CNN Documentation

Release 0.1

Jan-Oliver Joswig

Dec 03, 2018

CONTENTS:

CHAPTER

ONE

INSTALLATION

Get the source:

git clone https://github.com/janjoswig/CNN.git

To use the cnn module add it to your PYTHONPATH:

export PYTHONPATH=$PYTHONPATH:$YOUR_DIR/CNN/cnn

To use the executables:

export PATH=$PATH:$YOUR_DIR/CNN/bin

or create a symlink, e.g.

ln -s $YOUR_DIR/CNN/bin/cnn $YOUR_FAVORITE_DIR/cnn

CHAPTER

TWO

GETTING STARTED

The cnn module provides a data set based API for common-nearest-neighbour clustering. The core functionality is bundled in a class `cnn.CNN()`. Here is a minimal example of its usage:

```
>>> import cnn
...
>>> cobj = cnn.CNN()
>>> cobj.load('path_to_data/data')
>>> cobj.fit()
```

After importing the main module, this creates an instance of the cnn cluster class. Data is loaded into the object and fitted, i.e. clustered by the cnn algorithm.

Let's go a bit deeper. Call of a cluster objects `__str__()` gives us an overview of its properties.

```
>>> cobj = cnn.CNN()
>>> print(cobj)
cnn.CNN() cluster object
alias :                                 root
hierachy level:                         0
test data loaded :                      False
test data shape :                       None
train data loaded :                     False
train data shape :                      None
distance matrix calculated (train):     False
distance matrix calculated (test):      False
clustered :                             False
children :                              False
```

A freshly created cluster object is by default called *root*. It has the highest possible hierarchy level 0 (more on this later). No data is present as indicated by *data loaded : False* and nothing has been done so far. Data is either treated as *test* or *train* to allow for clustering on one set and interpolation on another. We will see how, but first we need some data:

```
>>> from sklearn import datasets
>>> from sklearn.preprocessing import StandardScaler
...
>>> blobs, _ = datasets.make_blobs(n_samples=10000,
...                                cluster_std=[1.0, 2.1, 0.25],
...                                random_state=1)
>>> blobs = StandardScaler().fit_transform(blobs)
>>> cobj = cnn.CNN(test=blobs)
```

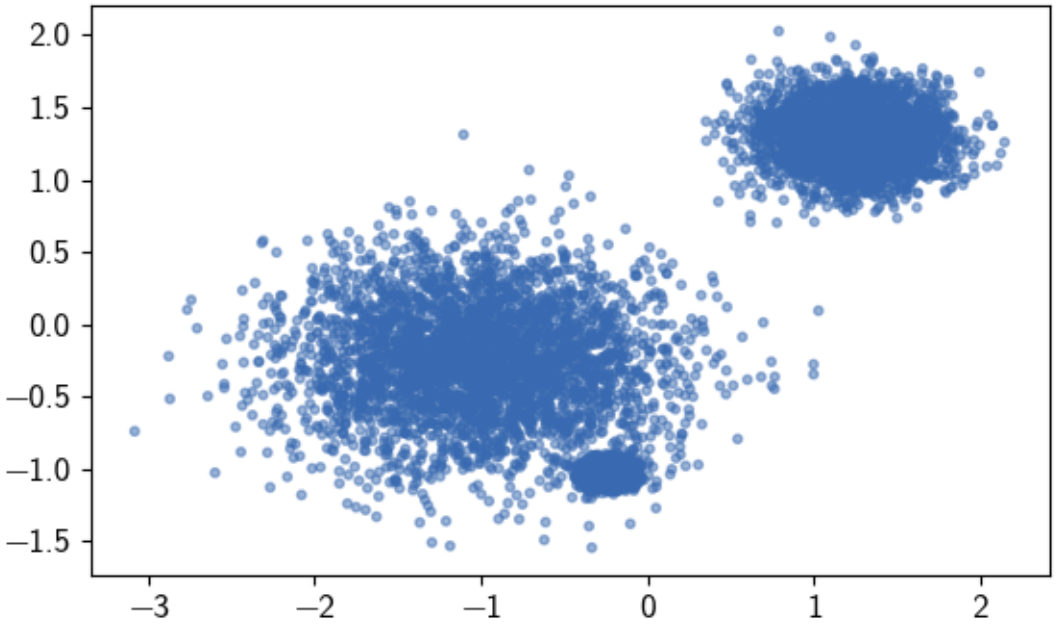
When we now look at our object again, its properties have changed:

```
>>> print(cobj)
cnn.CNN() cluster object
alias :                                 root
hierachy level:                         0
test data loaded :                      True
test data shape :                       {'parts': 1, 'points': [10000], 'dimensions':␣
↪2}
train data loaded :                     False
train data shape :                      None
distance matrix calculated (train):     False
distance matrix calculated (test):      False
clustered :                             False
children :                              False
```

Data is now present as *test*. All data is processed as numpy array of the form (*parts* x *points* x *dimensions*). Data can be passed to the CNN() class as 1-D (only one part, points in only one dimension) or 2-D (one part, points in *n* dimensions) array-like structure, but will be processed internally in this general shape. In the *shape* dictionary, `cobj.test_shape`, `'points'` is associated with a list of data points per part.

We can get an impression of the loaded data by plotting the points.

```
>>> cobj.evaluate(mode='test')
```



Let's reduce the large *test* data set to make the clustering faster.

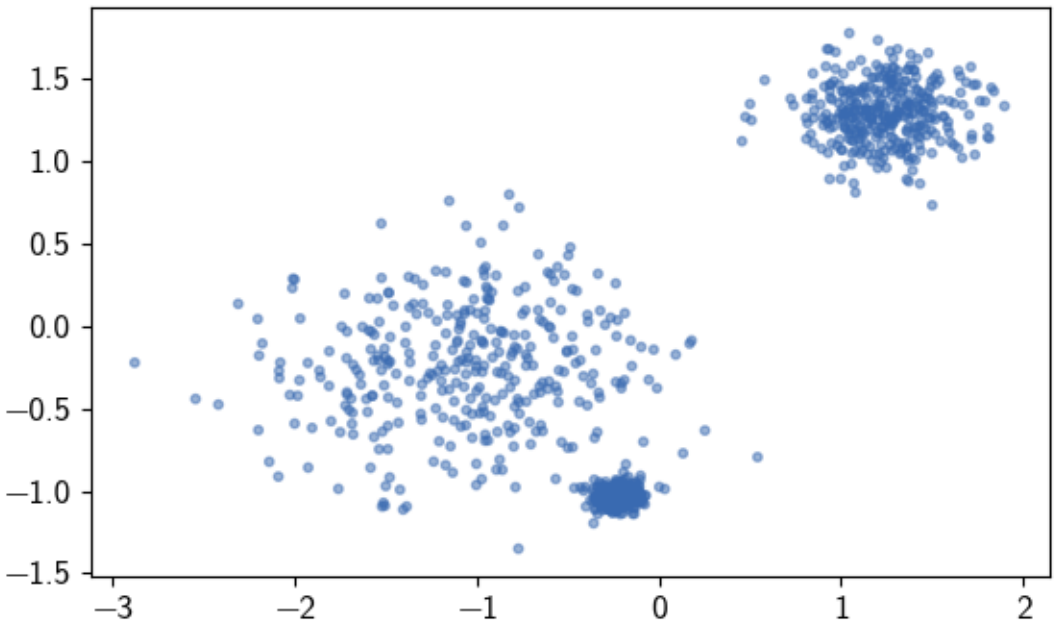
```
>>> cobj.cut(points=(None, None, 10))
...
>>> print(cobj)
cnn.CNN() cluster object
alias :                                 root
hierachy level:                         0
test data loaded :                      True
```

(continues on next page)

(continued from previous page)

```
test data shape :                          {'parts': 1, 'points': [10000], 'dimensions':␣
↪2}
train data loaded :                        True
train data shape :                         {'parts': 1, 'points': [1000], 'dimensions':␣
↪2}
distance matrix calculated (train):        False
distance matrix calculated (test):         False
clustered :                                False
children :                                 False
```

```
>>> cobj.evaluate()
```



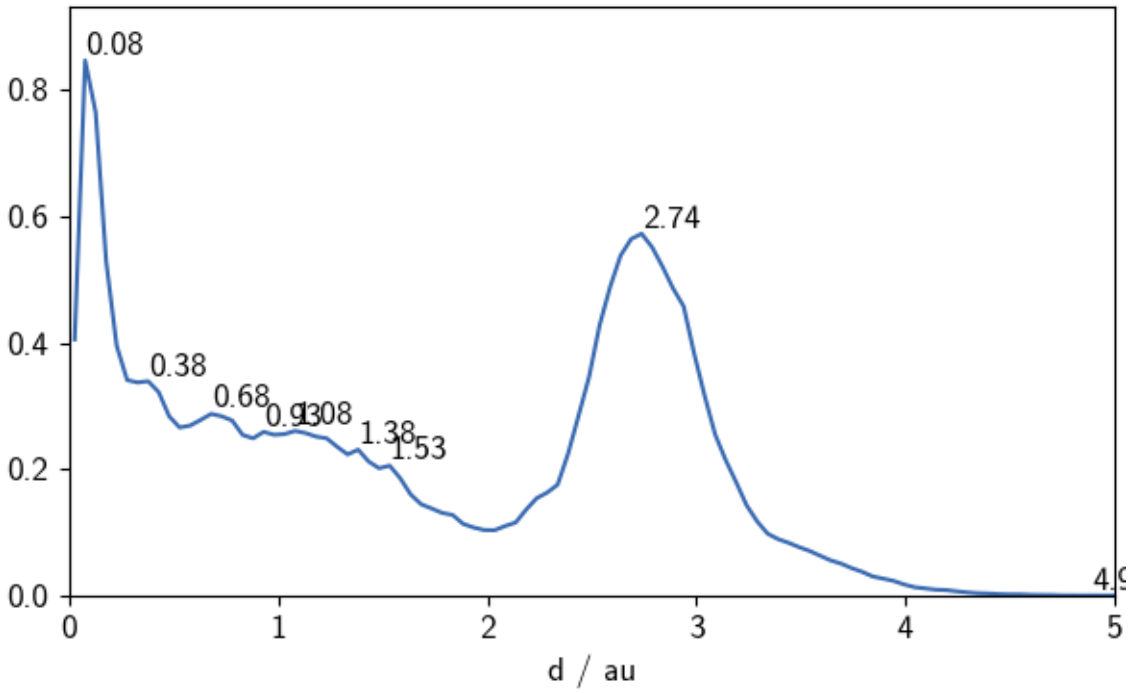
Plotting the pairwise distance distribution of the data set can be useful, too. Multiple peaks in this distribution hint to the presence of more than one cluster and the location of the peaks can help in finding an appropriate `radius_cutoff` to start with. If we want to separate a cluster from the bulk, the `radius_cutoff` needs to be smaller than the maximum of the peak associated with it.

```
>>> cobj.dist_hist(maxima=True)
Train distance matrix not calculated. Calculating distance matrix.
Calculating nxn distance matrix for 1000 points
Execution time for call of dist(): 0 hours, 0 minutes, 0.0165 seconds
```

So let's try fitting our data to some parameters.

```
>>> cobj.fit(radius_cutoff=1, cnn_cutoff=20)
Execution time for call of fit(): 0 hours, 0 minutes, 0.2588 seconds
recording: ...
points               1000
radius_cutoff           1
cnn_cutoff             20
member_cutoff           1
```

(continues on next page)



(continued from previous page)

```
max_clusters          None
n_clusters               2
largest              0.658
noise                0.001
time              0.258785
dtype: object
```

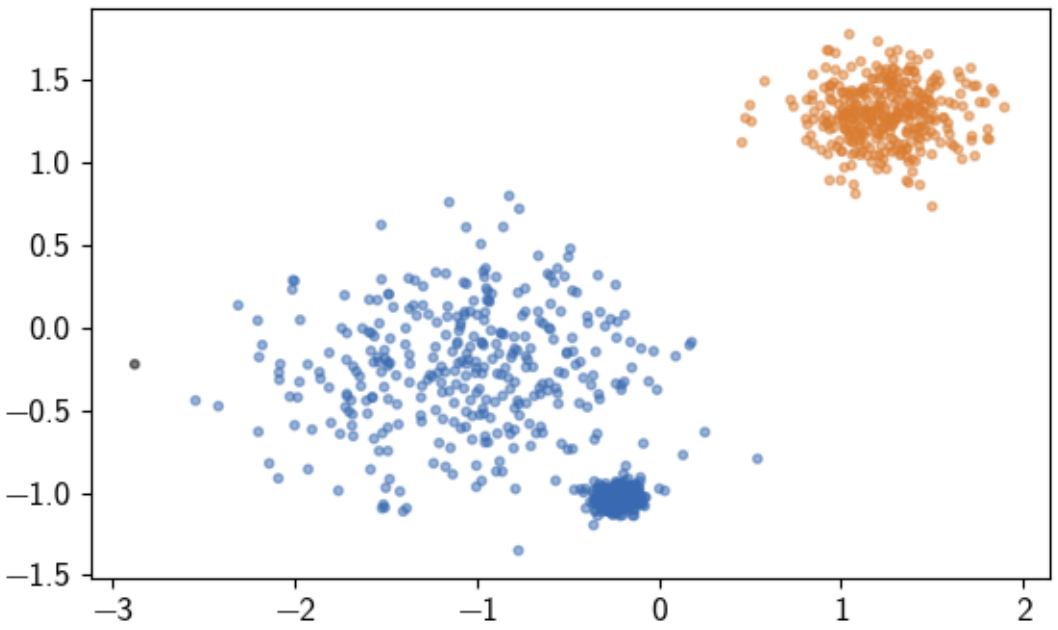
```
>>> cobj.evaluate()
```

All calls of `cobj.fit()` are recorded and stored in a pandas data frame.

```
>>> cobj.summary.sort_values('n_clusters')
  points radius_cutoff cnn_cutoff member_cutoff max_clusters n_clusters  largest␣
↪noise      time
0   1000             2         10             1         None          1    1.000  0.
↪000  0.374373
1   1000             2         20             1         None          1    1.000  0.
↪000  0.365650
2   1000           1.5         20             1         None          1    1.000  0.
↪000  0.362920
3   1000             1         20             1         None          2    0.658  0.
↪001  0.258785
```

The cluster result itself is stored in two instance variables, `cobj.train_labels` (cluster label assignments for each point) and `cobj.train_clusterdict` (points associated to cluster label keys). Noise points are labeled by 0.

```
>>> print(f"Cluster labels: {cobj.train_labels[:10]}, \
... Shape: {np.shape(cobj.train_labels)}, \
... Type: {type(cobj.train_labels)}")
Cluster labels: [2 2 1 1 2 1 2 1 1 1], Shape: (1000,), Type: <class 'numpy.ndarray'>
```




```
>>> print(f"Cluster dictionary: {cobj.train_clusterdict.keys()}, \
... Shape: {[len(x) for x in cobj.train_clusterdict.values()]}, \
... Type: {type(cobj.train_clusterdict)}")
Cluster dictionary: dict_keys([0, 1, 2]), Shape: [1, 658, 341], Type: <class
↪'collections.defaultdict'>
```

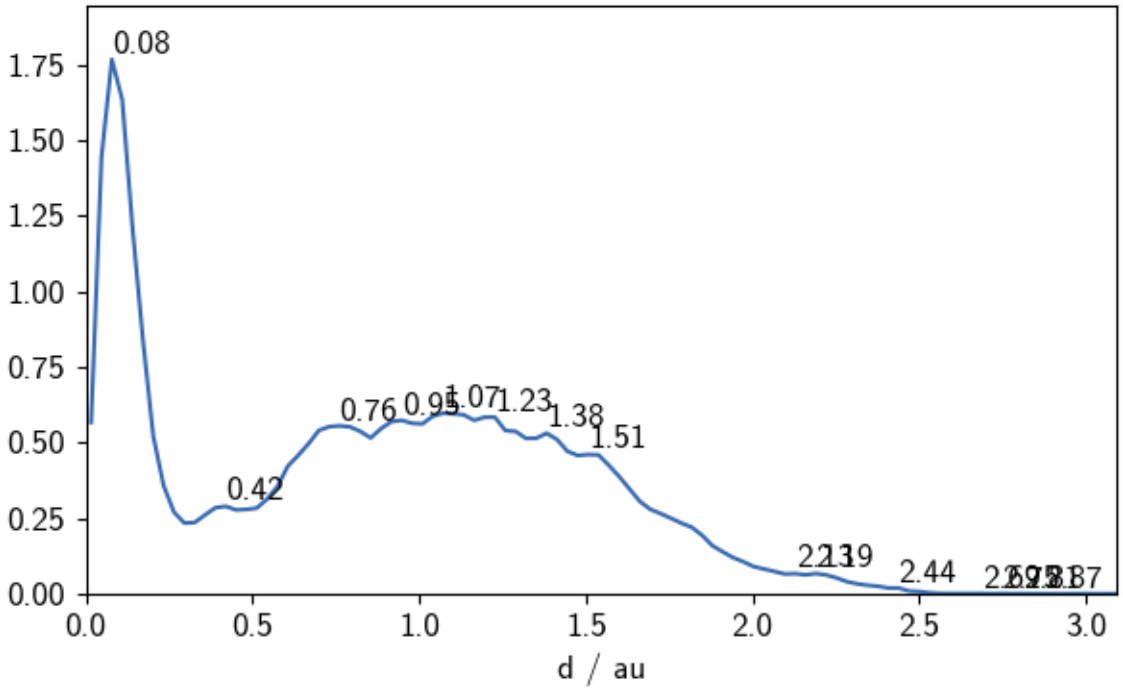
This first fit devided the data set into two clusters. As can be clearly seen from the evaluation above, the blue cluster (label 1) could be further splitted. Before we attempt this, we need to isolate the clusters found, i.e. we create a new cluster object for each one of them. These *child* objects of our *root* data set are stored in a dictionary `cobj.train_children`.

```
>>> cobj.isolate()
>>> cobj.train_children
defaultdict(<function cnn.CNN.isolate.<locals>.<lambda>()>,
            {0: <cnn.CNNChild at 0x7f1397bdf470>,
             1: <cnn.CNNChild at 0x7f1397bc2cf8>,
             2: <cnn.CNNChild at 0x7f1397b58940>})
>>> print(cobj.train_children[1])
cnn.CNN() cluster object
alias :                                 child No. 1
hierachy level:                         1
test data loaded :                      False
test data shape :                       None
train data loaded :                     True
train data shape :                      {'parts': 1, 'points': [658], 'dimensions': 2}
distance matrix calculated (train):     False
distance matrix calculated (test):      False
clustered :                             False
children :                              False
```

A child cluster class instance of cnn.CNNChild() is a fully functional cluster object itself. New, as shown above, is here that the hierarchy level was incremented by one. We can now look at the distance distribution of the data subset

in *child No. 1*.

```
>>> cobj.train_children[1].dist_hist(maxima=True)
Train distance matrix not calculated. Calculating distance matrix.
Calculating nxn distance matrix for 658 points
Execution time for call of dist(): 0 hours, 0 minutes, 0.0073 seconds
```



And we can fit with adjusted parameters.

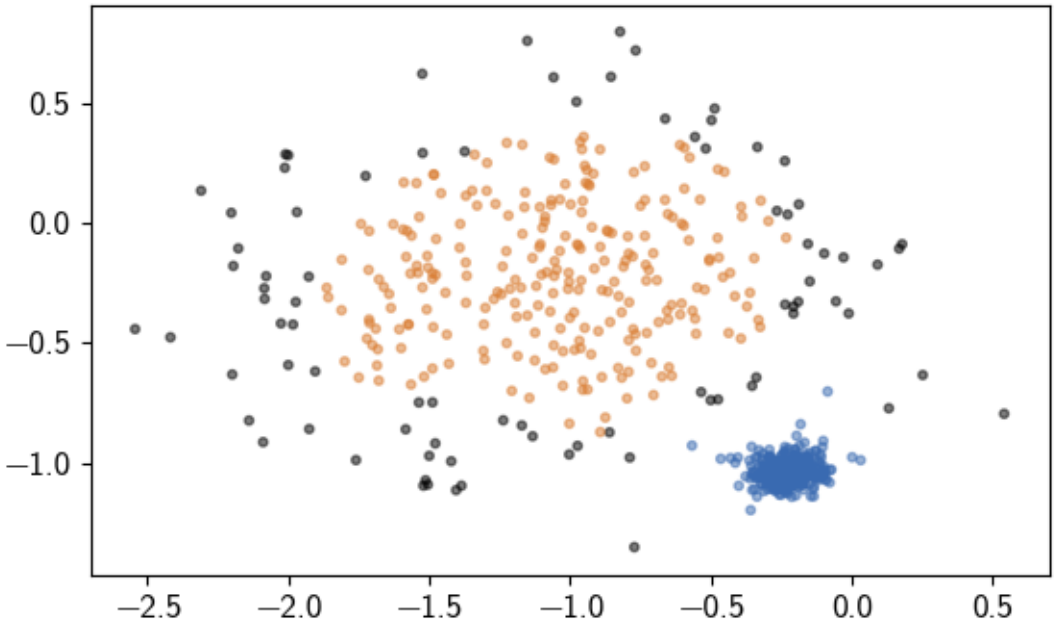
```
>>> cobj.train_children[1].fit(radius_cutoff=0.3,
                               cnn_cutoff=20,
                               member_cutoff=5)
Execution time for call of fit(): 0 hours, 0 minutes, 0.1330 seconds
recording: ...
points                 658
radius_cutoff          0.3
cnn_cutoff              20
member_cutoff            5
max_clusters          None
n_clusters               2
largest                0.5
noise              0.12766
time              0.132971
dtype: object
```

```
>>> cobj.evaluate()
```

When we are satisfied by the outcome, putting everything back together is easy.

```
>>> cobj.train_children[1].train_clusterdict.keys()
... dict_keys([0, 1, 2])
>>> cobj.train_clusterdict.keys()
```

(continues on next page)



(continued from previous page)

```
... dict_keys([0, 1, 2])
>>> cobj.reel()
>>> cobj.train_clusterdict.keys()
... dict_keys([0, 1, 2, 3])
>>> cobj.evaluate()
```

Lastly we want to map the larger *test* data set onto our result. While this is possible for all clusters at once, it can be nice to predict the assignement of *test* points to the *train* clusters for each set using individual parameters.

```
>>> cobj.predict(radius_cutoff=0.9, cnn_cutoff=22, cluster=[1])
Predicting cluster for point  10000 of 10000
Execution time for call of predict(): 0 hours, 0 minutes, 77.2176 seconds
>>> cobj.evaluate(mode='test')
```

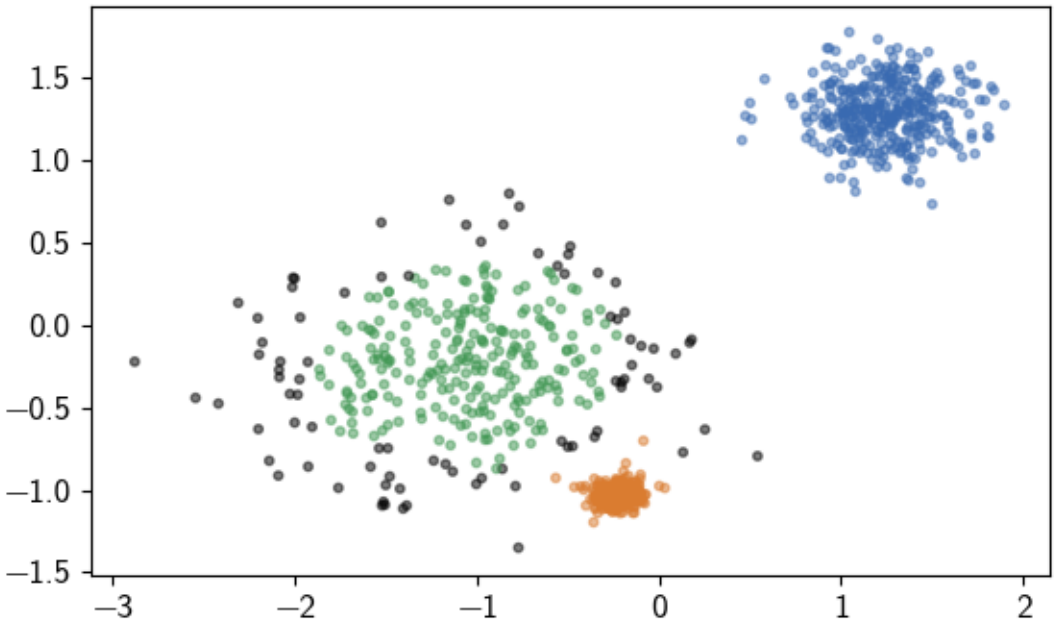
```
>>> cobj.predict(radius_cutoff=0.25, cnn_cutoff=22, cluster=[2])
Predicting cluster for point  10000 of 10000
>>> cobj.predict(radius_cutoff=0.4, cnn_cutoff=22, cluster=[3])
Predicting cluster for point  10000 of 10000
>>> cobj.evaluate(mode='test')
```

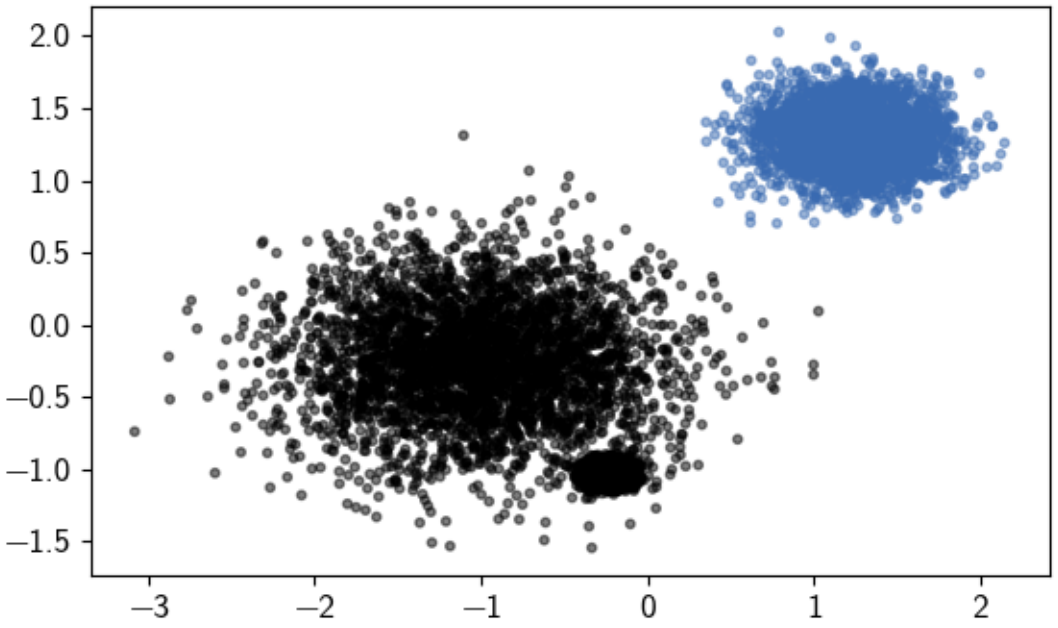
The predicted cluster result is then stored in the complementary instance variables `cobj.test_labels` and `cobj.test_clusterdict`. Et voilà!

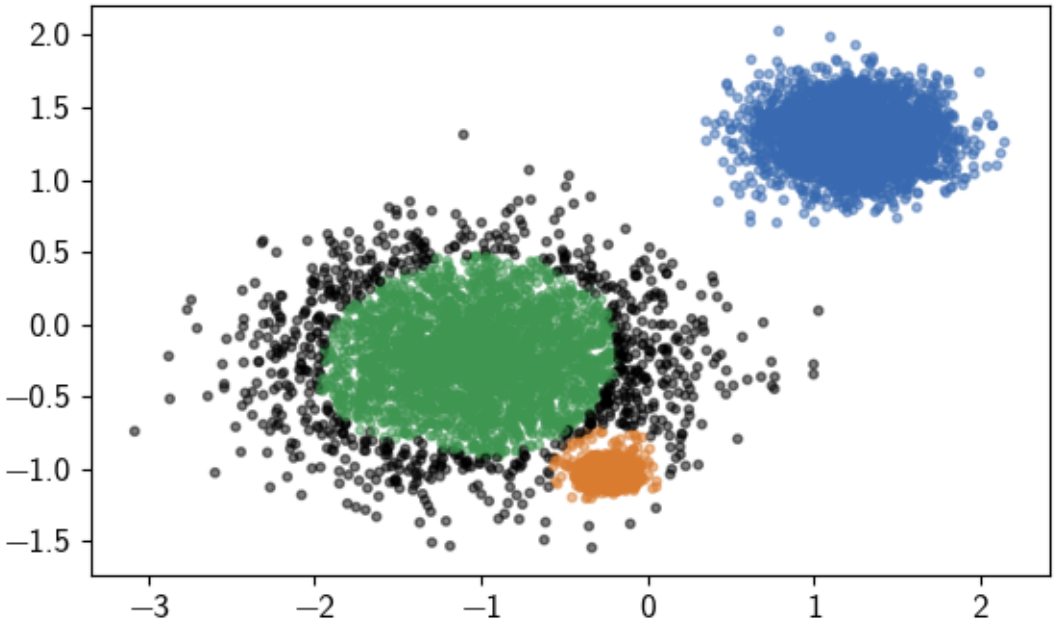
How certain aspects of the module behave is defined by a config file, which is automatically tried to be saved in the users home directory as .cnnrc. A config file in the current working directory overides all settings.

```
>>> import pathlib
...
>>> with open(f"{pathlib.Path.home()}/.cnnrc", 'r') as file_:
...     for line in file_:
...         print(line)
```

(continues on next page)


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2



(continued from previous page)

```
[settings]
record_points = points
record_radius_cutoff = radius_cutoff
record_cnn_cutoff = cnn_cutoff
record_member_cutoff = member_cutoff
record_max_cluster = max_cluster
record_n_cluster = n_cluster
record_largest = largest
record_noise = noise
record_time = time
default_radius_cutoff = 1
default_cnn_cutoff = 1
default_member_cutoff = 0
color = #000000 #396ab1 #da7c30 #3e9651 #cc2529 #535154
        #6b4c9a #922428 #948b3d #7293cb #e1974c #84ba5b
        #d35e60 #9067a7 #ab6857 #ccc210 #808585
```

CHAPTER

THREE

DOCUMENTATION

This is cnn v0.1

The functionality provided in this module is based on code implemented by Oliver Lemke in the script collection CNNClustering available on git-hub (https://github.com/BDGSoftware/CNNClustering.git).

Author: Jan-Oliver Joswig, first released: 03.12.2018

class cnn.**CNN**(*alias='root'*, *train=None*, *test=None*, *train_dist_matrix=None*, *test_dist_matrix=None*, *map_matrix=None*)

CNN cluster object class

cut(*parts=(None*, *None*, *None)*, *points=(None*, *None*, *None)*, *dimensions=(None*, *None*, *None)*)

Alows data set reduction. For each data set level (parts, points, dimensions) a tuple (start:stop:step) can be specified.

dist(*mode='train'*, *low_memory=False*)

Computes a distance matrix (points x points) for points in given data of standard shape (parts, points, dimensions)

dist_hist(*mode='train'*, *bins=100*, *range=None*, *density=True*, *weights=None*, *xlabel='d / au'*, *ylabel=''*, *show=True*, *save=False*, *output='dist_hist.pdf'*, *dpi=300*, *maxima=False*)

Shows/saves a histogram plot for distances in a given distance matrix

evaluate(*mode='train'*, *max_clusters=None*, *plot='scatter'*, *parts=(None*, *None*, *None)*, *points=(None*, *None*, *None)*, *dim=None*, *show=True*, *save=False*, *output='evaluation.pdf'*, *dpi=300*)

Shows/saves a 2D histogram or scatter plot of a cluster result

fit(*radius_cutoff=None*, *cnn_cutoff=None*, *member_cutoff=None*, *max_clusters=None*, *rec=True*)

Performs a CNN clustering of points in a given train distance matrix

get_shape(*data*)

Analyses the format of given data and fits it into standard format (parts, points, dimensions).

isolate(*mode='train'*, *purge=True*)

Isolates points per clusters based on a cluster result

load(*file_*, *mode='train'*)

Loads file content and return data and shape

map(*nearest=None*)

Computes a map matrix that maps an arbitrary data set to a reduced to set

query_data(*mode='train'*)

Helper function to evaluate user input. If data is required as keyword argument and data=None is passed, the default data used is either self.rdata or self.data.

save(*file_*, *content*)
Saves content to file

class cnn.**CNNChild**(*parent*)
CNN cluster object subclass. Increments the hierarchy level of the parent object when instanciated.

cnn.**dist**(*data*)
High level wrapper function for cnn.CNN().dist(). Takes data and returns a distance matrix (points x points).

cnn.**recorded**(*function_*)
Decorator to format function feedback. Feedback needs to be pandas series in record format. If execution time was measured, this will be included in the summary.

cnn.**timed**(*function_*)
Decorator to measure execution time. Forwards the output of the wrapped function and measured excecution time.

CHAPTER

FOUR

CHANGELOG

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX