DeepPharmacophore

Janani Rajadurai

October 22, 2018

1 Calculate Pharmacophore Fingerprints

In this notebook we use a ML approach to build a classifier of MOR ligands between agonists and antagonsits based on pharmacophore fingerprints. In the first section we create the pharmacophoric fingerprints, and in the second one we setup and fit a deep network to classify the fingerprints.

1.1 Calculate pharmacophore fingerprint from Schrodinger output

In order to be able to run automatic inferences on the pharmacophores, we have to convert the Schrodinger output files to a linear fingerprint that encodes in a standardized format the geometrical features. We do that using the `fp` and `ligand` classes defined below.

In [1]:
```
# libraries
import os
import re

# libraries for math and database management
import numpy as np
import pandas as pd

# libraries for machine learning
import keras
import sklearn.model_selection
from sklearn.preprocessing import LabelBinarizer
from sklearn.utils import class_weight

# liberaries for graphs
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

1.1.1 Define classes for store fingerprints (`fp`) and ligand data (`ligand`)

In [2]:
```
def alltrips(n):
    """choose of 3 out of n objects, order matters"""
```

```
    tt = np.math.factorial(n)/(np.math.factorial(n-3))
    res = [[i,j,k]
           for i in range(n)
           for j in range(n)
           for k in range(n)
           if not i==j and not i==k and not j==k]
    assert len(res)==tt
    return res

def gen_trips(p_types):
    """
    generates all the triples of pharmacophoric
    points taken from the p_types types
    """
    n = len(p_types)
    res = [ [p_types[i], p_types[j], p_types[k]]
           for i in range(n)
           for j in range(n)
           for k in range(n)]
    return res

def get_list(lig_beads, p_type):
    """
    extracts all triplets of given type from a ligand
    """
    n_beads = len(lig_beads)
    trips = alltrips(n_beads)
    res=[]
    for i,tt in enumerate(trips):
        if all(np.array(lig_beads)[tt]==p_type):
            #print(tt)
            res.append(tt)
    return res


def remap1(types,map1):
    r=[]
    if map1:
        for t in types:
            if t in map1.keys():
                r.append(map1[t])
            else:
                r.append(t)
    else:
        r=types
    return r

def binning(dist, dmin, dmax, n_bins):
```

```
        """converts a set of distances into bins
        for a give choice of the binning
        """
        if dist.shape[0]==0:
            return []
        else:
            assert dist.max() <= dmax and dist.min() >= dmin
            return ((n_bins-1)*(dist-dmin)/(dmax-dmin)).astype('int')

    def block(distances,n_bins):
        """takes a set of distances and
        maps it onto a block
        """
        t = np.zeros(shape=(3,n_bins))

        for d in distances:
            for i in range(3):
                t[i,d[i]] += 1./distances.shape[0]
        return np.hstack(t)
```

In [4]:
```
## fp class sets the parameters of the FP

class fp(object):

    def __init__(self, typesR, n_bins, dmin=0, dmax=25, remap=None):

        types=list(set(remap1(typesR,remap)))
        p_types=gen_trips(types)
        print('types: '+','.join(types))

        #n_bins = 10
        dx = (dmax-dmin)/n_bins
        print("bins ->  min:%f, max:%f, n:%d / dx:%f"%(dmin,dmax,n_bins,dx))
        fp_len = len(p_types)*3*n_bins
        print("fingerprint has %d sections"% len(p_types))
        print("fingerptins is %d long"%fp_len)

        self.p_types=p_types
        self.n_bins=n_bins
        self.dmin = dmin
        self.dmax = dmax
        self.fp_len=fp_len
        self.remap=remap


        headers=[]
        for p_type in p_types:
            blockname = '/'.join(p_type)
```

```
            for dist_name in ['d1','d2','d3']:
                for bi in range(n_bins):
                    headers.append("%s_%s_%d"%(blockname,dist_name,bi))

        self.headers=headers

    def binning(self,dist):
        return binning(
            dist,
            dmin = self.dmin,
            dmax = self.dmax,
            n_bins = self.n_bins)

    def block(self, distances):
        return block(distances, self.n_bins)



class ligand(object):

    def __init__(self, filen=None,
                 beads=None,
                 coords=None,
                 name=None,
                 verbose = False):
        if filen==None:
            self.from_data(beads,coords,verbose)
            self.name=None
        else:
            y = np.genfromtxt(filen,dtype=None)
            beads = [x[1].decode("utf-8")  for x in y]
            coords = np.array([[x[2],x[3],x[4]]  for x in y])
            self.name=filen
            self.from_data(beads,coords,verbose)

    def from_data(self,beads,coords,verbose):
        self.beads=beads
        self.coords=coords
        n = len(beads)
        d = np.zeros(shape=(n,n))
        for i in range(n):
            for j in range(i,n):
                d[i,j]=np.sqrt(((coords[i,:]-coords[j,:])**2).sum())
                d[j,i]=d[i,j]
        self.dists=d
        mmin = d[np.triu_indices(n,1)].min()
        mmax = d.max()
        if verbose:
```

```
            print("max: %f, min:%f"%(mmax,mmin))
        self.n = n
        self.mmin = mmin
        self.mmax = mmax

    def get_dists3(self,ats):
        """
        returns d1,d2,d3 for three points
        """
        return np.array([self.dists[ats[0],ats[1]],
                self.dists[ats[1],ats[2]],
                self.dists[ats[0],ats[2]]])

    def get_dists_triplet_types(self,p_types,remap):
        """
        get all the distances for a given triplet type
        """
        ## all the atoms trips that match the given type
        bs = remap1(self.beads,remap)
        trips=get_list(bs,p_types)
        #return trips
        return np.array([self.get_dists3(t) for t in trips])

    ## loop over all ptypes in a set and return the fingerprint

    def get_fp(self,fp1,verbose=False):

        p_types = fp1.p_types

        fp = []
        for p_type in p_types:
            get1_tr = self.get_dists_triplet_types(p_type,fp1.remap)
            f =  fp1.binning(get1_tr)
            fp.append(fp1.block(f))
            if verbose:
                print('/'.join(p_type), block(f))
        return np.hstack(fp)
```

1.2 Read pharmacophores from files and create ligand instances

In this step we use the functions defined above to read the pharmacophore files for each ligand for both the antagonists and the agonists. Then we identify the ligand class (agonist/antogonist), the ligand name, and the progressive conformation number for the ligands with more than one conformation.

```
In [5]: dir1 = 'data/antagonists67Coords/'
        dir2 = 'data/agonists412Coords/'
```

```
        ligs=[]
        for dir in [dir1,dir2]:
            for fname in os.listdir(dir):
                ligs.append(ligand(filen=dir+fname))
        ## collect max, min and types and create fp object

        n_lig = len(ligs)

        b_type = np.unique(
            [b for l in ligs for b in l.beads ])
        b_min= np.min([l.mmin for l in ligs])
        b_max= np.max([l.mmax for l in ligs])

        print("ligands found: %d"%n_lig )

        print("types found:" + ','.join(b_type))
        print("max, min distances: %f,%f"% (b_max,b_min))
```

```
ligands found: 479
types found:A,D,H,N,P,R
max, min distances: 29.115743,0.075799
```

1.2.1 Obtain ligand name, ligand class and ligand number

```
In [169]: def getInfo(lig):
              ligandstring = lig.name.split('/')[-1].split(".")[0]
              r = re.match(u'(.+)_(antagonist|agonist)_([0-9]*)$',ligandstring,re.M)
              if r:
                  return r.groups() + (ligandstring,)
              else:
                  r = re.match(u'(.+)_(antagonist|agonist)',  ligandstring, re.M)
                  if r:
                      return r.groups() + (0,ligandstring,)
                  else:
                      raise


          ligand_IDs = [getInfo(lig) for lig in ligs]
          len(ligand_IDs)
          lID = pd.DataFrame({
              'name':[l[0] for l in ligand_IDs],
              'class':[l[1] for l in ligand_IDs],
              'num': [l[2]  for l in ligand_IDs],
              'string': [l[3] for l in ligand_IDs]})


          lID.sort_values(['class','name','num'])
```

Out[169]:

	name	class	num	\
398	3-HO-PCP	agonist	1	
219	3-HO-PCP	agonist	2	
370	7-Hydroxymitragynine	agonist	1	
162	7-Hydroxymitragynine	agonist	2	
449	AH-7921	agonist	0	
296	Acetyldihydrocodeine	agonist	1	
174	Acetyldihydrocodeine	agonist	2	
154	Acetyldihydrocodeine	agonist	3	
280	Acetylfentanyl	agonist	1	
142	Acetylfentanyl	agonist	2	
151	Acetylfentanyl	agonist	3	
253	Acrylfentanyl	agonist	1	
275	Acrylfentanyl	agonist	2	
264	Acrylfentanyl	agonist	3	
284	Akuammicine	agonist	1	
347	Akuammicine	agonist	2	
289	Akuammicine	agonist	3	
286	Akuammicine	agonist	4	
211	Akuammicine	agonist	5	
144	Akuammicine	agonist	6	
403	Akuammicine	agonist	7	
455	Akuammicine	agonist	8	
404	Alfentanil	agonist	1	
78	Alfentanil	agonist	2	
70	Alfentanil	agonist	3	
328	Anileridine	agonist	1	
473	Anileridine	agonist	2	
326	Anileridine	agonist	3	
150	Apparicine	agonist	1	
338	Apparicine	agonist	2	
..	...	...	..	
9	Methylnaltrexone	antagonist	1	
57	Methylnaltrexone	antagonist	2	
6	Naldemedine	antagonist	1	
63	Naldemedine	antagonist	2	
1	Nalmefene	antagonist	0	
30	Nalodeine	antagonist	1	
62	Nalodeine	antagonist	2	
41	Nalodeine	antagonist	3	
8	Nalodeine	antagonist	4	
14	Nalorphine	antagonist	0	
49	Nalorphine_dinicotinate	antagonist	0	
36	Naloxazone	antagonist	1	
21	Naloxazone	antagonist	2	
40	Naloxazone	antagonist	3	
17	Naloxazone	antagonist	4	
28	Naloxegol	antagonist	0	

```
26              Naloxonazine  antagonist   0
11                  Naloxone  antagonist   0
4                 Naltrexone  antagonist   0
24                Naltrindole  antagonist   0
39                Naringenin  antagonist   0
19                Noribogaine  antagonist   0
59                Oxilorphan  antagonist   0
29                Pawhuskin_A  antagonist   0
64                  Picenadol  antagonist   0
45                Quadazocine  antagonist   1
37                Quadazocine  antagonist   2
51                 Rimonabant  antagonist   0
20                Samidorphan  antagonist   0
50                  Taxifolin  antagonist   0

                                    string
398                    3-HO-PCP_agonist_1
219                    3-HO-PCP_agonist_2
370      7-Hydroxymitragynine_agonist_1
162      7-Hydroxymitragynine_agonist_2
449                     AH-7921_agonist
296      Acetyldihydrocodeine_agonist_1
174      Acetyldihydrocodeine_agonist_2
154      Acetyldihydrocodeine_agonist_3
280            Acetylfentanyl_agonist_1
142            Acetylfentanyl_agonist_2
151            Acetylfentanyl_agonist_3
253             Acrylfentanyl_agonist_1
275             Acrylfentanyl_agonist_2
264             Acrylfentanyl_agonist_3
284               Akuammicine_agonist_1
347               Akuammicine_agonist_2
289               Akuammicine_agonist_3
286               Akuammicine_agonist_4
211               Akuammicine_agonist_5
144               Akuammicine_agonist_6
403               Akuammicine_agonist_7
455               Akuammicine_agonist_8
404                Alfentanil_agonist_1
78                 Alfentanil_agonist_2
70                 Alfentanil_agonist_3
328               Anileridine_agonist_1
473               Anileridine_agonist_2
326               Anileridine_agonist_3
150                Apparicine_agonist_1
338                Apparicine_agonist_2
..                                  ...
9         Methylnaltrexone_antagonist_1
```

```
57        Methylnaltrexone_antagonist_2
6              Naldemedine_antagonist_1
63             Naldemedine_antagonist_2
1                  Nalmefene_antagonist
30               Nalodeine_antagonist_1
62               Nalodeine_antagonist_2
41               Nalodeine_antagonist_3
8                Nalodeine_antagonist_4
14                Nalorphine_antagonist
49   Nalorphine_dinicotinate_antagonist
36              Naloxazone_antagonist_1
21              Naloxazone_antagonist_2
40              Naloxazone_antagonist_3
17              Naloxazone_antagonist_4
28                 Naloxegol_antagonist
26              Naloxonazine_antagonist
11                  Naloxone_antagonist
4                 Naltrexone_antagonist
24               Naltrindole_antagonist
39                Naringenin_antagonist
19               Noribogaine_antagonist
59                Oxilorphan_antagonist
29               Pawhuskin_A_antagonist
64                 Picenadol_antagonist
45             Quadazocine_antagonist_1
37             Quadazocine_antagonist_2
51                Rimonabant_antagonist
20               Samidorphan_antagonist
50                  Taxifolin_antagonist

[479 rows x 4 columns]
```

1.3 Calculate fingerprints

Using the ligand objects created above, we generate a linear pharmacophoric fingerprint. To reduce the size of the fingerprint we define a simplified set of pharmacophoric points mapping the positive and negative beads (`P/N`) to a common "charged" bead (`c`). We also define the distance ranges so that all distances are binned in 5 bins. Finally we check how long are the fingerprints.

```
In [7]: ## define fp object
        ## map N,P -> c A,D -> hb
        #charged_hb = {'N':'c','P':'c','A':'hb','D':'hb'}
        charged_ = {'N':'c','P':'c'}

        fp2 = fp(b_type, n_bins= 5,
                 dmin = int(b_min),
                 dmax = 1+int(b_max),
                 remap=charged_)
```

```
types: c,D,R,A,H
bins ->  min:0.000000, max:30.000000, n:5 / dx:6.000000
fingerprint has 125 sections
fingerptins is 1875 long
```

```
In [ ]: ## calculate all fps
        all_fps = []
        for i,l in enumerate(ligs):
            if i % 10 == 0:
                #print("%f done"% (float(i)/len(ligs)))
                pass
            all_fps.append(l.get_fp(fp2))
```

1.4 Remove trivial bits

We remove the fingerprint columns that have null variance (i.e. that have the same value for all ligands). these are not useful in explaining the differences.

```
In [9]: trivials = np.array(all_fps).var(axis=0)==0
        print("trivial bits %d"%sum(trivials))
        np.array(fp2.headers)[~trivials]
```

```
trivial bits 633
```

```
Out[9]: array(['c/c/c_d1_0', 'c/c/c_d1_1', 'c/c/c_d2_0', ..., 'H/H/H_d3_0',
               'H/H/H_d3_1', 'H/H/H_d3_2'],
              dtype='<U10')
```

```
In [170]: fp = pd.DataFrame(np.array(all_fps)[:,~trivials])
          fp.columns = np.array(fp2.headers)[~trivials]

          # print out the first fingerprints
          #fp.head();
          #lID.head()
```

1.5 Define the final dataset

We create a `pandas` database with the fingerprints and the ligand information; we also use the `groupby` function to count how many fingerprints (i.e. conformations) we have for each ligand.

```
In [12]: data = pd.concat([lID, fp], axis=1, sort=False)
         d2 = data.sort_values(['class','name','num'])
         d2.index = d2['string']
```

```
In [188]: d2.groupby(['class','name']).size()
```

Out[188]:

class	name	
agonist	3-HO-PCP	2
	7-Hydroxymitragynine	2
	AH-7921	1
	Acetyldihydrocodeine	3
	Acetylfentanyl	3
	Acrylfentanyl	3
	Akuammicine	8
	Alfentanil	3
	Anileridine	3
	Apparicine	3
	BW373U86	8
	Benzhydrocodone	3
	Benzylmorphine	3
	Bezitramide	3
	Biphalin	3
	Buprenorphine	2
	Butorphanol	3
	Butyrfentanyl	3
	Carfentanil	3
	Casokefamide	2
	Cebranopadol	2
	Chlornaltrexamine	5
	Chloroxymorphamine	12
	Codeine	3
	DADLE	2
	Dermorphin	2
	Desmetramadol	4
	Desomorphine	5
	Dextromoramide	2
	Dextropropoxyphene	1
		..
antagonist	Epicatechin_gallate	1
	Eptazocine	1
	Gallocatechol	1
	Gemazocine	2
	Hyperoside	1
	Ibogaine	1
	LY-255582	1
	Levallorphan	1
	Lobeline	1
	Methylnaltrexone	2
	Naldemedine	2
	Nalmefene	1
	Nalodeine	4
	Nalorphine	1
	Nalorphine_dinicotinate	1
	Naloxazone	4

```
                    Naloxegol                   1
                    Naloxonazine                1
                    Naloxone                    1
                    Naltrexone                  1
                    Naltrindole                 1
                    Naringenin                  1
                    Noribogaine                 1
                    Oxilorphan                  1
                    Pawhuskin_A                 1
                    Picenadol                   1
                    Quadazocine                 2
                    Rimonabant                  1
                    Samidorphan                 1
                    Taxifolin                   1
Length: 168, dtype: int64
```

1.6 Load weights and average fingerprints

We now use the weigths of each conformation to average the different conformations. First we read the weights from external files, and add them to the database, then use them to perform the average

```
In [189]: weights=[]
          for file in [
              "data/pharmacophore_scoringPenaltyWeightsAntagonists.csv",
              "data/pharmacophore_scoringPenaltyWeightsAgonists.csv"]:

              weights1=pd.read_csv(
                  file,
                  delimiter=' ',
                  header=None,
                  skipinitialspace=True,
                  names=["ligand_filename","weight"])

              weights.append(weights1)


          w = pd.concat(weights,axis=0)
          l_string = [l.split(".")[0] for l in w['ligand_filename']]
          w.index=l_string
          # show the first 30 weights
          w['weight'].head(30)

Out[189]: 18-Methoxycoronaridine_antagonist    0.0190
          Diacetylnalorphine_antagonist_4      0.0000
          4-Caffeoyl-1,5-quinide_antagonist    0.0027
          4,7-Dihydroxyflavone_antagonist      0.0117
          6beta-Naltrexol_antagonist           0.0304
```

```
6beta-Naltrexol-d4_antagonist        2.5582
Akuammine_antagonist                 0.2067
Alvimopan_antagonist                 0.0048
AM-251_antagonist                    0.0000
Apigenin_antagonist                  0.0397
AT-076_antagonist                    0.1536
Axelopran_antagonist_2               0.0000
Axelopran_antagonist_3               0.0000
Axelopran_antagonist_4               0.0000
Beta-Funaltrexamine_antagonist       0.0351
Bevenopran_antagonist                0.0064
Catechin_antagonist_1                0.0000
Catechin_antagonist_2                0.0000
Catechin_antagonist_3                0.0000
Catechin_antagonist_4                0.0000
Chlornaltrexamine_antagonist         0.0282
Clocinnamox_antagonist               0.1414
Cyprodime_antagonist                 0.0713
Diacetylnalorphine_antagonist_1      0.3078
Diacetylnalorphine_antagonist_2      0.3078
Diacetylnalorphine_antagonist_3      0.5352
Diprenorphine_antagonist             0.0000
Epicatechin_gallate_antagonist       0.0725
Eptazocine_antagonist                0.0000
Gallocatechol_antagonist             0.0068
Name: weight, dtype: float64
```

In [16]:
```
## add the weights to the database
d4 = d2.join(w)

# define the weigthed average and perform the average by grouping over the ligand name
wm = lambda x: np.average(x, weights=np.exp(d4.loc[x.index, "weight"]))
d5 = d4.groupby(['class','name']).agg(wm)
```

In [177]:
```
# as an example, show the first 5 columns for the first 15 ligands
d5[d5.columns[0:5]].head(15)
```

Out[177]:

class	name	A/A/A_d1_0	A/A/A_d1_1	A/A/A_d1_2	A/A/A_d1_3	\
agonist	3-HO-PCP	0.000000	0.000000	0.000000	0.0	
	7-Hydroxymitragynine	0.622222	0.377778	0.000000	0.0	
	AH-7921	0.000000	0.000000	0.000000	0.0	
	Acetyldihydrocodeine	1.000000	0.000000	0.000000	0.0	
	Acetylfentanyl	0.000000	0.000000	0.000000	0.0	
	Acrylfentanyl	0.000000	0.000000	0.000000	0.0	
	Akuammicine	1.000000	0.000000	0.000000	0.0	
	Alfentanil	0.542407	0.457593	0.000000	0.0	
	Anileridine	1.000000	0.000000	0.000000	0.0	

	Apparicine	0.000000	0.000000	0.000000	0.0
	BW373U86	0.369192	0.630808	0.000000	0.0
	Benzhydrocodone	0.964286	0.035714	0.000000	0.0
	Benzylmorphine	1.000000	0.000000	0.000000	0.0
	Bezitramide	0.654653	0.272674	0.072674	0.0
	Biphalin	0.444001	0.475276	0.080723	0.0

		A/A/A_d2_0
class	name	
agonist	3-HO-PCP	0.000000
	7-Hydroxymitragynine	0.622222
	AH-7921	0.000000
	Acetyldihydrocodeine	1.000000
	Acetylfentanyl	0.000000
	Acrylfentanyl	0.000000
	Akuammicine	1.000000
	Alfentanil	0.542407
	Anileridine	1.000000
	Apparicine	0.000000
	BW373U86	0.369192
	Benzhydrocodone	0.964286
	Benzylmorphine	1.000000
	Bezitramide	0.654653
	Biphalin	0.444001

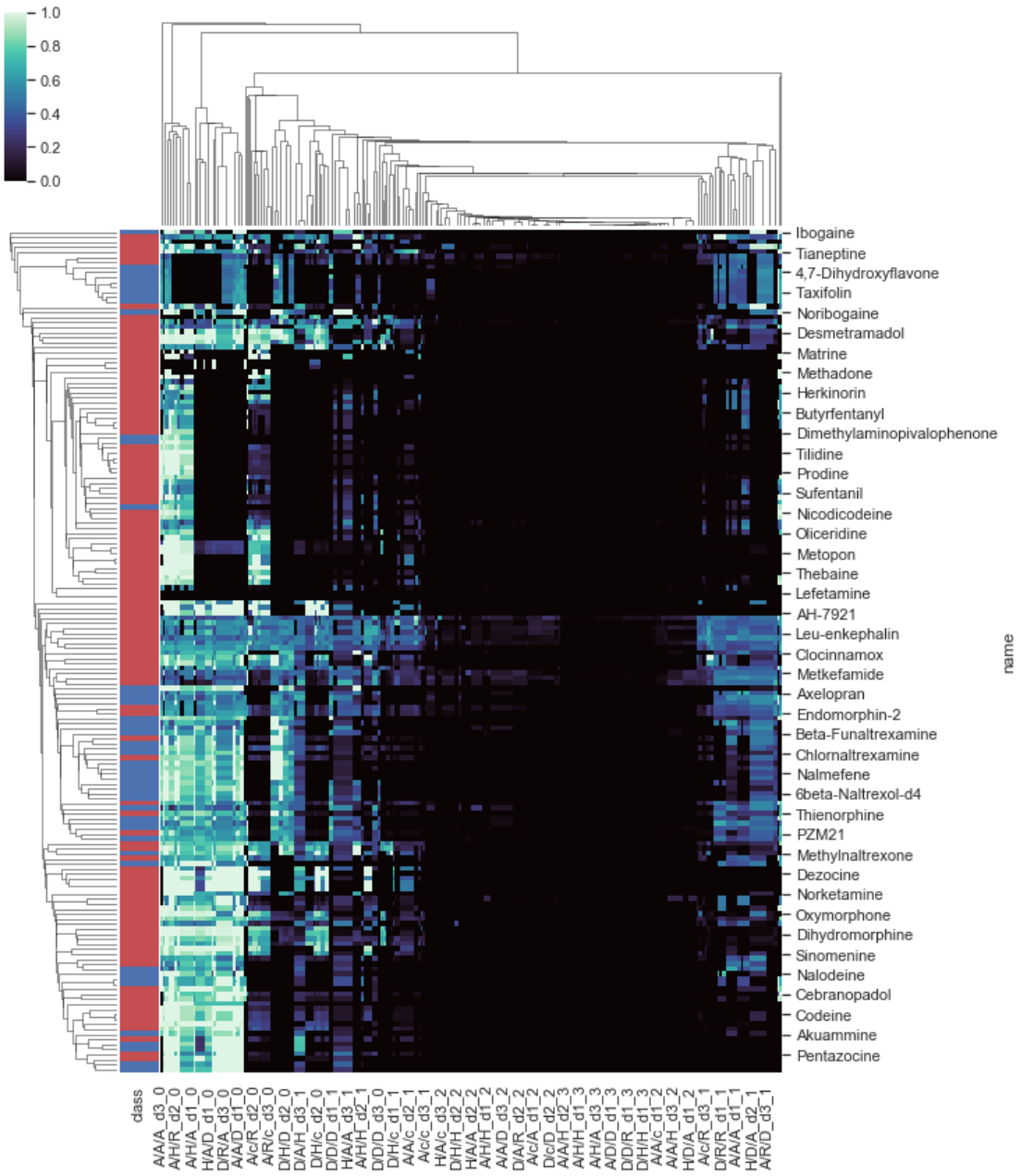
2 Machine learning of the pharmacophores

we are now ready to set up the deep learning of the fingerprints. First we inspect the clustering of the fingerprints to see if there is any evident visual separation. We plot the fingerprints as a heatmap (light = populated distance bin, dark = unpopulated distance bin). We also indicate on the plot the class of the ligand (red=agonist, blue=antagonist).

2.1 Plot fingerprint heatmap

In [179]:
```
import seaborn as sns;
sns.set(color_codes=True)

d6b = d6
d7 = d6b[regressors_col[0:600]]
d7.index = d6b['name']

classes = d6b['class']
lut = dict(zip(classes.unique(), "rbg"))

row_colors = classes.map(lut)
row_colors.index = d6b['name']
g = sns.clustermap(
```

```
        d7,
        method="single",
        row_colors=row_colors,
        figsize=(10, 14),
        cmap="mako")

g.savefig('heatmap.pdf')
```



2.2 Prepare data for network

We prepare the data for the netowrk. Convert the class (ago/antagonist) to a binary value `y`, and define the input `x` using the fingerprint bins. We split the dataset into train and validation

```
In [20]: regressors_col = [c for c in d5.columns if not (c=='weight' or c=='name')]
         d6 = d5.reset_index()

         ## we define the input (x) and output (y) for the learning
         x = np.array(d6[regressors_col],dtype='float64')
         x1 = sklearn.preprocessing.scale(np.array(x,dtype='float64'))


         encoder = LabelBinarizer()
         y = encoder.fit_transform(d6['class'])

In [22]: x_train, x_valid, y_train, y_valid = sklearn.model_selection.train_test_split(
             x1,
             y,
             test_size=0.3,
             random_state=0)
```

2.3 Create network

Now we create the neural network to do the learning, using the `keras` library. The input layer has the same number of nodes as the fingerprints, then we add a layer with 32 nodes, connected with the `relu` activation function, followed by two layers with 16 nodes, and the one output layer with one single node. this node will contain the probability that the input fingerprint corresponds to an antagonist (since we encoded the antagonist class with 1 and the agonist with 0). Then we run the learning step, by calling the `model.fit` method on the training data.

```
In [68]: def create_model(x_train):

             model = keras.models.Sequential([
                 keras.layers.normalization.BatchNormalization(
                     input_shape=tuple([x_train.shape[1]])),
                 keras.layers.core.Dense(32, activation='relu'),
                 keras.layers.core.Dense(16, activation='relu'),
                 keras.layers.core.Dense(16, activation='relu'),
                 keras.layers.core.Dense(1,  activation='sigmoid')
             ])

             model.compile(keras.optimizers.Adam(lr=0.001),
                       loss='binary_crossentropy',
                       metrics=["accuracy"])

             #print(model.summary())
             return model
```

```
# to run single
callback_early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=20,
    verbose=0,
    mode='auto')

model = create_model(x_train)
model.fit(x_train, y_train,
          batch_size=20,
          epochs=500,
          validation_data=(x_valid, y_valid),
          verbose=0,
          callbacks=[callback_early_stopping])
```

```
Out[68]: <keras.callbacks.History at 0x7f925c381978>
```

2.4 Check network predictions

We use the network to see how our predictions would match the real classes. We see that on the complete set, we misclassify 6 ligands, and correctly classify 116+46=162 ligands, so that the error is only 3%. We print a table with the misclassified ligands.

```
In [181]: ## calculate the predictions

          x_test = x1
          y_test = model.predict(x_test)
          preds = pd.DataFrame({
              'name':d6['name'],
              'real_class':d6['class'],
              'pred_class': ["agonist" if y_<0.5 else "antagonist" for y_ in y_test[:,0]],
              'prob_pred': y_test[:,0]
          })

          preds['misclass'] = -(preds['real_class'] == preds['pred_class'])
          preds.groupby(['misclass','real_class', 'pred_class']).size()
```

```
Out[181]: misclass  real_class  pred_class
          False     agonist     agonist       116
                    antagonist  antagonist     46
          True      agonist     antagonist      3
                    antagonist  agonist         3
          dtype: int64
```

```
In [156]: preds[preds['misclass']]
```

```
Out[156]:                         name  real_class  pred_class  prob_pred  misclass
          16                Butorphanol     agonist  antagonist   0.887866      True
          39                Embutramide     agonist  antagonist   0.583251      True
```

```
74   Mitragynine_pseudoindoxyl     agonist  antagonist   0.729355      True
135                  Cyprodime  antagonist     agonist   0.001739      True
136         Diacetylnalorphine  antagonist     agonist   0.005922      True
152    Nalorphine_dinicotinate  antagonist     agonist   0.006000      True
```

```
In [168]: g2 = sns.catplot(x="real_class", y="prob_pred", hue="misclass", data=preds);
          g2.savefig('prediction.pdf')
```

The figure above shows the predicted probablity to be an antagonist for the two classes. the probablities for agonists are very small, as expected, and the ones for the antagonists are big. the six misclassified ligands are reported in yellow.