北京航空航天大學
BEIHANG UNIVERSITY

Pattern Recognition and Machine Learning Assignment

院（系）名称	自动化科学与电气工程学院
专业名称	模式识别与智能系统
学生学号	14031259
学生姓名	孔昭宁

2017 年 5 月 9 日

Spiral and Concentric Dataset Clustering

1. Introduction

Unsupervised learning is an indispensable domain in pattern recognition and machine learning. In contrast with supervised learning, the samples in unsupervised learning does not come with a label with which we can classify them. However, we can instead analyze them on the basis of their intrinsic traits, and clustering is the most commonly used method in unsupervised learning.

Clustering can be an individual task, but it also serves as dataset preprocessing algorithms in some supervised learning tasks, in some cases. Many clustering algorithms have been designed to tackle different tasks.

K-means is the simplest and most commonly used clustering algorithm. It is comparably effective and very easy to implement. It belongs to the regime of prototype-based clustering.

For the other two task we are handling, nevertheless, this algorithm does not perform well, as the samples which should be assigned the same labels are sparsely located in space. Therefore, we introduce the following algorithms to correctly cluster the dataset.

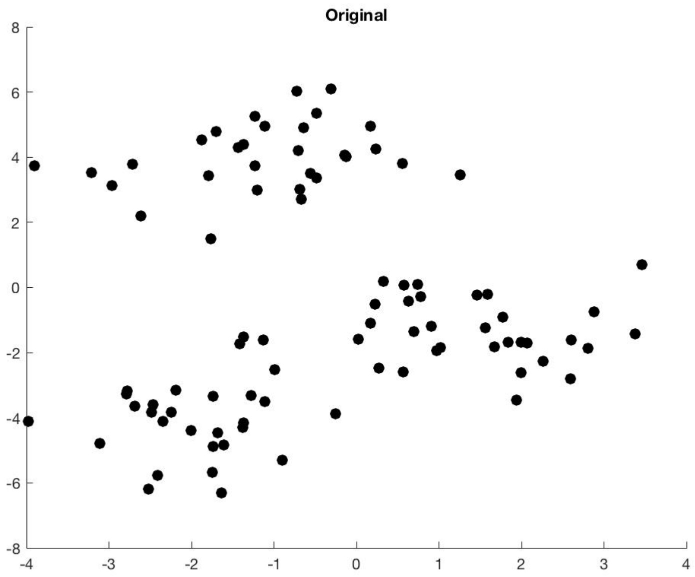
Isometric Mapping (ISOMAP) algorithm assumes that the higher dimension space is not necessarily a linear transformation of the lower ones, whereas the neighboring samples are subject to Euclidean distance. Thus, we can nonlinearly reduce the dataset into lower dimensions based on their distances, where they can be easily separated into two or more types.

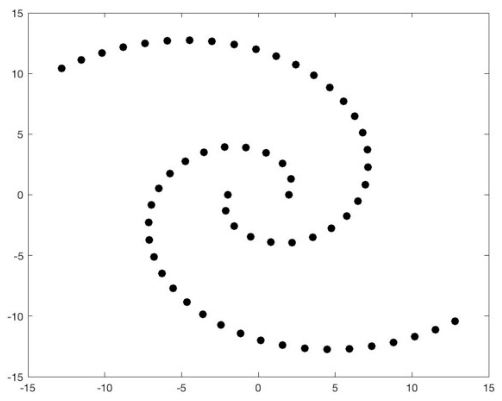
Density-based Clustering algorithm assumes that the relation between the samples can be determined by how close they are. Therefore, by calculating the connectivity between the samples, we can conduct depth first search (DFS) or breadth first search (BFS) on the dataset, which selects a group of samples which are closely located. The number of clusters varies with the threshold assigned at the beginning of the algorithm, which we can modify to regularize the behavior of the algorithm.

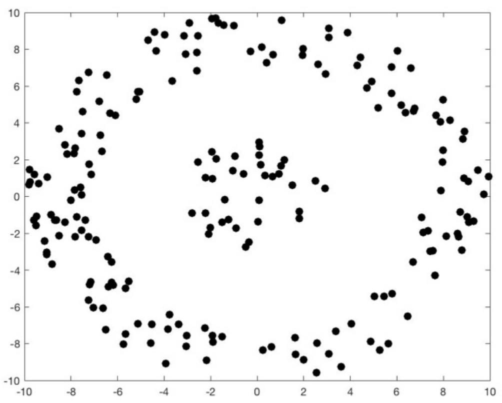
2. Goals

The first goal of this assignment is to cluster a dataset which appears in 3 clusters, each of which subject to 2-d Gaussian distribution.

The other goal of this assignment is to cluster two datasets, each of two classes. The first dataset is a spiral dataset, while the second one is of two concentric circles.







3. K-means

Given dataset $D = \{x_1, x_2, \dots, x_m\}$, the K-means algorithm clusters them into $C = \{C_1, C_2, \dots, C_k\}$ by minimizing the error term

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|_2^2$$

where $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ is the mean vector of cluster C_i.

It is NP-hard to minimize the error term above. However, there exists heuristic algorithms that efficiently converges to local optima. A simplified description of the algorithms is as follows:

1. **Assignment Step:** Assign n centroids within the sample space, where n is the number of clusters we wish to classify.
2. **Update Step:** Each sample point is temporarily assigned to the centroid to which it is closest to. Samples assigned to the same centroid temporarily

belong to the same cluster. Each centroid is then moved towards the mean location of all samples belonging to this cluster. Iterate until convergence.

4. Isometric Mapping

ISOMAP is a nonlinear dimension reduction method. A simplified description of the ISOMAP algorithm is described below.

1. Determine the neighbors of each point within some fixed radius
2. Construct a neighborhood graph. Each point is connected to points within some fixed radius with the length equal to Euclidean distance. Other points are not connected.
3. Compute shortest path between every two nodes with Dijkstra's Algorithm
4. Computer the lower-dimension embedding with Multidimensional Scaling (MDS) algorithm.

Having established the lower-dimension embedding of the dataset, the dataset can be easily clustered in lower-dimension.

5. Density-based Clustering

Density-based Spatial Clustering of Applications with Noise (DBSCAN) is a data clustering algorithm based on the connectivity between the sample points.

DBSCAN algorithm classifies sample points as three types: Core points, reachable points and outliers, as follows:

- A point p is a core point if at least $minPts$ points are within the distance ε. Those points are said to be *directly reachable* from p. No points are *directly reachable* from non-core point.
- A point q is reachable from p if there is a path $p_1, p_2, \ldots, p_n$ with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i.
- All points not reachable from other points are outliers.

A cluster then satisfies the following two properties:

- All points within the cluster are mutually density-connected
- If a point is density-reachable from any point from any point of the cluster, it is part of the cluster as well.

A simplified description of the DBSCAN algorithm is as follows.

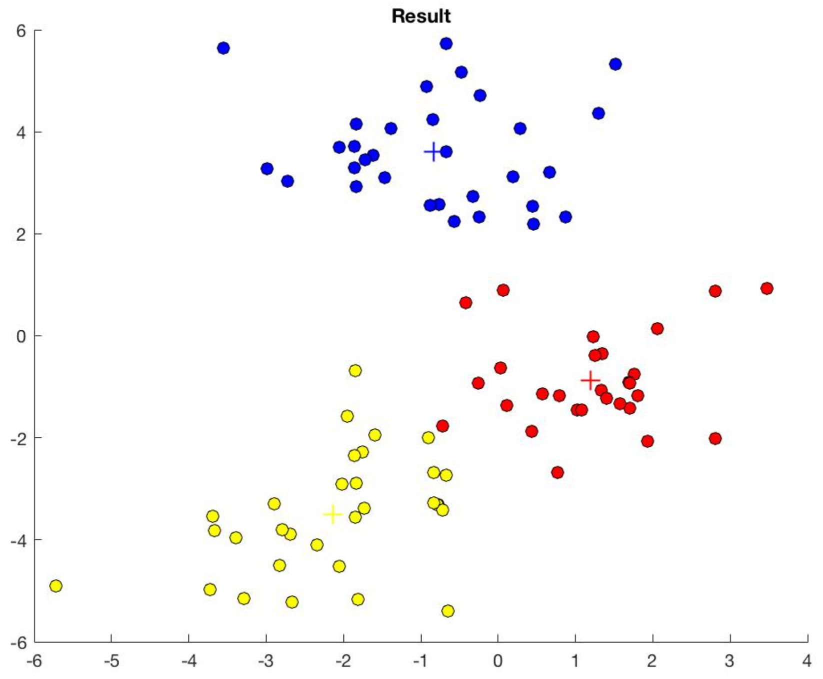
1. Compute the distance between any two points, and select the core points.

2. For every core point, traverse all points in dataset that are density reachable, which belong to the same cluster.
3. Iterate step 2 until all points have been visited, each time a new cluster is produced.

6. Experiment Results

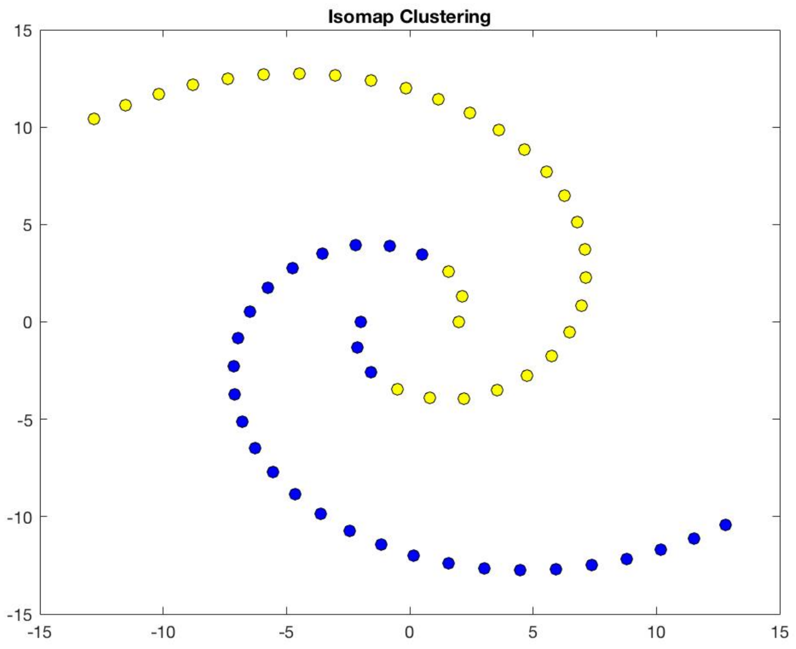
6.1 K-means Algorithm

The clustered dataset is as follows, where the '+' sign corresponds to the location of the centroids after having successfully clustered the dataset.



6.2 Isometric Mapping (ISOMAP)

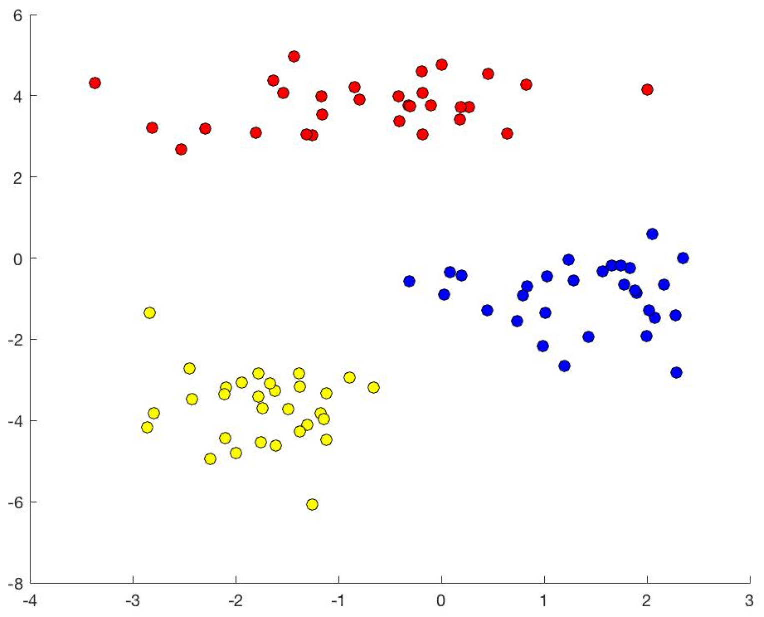
The spiral dataset can be well clustered with the ISOMAP algorithm. The original 2-dimensional dataset is nonlinearly reduced to 1 dimension with the MDS (Multidimensional Scaling) algorithm. Having reduced the dimension of the dataset, we can subsequently separate the 1-d dataset with an arbitrary threshold. As for this dataset, I assigned those samples with a positive 1-d coordinate with one label, and the negative ones with another. The following plot depicts the result of this algorithm.



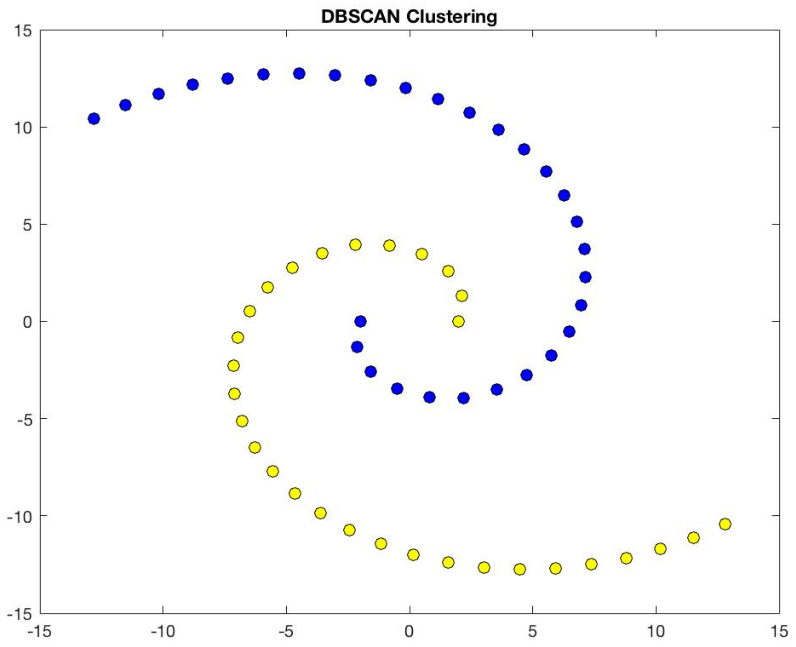
As it can be observed in the plot above, however, a few samples in the center are not correctly clustered as expected. This is probably because the samples in the center messed up during dimension reduction, as their relationships are difficult to resemble with one dimension coordinates.

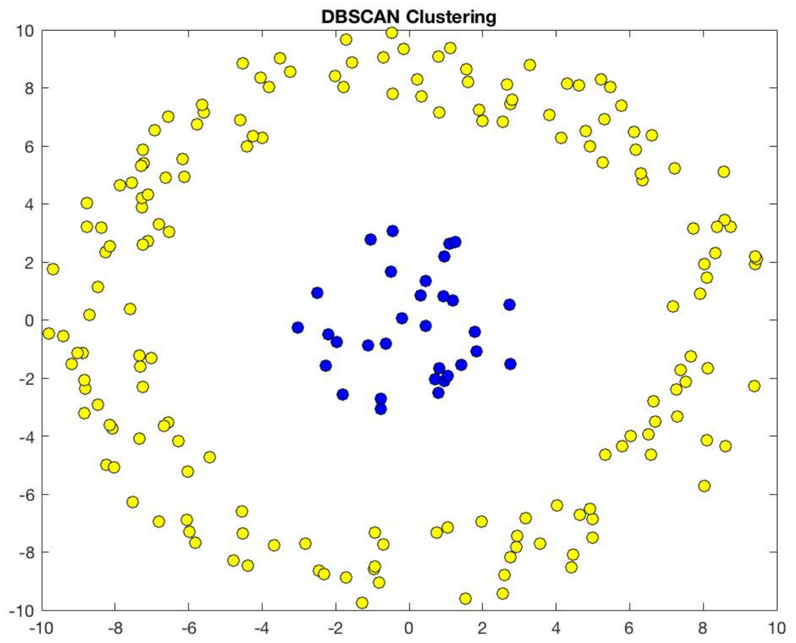
6.3 Density-based Clustering (DBSCAN)

DBSCAN algorithm can be a substitute for k-means algorithm for our first dataset. The result is as follows.



In addition, both the spiral and concentric dataset can be well clustered with the Density-based Clustering (DBSCAN) algorithm. The results are as follows.





As it can be observed in the two plots above, all the samples have been correctly clustered with DBSCAN algorithm, including the samples that the ISOMAP algorithm fails to cluster.

7. Conclusion

K-means successfully clustered our dataset into 3 clusters. As a prototype-based clustering algorithm, it works well on dataset that have intrinsic cluster-like distributions. The drawback, however, is that the number of clusters we wish to assign are fixed. Nevertheless, it is easy to implement and trivial in computation.

It turns out that DBSCAN is a better algorithm for the other datasets, as it correctly clusters all samples. This algorithm is easy to implement, efficient to run, thus catering to many real-time or online clustering tasks. In addition, it is fairly intuitive by clustering the samples based on their mutual connectivity, without having to nonlinearly transform or dimension-reduce the coordinates with which we observe the samples with.

Admittedly, slight variations on the spiral dataset could sway the effectiveness of the DBSCAN algorithm. For example, if the sample points in the center of the spiral dataset is close enough to establish the connectivity between the two curves, the DBSCAN algorithm would eventually consider all samples of the dataset as one cluster.

That's where the ISOMAP algorithm comes in. With two curves close enough, they tend to blend into one curve of 'S' shape. In this case, the ISOMAP algorithm would dimension-reduce the dataset, along this curve, into a 1-dimensional dataset. As for the specific dataset we are dealing with, which consists of equal number of sample from the two classes, the midpoint of the dimension-reduced dataset might effectively serve as a boundary to separate the dataset into two groups.

8. Code

8.1 K-means

```
%% Generate Dataset
mean1_init = [-2 -3.5];
cov1_init = [1 0.5; 0.5 2];
mean2_init = [-1 4];
cov2_init = [2 0; 0 1];
mean3_init = [1.5, -1];
cov3_init = [1 0; 0 1];
num_of_samples_per_class = 30;
w1 = mvnrnd(mean1_init, cov1_init, num_of_samples_per_class);
w2 = mvnrnd(mean2_init, cov2_init, num_of_samples_per_class);
w3 = mvnrnd(mean3_init, cov3_init, num_of_samples_per_class);
w = [w1; w2; w3];

%% k-means clustering
label = k_means(w);
% Plot original Data
title('Result')
figure(2)
hold on
```

```matlab
% plot(w1(:, 1), w1(:, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7)
% plot(w2(:, 1), w2(:, 2), 'ko', 'MarkerFaceColor', 'b', 'MarkerSize', 7)
% plot(w3(:, 1), w3(:, 2), 'ko', 'MarkerFaceColor', 'r', 'MarkerSize', 7)
plot(w(:, 1), w(:, 2), 'ko', 'MarkerFaceColor', 'k', 'MarkerSize', 7)
title('Original')

%% Functions
function label = k_means(w)
minPos = min(w);
maxPos = max(w);
% Init centroids and normalize
centroids = rand(3, size(w, 2));
centroids(:, 1) = centroids(:, 1) * (maxPos(1) - minPos(1)) + minPos(1);
centroids(:, 2) = centroids(:, 2) * (maxPos(2) - minPos(2)) + minPos(2);
figure(1)
hold on
label = zeros(size(w, 1), 1);
while(1)
    label_prev = label;
    label = findNearest(w, centroids);
    % Plot
    plt1 = plot(w(label == 1, 1), w(label == 1, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7);
    plt2 = plot(w(label == 2, 1), w(label == 2, 2), 'ko', 'MarkerFaceColor', 'b', 'MarkerSize', 7);
    plt3 = plot(w(label == 3, 1), w(label == 3, 2), 'ko', 'MarkerFaceColor', 'r', 'MarkerSize', 7);
    plt4 = plot(centroids(1, 1), centroids(1, 2), 'y+', 'LineWidth', 1, 'MarkerSize', 10);
    plt5 = plot(centroids(2, 1), centroids(2, 2), 'b+', 'LineWidth', 1, 'MarkerSize', 10);
    plt6 = plot(centroids(3, 1), centroids(3, 2), 'r+', 'LineWidth', 1, 'MarkerSize', 10);
    pause(0.5);
    % Assign new pos to centroids
    centroids = updatePosition(w, label, centroids);
    % Check convergence
    if(sum(label == label_prev) == size(w, 1))
        disp('Converged')
        break;
    end
    delete(plt1)
    delete(plt2)
    delete(plt3)
    delete(plt4)
    delete(plt5)
    delete(plt6)
end
end

function label = findNearest(w, centroids)
label = zeros(size(w, 1), 1);
for i = 1 : size(w, 1)
    minDistance = 1e+5;
    minDistCentroid = 1;
    for j = 1 : size(centroids, 1)
        distance = 0;
        for k = 1 : size(w, 2)
            distance = distance + (w(i, k) - centroids(j, k)) ^ 2;
        end
        if(distance < minDistance || j == 1)
            minDistance = distance;
            minDistCentroid = j;
        end
    end
    label(i) = minDistCentroid;
end
end

function centroids_new = updatePosition(w, label, centroids)
centroids_new = zeros(size(centroids));
for i = 1 : size(centroids, 1)
    centroids_new(i, :) = mean(w(label == i, :));
end
end
```

8.2 Isometric Mapping (ISOMAP)

```
w = generate_spiral();
figure(1)
label = isomapClustering(w, 4);
plot(w(label == 1, 1), w(label == 1, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7)
hold on
plot(w(label == 2, 1), w(label == 2, 2), 'ko', 'MarkerFaceColor', 'b', 'MarkerSize', 7)
title('Isomap Clustering')

%% functions
function label = isomapClustering(w, epsilon)
distance = calculateDistance(w, epsilon);
distance = calculateManifoldDistance(distance);
X = mds(distance, 1);
label = zeros(size(w, 1), 1);
label(X >= 0) = 1;
label(X < 0) = 2;
end

function X = mds(distance, target_dimension)
B = zeros(size(distance));
n = size(B, 1);
for i__ = 1 : size(distance, 1)
    for j__ = 1 : size(distance, 1)
        B(i__, j__) = -0.5 * (distance(i__, j__) ^ 2 - distance(i__, :) * distance(i__, :)' / n - 
distance(:, j__)' * distance(:, j__) / n + sum(sum(distance .^ 2)) / (n ^ 2));
    end
end
[V, D] = eig(B);
X = V(:, 1 : target_dimension) * D(1 : target_dimension, 1 : target_dimension) .^ (1 / 2);
end

% Calculate the distance of k-nearest-neighbors, otherwise infinite
function distance = calculateDistance(w, epsilon)
distance = zeros(size(w, 1), size(w, 1));
for i_ = 1 : size(w, 1)
    for j_ = i_ : size(w, 1)
        tmp = 0;
        for k = 1 : size(w, 2)
            tmp = tmp + (w(i_, k) - w(j_, k)) ^ 2;
        end
        tmp = sqrt(tmp);
        if(tmp < epsilon)
            distance(i_, j_) = tmp;
            distance(j_, i_) = tmp;
        else
            distance(i_, j_) = inf;
            distance(j_, i_) = inf;
        end
    end
end
end

% Calculate shortest path between paths
function manifoldDistance = calculateManifoldDistance(distance)
manifoldDistance = zeros(size(distance));
num_samples = size(distance, 1);
distance(isinf(distance)) = 0;
distance = sparse(distance);
for i_ = 1 : num_samples
    [d, ~, ~] = graphshortestpath(distance, i_);
    manifoldDistance(i_, :) = d;
end
end

function w = generate_spiral()
```

```
r = 2;
theta = 0;
w1 = [];
w2 = [];
for i = 1 : 30
    x = [r * cos(theta * pi / 180), r * sin(theta * pi / 180)];
    w1 = [w1; x];
    w2 = [w2; -x];
    r = r + 0.5;
    theta = theta + 80 / r;
end
w = [w1; w2];
end

function w = generate_concentric()
w = [];
i = 0;
while(i < 200)
    x = rand() * 20 - 10;
    y = rand() * 20 - 10;
    if(x ^ 2 + y ^ 2 <= 10)
        w = [w; [x y]];
    elseif(x ^ 2 + y ^ 2 >= 50 && x ^ 2 + y ^ 2 <= 100)
        w = [w; [x y]];
    else
        continue;
    end
    i = i + 1;
end
end
```

8.3 Density-based Clustering (DBSCAN)

```
%% Generate dataset
% w = generate_concentric();
% label = dbscan(w, 5, 2);
w = generate_spiral();
label = dbscan(w, 2, 2);
figure(1)
plot(w(label == 1, 1), w(label == 1, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7)
hold on
plot(w(label == 2, 1), w(label == 2, 2), 'ko', 'MarkerFaceColor', 'b', 'MarkerSize', 7)
title('DBSCAN Clustering')

%% DBSCAN
function label = dbscan(w, minPts, epsilon)
import java.util.LinkedList
% Parameters
distance = calculateDistance(w);
class_cnt = 1;
visited = zeros(size(w, 1), 1);
isCentroid = zeros(size(w, 1), 1);
label = zeros(size(w, 1), 1);
% Determine centroids
for i = 1 : size(w, 1)
    D = distance(i, :);
    neighbor = find(D <= epsilon);
    if(length(neighbor) >= minPts)
        isCentroid(i) = 1;
    end
end
while(sum(visited == 0) > 0)
    visited_prev = visited;
    % Find a random centroid
    randomCentroidPos = randi(size(w, 1));
    while(isCentroid(randomCentroidPos) == 0)
        randomCentroidPos = randomCentroidPos + 1;
        if(randomCentroidPos > size(w, 1))
```

```matlab
                randomCentroidPos = 1;
            end
        end
        % Initialize queue
        Q = LinkedList();
        Q.add(randomCentroidPos);
        visited(randomCentroidPos) = 1;
        while(Q.size() > 0)
            q = Q.pop();
            if(isCentroid(q) == 1)
                for i = 1 : size(w, 1)
                    if(distance(q, i) <= epsilon && visited(i) == 0 && i ~= q)
                        visited(i) = 1;
                        Q.add(i);
                    end
                end
            end
        end
        label(visited ~= visited_prev) = class_cnt;
        isCentroid(visited ~= visited_prev) = 0;
        class_cnt = class_cnt + 1;
    end
end

%% Generate distance matrix of the samples
function distance = calculateDistance(w)
distance = zeros(size(w, 1), size(w, 1));
for i_ = 1 : size(w, 1)
    for j_ = i_ : size(w, 1)
        tmp = 0;
        for k = 1 : size(w, 2)
            tmp = tmp + (w(i_, k) - w(j_, k)) ^ 2;
        end
        tmp = sqrt(tmp);
        distance(i_, j_) = tmp;
        distance(j_, i_) = tmp;
    end
end
end

function w = generate_spiral()
r = 2;
theta = 0;
w1 = [];
w2 = [];
for i = 1 : 30
    x = [r * cos(theta * pi / 180), r * sin(theta * pi / 180)];
    w1 = [w1; x];
    w2 = [w2; -x];
    r = r + 0.5;
    theta = theta + 80 / r;
end
w = [w1; w2];
end

function w = generate_concentric()
w = [];
i = 0;
while(i < 200)
    x = rand() * 20 - 10;
    y = rand() * 20 - 10;
    if(x ^ 2 + y ^ 2 <= 10)
        w = [w; [x y]];
    elseif(x ^ 2 + y ^ 2 >= 50 && x ^ 2 + y ^ 2 <= 100)
        w = [w; [x y]];
    else
        continue;
    end
    i = i + 1;
```

```
end
end
```