each molecule

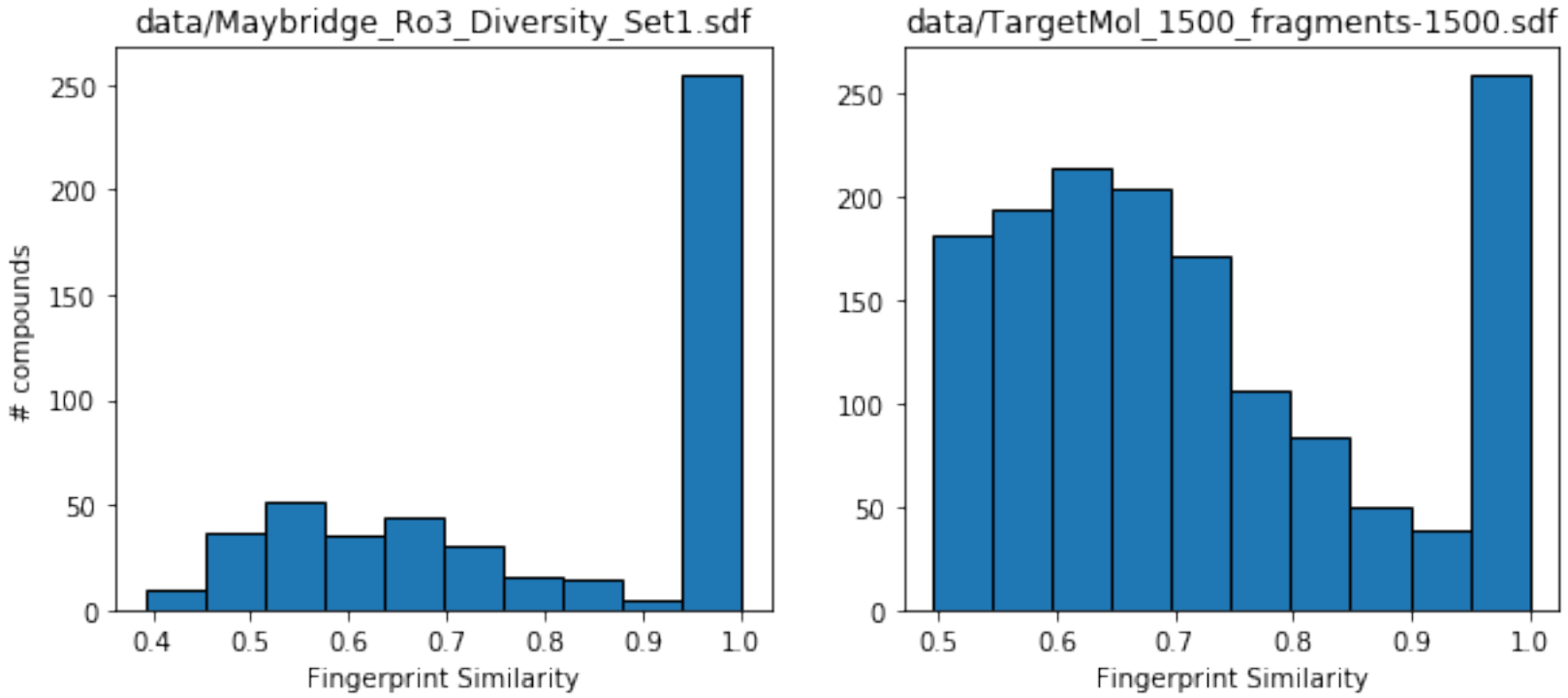
- Here, scoring ranges from 0.0 (no similarity) to 1.0 (structurally identical)
- E.g. if a compound is associated with a score of 1.0, this means the same compound exists in both libraries

```
In [13]: def calc_nearest_neighbors(fpsA, fpsB):
             '''
             Calculate lists of nearest-neighbor similarities between to compound collections
             '''
             molsA = load_library(libraryA)
             molsB = load_library(libraryB)
             fpsA = []
             for x in range(len(molsA)):
                 try:
                     fpsA.append(FingerprintMols.FingerprintMol(molsA[x]))
                 except:
                     pass
             fpsB = []
             for x in range(len(molsB)):
                 try:
                     fpsB.append(FingerprintMols.FingerprintMol(molsB[x]))
                 except:
                     pass
             nnA = np.zeros(len(fpsA))
             for x, tq in zip(range(len(fpsA)), tqdm(range(len(fpsA)), desc='calculating A...')):
                 similarity = 0.
                 for y in range(len(fpsB)):
                     tmp = DataStructs.FingerprintSimilarity(fpsA[x], fpsB[y])
                     if tmp > similarity:
                         similarity = tmp
                     nnA[x] = similarity
             nnB = np.zeros(len(fpsB))
             for x, tq in zip(range(len(fpsB)), tqdm(range(len(fpsB)), desc='calculating B...')):
                 similarity = 0.
                 for y in range(len(fpsA)):
                     tmp = DataStructs.FingerprintSimilarity(fpsB[x], fpsA[y])
```

```
                if tmp > similarity:
                    similarity = tmp
                nnB[x] = similarity
        return nnA, nnB

    def plot_nearest_neighbors(libraryA, libraryB):
        '''
        plot nearest-neighbor similarities between two compound collections
        '''
        nnA, nnB = calc_nearest_neighbors(join('data', libraryA), join('data', libraryB))
        fig, ax = plt.subplots(1, 2, figsize=(10, 4))
        ax[0].hist(nnA, edgecolor='k')
        ax[0].set_title(libraryA)
        ax[0].set_ylabel('# compounds')
        ax[0].set_xlabel('Fingerprint Similarity')
        ax[1].hist(nnB, edgecolor='k')
        ax[1].set_title(libraryB)
        ax[0].set_xlabel('Fingerprint Similarity')
        ax[1].set_xlabel('Fingerprint Similarity')
        plt.show()
        plt.close()
```

In [14]:
```
libraryA = 'data/Maybridge_Ro3_Diversity_Set1.sdf' # Note: can change this to any other  structure file
libraryB = 'data/TargetMol_1500_fragments-1500.sdf' # Note: can change this to any other  structure file
plot_nearest_neighbors(libraryA, libraryB)
```



6 2D depictions of molecular fingerprints:

- The dimensionality of fingerprint bitmaps (here we used 2048-bit Morgan fingerprints) can be reduced using statistical techniques like Principle component analysis (PCA) to allow for visualization and comparison of library diversity.
- Here we fit a pca model on a set of ~348 million fragments (MW $\leq$ 350 g/mol) downloaded from the ZINC database
- The model (loaded here as 'ipca.pkl') is then used to project 2048 bit Morgan fingerprints calculated from compound libraries onto two dimensions for visual comparison of diversity.

```
In [15]: ipca = pickle.load(open('ipca.pkl', 'rb'))
```

```
def decompose_mols(library):
    '''
    calculate Morgan fingerprints for each molecule in a collection and
    reduce 2048 bit fingerprints to 20 principle components using a pre-fit incremental PCA model
    '''
    mols = load_library(library)
    bits = []
    for x in mols:
        try:
            bits.append(AllChem.GetMorganFingerprintAsBitVect(x, 2))
        except:
            pass
    X = np.zeros((len(bits), len(bits[0])))
    for i, b in enumerate(bits):
        X[i] = [int(x) for x in b]
    x = ipca.transform(X)
    return x.T

def plot_pcs(libraryA, libraryB):
    '''
    plot 2D molecular fingerprint depictions to compare two compound collections
    '''
    XA, XB = decompose_mols(libraryA), decompose_mols(libraryB)
    xmin, xmax = np.min([np.min(XA[0]), np.min(XB[0])]), np.max([np.max(XA[0]), np.max(XB[0])])
    ymin, ymax = np.min([np.min(XA[1]), np.min(XB[1])]), np.max([np.max(XA[1]), np.max(XB[1])])
    fig, ax = plt.subplots(1, 2, figsize=(10, 4))
    ax[0].scatter(XA[0], XA[1], c=gaussianKDE(XA[0], XA[1]), cmap='jet', s=5, alpha=0.8)
    ax[0].set_ylabel('PC2')
    ax[0].set_xlabel('PC1')
    ax[0].set_xlim([xmin, xmax])
    ax[0].set_ylim([ymin, ymax])
    ax[0].set_title(libraryA)
```

```
    ax[1].scatter(XB[0], XB[1], c=gaussianKDE(XB[0], XB[1]), cmap='jet', s=5, alpha=0.8)
    ax[1].set_xlabel('PC1')
    ax[1].set_xlim([xmin, xmax])
    ax[1].set_ylim([ymin, ymax])
    ax[1].set_title(libraryB)
    plt.tight_layout()
    plt.show()
    plt.close()
```

In [16]:
```
libraryA = 'data/Zenobia_968.sdf' # Note: can change this to any other  structure file
libraryB = 'data/LifeChemicals_3D_Fragment_Library.sdf' # Note: can change this to any other  structure file
plot_pcs(libraryA, libraryB)
```