Exercise 2

1. Density Based Clustering: DENCLUE Write a script to implement the DENCLUE density-based clustering algorithm Algorithm 15.2 in chapter 15. The script should take as input a dataset D, the minimum density ξ, the tolerance for convergence ϵ, and the width h. Do not make any assumptions about the data (i.e., column names, etc), except that the last column gives the "true" cluster id.

Run your script on the iris.txt dataset, with ϵ=0.0001. Your script should output the following:

The number of clusters, and the size of each cluster. The density attractor, followed by the set of point in that cluster. Purity of the clustering, based on the true id. For Iris, you should use a value of ξ that gives you 3 clusters in the end, i.e., try different values and then finally report only the results for the value that gives you 3 clusters, since there are 3 true clusters in the data. Select the value of h empirically.

To speed up the computation for estimating the density at a point, you may want to first identify the K nearest neighbors, and use only those neighbors.

In [29]:
```
import pandas as pd
import numpy as np
import networkx as nx
from scipy.spatial import distance
from sklearn.base import BaseEstimator, ClusterMixin
print("Import Complete")
```

```
Import Complete
```

In [30]:
```
FILE_NAME = "iris.txt"
```

In [31]:
```
## Code adapted from @author: mgarrett

def _hill_climb(x_t, X, W=None, h=0.3, eps=0.0001):
    """
    This function climbs the 'hill' of the kernel density function
    and finds the 'peak', which represents the density attractor
    """
    error = 99.
    prob = 0.
    x_l1 = np.copy(x_t)

    #Sum of the last three steps is used to establish radius
    #of neighborhood around attractor. Authors suggested two
    #steps works well, but I found three is more robust to
    #noisey datasets.
    radius_new = 0.
    radius_old = 0.
    radius_twiceold = 0.
    iters = 0.
    while True:
        radius_thriceold = radius_twiceold
        radius_twiceold = radius_old
        radius_old = radius_new
        x_l0 = np.copy(x_l1)
        x_l1, density = _step(x_l0, X, W=W, h=h)
        error = density - prob
        prob = density
        radius_new = np.linalg.norm(x_l1-x_l0)
        radius = radius_thriceold + radius_twiceold + radius_old + radius_new
        iters += 1
        if iters>3 and error < eps:
            break
    return [x_l1, prob, radius]

def _step(x_l0, X, W=None, h=0.2):
    n = X.shape[0]
    d = X.shape[1]
    superweight = 0. #superweight is the kernel X weight for each item
    x_l1 = np.zeros((1,d))
    if W is None:
        W = np.ones((n,1))
    else:
        W = W
    for j in range(n):
        kernel = kernelize(x_l0, X[j], h, d)
        kernel = kernel * W[j]/(h**d)
        superweight = superweight + kernel
        x_l1 = x_l1 + (kernel * X[j])
    x_l1 = x_l1/superweight
    density = superweight/np.sum(W)
    return [x_l1, density]

def kernelize(x, y, h, degree):
    kernel = np.exp(-(np.linalg.norm(x-y)/h)**2./2.)/((2.*np.pi)**(degree/2))
    return kernel

def density(X,D,h,degree):
    sum1=0
    for i in range(D.shape[0]):
        k=kernelize(X,D[i],h,degree)
        sum1=sum1+k
    d = 1./D.shape[0]/h**degree*sum1
    return d
def DENCLUE(D,h,xi,eps):
```

In [32]:
```
class DENCLUE(BaseEstimator, ClusterMixin):

    def __init__(self, h=0.2, eps=0.0001, min_density=0., metric='euclidean'):
        self.h = h
        self.eps = eps
        self.min_density = min_density
        self.metric = metric

    def fit(self, X, y=None, sample_weight=None):
        if not self.eps > 0.0:
            raise ValueError("eps must be positive.")
        self.n_samples = X.shape[0]
        self.n_features = X.shape[1]
        density_attractors = np.zeros((self.n_samples,self.n_features))
        radii = np.zeros((self.n_samples,1))
        density = np.zeros((self.n_samples,1))

        #create default values
        if self.h is None:
            self.h = np.std(X)/5
        if sample_weight is None:
            sample_weight = np.ones((self.n_samples,1))
        else:
            sample_weight = sample_weight

        #initialize all labels to noise
        labels = -np.ones(X.shape[0])

        #climb each hill
        for i in range(self.n_samples):
            density_attractors[i], density[i], radii[i] = _hill_climb(X[i], X, W=sa
mple_weight,
                                                      h=self.h, eps=self.eps)

        #initialize cluster graph to finalize clusters. Networkx graph is
        #used to verify clusters, which are connected components of the
        #graph. Edges are defined as density attractors being in the same
        #neighborhood as defined by our radii for each attractor.
        cluster_info = {}
        num_clusters = 0
        cluster_info[num_clusters]={'instances': [0],
                                    'centroid': np.atleast_2d(density_attractors
[0])}
        g_clusters = nx.Graph()
        for j1 in range(self.n_samples):
            g_clusters.add_node(j1, attr_dict={'attractor':density_attractors[j1],
'radius':radii[j1],
                                   'density':density[j1]})

        #populate cluster graph
        for j1 in range(self.n_samples):
            for j2 in (x for x in range(self.n_samples) if x != j1):
                if g_clusters.has_edge(j1,j2):
                    continue
                diff = np.linalg.norm(g_clusters.node[j1]['attractor']-g_clusters.n
ode[j2]['attractor'])
                if diff <= (g_clusters.node[j1]['radius']+g_clusters.node[j1]['radi
us']):
                    g_clusters.add_edge(j1, j2)

        #connected components represent a cluster
        clusters = list(nx.connected_component_subgraphs(g_clusters))
        num_clusters = 0
```

In [39]:
```
## Main function

# reading the file
iris = pd.read_csv(FILE_NAME, header=None)


data = iris
data = np.array(data)
samples = np.mat(data[:,0:4])
true_labels=data[:,-1]
labels=list(set(true_labels))
true_ID=np.zeros((3,50))
index=range(len(true_labels))
for i in range(len(labels)):
    true_ID[i]=[j for j in index if true_labels[j]==labels[i]]


d = DENCLUE(0.2, 0.0001)



print(d)
```

```
DENCLUE(eps=0.0001, h=0.2, metric='euclidean', min_density=0.0)
```

In [ ]: