Machine Learning Homework 6

Kernel K-means and Spectral Clustering

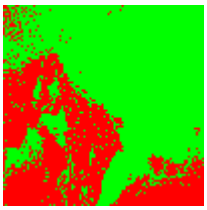
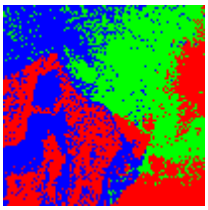
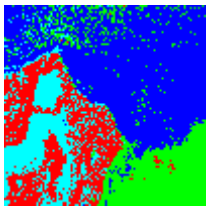
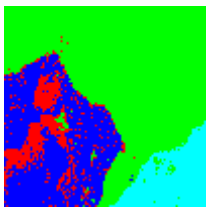
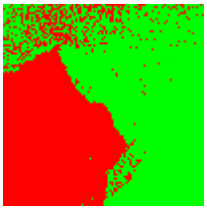
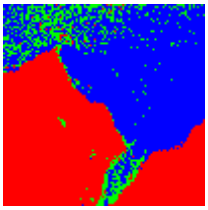
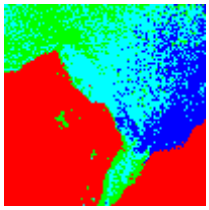
ID: 0886006 Name: 張又允

(1) You need to make videos or GIF images to show the clustering procedure (visualize the cluster assignments of data points in each iteration, colorize each cluster with different colors) of your kernel k-means and spectral clustering (both normalize cut and ratio cut) programs.

(2) In addition to cluster data into 2 clusters, try more clusters (e.g. 3 or 4) and show your results.

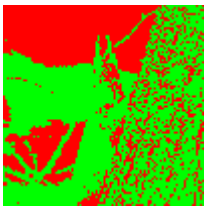
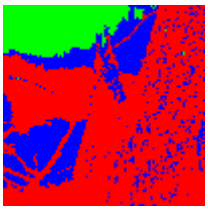
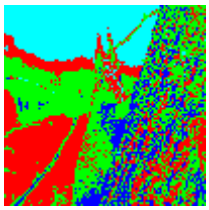
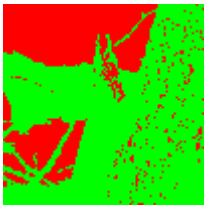
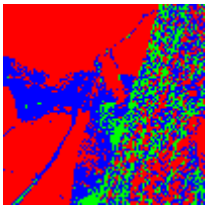
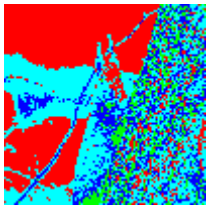
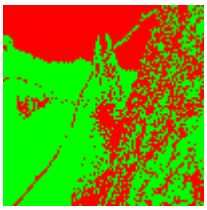
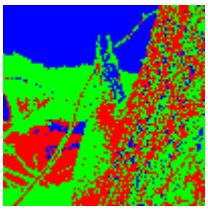
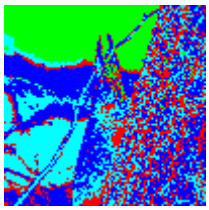
In the following discussion, $\gamma_s = 0.001$ and $\gamma_c = 0.001$, which are the best values we obtained in the grid search.
The clustering results are shown below:

- image1.png
  Naming rule: image1_(method)_(# clusters).gif
  method ∈ {kernelkmeans, unnormalized, normalized}

	# clusters = 2	# clusters = 3	# clusters = 4
Kernel k-means			
Unnormalized spectral clustering			
Normalized spectral clustering			

- image2.png
  Naming rule: image2_(method)_(# clusters).gifmethod ∈ {kernelkmeans,

unnormalized, normalized}

	# clusters = 2	# clusters = 3	# clusters = 4
K-means			
Unnormalized spectral clustering			
Normalized spectral clustering			

(3) For the initialization of k-means clustering used in kernel k-means and spectral clustering (both normalize cut and ratio cut), try different ways and show corresponding results, e.g. k-means++.

I have used k-means++ as the initialization method. The procedure of k-means++ is described as follows:
(Reference: https://en.wikipedia.org/wiki/K-means%2B%2B)
Step 1 Choose one center uniformly at random among the data points.
Step 2 For each data point x, compute $D(x)^2$, the squared distance between x and the nearest center that has already been chosen.
Step 3 Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$.
Step 4 Repeat Steps 2 and 3 until k centers have been chosen.

The code snippet is given as follows (initKmeans.m):

```
% k-means++
count = 1;
centers(count) = randi(numOfData);
while count < k
    weights = zeros(numOfData, 1);
    for i = 1:numOfData
        minV = 0;
        for j = 1:count
            dist = norm(data(centers(j)), data(i)) ^ 2;
            if (j == 1 || dist < minV)
                minV = dist;
            end
        end
        weights(i) = dist;
    end
    total = sum(weights);
    weights = weights / total;
    count = count + 1;
    centers(count) = randsample(1:numOfData, 1, true, weights);
end
```

Choose one center uniformly at random among the data points

For each data point x, compute $D(x)^2$, the squared distance between x and the nearest center that has already been chosen

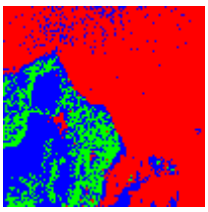
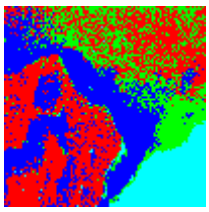
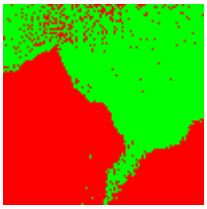
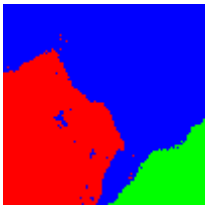
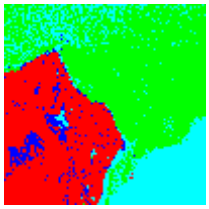
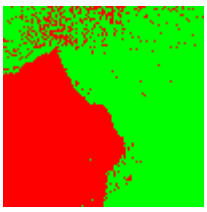
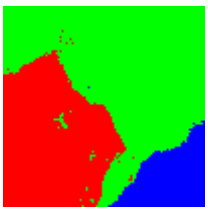
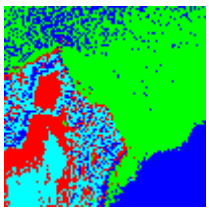
Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$

The clustering results are shown below:

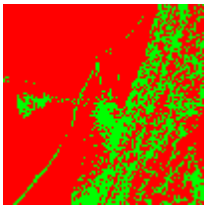
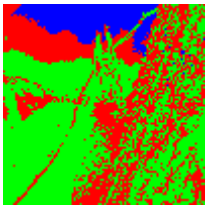
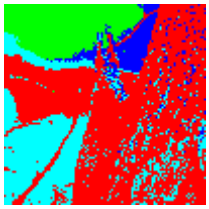
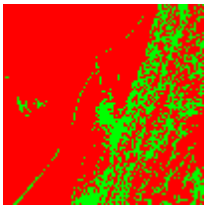
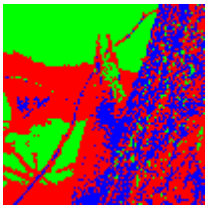
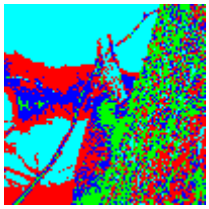
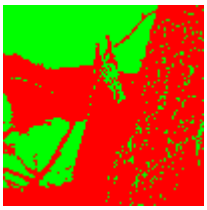
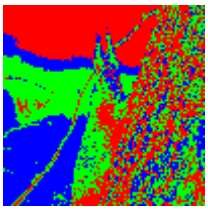
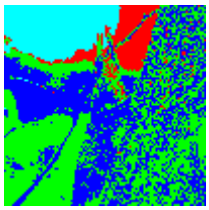
- image1.png

  Naming rule: image1_(method)_kmeansplusplus_(# clusters).gif

  method ∈ {kernelkmeans, unnormalized, normalized}

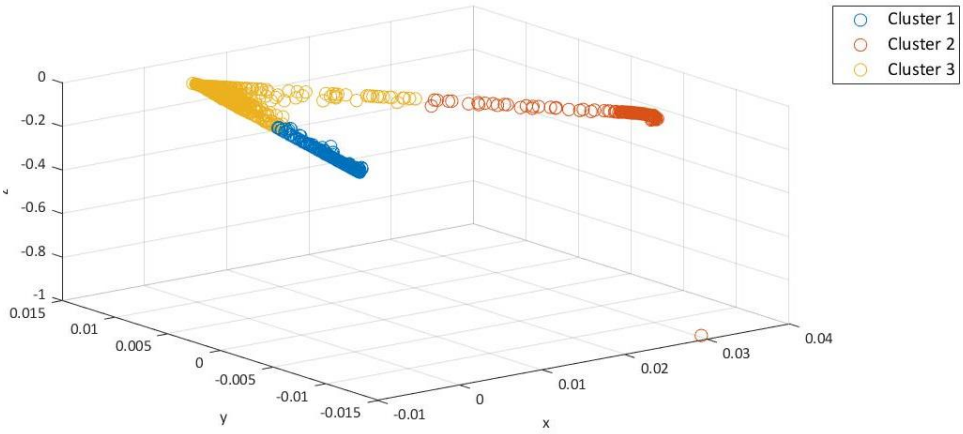
	# clusters = 2	# clusters = 3	# clusters = 4
Kernel k-means			
Unnormalized spectral clustering			
Normalized spectral clustering			

- image2.png
  Naming rule: image2_(method)_kmeansplusplus_(# clusters).gif
  method ∈ {kernelkmeans, unnormalized, normalized}

	# clusters = 2	# clusters = 3	# clusters = 4
Kernel k-means			
Unnormalized spectral clustering			
Normalized spectral clustering			

(4) For spectral clustering (both normalize cut and ratio cut), you can try to examine whether the data points within the same cluster do have the same coordinates in the eigenspace of graph Laplacian or not. You should plot the result and discuss it in the report.

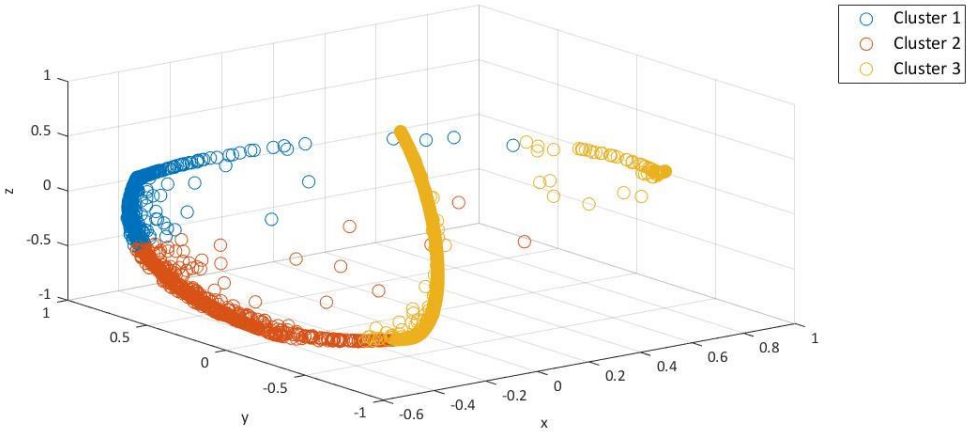
- image1.png
  <u>Unnormalized spectral clustering</u>



We found that the data points in the same cluster have the similar

coordinates and the boundaries between different clusters can be easily recognized.

<u>Normalized spectral clustering</u>

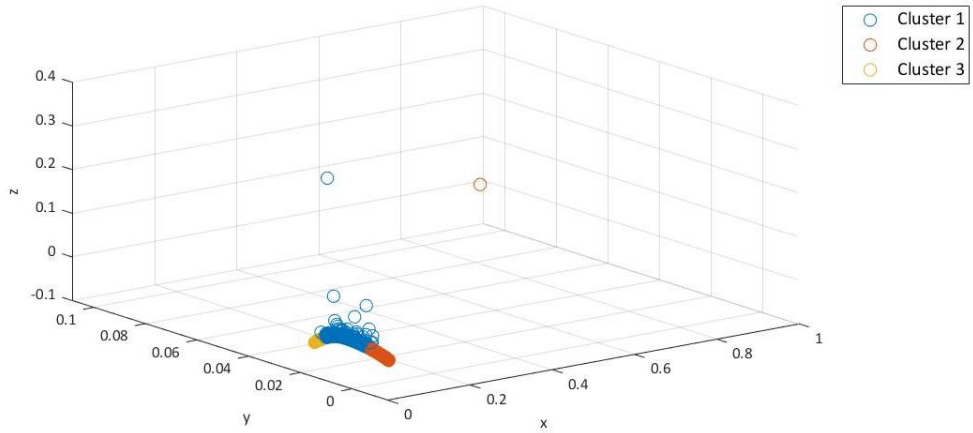


We found that the data points in the same cluster have the similar coordinates and the spread range of one cluster is wider than the unnormalized one.

The sum of the coordinates of all data points approaches to zero in the both cases.

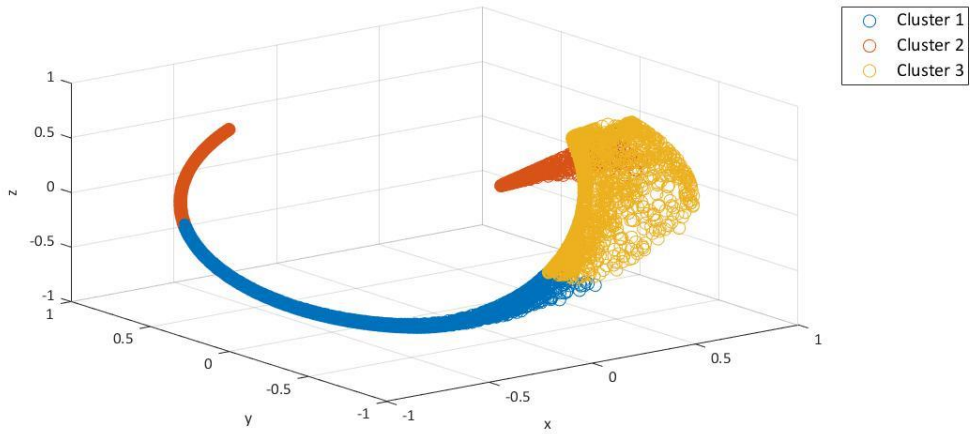
- Image2.png

  <u>Unnormalized spectral clustering</u>



As we have observed in the case of image1.png, the data points in the same cluster have the similar coordinates and is more centralized. The boundaries between different clusters can be easily recognized.

Normalized spectral clustering



As we have observed in the case of image1.png, the data points in the same cluster have the similar coordinates and the spread range of one cluster is wider than the unnormalized one.

The sum of the coordinates of all data points approaches to zero in the both cases.

(5) Code explanation

We use MATLAB as our programming language in this project.

- Main function (main.m)

  The main function consists of five parts:

  1. Read the image file and transfer 100 X 100 pixels to a vector with 10000 elements.
  2. Calculate the gram matrix.
  3. Perform the kernel k-means algorithm and save the clustering process in a GIF file.
  4. Perform the unnormalized spectral clustering algorithm and save the clustering process in a GIF file.
  5. Perform the normalized spectral clustering algorithm and save the clustering process in a GIF file.

```
for file = 1:2
    filename = ['image' num2str(file) '.png'];
    index = find(filename == '.');
    last = index - 1;
    prefix = filename(1:last);
    image = imread(filename);
    height = size(image, 1);
    width = size(image, 2);
    numOfData = height * width;

    data = zeros(numOfData, 3);
    for i = 1:height
        for j = 1:width
            data((i - 1) * width + j, :) = image(i, j, :);
        end
    end

    %% calculate kernel matrix
    gammaS = 0.001;
    gammaC = 0.001;
    gram = calKernel(data, width, gammaS, gammaC);

    %% kernel k-means algorithm
    for k = 2:4
        output = [filename(1:last) '_kernelkmeans_kmeansplusplus_' num2str(k) '.gif'];
        [curr, count] = kernelKmeans(data, gram, k, height, width, output);
    end
    %% unnormalized spectral clustering
    for k = 2:4
        output = [prefix '_unnormalized_kmeansplusplus_' num2str(k) '.gif'];
        [clusters, count, means] = unnormalizedSpectralClustering(gram, k, height, width, output);
    end
    %% normalized spectral clustering
    for k = 2:4
        output = [prefix '_normalized_kmeansplusplus_' num2str(k) '.gif'];
        [clusters, count, means] = normalizedSpectralClustering(gram, k, height, width, output);
    end
end
```

Read the image file and transfer 100 X 100 pixels to a vector with 10000 elements

Calculate the gram matrix

Kernel k-means algorithm

Unnormalized spectral clustering

Normalized spectral clustering

- Kernel k-means (kernelKmeans.m)

  1. Initialize the cluster centers (randomly or using k-means++)

  cluster centers ← initKmeans (data, # clusters, initialization method)

  initialization method: default using k-means++, or set 1 to select cluster centers randomly.

```
points = initKmeans(data, k, 1);
numOfData = size(data, 1);
curr = zeros(numOfData, 1);
c = 1;
for i = 1:length(points)
    curr(points(i)) = c;
    c = c + 1;
end
```

  2. Use the kernel k-means algorithm to perform clustering until there is no change in cluster assignment (i.e., the algorithm converges).

```
while 1
    prev = curr;
    % kernel k-means algorithm
    clusters = cell(k, 1);
    squareTerms = zeros(k, 1);
    % pre-compute the last term in the distance formula to speed up
    for class = 1:k...
    for i = 1:numOfData...
    % generate the cluster image
    generateClusterImage(curr, height, width, filename, count);
    % there is no change in cluster assignment
    if prev == curr
        break;
    end
    count = count + 1;
end
```

3. To speed up the calculation, we found that the last term in the kernel k-mean formula is only related to the cluster members and not related to the data point we are checking for and thus we can pre-compute the last term and save the result in a lookup table.

$$\left\|\phi\left(x_j\right)-\mu_k^{\phi}\right\|^2=\mathbf{k}\left(x_j,x_j\right)-\frac{2}{|C_k|}\sum_n \alpha_{kn}\mathbf{k}\left(x_j,x_n\right)+\frac{1}{|C_k|^2}\sum_p\sum_q \alpha_{kp}\alpha_{kq}\mathbf{k}\left(x_p,x_q\right)$$

```
% pre-compute the last term in the distance formula to speed up
for class = 1:k
    members = find(prev == class);
    clusters{class} = members;
    clusterSize = size(members, 1);
    for m = 1:clusterSize
        for n = 1:clusterSize
            squareTerms(class) = squareTerms(class) + gram(members(m), members(n));
        end
    end
    squareTerms(class) = squareTerms(class) / (clusterSize^2);
end
```

4. Assign the data point to the cluster which has the smallest distance between the data point and the cluster center.

```
for i = 1:numOfData
    minI = 0;
    minV = 0;
    for j = 1:k
        mid = 0;
        members = clusters{j};
        clusterSize = size(members, 1);
        for m = 1:clusterSize
            mid = mid + gram(i, members(m));
        end
        value = gram(i, i) - (2 / clusterSize) * mid + squareTerms(j);
        if (minI == 0 || value < minV)
            minI = j;
            minV = value;
        end
    end
    curr(i) = minI;
end
```

5. Generate the cluster image.

```
% generate the cluster image
generateClusterImage(curr, height, width, filename, count);
```

- Unnormalized spectral clustering (unnormalizedSpectralClustering.m)

1. Compute the unnormalized Laplacian L by using the formula $L = D - W$.

```
numOfData = size(W, 1);
D = zeros(numOfData, numOfData);
for i = 1:numOfData
    D(i, i) = sum(W(i, :));
end
L = D - W;
```

2. Generate the eigenvalues and the corresponding eigenvectors of L, and then sort the eigenvalues and the corresponding eigenvectors to get the first k eigenvectors and put them into the matrix U.

```
[eigenvectors, eigenvalues] = eig(L);
% val: the diagonal elements are eigenvalues
% vec: the columns are the corresponding eigenvectors
[d, ind] = sort(diag(eigenvalues));
eigenvalues = eigenvalues(ind, ind);
eigenvectors = eigenvectors(:, ind);
U = eigenvectors(:, 2:(k + 1));
```

3. Use the k-means algorithm to cluster the data points.

```
[clusters, count, means] = kmeans(U, k, height, width, filename);
```

- Normalized spectral clustering (normalizedSpectralClustering.m)

1. Compute the normalized Laplacian $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.

```
numOfData = size(W, 1);
D = zeros(numOfData, numOfData);
for i = 1:numOfData
    D(i, i) = sum(W(i, :));
end
L = D - W;
Q = D^(-1/2);
Lsym = Q * L * Q;
```

2. Generate the eigenvalues and the corresponding eigenvectors of L_{sym}, and then sort the eigenvalues and the corresponding eigenvectors to get the first k eigenvectors and put them into the matrix U.

```
[eigenvectors, eigenvalues] = eig(Lsym);
% val: the diagonal elements are eigenvalues
% vec: the columns are the corresponding eigenvectors
[d, ind] = sort(diag(eigenvalues));
eigenvalues = eigenvalues(ind, ind);
eigenvectors = eigenvectors(:, ind);
U = eigenvectors(:, 2:(k + 1));
```

3. Form the matrix by normalizing the row to norm 1, that is set

$$u_{ij} = \frac{u_{ij}}{\sqrt{\sum_k u_{ik}^2}}.$$

```
U = eigenvectors(:, 2:(k + 1));
for i = 1:numOfData
    len = norm(U(i, :));
    U(i, :) = U(i, :) ./ len;
end
```

4. Use the k-means algorithm to cluster the data points.

```
[clusters, count, means] = kmeans(U, k, height, width, filename);
```