# Worksheet 07

Name: Rishab Sudhir

UID: U64819615

## Topics

- Density-Based Clustering

## Density-Based Clustering

Follow along with the live coding of the DBScan algorithm.

In [3]:
```
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=
                           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 distance(self,i,j):
        return np.linalg.norm(self.dataset[i] - self.dataset[j])


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

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

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


    def get_unassigned_neighborhood(self,i):
        neighborhood = self.get_neighborhood(i)
        return [point for point in neighborhood if self.is_unassigned(p

    def make_cluster(self, i, clusterNum):
        self.assignments[i] = clusterNum
        neighborhood_queue = self.get_neighborhood(i)

        while neighborhood_queue:
            next_pt = neighborhood_queue.pop()

            if not self.is_unassigned(next_pt):
                continue

            self.assignments[next_pt] = clusterNum

            if self.is_core(next_pt):
                neighborhood_queue += self.get_unassigned_neighborhood(

        return
```

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

        clusterNum = 0

        for i in range(len(self.dataset)):
            if self.assignments[i] != -1:
                continue

            if self.is_core(i):
                # start building a new cluster
                self.make_cluster(i, clusterNum)
                clusterNum += 1

        return self.assignments

clustering = DBC(X, 3, .2).dbscan()
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 100)
plt.scatter(X[:, 0], X[:, 1], color=colors[clustering].tolist(), s=10,
plt.show()
```

```
---------------------------------------------------------------------
------
ModuleNotFoundError                       Traceback (most recent call
last)
/Users/rsudhir/Documents/GitHub/Data-Science-Fundamentals/lecture_07/
worksheet_07.ipynb Cell 2 line 1
----> <a href='vscode-notebook-cell:/Users/rsudhir/Documents/GitHub/D
ata-Science-Fundamentals/lecture_07/worksheet_07.ipynb#W1sZmlsZQ%3D%3
D?line=0'>1</a> import numpy as np
      <a href='vscode-notebook-cell:/Users/rsudhir/Documents/GitHub/D
ata-Science-Fundamentals/lecture_07/worksheet_07.ipynb#W1sZmlsZQ%3D%3
D?line=1'>2</a> import matplotlib.pyplot as plt
      <a href='vscode-notebook-cell:/Users/rsudhir/Documents/GitHub/D
ata-Science-Fundamentals/lecture_07/worksheet_07.ipynb#W1sZmlsZQ%3D%3
D?line=2'>3</a> import sklearn.datasets as datasets

ModuleNotFoundError: No module named 'numpy'
```

In [ ]:

In [ ]:

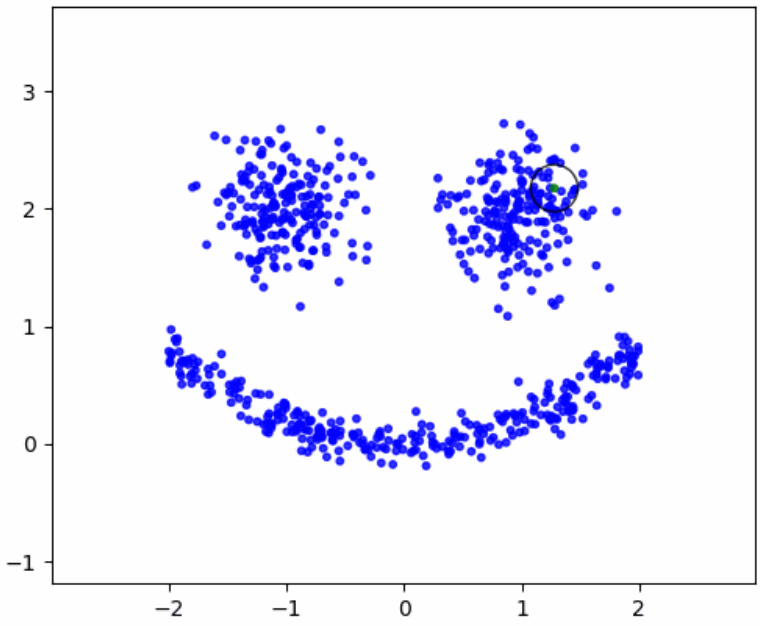
In [ ]:

# Challenge Problem

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

In [2]:
```
from IPython.display import Image
Image(filename="dbscan_2.gif", width=500, height=500)
```

Out[2]:




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 [7]:
```
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.assignments = [-1 for _ in range(len(self.dataset))]
        self.snaps = []

    def snapshot(self, i):
        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

        cir = plt.Circle(self.dataset[i], 0.2, fill = False) # create c

        ax.add_patch(cir)

        ax.set_xlim(-3, 3)
        ax.set_ylim(-1, 4)
        ax.set_aspect('equal') # necessary or else the circles appear t

        fig.savefig(TEMPFILE)
        plt.close()

        self.snaps.append(im.fromarray(np.asarray(im.open(TEMPFILE))))

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

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

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

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

    def get_unassigned_neighbors(self, i):
        n = self.get_neighbors(i)
        return [pt for pt in n if self.is_unassigned(pt)]

    def make_cluster(self, i, cluster_num):
```

```
        self.assignments[i] = cluster_num
        self.snapshot(i)
        queue = self.get_unassigned_neighbors(i)

        while queue:
            next_pt = queue.pop()
            if not self.is_unassigned(next_pt):
                continue
            self.assignments[next_pt] = cluster_num
            self.snapshot(next_pt)

            if self.is_core(next_pt):
                queue += self.get_unassigned_neighbors(next_pt)

    def dbscan(self):
        cluster_num = 0
        for i in range(len(self.dataset)):
            if self.assignments[i] != -1:
                continue
            if self.is_core(i):
                self.make_cluster(i, cluster_num)
                cluster_num += 1
        return self.assignments


centers = [[-1, 2], [1, 2]]
eyes, _ = datasets.make_blobs(n_samples=500, centers=centers, cluster_s

mouth_x = 4 * np.random.random(400) - 2

mouth_y = [0.2 * x**2 for x in mouth_x] + .1 * np.random.randn(400)

mouth = [list(l) for l in zip(mouth_x, mouth_y)]

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

dbc = DBC(face, 5, 0.3)
clustering = dbc.dbscan()

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