worksheet_07

February 25, 2024

1 Worksheet 07

Name:
UID:

1.0.1 Topics

- Density-Based Clustering

1.0.2 Density-Based Clustering

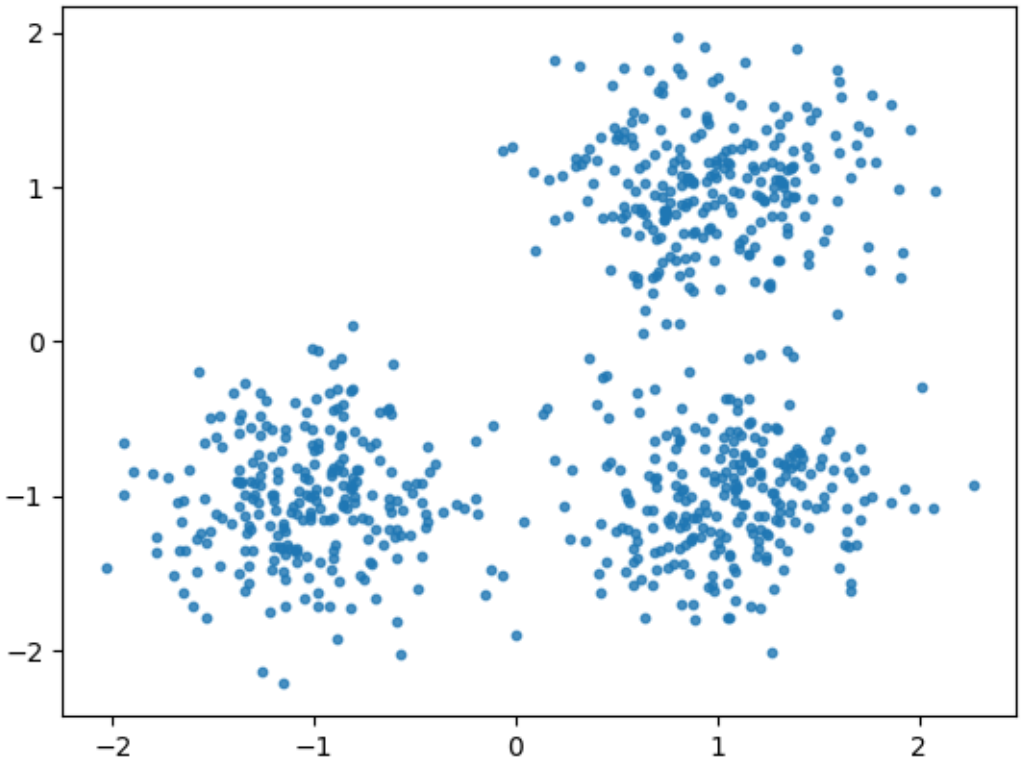
Follow along with the live coding of the DBScan algorithm.

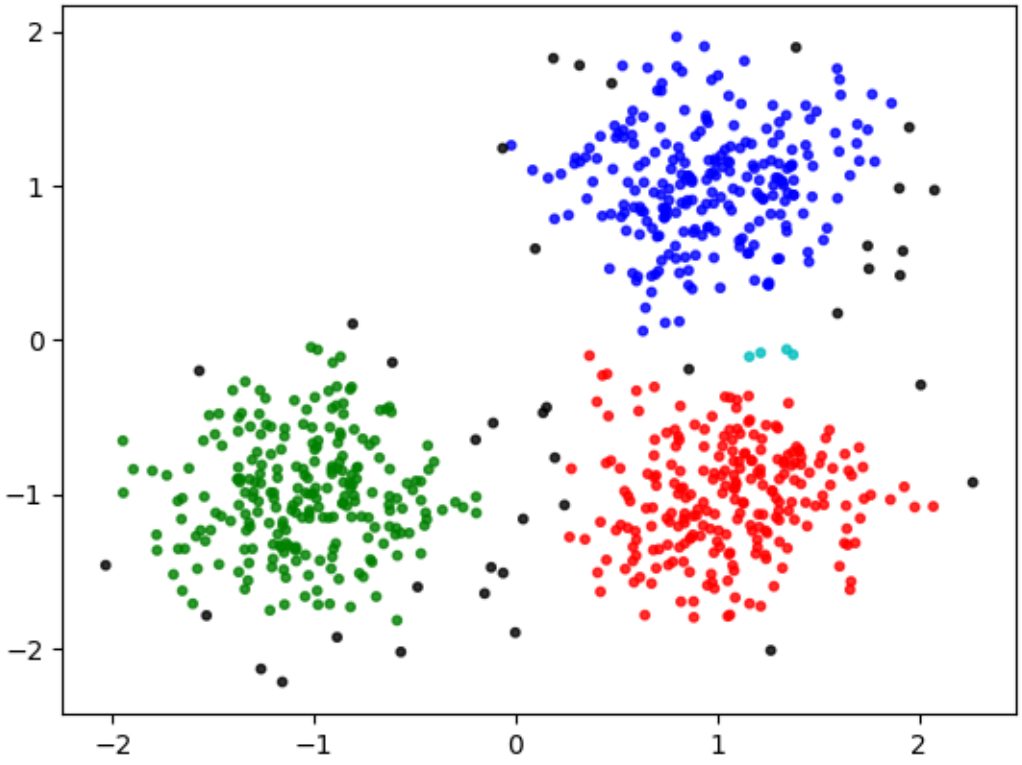
[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

centers = [[1, 1], [-1, -1], [1, -1]]
X, _ = datasets.make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                           random_state=0)

print(X)
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 x in range(len(self.dataset))]

    def dbscan(self):
        """
        returns a list of assignments. The index of the
        assignment should match the index of the data point
```

```
        in the dataset.
        """


        cluster_id = 0 #changed cluster num to cluster id cause it makes more sense to me
        for i in range(len(self.dataset)):
            if not self.is_unassigned(i) :
                continue
            if self.is_core_point(i):
                #start building new cluster
                self.make_cluster(i, cluster_id=cluster_id)
                cluster_id += 1


        return self.assignments


    def is_core_point(self, i):

        return len(self.get_neigborhood(i)) >= self.min_pts

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

    def get_neigborhood(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_neigborhood(self, i):
        neighborhood = self.get_neigborhood(i)
        return [point for point in neighborhood if self.is_unassigned(point)]


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

    def make_cluster(self, i, cluster_id):
        self.assignments[i] = cluster_id
        neighbors_queue = self.get_unassigned_neigborhood(i) #maybe get a stack or a double ended queue

        while len(neighbors_queue) > 0:
```

```
            next_pt = neighbors_queue.pop()
            if not self.is_unassigned(next_pt): #Todo: Make this a function and␣
 ↪improve data structure
                continue
            self.assignments[next_pt] = cluster_id #Note: border points will be␣
 ↪assigmend to the cluster in a last come last serve basis
            if self.is_core_point(next_pt):
                #self.assignments[next_pt] = cluster_id
                neighbors_queue += self.get_unassigned_neigborhood(next_pt)


        return

clustering = DBC(X, 3, .2).dbscan()
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 200)
plt.scatter(X[:, 0], X[:, 1], color=colors[clustering].tolist(), s=10, alpha=0.
 ↪8)
plt.show()
```

```
[[ 0.84022039  1.14802236]
 [-1.15474834 -1.2041171 ]
 [ 0.67863613  0.72418009]
 …
 [ 0.26798858 -1.27833405]
 [-0.88628813 -0.30293249]
 [ 0.60046048 -1.29605472]]
```


2
1
0
−1
−2
−2
−1
0
1
2


2
1
0
−1
−2
−2
−1
0
1
2

1.1 Challenge Problem

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

[6]:
```
from IPython.display import Image
Image(filename="dbscan_2.gif", width=500, height=500)
```

[6]:
```
<IPython.core.display.Image object>
```

Hints:

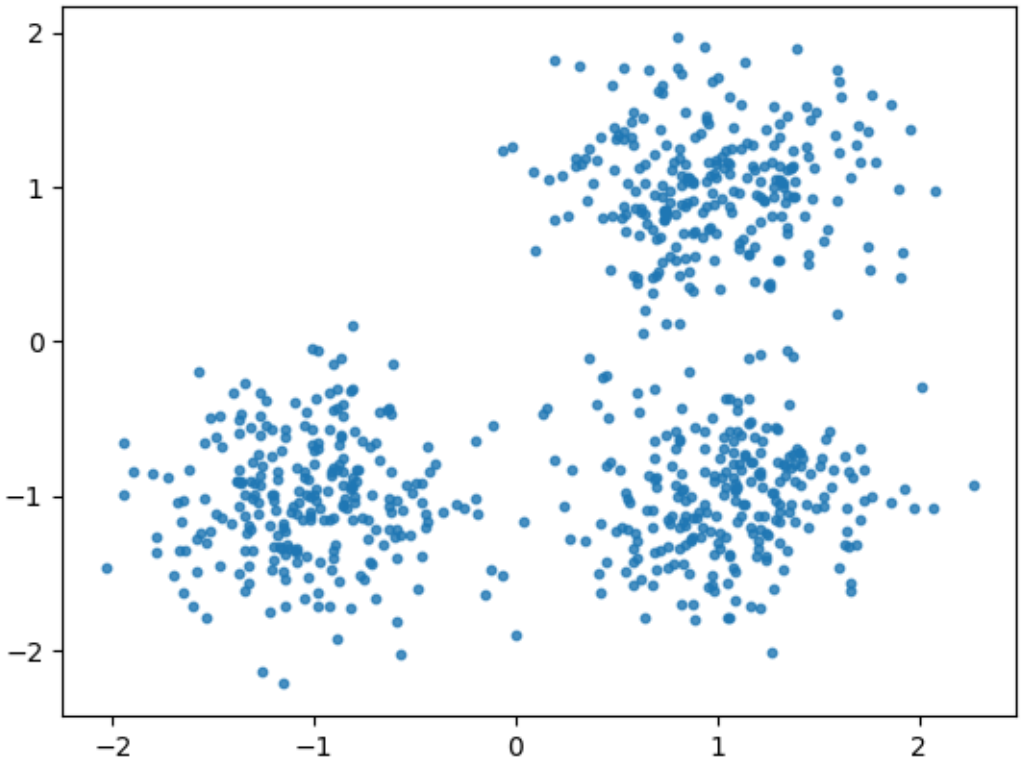
- First animate the dbscan algorithm for the dataset used in class (before trying to create the above dataset)
- Take a snapshot of the assignments when the point gets assigned to a cluster
- Confirm that the snapshot works by saving it to a file
- Don't forget to close the matplotlib plot after saving 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

[30]:
```
import numpy as np
from PIL import Image as im
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()

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 = []
```

```
        self.assignments = [-1 for x in range(len(self.dataset))]
        self.positionX = .5
        self.positionY = 1



    def getX(self,i):
        return self.dataset[i][0]

    def getY(self,i):
        return self.dataset[i][1]


    def snapshot(self,i):
        fig, ax = plt.subplots()
        colors = ...
        x = self.dataset[:,0]
        y = self.dataset[:,1]

        ax.scatter(x, y)
       # cir = plt.Circle(self.positionX - self.getX(i), self.positionY- self.
↪getY(i) ) # create circle around the point assigned
        #ax.add_patch(cir)




        ax.set_xlim(min(x)+1,max(x)+1)
        ax.set_ylim(min(y)+1, max(y)+1)
        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):
        cluster_id = 0 #changed cluster num to cluster id cause it makes more␣
↪sense to me
        for i in range(len(self.dataset)):
            if not self.is_unassigned(i) :
                continue
            if self.is_core_point(i):
                #start building new cluster
                self.make_cluster(i, cluster_id=cluster_id)
```

```
                self.snaps.append(self.snapshot(i))
                cluster_id += 1


        return self.assignments


    def is_core_point(self, i):

        return len(self.get_neigborhood(i)) >= self.min_pts

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

    def get_neigborhood(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_neigborhood(self, i):
        neighborhood = self.get_neigborhood(i)
        return [point for point in neighborhood if self.is_unassigned(point)]


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

    def make_cluster(self, i, cluster_id):
        self.assignments[i] = cluster_id
        neighbors_queue = self.get_unassigned_neigborhood(i) #maybe get a stack␣
↪or a double ended queue

        while len(neighbors_queue) > 0:
            next_pt = neighbors_queue.pop()
            if not self.is_unassigned(next_pt): #Todo: Make this a function and␣
↪improve data structure
                continue
            self.assignments[next_pt] = cluster_id #Note: border points will be␣
↪assigmend to the cluster in a last come last serve basis
            if self.is_core_point(next_pt):
                #self.assignments[next_pt] = cluster_id
                neighbors_queue += self.get_unassigned_neigborhood(next_pt)
```

```
        return



#centers = [...]
#eyes, _ = datasets.make_blobs(...)

#mouth_x = ... * np.random.random(...)
#mouth_y = ... + .1 * np.random.randn(...)

#face = np.append(eyes, ..., axis=0)



dbc = DBC(X, 3, 3)
clustering = dbc.dbscan()

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


2
1
0
−1
−2
−2
−1
0
1
2