Machine Learning HW5

0556562 陳鴻君

(All programs are implemented in Python 2.7 in Linux.)

(Videos is made by “ffmpeg”.

The usage is “ffmpeg –y –framerate 3 –i $sourgeImages $outputpath”)

1. K-means clustering

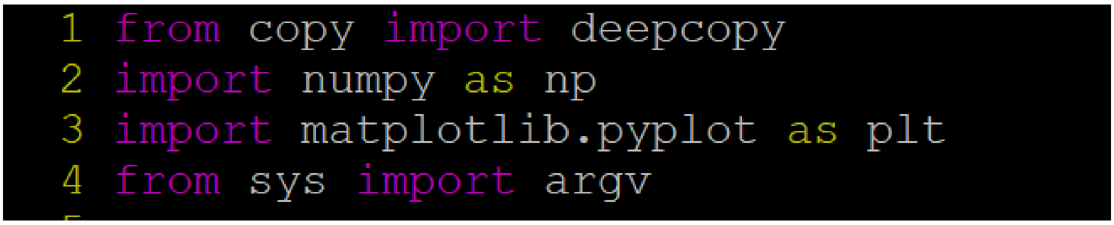
Here’s import part.

“deepcopy” is used to copy array elements in detail.

“numpy” is a powerful matrix operation library.

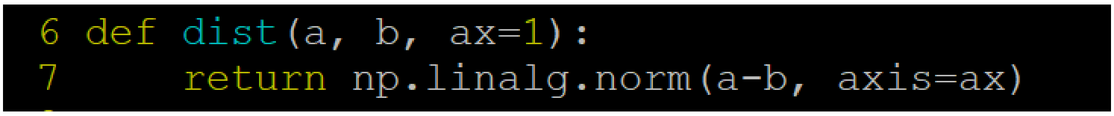
“matplotlib” is used to visualize data.

“argv” is input argument list.



```
from copy import deepcopy
import numpy as np
import matplotlib.pyplot as plt
from sys import argv
```

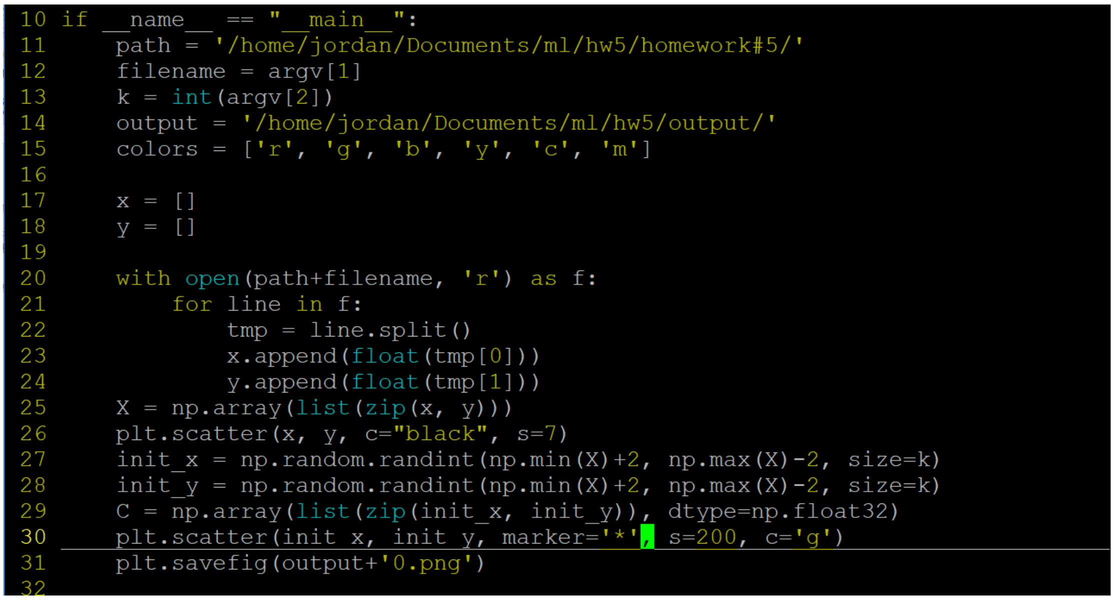
A function named “dist” is used to calculate norm from all elements in matrix a to all elements in matrix b.



```
def dist(a, b, ax=1):
    return np.linalg.norm(a-b, axis=ax)
```

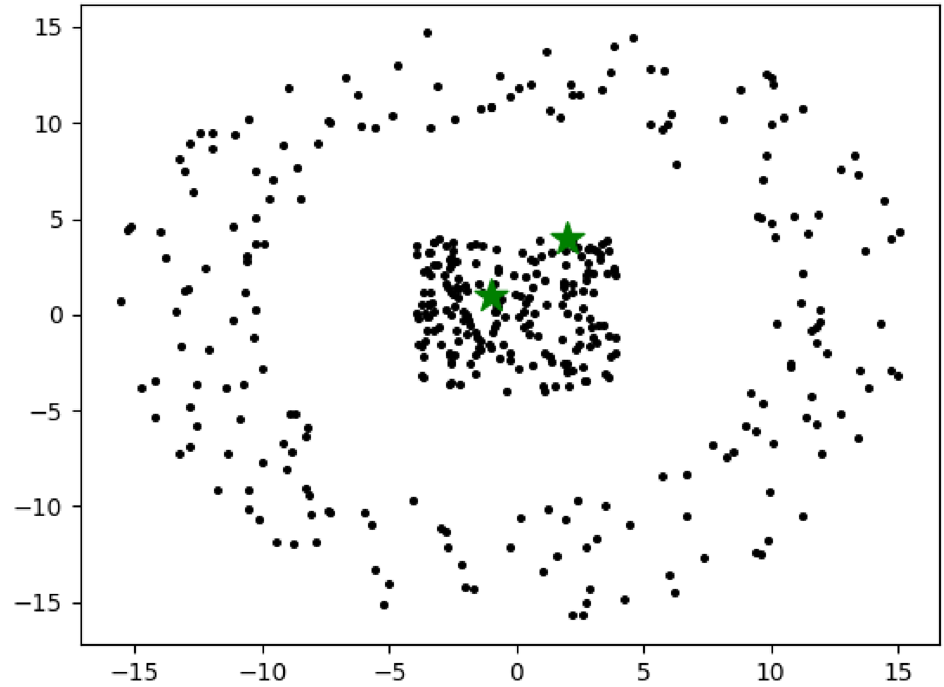
In the first part of main function, save initial in following steps:

- Read data from test file, and k from arguments.
- Initial centroid matrix C, plot and save them in a figure.



```
if __name__ == "__main__":
    path = '/home/jordan/Documents/ml/hw5/homework#5/'
    filename = argv[1]
    k = int(argv[2])
    output = '/home/jordan/Documents/ml/hw5/output/'
    colors = ['r', 'g', 'b', 'y', 'c', 'm']

    x = []
    y = []

    with open(path+filename, 'r') as f:
        for line in f:
            tmp = line.split()
            x.append(float(tmp[0]))
            y.append(float(tmp[1]))
    X = np.array(list(zip(x, y)))
    plt.scatter(x, y, c="black", s=7)
    init_x = np.random.randint(np.min(X)+2, np.max(X)-2, size=k)
    init_y = np.random.randint(np.min(X)+2, np.max(X)-2, size=k)
    C = np.array(list(zip(init_x, init_y)), dtype=np.float32)
    plt.scatter(init_x, init_y, marker='*', s=200, c='g')
    plt.savefig(output+'0.png')

```

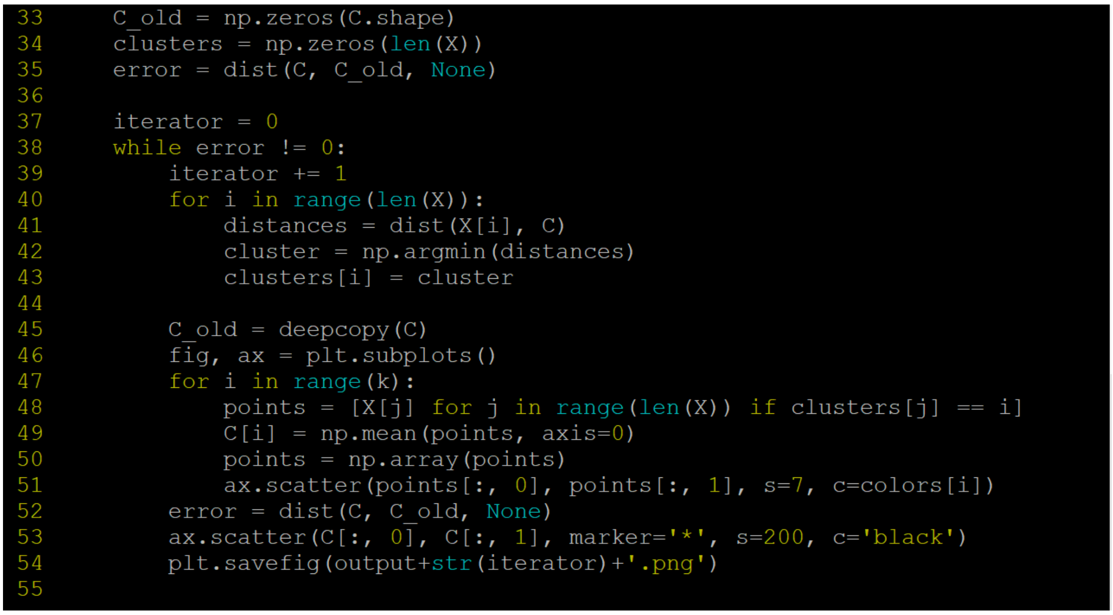
For example:



In the second part of main function, keep clustering data into k clusters until centroid fixed.

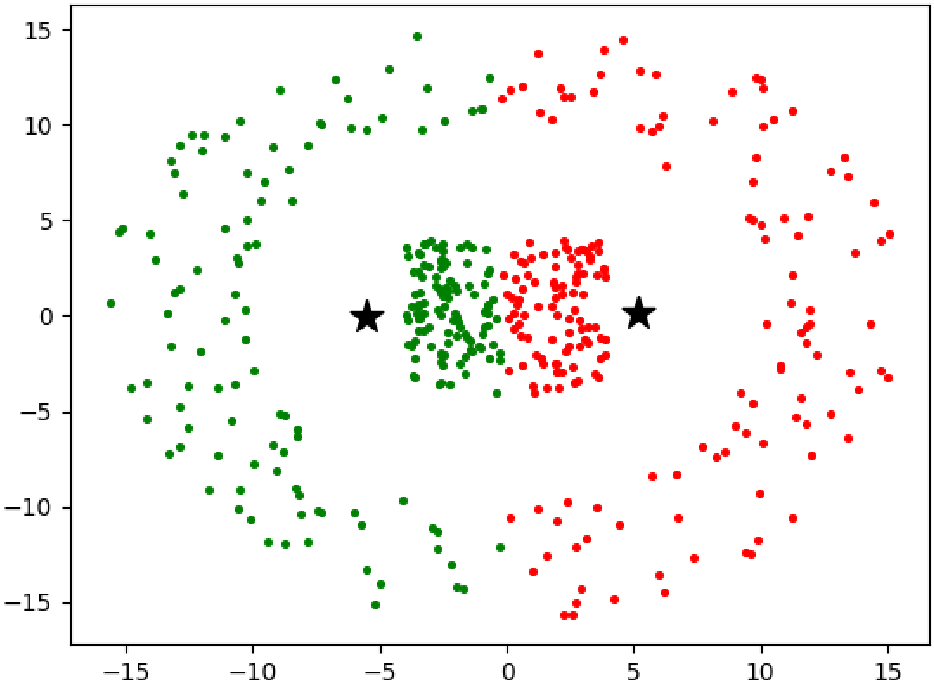
- Define “error” is the converge condition, which is the difference from this turn’s centroid to previous one.
- The data are clustered into k clusters by their distance to centroids. They will be clustered into the shortest one.

- Update centroids, by calculating the mean of all data belong to this cluster.
- Save clustering result figure in each iterator.



```
    C_old = np.zeros(C.shape)
    clusters = np.zeros(len(X))
    error = dist(C, C_old, None)

    iterator = 0
    while error != 0:
        iterator += 1
        for i in range(len(X)):
            distances = dist(X[i], C)
            cluster = np.argmin(distances)
            clusters[i] = cluster

        C_old = deepcopy(C)
        fig, ax = plt.subplots()
        for i in range(k):
            points = [X[j] for j in range(len(X)) if clusters[j] == i]
            C[i] = np.mean(points, axis=0)
            points = np.array(points)
            ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])
        error = dist(C, C_old, None)
        ax.scatter(C[:, 0], C[:, 1], marker='*', s=200, c='black')
        plt.savefig(output+str(iterator)+'.png')

```

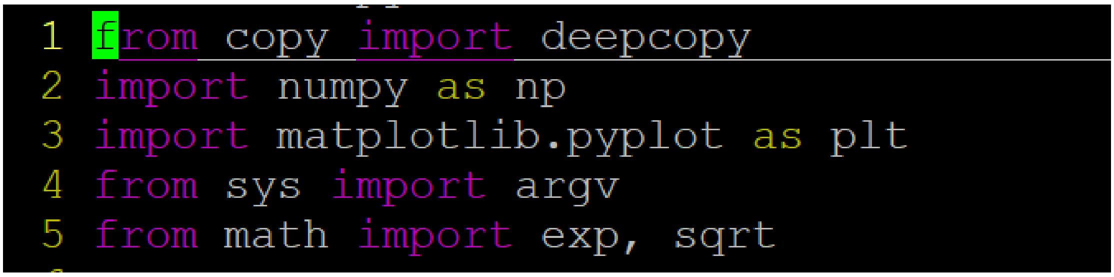
The result of test 2:



2. Kernel K-means clustering with RBF kernel

The import is similar to K-means.

“exp” is the exponent of e. “sqrt” is square root.

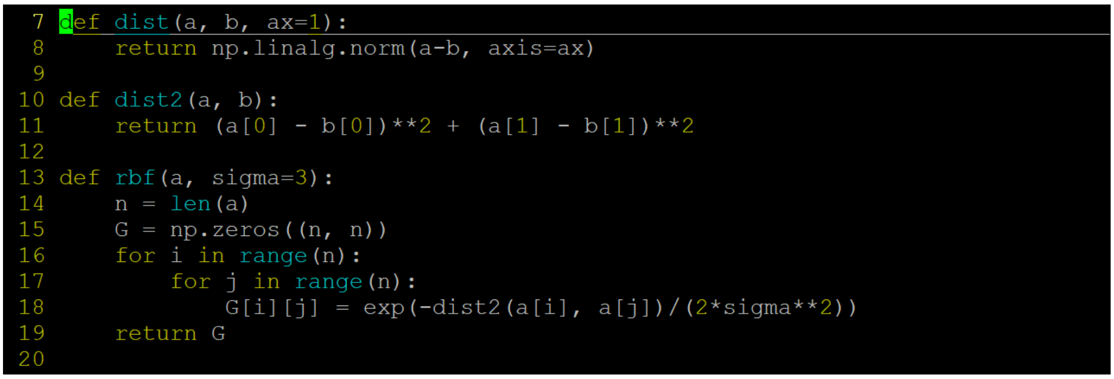


```python
from copy import deepcopy
import numpy as np
import matplotlib.pyplot as plt
from sys import argv
from math import exp, sqrt
```

“dist” function is the same of k-means.

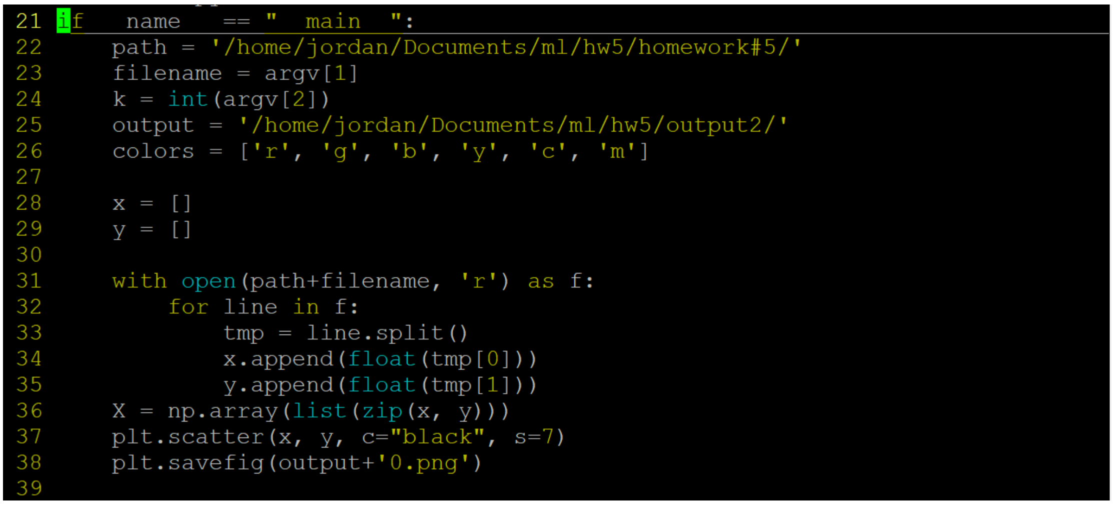
“dist2” function is used to calculate norm between two data points.

“rbf” function is used to calculate gram matrix G. Sigma default equals to 3.



```python
def dist(a, b, ax=1):
    return np.linalg.norm(a-b, axis=ax)

def dist2(a, b):
    return (a[0] - b[0])**2 + (a[1] - b[1])**2

def rbf(a, sigma=3):
    n = len(a)
    G = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            G[i][j] = exp(-dist2(a[i], a[j])/(2*sigma**2))
    return G

```

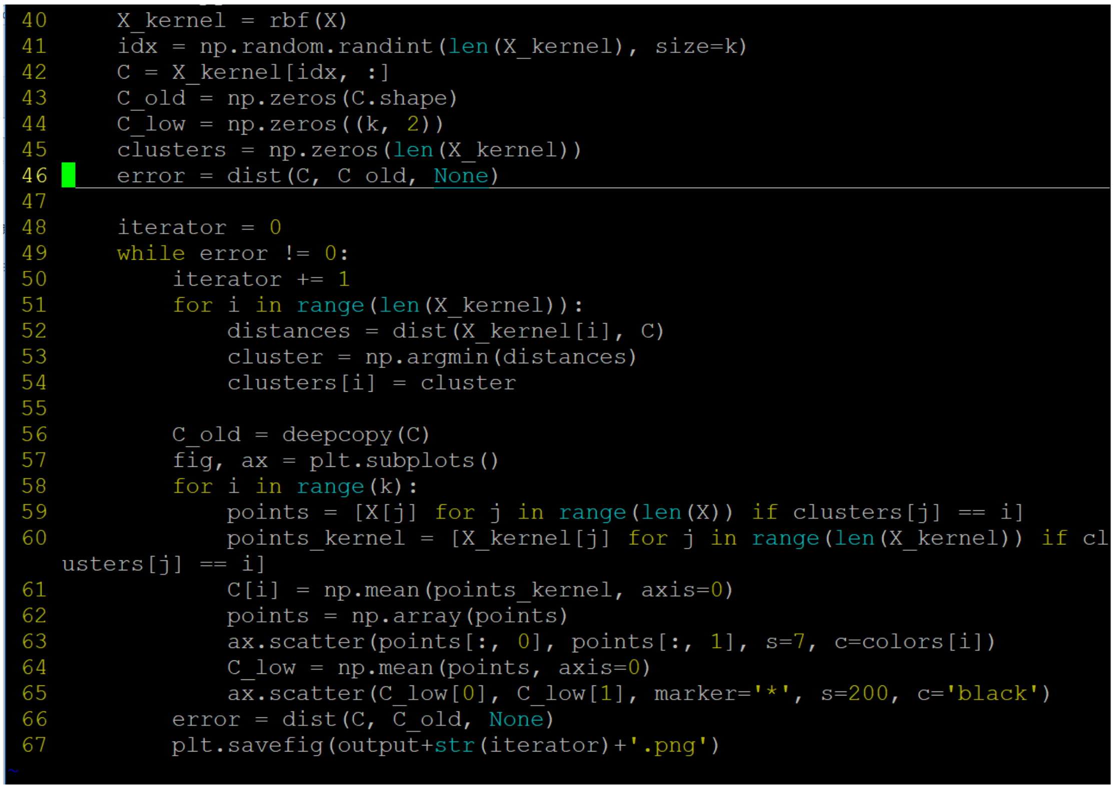
The first part of main function is same as k-means. Load and plot data.



```python
if __name__ == "__main__":
    path = '/home/jordan/Documents/ml/hw5/homework#5/'
    filename = argv[1]
    k = int(argv[2])
    output = '/home/jordan/Documents/ml/hw5/output2/'
    colors = ['r', 'g', 'b', 'y', 'c', 'm']

    x = []
    y = []

    with open(path+filename, 'r') as f:
        for line in f:
            tmp = line.split()
            x.append(float(tmp[0]))
            y.append(float(tmp[1]))
    X = np.array(list(zip(x, y)))
    plt.scatter(x, y, c="black", s=7)
    plt.savefig(output+'0.png')

```

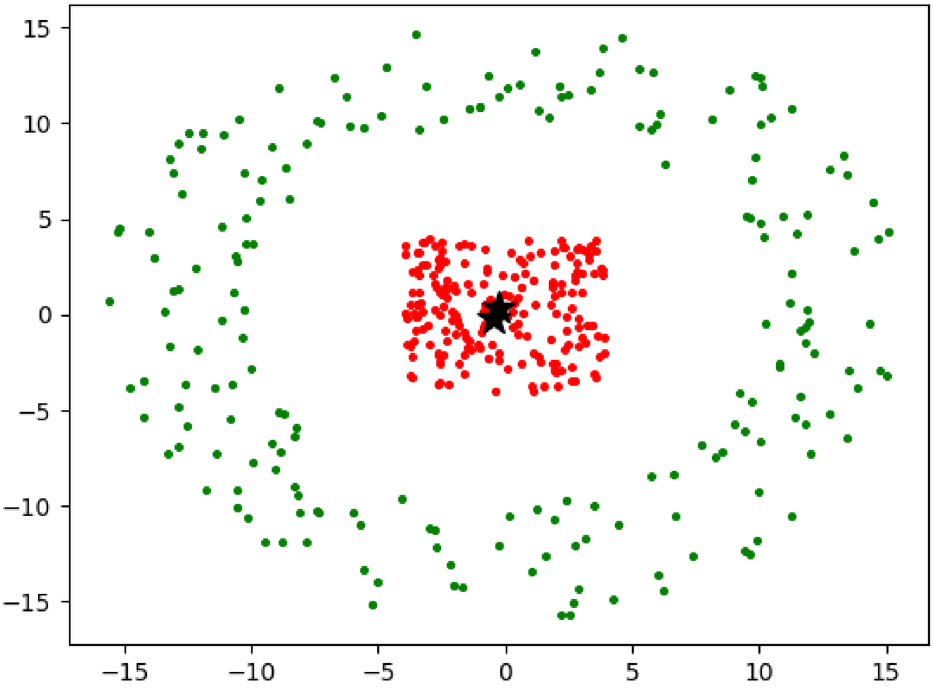
In the second part of main function, plotting data points into high dimensional space, and doing k-means with high dimension data.

- Project data into high dimensional space by RBF kernel, and randomly select k rows as initial centroids C. (C_old is last round’s C, and it is used to calculate error. C_low is the mean of corresponding original data points in each cluster.)
- For clustering phase, meaningful, it’s using high dimensional data to do k-means clustering. Therefore, the progress is the same.
- Plotting corresponding original data points and their centroids.



```
    X_kernel = rbf(X)
    idx = np.random.randint(len(X_kernel), size=k)
    C = X_kernel[idx, :]
    C_old = np.zeros(C.shape)
    C_low = np.zeros((k, 2))
    clusters = np.zeros(len(X_kernel))
    error = dist(C, C_old, None)

    iterator = 0
    while error != 0:
        iterator += 1
        for i in range(len(X_kernel)):
            distances = dist(X_kernel[i], C)
            cluster = np.argmin(distances)
            clusters[i] = cluster

        C_old = deepcopy(C)
        fig, ax = plt.subplots()
        for i in range(k):
            points = [X[j] for j in range(len(X)) if clusters[j] == i]
            points_kernel = [X_kernel[j] for j in range(len(X_kernel)) if clusters[j] == i]
            C[i] = np.mean(points_kernel, axis=0)
            points = np.array(points)
            ax.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])
            C_low = np.mean(points, axis=0)
            ax.scatter(C_low[0], C_low[1], marker='*', s=200, c='black')
        error = dist(C, C_old, None)
        plt.savefig(output+str(iterator)+'.png')
```

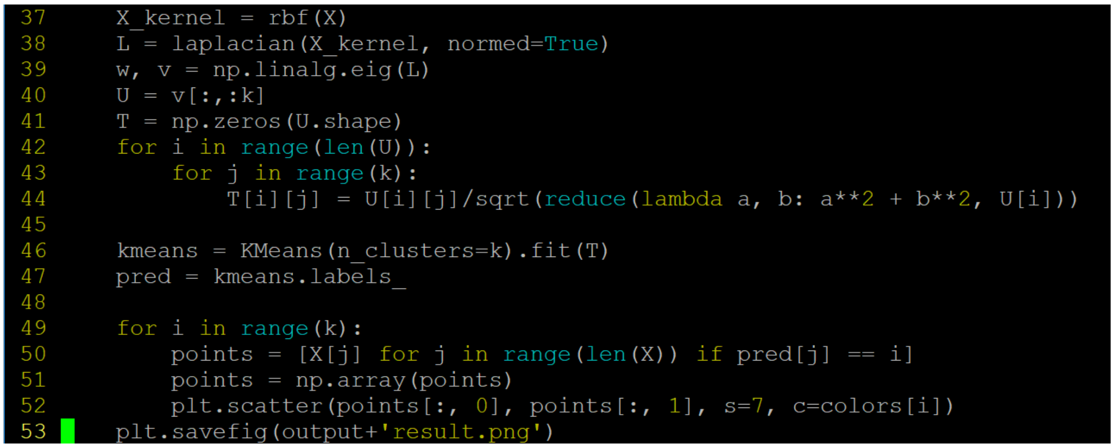
The result of test 2:



3. Spectral clustering

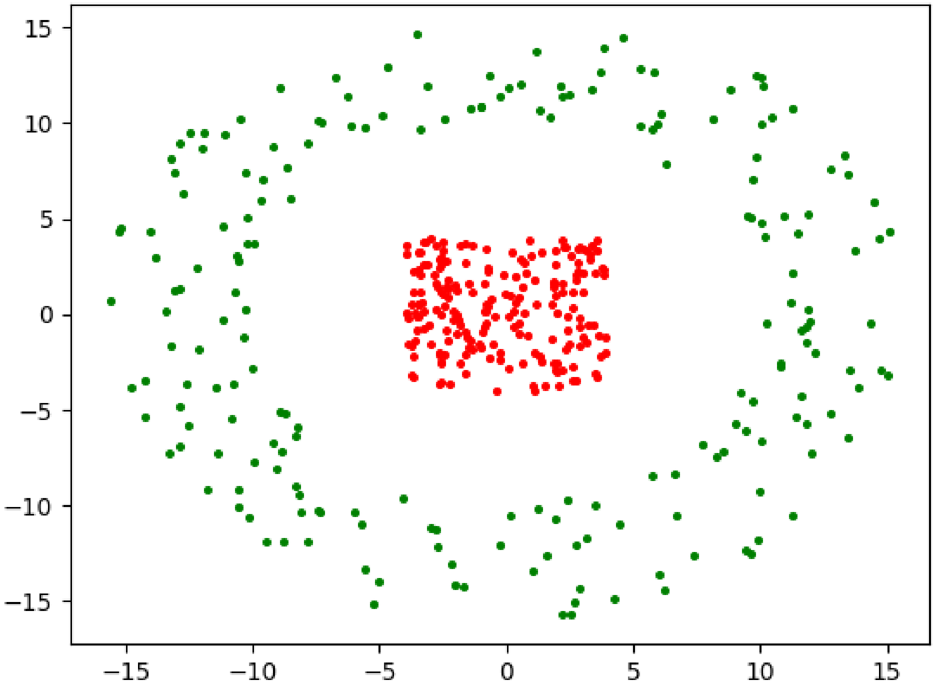
Import, “dist” function, and “rbf” function is skipped because they are almost the same with K-means and kernel K-means.

- Project data into high dimensional space with RBF kernel.
- Calculate normalized Laplacian matrix L.
- Eigenvalue matrix of L is called v, and select first k columns as U.
- T is the row normalized matrix of U.
- Do k-means with matrix T



```
    X_kernel = rbf(X)
    L = laplacian(X_kernel, normed=True)
    w, v = np.linalg.eig(L)
    U = v[:,:k]
    T = np.zeros(U.shape)
    for i in range(len(U)):
        for j in range(k):
            T[i][j] = U[i][j]/sqrt(reduce(lambda a, b: a**2 + b**2, U[i]))

    kmeans = KMeans(n_clusters=k).fit(T)
    pred = kmeans.labels_

    for i in range(k):
        points = [X[j] for j in range(len(X)) if pred[j] == i]
        points = np.array(points)
        plt.scatter(points[:, 0], points[:, 1], s=7, c=colors[i])
    plt.savefig(output+'result.png')
```

The result of test 2:

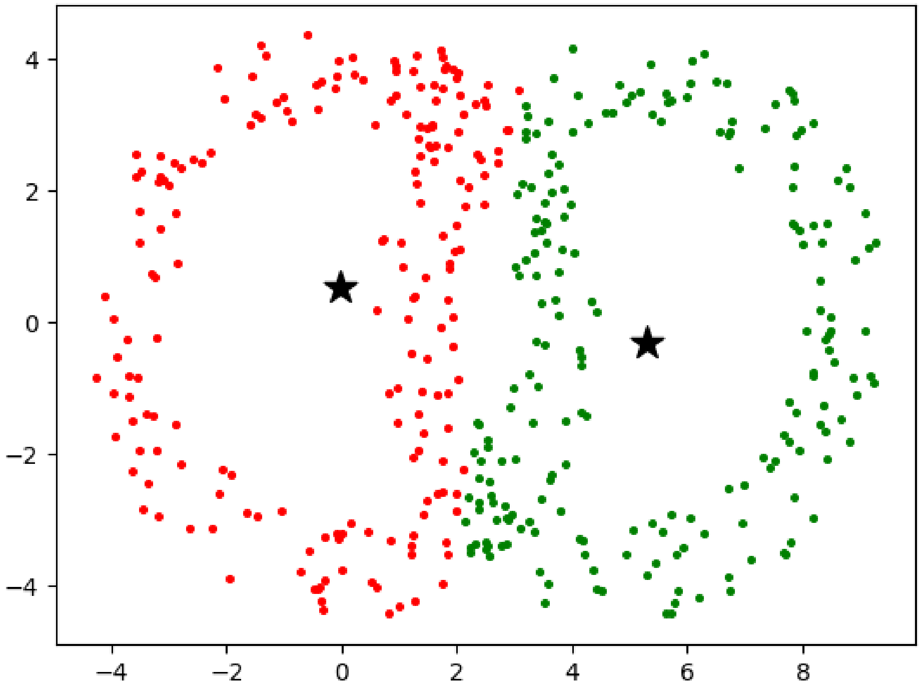


4. More clusters result

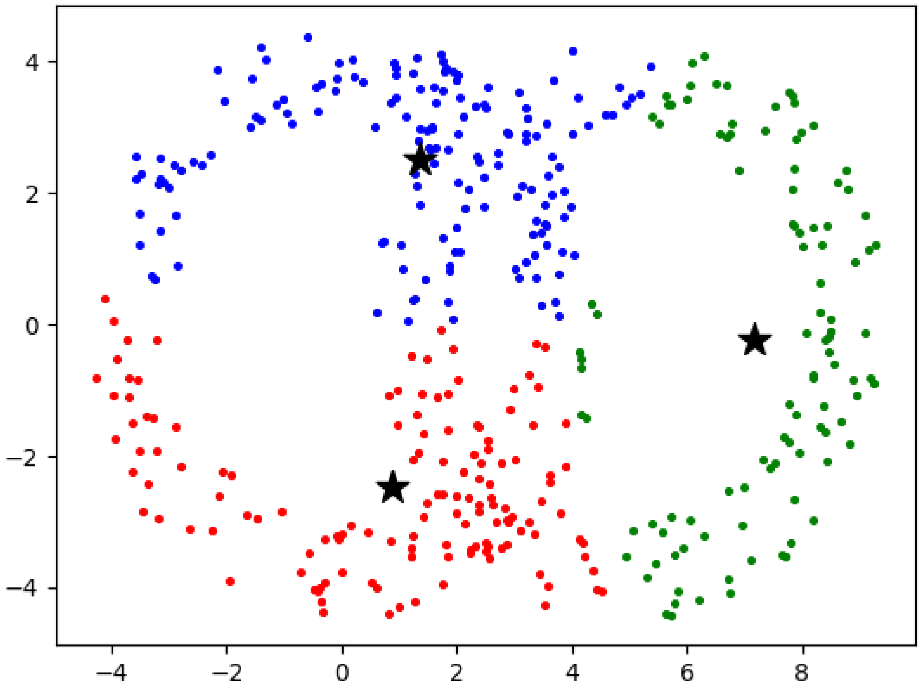
K-means:

Data = test_1

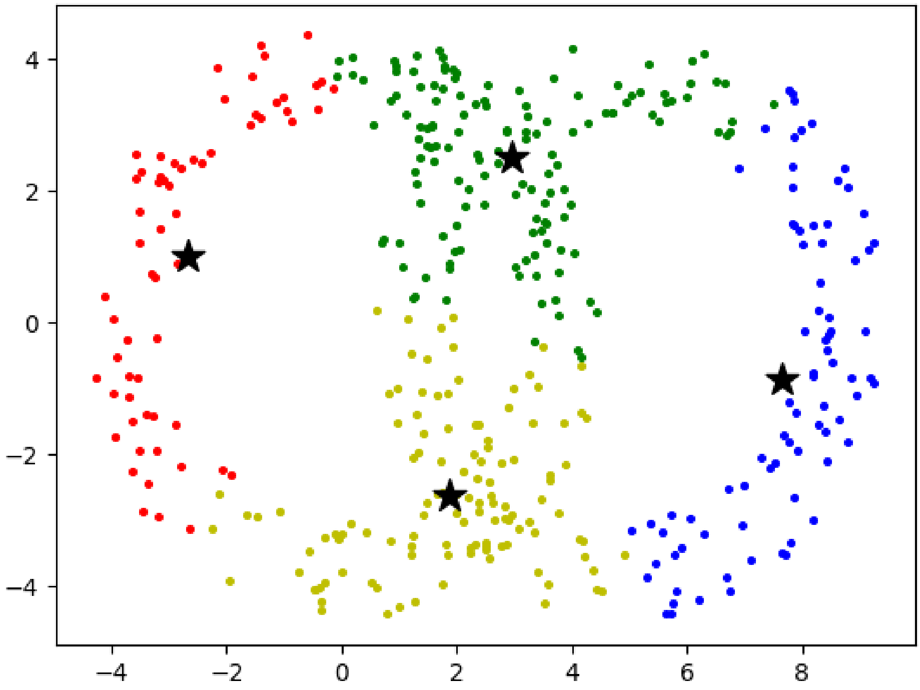
Cluster = 2



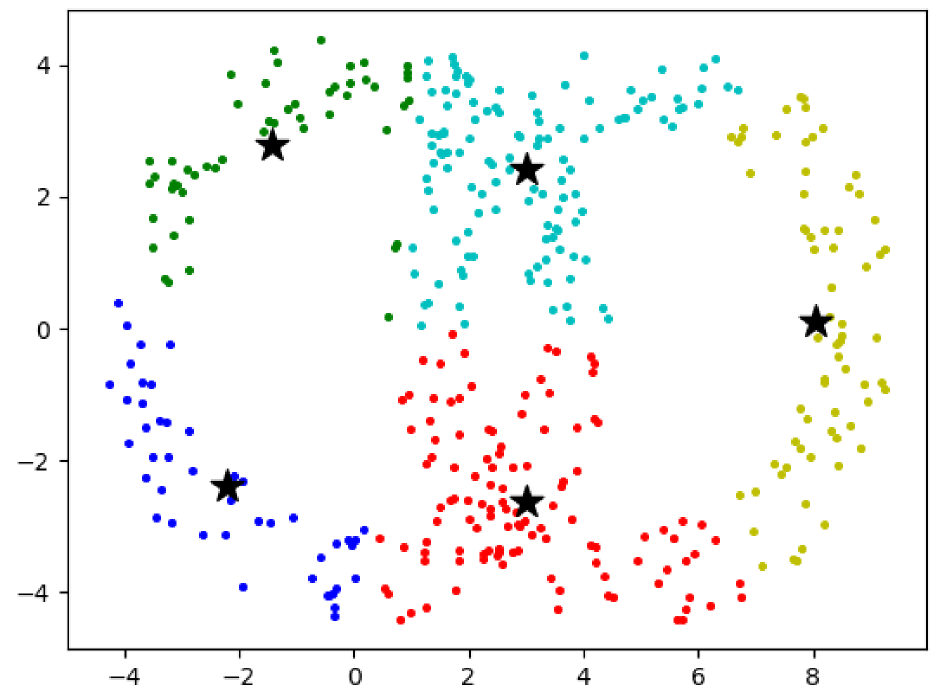
Cluster = 3



Cluster = 4

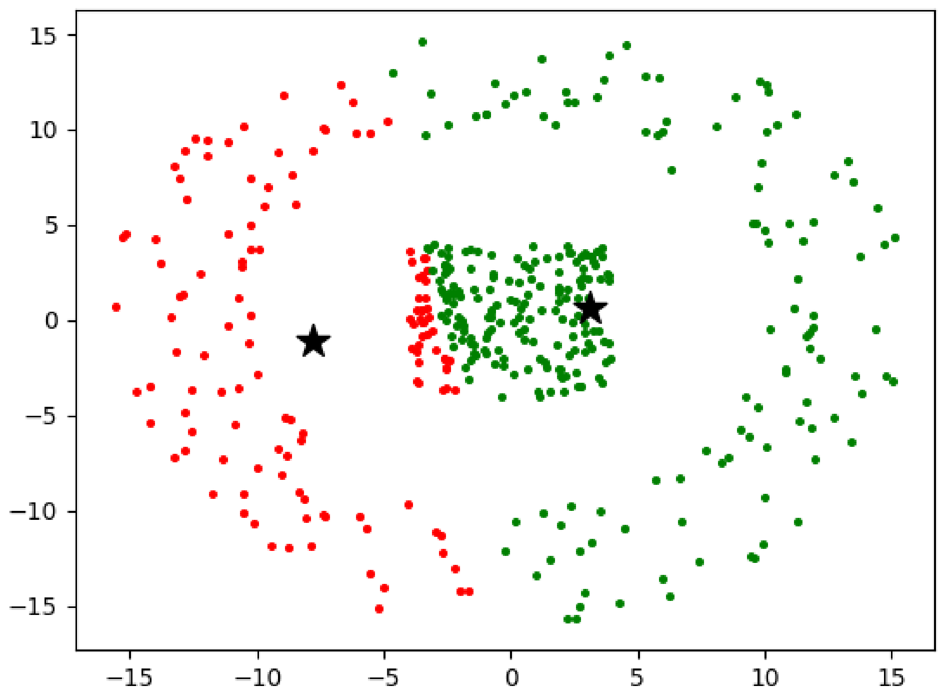


Cluster = 5

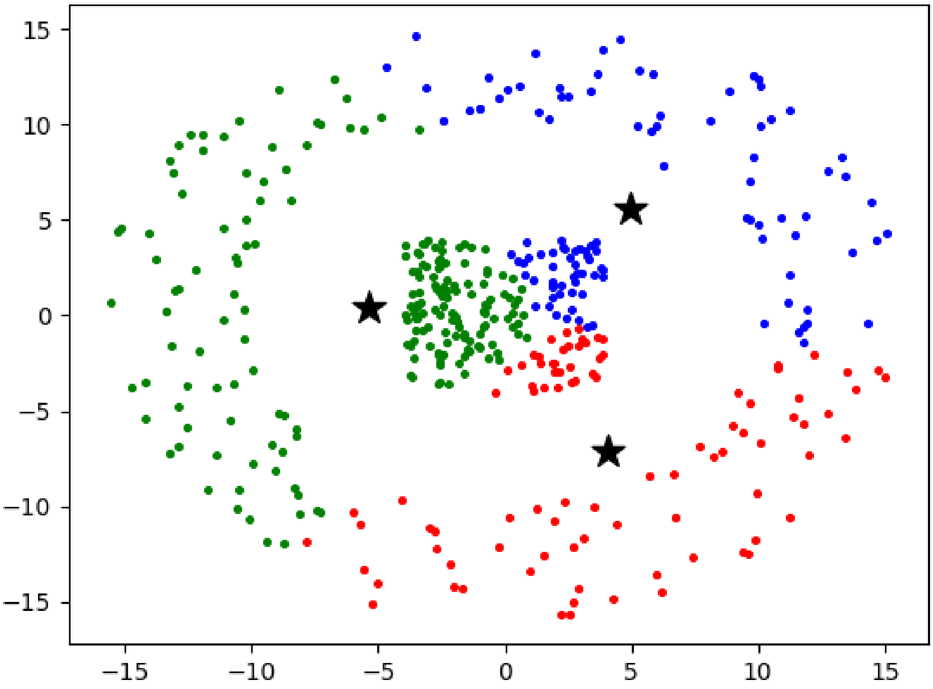


Data = test_2

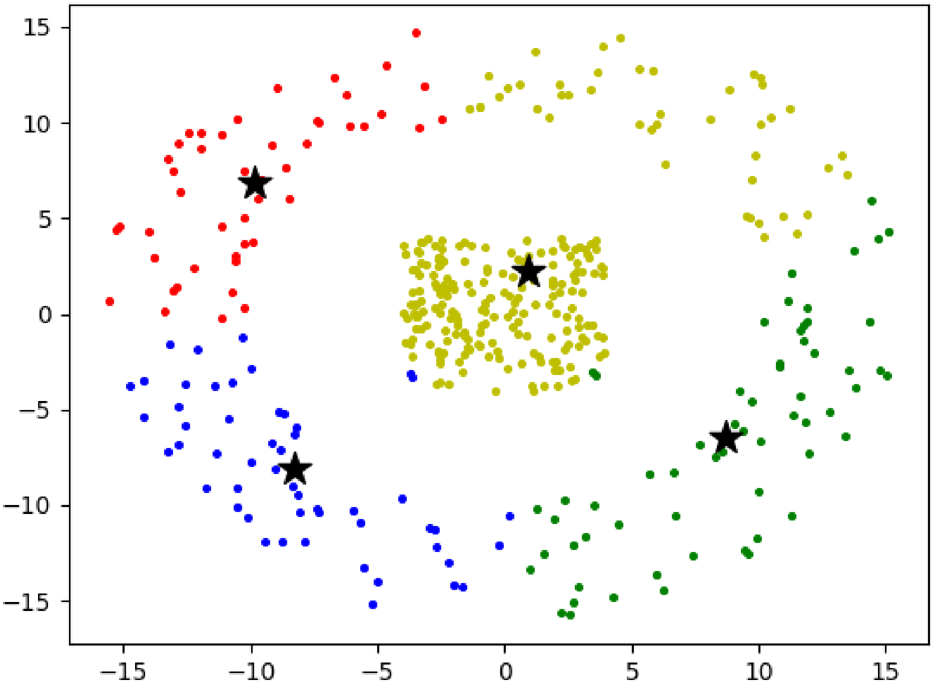
Cluster = 2



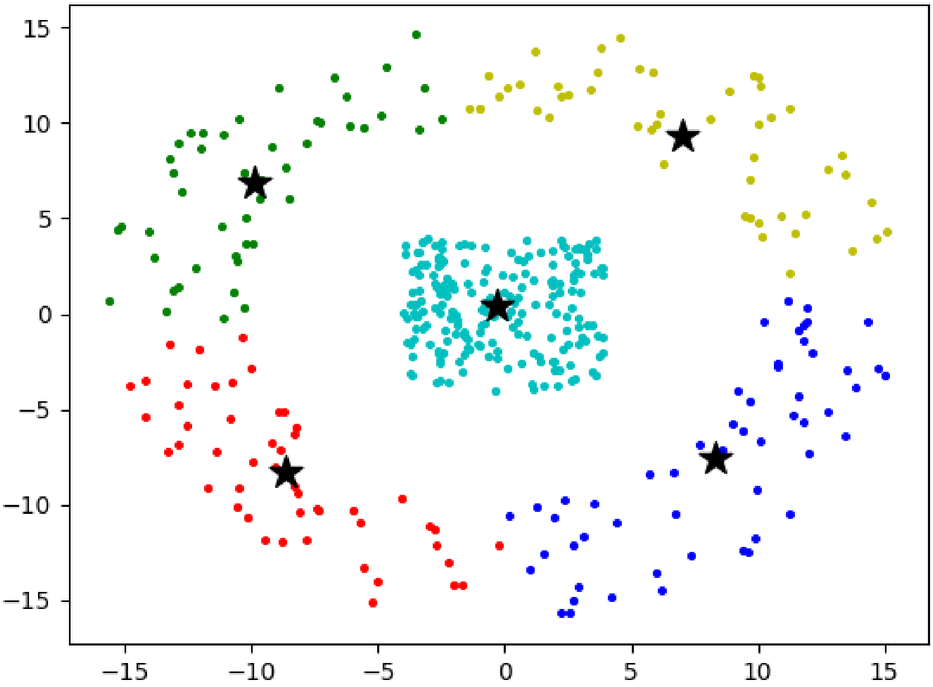
Cluster = 3



Cluster = 4



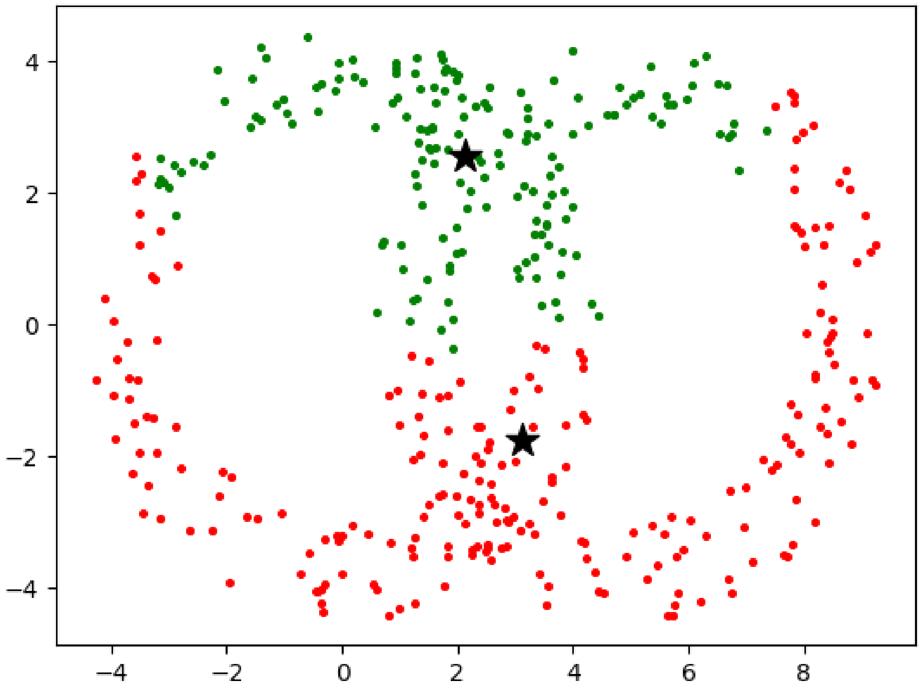
Cluster = 5



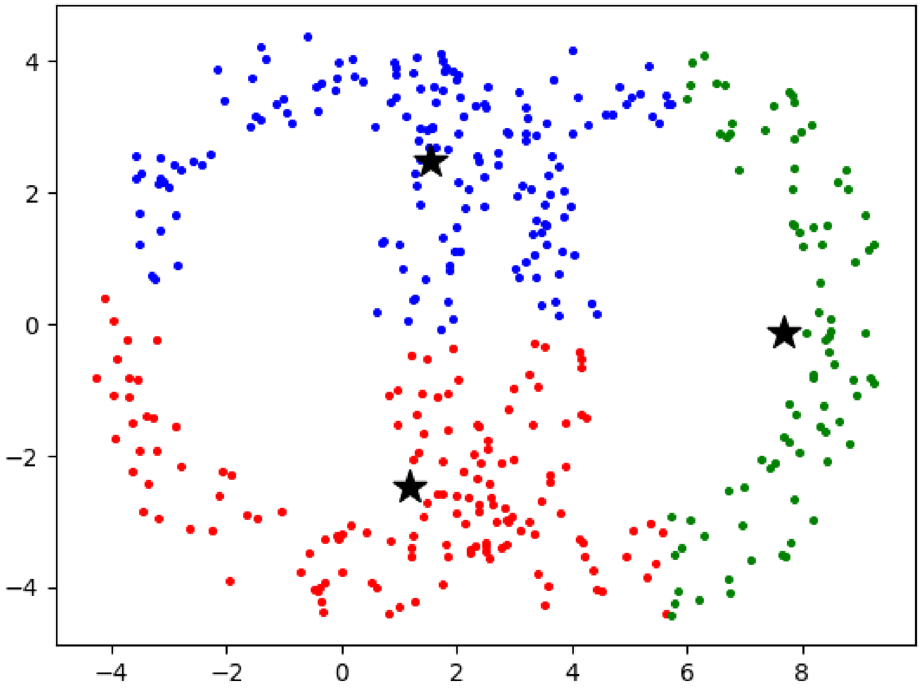
Kernel K-means:

Data = test_1

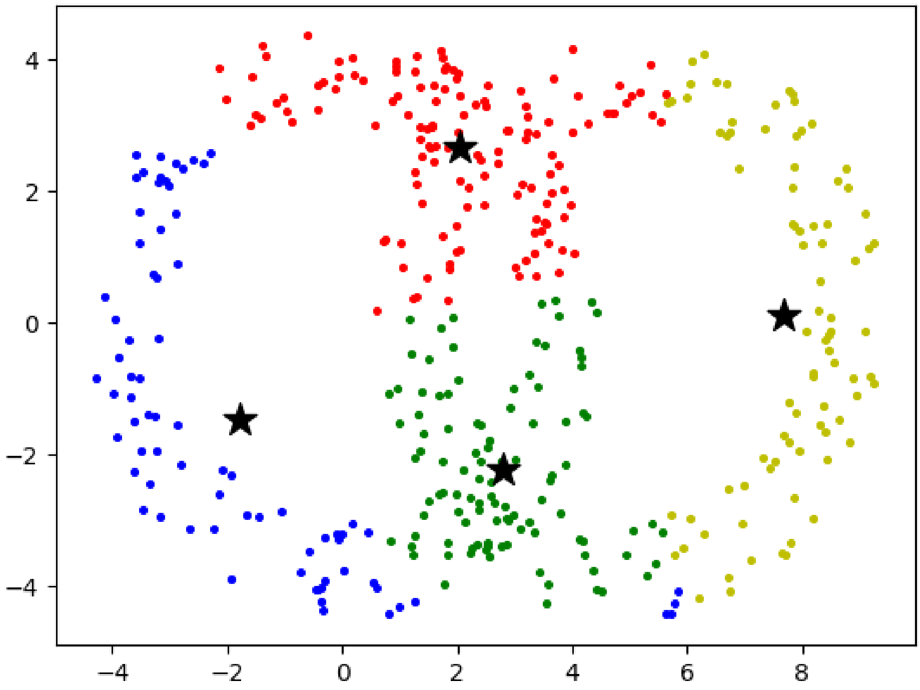
Cluster = 2



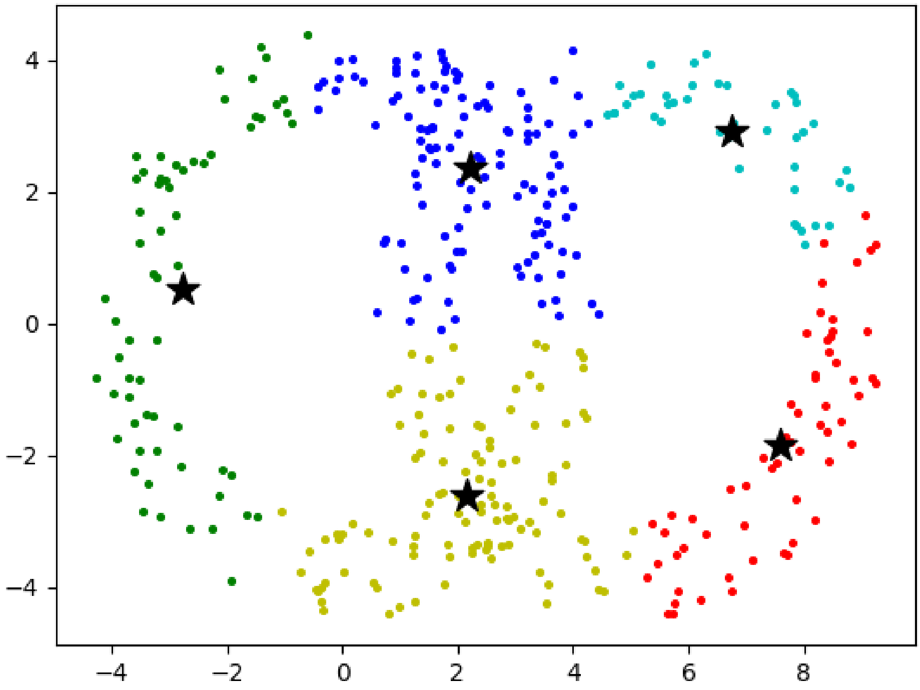
Cluster = 3



Cluster = 4

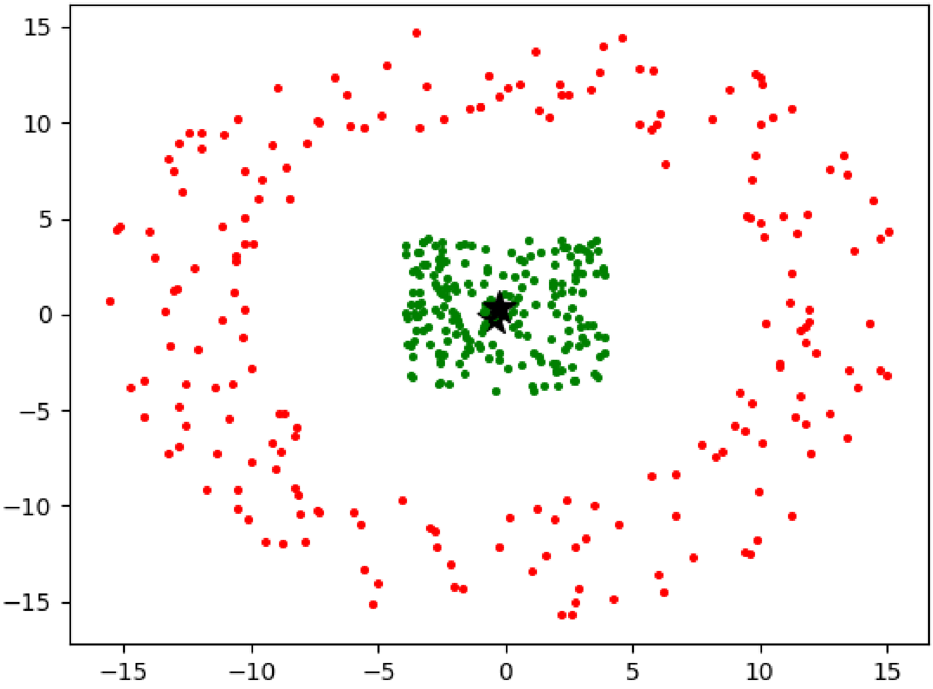


Cluster = 5

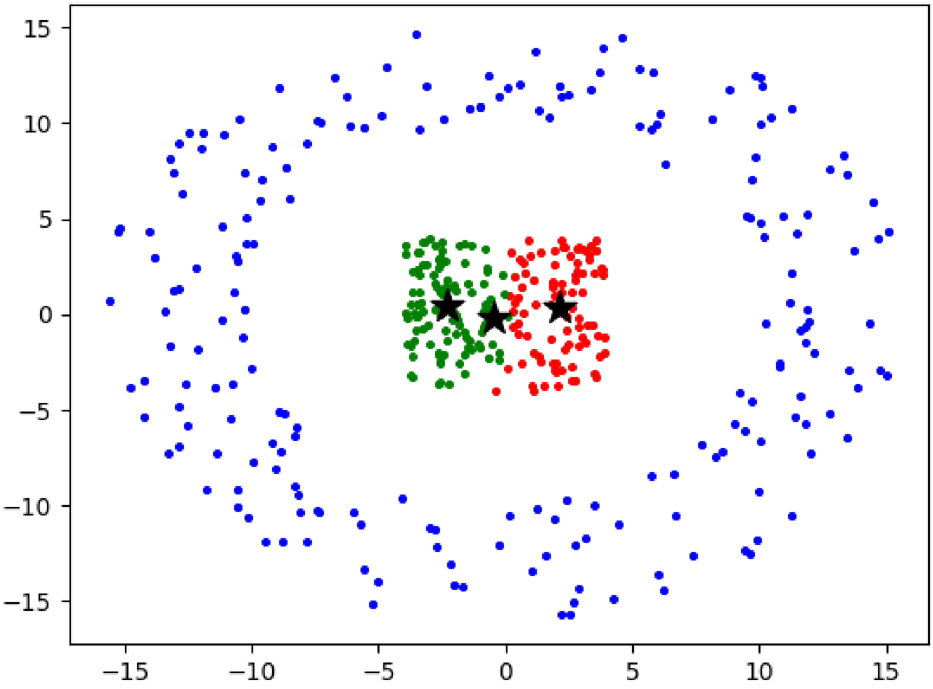


Data = test_2

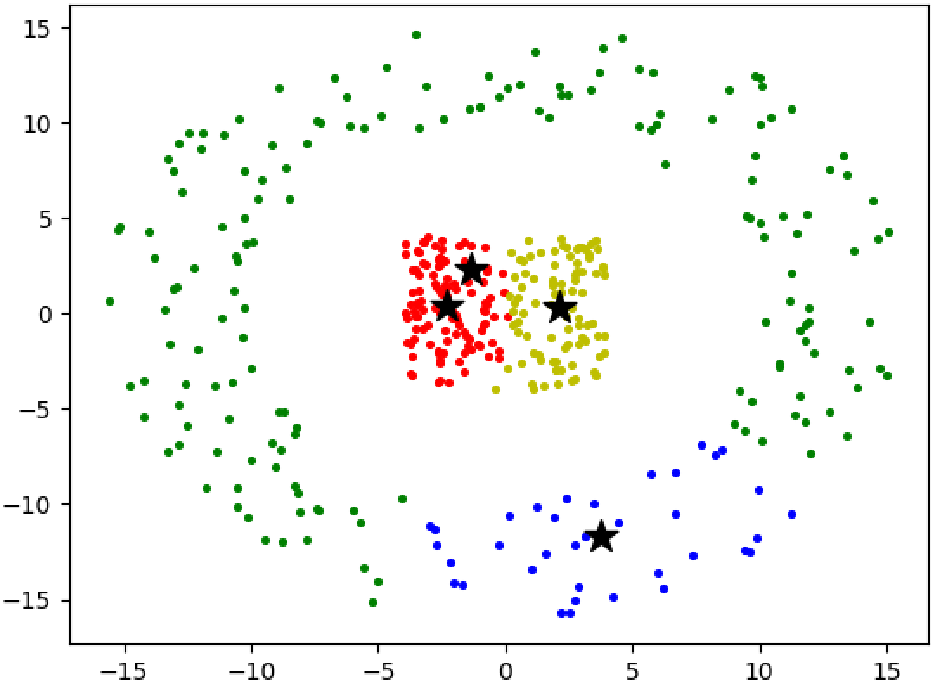
Cluster = 2



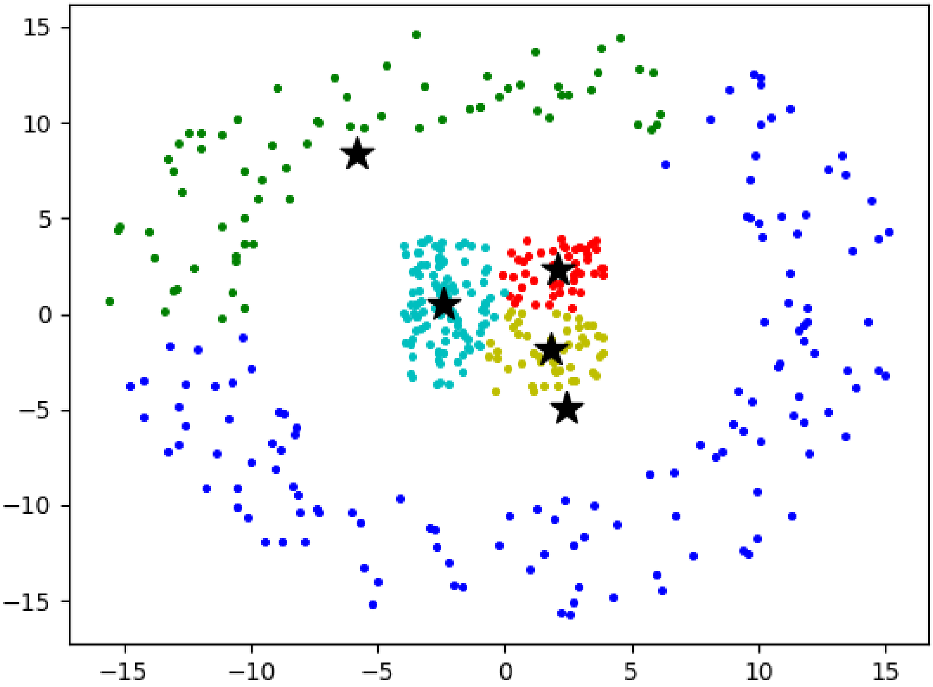
Cluster = 3



Cluster = 4



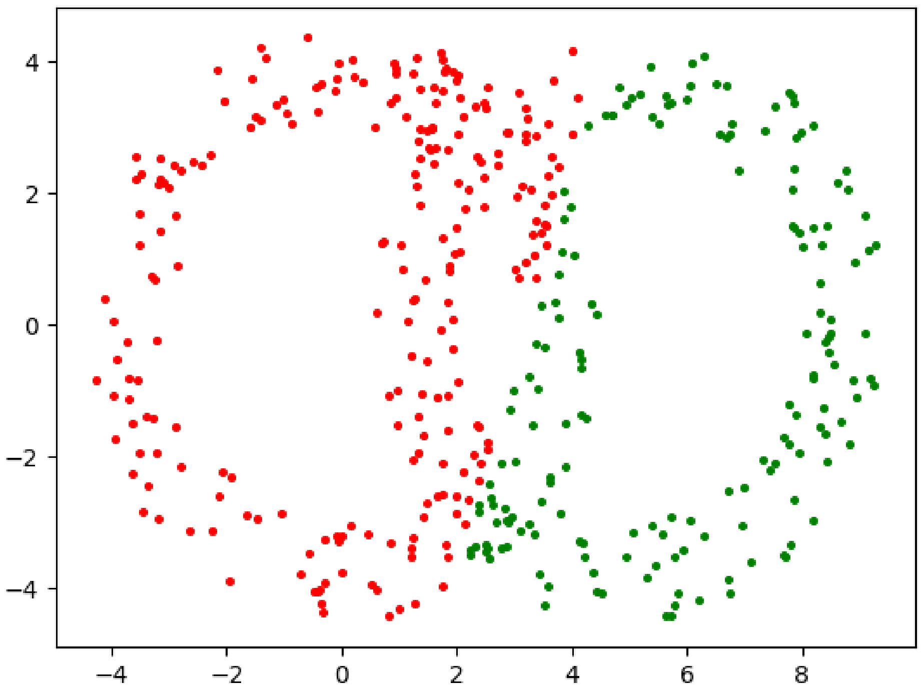
Cluster =5



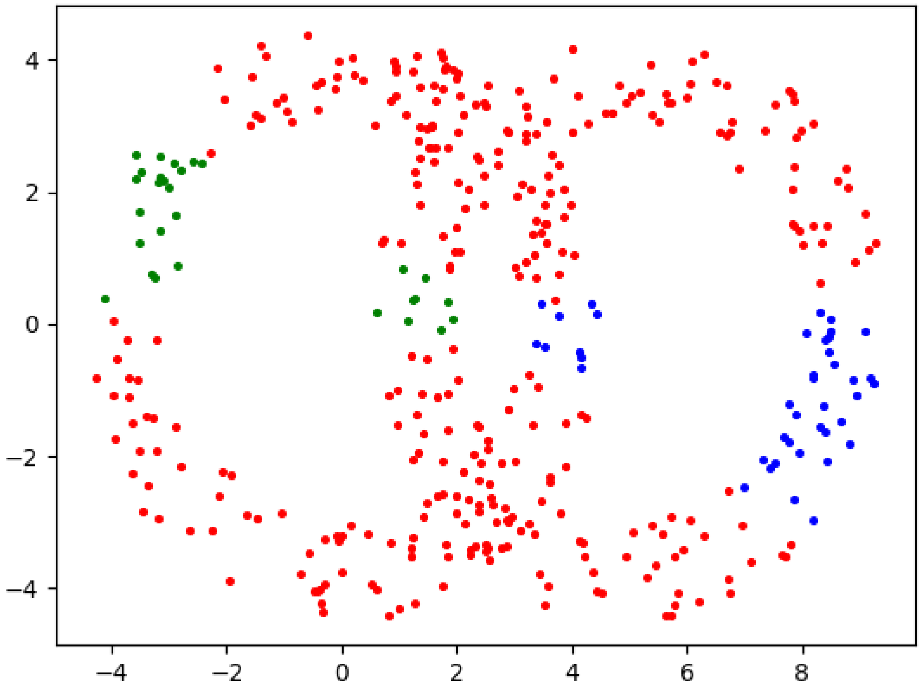
Spectral clustering:

Data = test_1

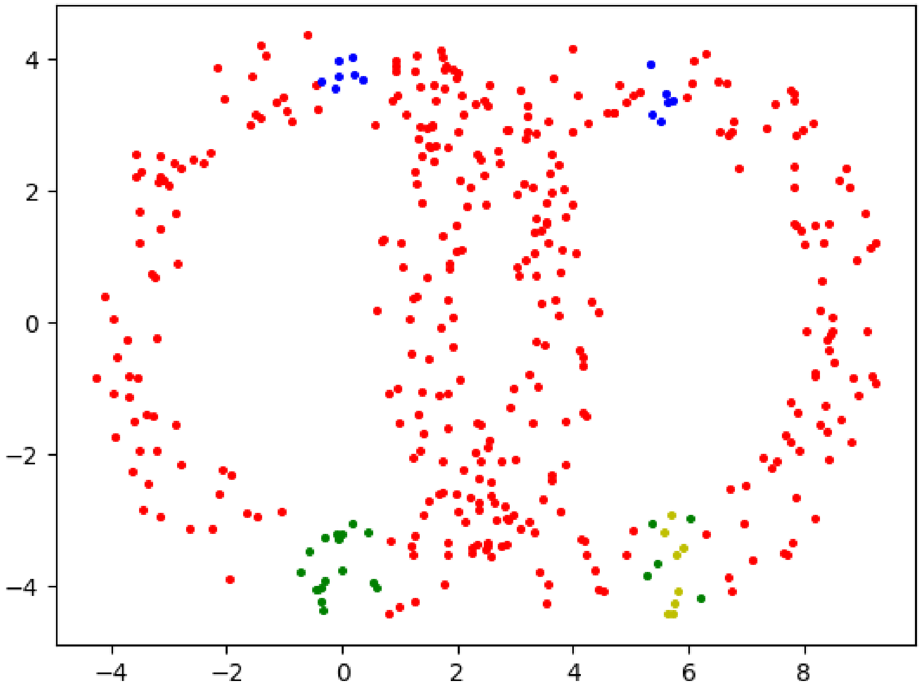
Cluster = 2



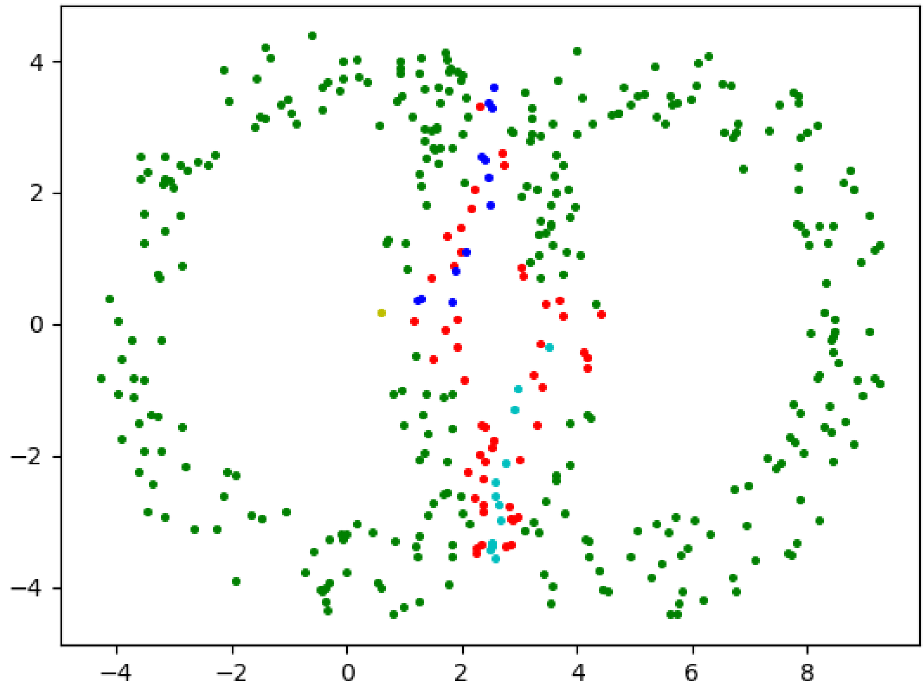
Cluster = 3



Cluster = 4

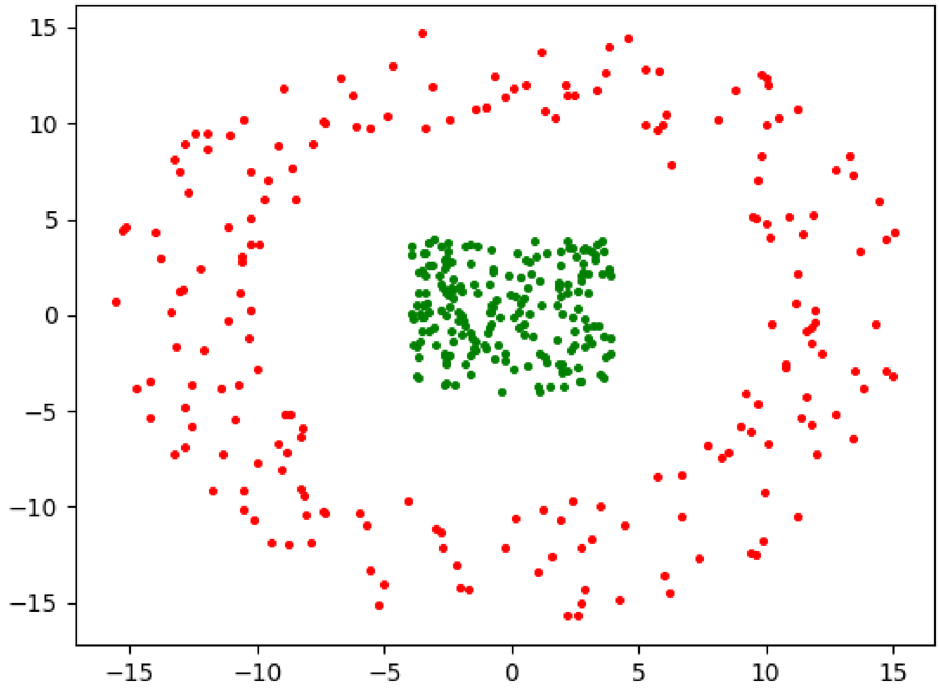


Cluster = 5

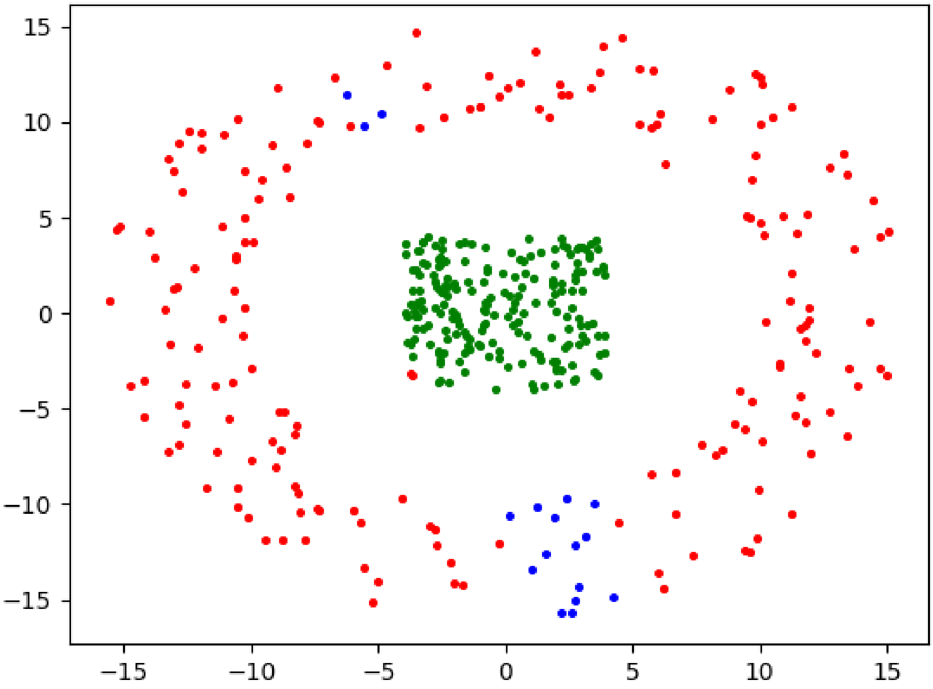


Data = test_2

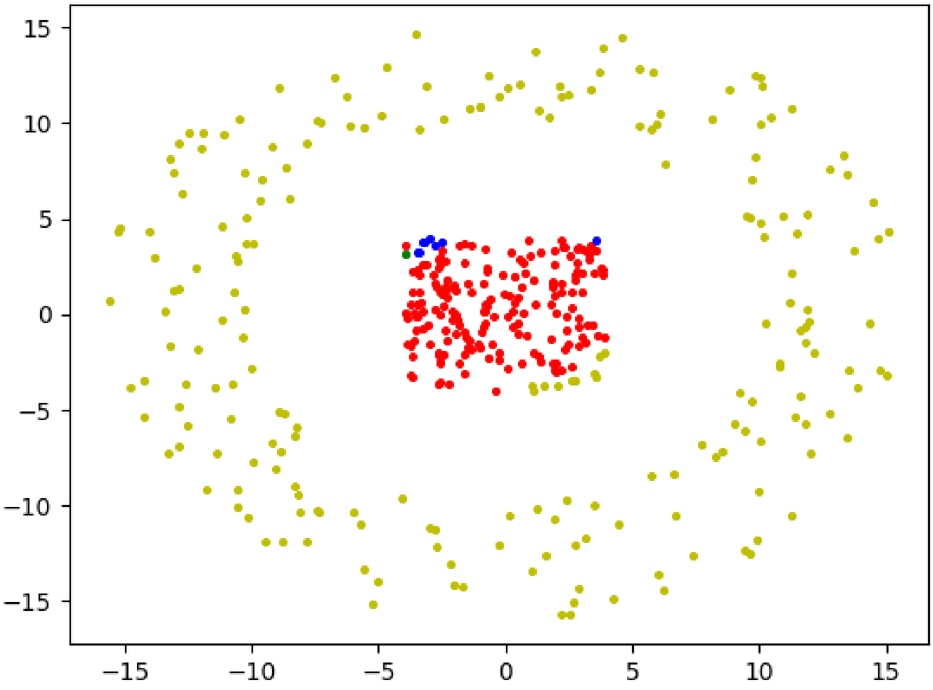
Cluster = 2



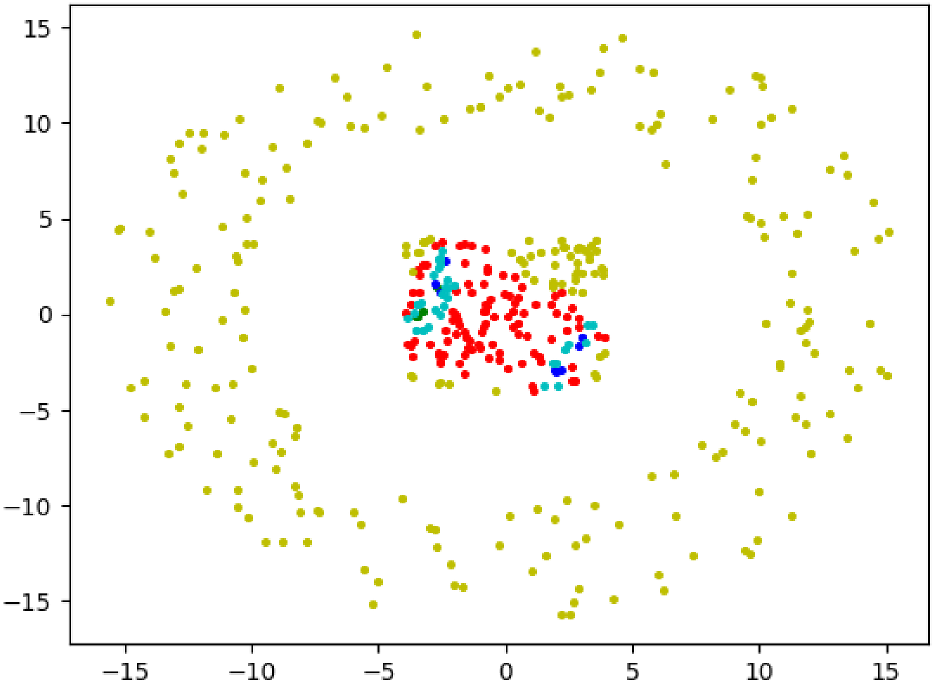
Cluster = 3



Cluster = 4



Cluster = 5


15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15