> Note: This report was made on `Hackmd.io` and restricted by the `.pdf` format, the `.gif` animation would not display. Please view it on https://hackmd.io/@swchiu/BJwOjuc3U, thanks.

Spectral Clustering

Sepctral clustering groups data by similarity graph which could be computed via kernel function. In the similarity graph, if two data are more similarity, then their distance (i.e., weight of edge) is larger. For example, if we compute similarity via `Gaussian` distribution (i.e., `rbf` kernel), the weight of edge illustrate that how likely are the two points.

But how do we group the data through similarity graph?
From the rich theorm supported, we could use `Graph Lapacian` matrix, `Rayleigh quotient`, and minimize the cut cost form similarity graph.

The cut cost is defined as follow:
$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$
But this might result in the unbalance numbers of cluster A and cluster B, so the two varies of cut cost are defined as follow:

$$Ratio\ cut(A, B) = \sum_{i \in A, j \in B} w_{ij}(\frac{1}{vol(A)} + \frac{1}{vol(B)})$$

$$Normal\ cut(A, B) = \sum_{i \in A, j \in B} w_{ij}(\frac{1}{|A|} + \frac{1}{|B|})$$

And the name of spetral clustering are also differrent, one is called `Unnormalize` another is called `Normalize`.

The different not only relate to the cut categories, but also the `Graph Lapacian` matrix (denote as L).

$$Unnormalized : L = D - W$$

$$Normalize : L = D^{\frac{-1}{2}}(D - W)D^{\frac{1}{2}}$$

Then we could compute the eigenvector and perform `k-means` algorithm on matrix H. Assume the eigenvectors is denoted as matrix T and it contains only corresbonding to the first k eigenvalues:

$$Unnormalized : H = T$$

$$Normalize : H = D^{\frac{-1}{2}}T$$

The Work

- kernel function: $e^{-\gamma_1||S(x)-S(x')||^2} \times e^{-\gamma_2||C(x)-C(x')||^2}$
- Input data: Two 100*100 images

> Cause that computing the eigenvalues and eigenvectors from a $10^4 \times 10^4$ similarity matrix are too time consumimg, I only perform the result of `k = 3` and `k = 4`.

Step 1

Prepare image data for precompute similarity matrix (`kernel`) and compute similarity matrix.

> This step is the same to `Kernel K-means` Step 1 and Step 2 two part, so I just ignore here.

Step 2

Compute eigenvalues and eigenvectors and sort the eigenvectors according to the first `k` eigenvalues.

```
class spectral_clustering():
    def __init__(self, data_similarity , \
            k=2, \
            normalize=False, \
            keep_log=False):
        self.k = k
        self.W = data_similarity
        self.D = np.sum(data_similarity, axis=1, keepdims=True) *
np.eye(data_similarity.shape[0])
        self.normalize = normalize
        self.keep_log = keep_log

    def __eig(self, A):
        if self.normalize:
            sqrt_D = np.sqrt(self.D)
            neg_sqrt_D = np.linalg.inv(sqrt_D)
            N = np.matmul(np.matmul(neg_sqrt_D, A), sqrt_D)
            eigenvals, eigenvecs = np.linalg.eig(N)
            eigenvecs = np.matmul(neg_sqrt_D, eigenvecs.real)
            return eigenvals, eigenvecs
        else:
            return np.linalg.eig(A)

    def __get_sorted_k_eigen(self, A, k):
        eigenvalues, eigenvectors = self.__eig(A)

        sorted_idx = np.argsort(eigenvalues)
        sorted_eigenvalues = []
        sorted_eigenvectors = []
        for i in range(k):
            vector = eigenvectors[:, sorted_idx[i]]
            sorted_eigenvectors.append(vector[:, None])
            sorted_eigenvalues.append(eigenvalues[sorted_idx[i]])
        sorted_eigenvalues = np.array(sorted_eigenvalues)
        sorted_eigenvectors = np.concatenate(sorted_eigenvectors, axis=1)

        return sorted_eigenvalues, sorted_eigenvectors

    def run(self):
        self.L = self.D - self.W
```

```
        k_eigenvalues, k_eigenvectors = self.__get_sorted_k_eigen(self.L,
self.k)
        km = kmeans(k_eigenvectors, k=self.k, keep_log=self.keep_log)
        return km.run(), k_eigenvalues, k_eigenvectors
```

Step 3

Perform `K-means` algorithm.

> The `K-means` algorithm and method is discussed above, I just call the same python-code I written.

```
def run(self):
    self.L = self.D - self.W
    k_eigenvalues, k_eigenvectors = self.__get_sorted_k_eigen(self.L, self.k)
    km = kmeans(k_eigenvectors, k=self.k, keep_log=self.keep_log)
    return km.run(), k_eigenvalues, k_eigenvectors
```

Step 4

Examine whether the data points within the same cluster do have the same coordinates in the eigenspace of graph Laplacian or not.

```
def reorder_by_cluster(c):
    new_order = np.array([])
    for i in range(c.shape[1]):
        new_order = np.append(new_order, np.where(c[:,i]==1)[0])
    new_order = new_order.astype('int32')
    return new_order

def show_vectors_by_clusters(vectors, clusters, fig_path):
    reorder_idx = reorder_by_cluster(clusters)
    num_cluster = np.sum(clusters, axis=0, dtype=int)

    iter_idx = 0
    for k in range(clusters.shape[1]):
        plt.subplot(1, clusters.shape[1], k+1)
        for j in range(num_cluster[k]):
            plt.plot(vectors[reorder_idx[iter_idx], :])
            plt.title('cluster '+str(k+1))
            iter_idx += 1
    plt.savefig(fig_path)
    plt.show(block = False)
    plt.pause(1)
    plt.close()
```

Step 5

Visualize the result of spectral clusering.

> This step is also the same to `Kernel K-means` Step 4 part.

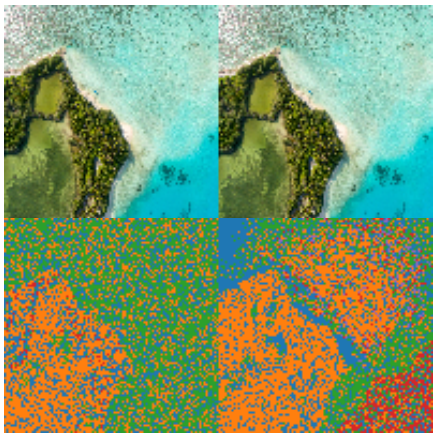
Screen Shot

```
processing image1...
compute eigenvalues and eigenvectors.....[complete]
running kernel k-means (k = 4, default).....[complete at [20] iterations]
visualizing.....[complete]
compute eigenvalues and eigenvectors.....[complete]
running kernel k-means (k = 4, default).....[complete at [30] iterations]
visualizing.....[complete]
faster.....[normalize]
compute eigenvalues and eigenvectors.....
```

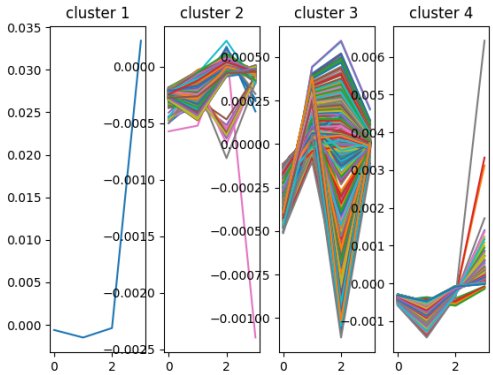
> The output log is too long as `Kernel K-means`, so I just upload part of the output.

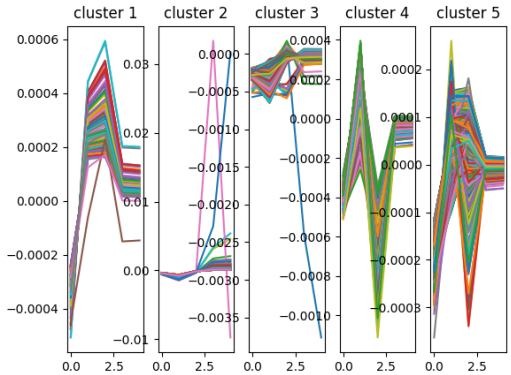
Result and The Discussion

- image1 (k = 4, 5)
  - Normalize:



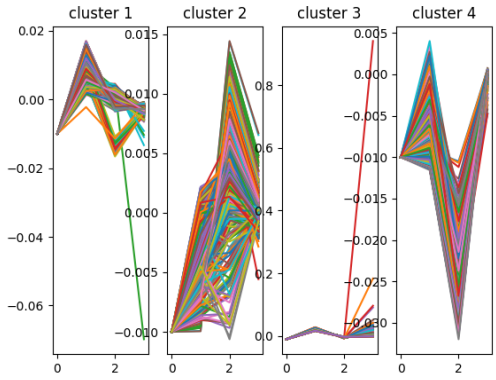
  - eigenvectors:

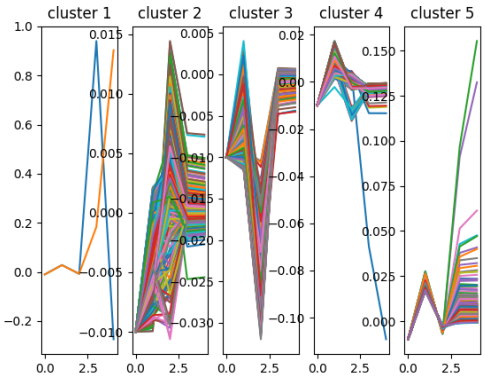




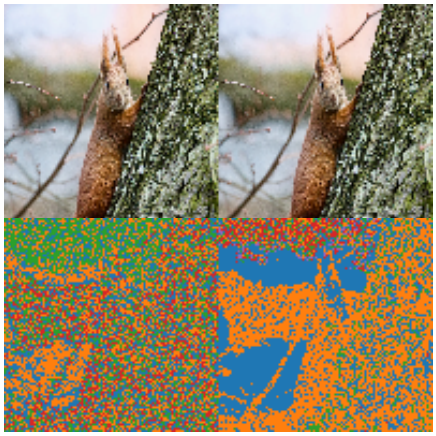
  - Unnormalize:

- eigenvectors:

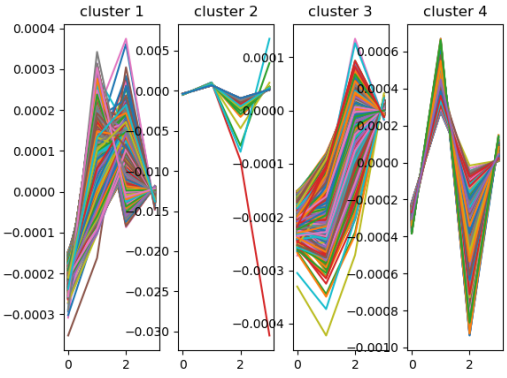


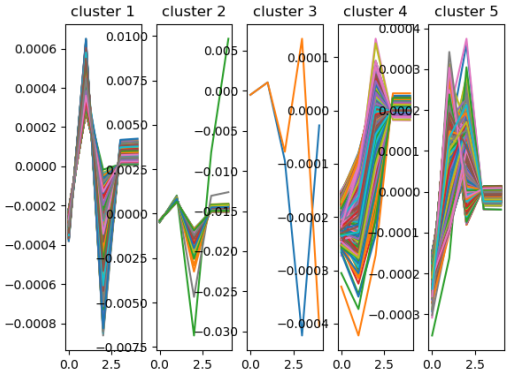


- image2 (k = 4, 5)
  - Normalize:

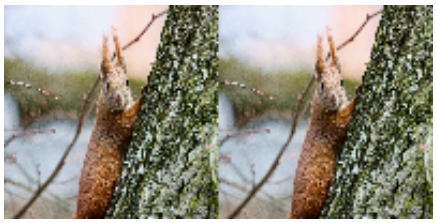


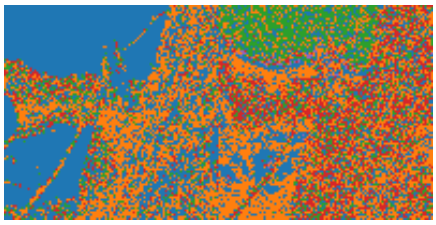
  - eigenvectors:



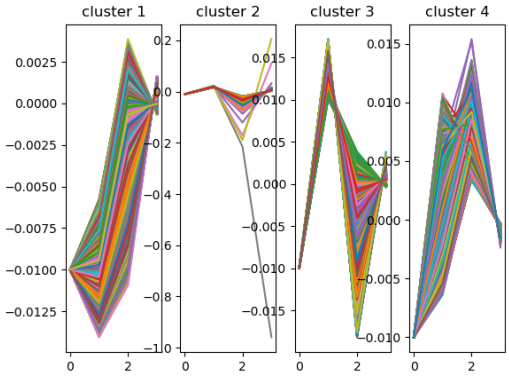


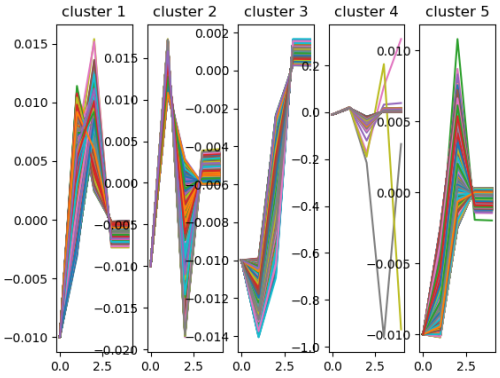
  - Unnormalize:





- eigenvectors:





According to the result shown above, we could not say that the data points within the same cluster do have the same coordinates in the eigenspace of `graph Laplacian` matrix L, they are different, but similar! `K-means` would group the similar vector to the same group.

Due to the result, the `unormalize` seems to perform better than `normalize`.