Worksheet 07

Name: Xi Chen UID: U23766637

Topics

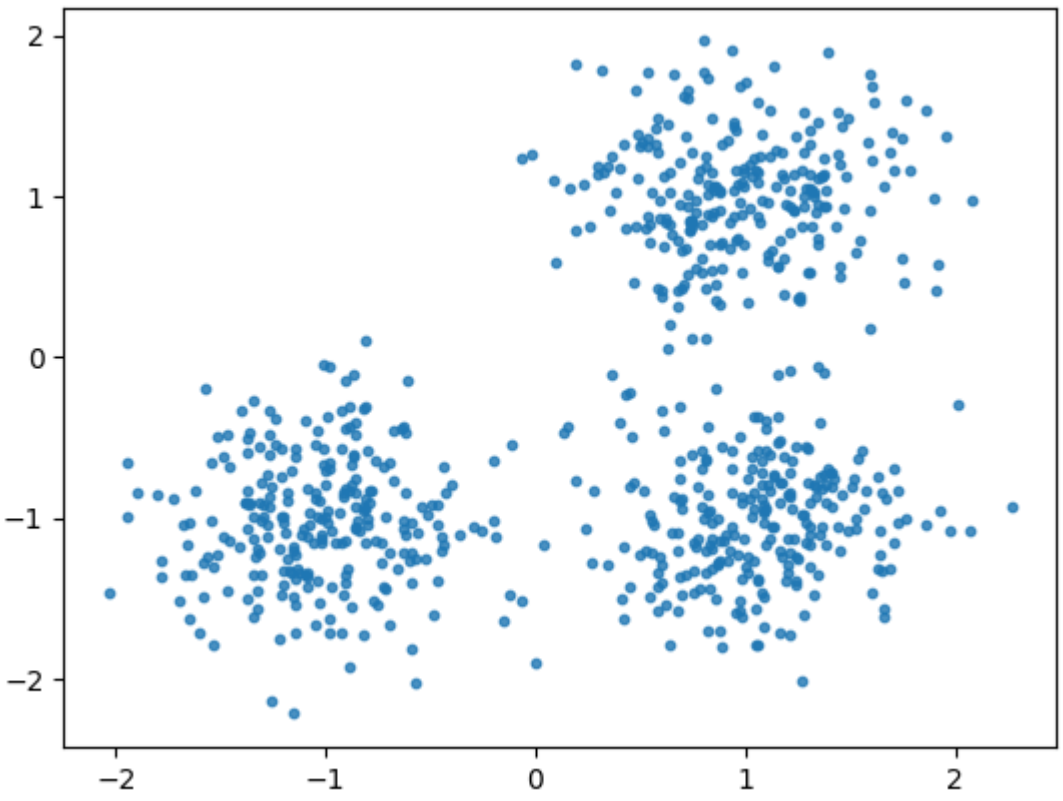
- Density-Based Clustering

Density-Based Clustering

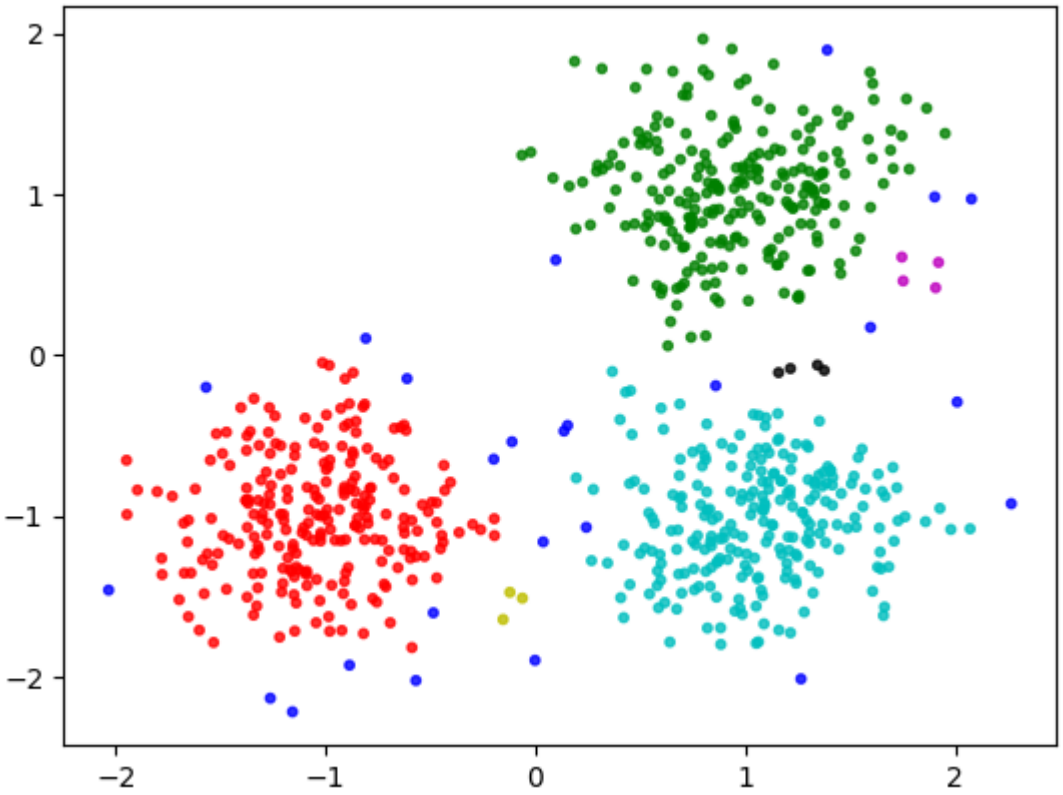
Follow along with the live coding of the DBScan algorithm.

In [20]:
```
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

    def search(self,now, id, assignments, core_points):
        assignments[now] = id
        for i in range(self.dataset.shape[0]):
            if np.linalg.norm(self.dataset[now] - self.dataset[i]) < self.epsilon:
                if assignments[i] == 0:
                    assignments[i] = id
                    if i in core_points:
                        self.search(i, id, assignments, core_points)
        return assignments

    def dbscan(self):
        """
        returns a list of assignments. The index of the
        assignment should match the index of the data point
        in the dataset.
        """
        core_points = []
        assignments = np.zeros(self.dataset.shape[0])
        for i in range(self.dataset.shape[0]):
            count = 0
            for j in range(self.dataset.shape[0]):
```

```
                if np.linalg.norm(self.dataset[i] - self.dataset[j]) < self.epsilon
                    count += 1
            if count >= self.min_pts:
                core_points.append(i)
        print(len(core_points))
        id = 1
        for i in range(len(core_points)):
            if assignments[core_points[i]] != 0:
                continue
            assignments = self.search(core_points[i], id, assignments, core_points)
            id += 1
        assignments = assignments.astype(int)
        return  assignments


clustering = DBC(X, 3, .2).dbscan()
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 20)

plt.scatter(X[:, 0], X[:, 1], color=colors[clustering].tolist(), s=10, alpha=0.8)
plt.show()
```



714



Challenge Problem

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

In [49]:
```
from IPython.display import Image
Image(filename="dbscan_2.gif", width=500, height=500)
```

Out[49]:
```
<IPython.core.display.Image object>
```

In [ ]:
```

```

Hints:

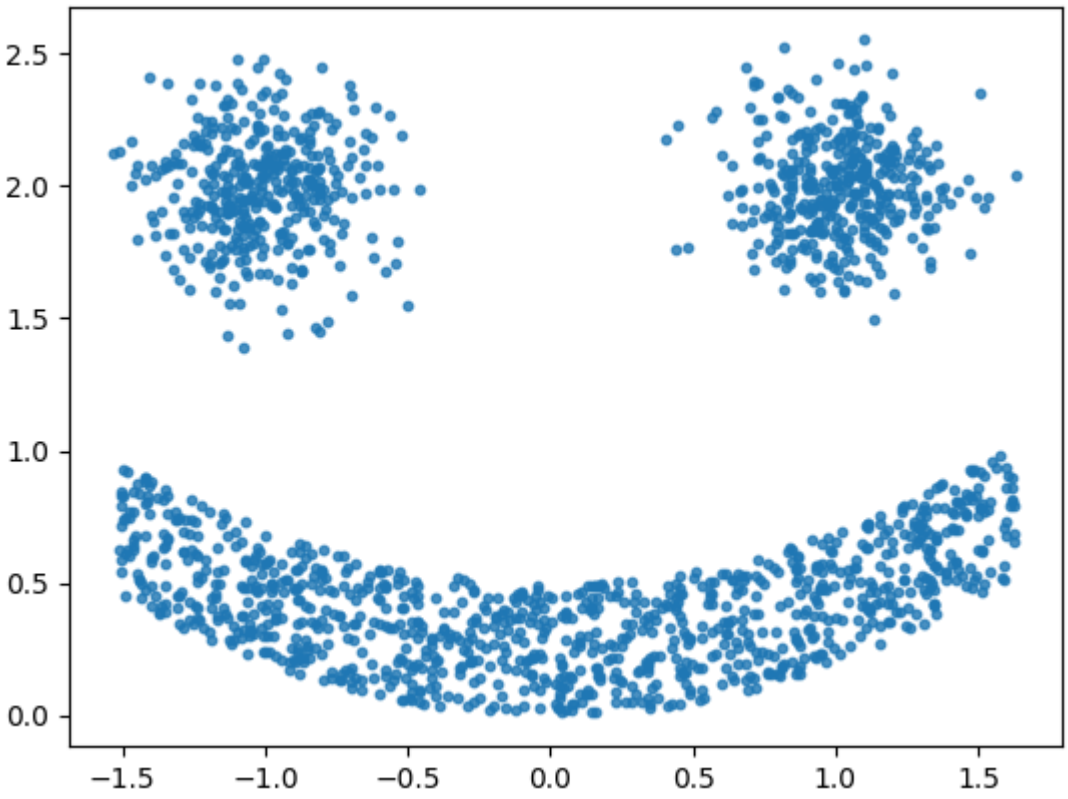
- First animate the dbscan algorithm for the dataset used in class (before trying to create the above dataset)
- Take a snapshot of the assignments when the point gets assigned to a cluster
- Confirm that the snapshot works by saving it to a file
- Don't forget to close the matplotlib plot after saving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

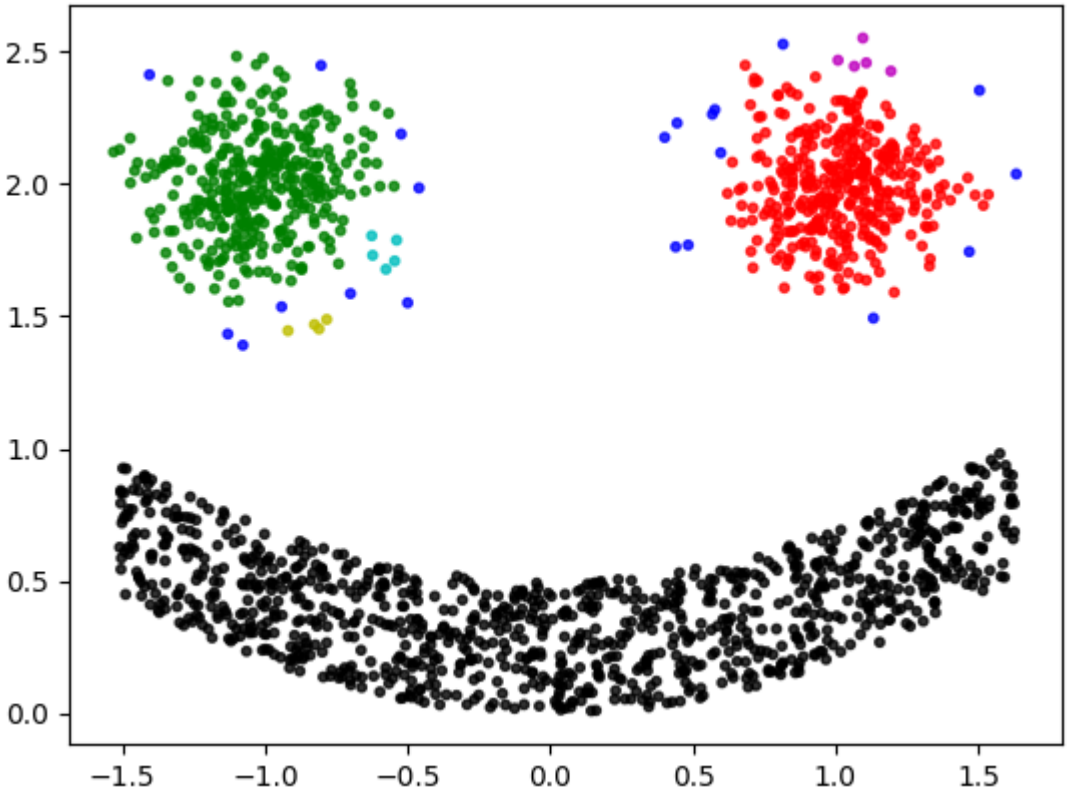
In [86]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets
import progressbar

TEMPFILE = 'temp.png'

class DBC():

    def __init__(self, dataset, min_pts, epsilon):
        self.dataset = dataset
        self.min_pts = min_pts
        self.epsilon = epsilon
        self.assignments = np.zeros(self.dataset.shape[0])
        self.snaps = []
        self.bar = progressbar.ProgressBar(maxval=len(dataset)).start()
        self.idx = 0

    def snapshot(self, point):
        fig, ax = plt.subplots()
        colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
        colors = np.hstack([colors] * 20)

        ax.scatter(self.dataset[:, 0], self.dataset[:, 1], color=colors[self.assign
        cir = plt.Circle(point,radius=self.epsilon,fill=False, edgecolor='black') #
        ax.add_patch(cir)
        ax.set_xlim(-2,2)
        ax.set_ylim(-0.5,3)
        ax.set_aspect('equal') # necessary or else the circles appear to be oval sh



        fig.savefig(TEMPFILE)

        plt.close()

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

    def search(self,now, id, core_points):
        self.assignments[now] = id
        self.snaps.append(self.snapshot(self.dataset[now]))
        for i in range(self.dataset.shape[0]):
            if np.linalg.norm(self.dataset[now] - self.dataset[i]) < self.epsilon:
                if self.assignments[i] == 0:
                    self.bar.update(self.idx)
                    self.idx += 1
                    self.assignments[i] = id
```

```
                    if i in core_points:
                        self.assignments[i] = 0
                        self.search(i, id, core_points)
        return


    def dbscan(self):
        core_points = []

        self.assignments = self.assignments.astype(int)
        for i in range(self.dataset.shape[0]):
            count = 0
            for j in range(self.dataset.shape[0]):
                if np.linalg.norm(self.dataset[i] - self.dataset[j]) < self.epsilon
                    count += 1
            if count >= self.min_pts:
                core_points.append(i)
        print("number of centers : ",len(core_points))
        id = 1
        for i in range(len(core_points)):
            if self.assignments[core_points[i]] != 0:
                continue
            self.search(core_points[i], id, core_points)
            id += 1
        self.bar.finish()
        return self.assignments



centers = [[-1, 2], [1,2]]
eyes, _ = datasets.make_blobs(n_samples=750, centers=centers, cluster_std=0.2,
                              random_state=0)
# 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 posi
# 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
mouth_x  = eyes[:,0].min() + np.random.rand(1000) * (eyes[:,0].max() - eyes[:,0].mi
mouth_y = .2 * mouth_x**2 + np.random.rand(1000) * 0.5
face = np.concatenate((eyes, np.array(list(zip(mouth_x, mouth_y)))))
# draw face
plt.scatter(face[:,0],face[:,1],s=10, alpha=0.8)
plt.show()
dbc = DBC(face, 4, 0.1)
clustering = dbc.dbscan()
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 20)

plt.scatter(face[:, 0], face[:, 1], color=colors[clustering].tolist(), s=10, alpha=
plt.show()
dbc.snaps[0].save(
    'dbscan.gif',
    optimize=False,
    save_all=True,
    append_images=dbc.snaps[1:],
    loop=0,
```

```
    duration=25
)
```




```
  0% |                                                                          |
number of centers :  1704
100% |##########################################################################|
```



In [ ]:

In [ ]: