Worksheet 07

Name: Jeong Yong Yang
UID: U95912941

Topics

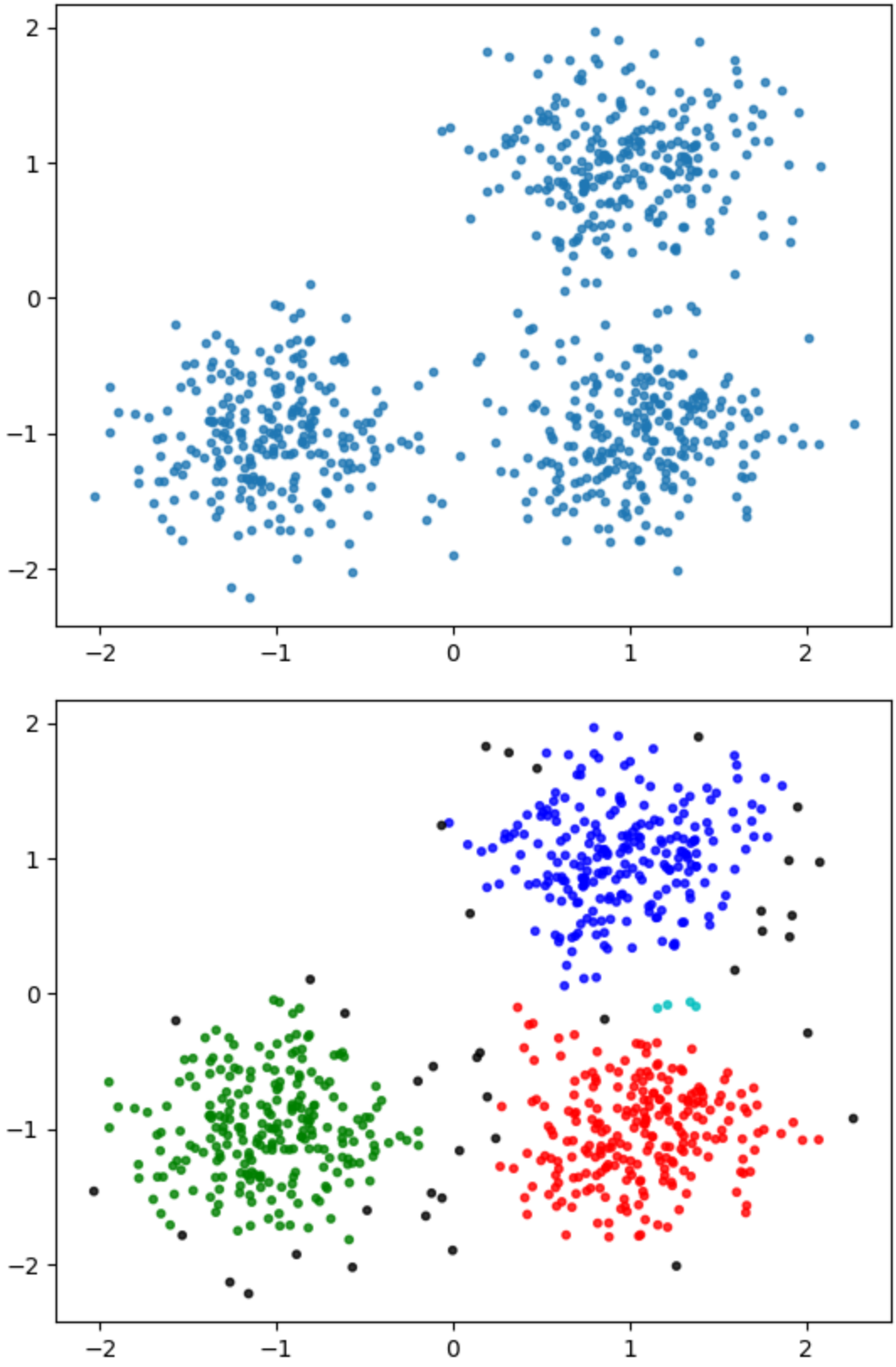
- Density-Based Clustering

Density-Based Clustering

Follow along with the live coding of the DBScan algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

centers = [[1, 1], [-1, -1], [1, -1]]
X, _ = datasets.make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                           random_state=0)
plt.scatter(X[:,0],X[:,1],s=10, alpha=0.8)
plt.show()

class DBC():

    def __init__(self, dataset, min_pts, epsilon):
        self.dataset = dataset
        self.min_pts = min_pts
        self.epsilon = epsilon
        self.assignments = [-1 for _ in range(len(self.dataset))]

    def is_unassigned(self, i):
        return self.assignments[i] == -1

    def distance(self, i, j):
        return np.linalg.norm(self.dataset[i] - self.dataset[j])

    def get_neighborhood(self, i):
        neighborhood = []
        for j in range(len(self.dataset)):
            if i != j and self.distance(i, j) <= self.epsilon:
                neighborhood.append(j)
        return neighborhood

    def get_unassigned_neighborhood(self, i):
        neighborhood = self.get_neighborhood(i)
        # return [point for point in neighborhood if self.assignments[point] == -1] # you
        return [point for point in neighborhood if self.is_unassigned(point)]

    def is_core(self, i):
        return len(self.get_neighborhood(i)) >= self.min_pts


    def make_cluster(self, i, cluster_num):
        self.assignments[i] = cluster_num
        neighborhood_queue = self.get_unassigned_neighborhood(i) #TODO: maybe make this a

        while neighborhood_queue:
            next_pt = neighborhood_queue.pop()
            if not self.is_unassigned(next_pt): #TODO: make this a function and improve c
                continue

            self.assignments[next_pt] = cluster_num #NOTE: border points will be assigned
            # self.snap
            if self.is_core(next_pt):
                neighborhood_queue += self.get_unassigned_neighborhood(next_pt)

        return
```

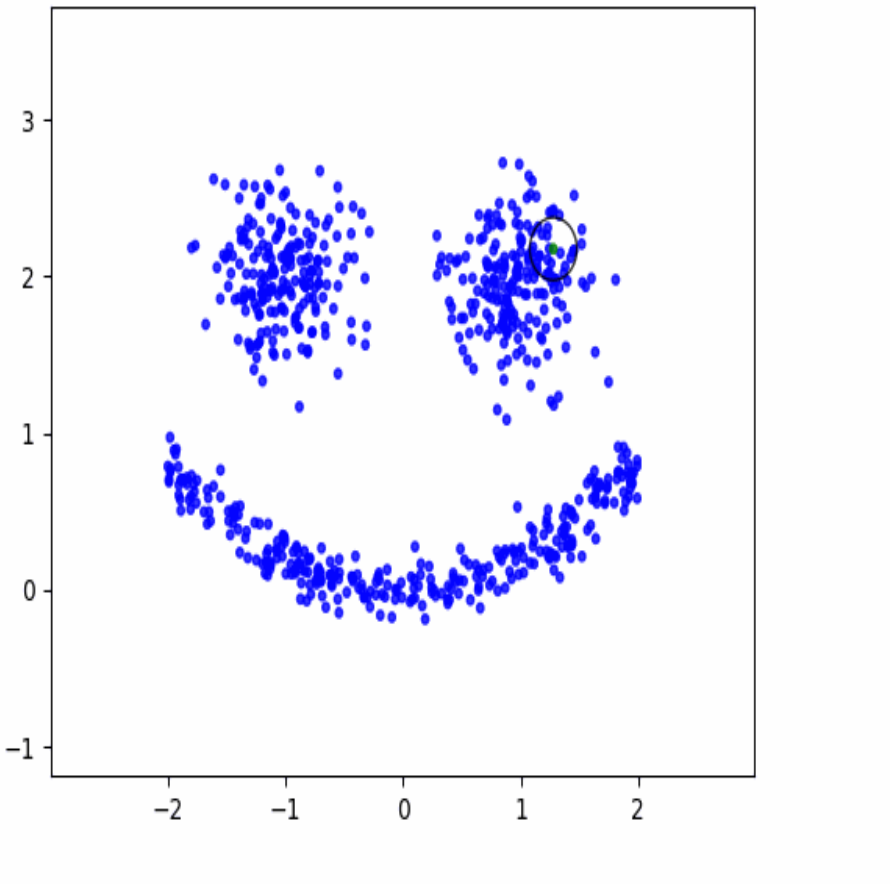
```
56
57     def dbscan(self):
58         """
59         returns a list of assignments. The index of the
60         assignment should match the index of the data point
61         in the dataset.
62         """
63
64         cluster_num = 0
65         for i in range(len(self.dataset)):
66             if self.assignments[i] != -1:
67                 continue
68
69             if self.is_core(i):
70                 # start building a new cluster
71                 self.make_cluster(i, cluster_num)
72                 cluster_num += 1
73
74         return self.assignments
75
76
77 clustering = DBC(X, 3, .2).dbscan()
78 colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
79 colors = np.hstack([colors] * 100)
80 plt.scatter(X[:, 0], X[:, 1], color=colors[clustering].tolist(), s=10, alpha=0.8)
```



Challenge Problem

Using the code above and the template provided below, create the animation below of the DBScan algorithm.

```
1 from IPython.display import Image
2 Image(filename="dbscan_2.gif", width=500, height=500)
```



Hints:

- First animate the dbscan algorithm for the dataset used in class (before trying to create the above dataset)
- Take a snapshot of the assignments when the point gets assigned to a cluster
- Confirm that the snapshot works by saving it to a file
- Don't forget to close the matplotlib plot after saving the figure
- Gather the snapshots in a list of images that you can then save as a gif using the code below
- Use `ax.set_aspect('equal')` so that the circles don't appear to be oval shaped
- To create the above dataset you need two blobs for the eyes. For the mouth you can use the following process to generate (x, y) pairs:
    - Pick an x at random in an interval that makes sense given where the eyes are positioned
    - For that x generate y that is 0.2 * x^2 plus a small amount of randomness
    - `zip` the x's and y's together and append them to the dataset containing the blobs

```python
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

TEMPFILE = 'temp.png'

class DBC():

    def __init__(self, dataset, min_pts, epsilon):
        self.dataset = dataset
        self.min_pts = min_pts
        self.epsilon = epsilon
        self.snaps = []


    def snapshot(self):
        fig, ax = plt.subplots()
        num_points = len(self.dataset)
        colors = np.random.rand(num_points, 3)

        ax.scatter(self.dataset[:, 0], self.dataset[:, 1], s=8, c=colors)
        cir = plt.Circle((0, 0), 1, edgecolor='black', facecolor='none') # create circle
        ax.add_patch(cir)
        ax.set_xlim(-10, 10)
        ax.set_ylim(-10, 10)
        ax.set_aspect('equal') # necessary or else the circles appear to be oval shaped

        fig.savefig(TEMPFILE)
        plt.close()

        return im.fromarray(np.asarray(im.open(TEMPFILE)))


    def dbscan(self):
        from sklearn.cluster import DBSCAN
        dbscan = DBSCAN(eps=self.epsilon, min_samples=self.min_pts).fit(self.dataset)
        return dbscan.labels_


centers = [[-5, 0], [5, 0]]
eyes, _ = datasets.make_blobs(n_samples=100, centers=centers, cluster_std=.7)

mouth_x = 3.5 * np.random.random(100)
mouth_y = np.abs(mouth_x - 1.75)**2 - 8 + .1 * np.random.randn(100)

face = np.append(eyes, np.column_stack((mouth_x, mouth_y)), axis=0)

dbc = DBC(face, min_pts=5, epsilon=3)
clustering = dbc.dbscan()

dbc.snaps.append(dbc.snapshot())
# for i in range(len(dbc.dataset)):
    # dbc.snaps.append(dbc.snapshot())

```

```
dbc.snaps[0].save(
    'dbscan.gif',
    optimize=False,
    save_all=True,
    append_images=dbc.snaps[1:],
    loop=0,
    duration=25
```

```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

TEMPFILE = 'temp.png'

class DBC():

    def __init__(self, dataset, min_pts, epsilon):
        self.dataset = dataset
        self.min_pts = min_pts
        self.epsilon = epsilon
```

```
 import numpy as np
 from PIL import Image as im
 import matplotlib.pyplot as plt
 import sklearn.datasets as datasets

 TEMPFILE = 'temp.png'

 class DBC():

     def __init__(self, dataset, min_pts, epsilon):
         self.dataset = dataset
         self.min_pts = min_pts
         self.epsilon = epsilon
         self.snaps = []


     def snapshot(self, assignments):
         fig, ax = plt.subplots()
         colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
         colors = np.hstack([colors] * 100)

         ax.scatter(self.dataset[:, 0], self.dataset[:, 1], color=colors[assignments].toli
         cir = plt.Circle((0, 0), 1, edgecolor='black', facecolor='none')
```