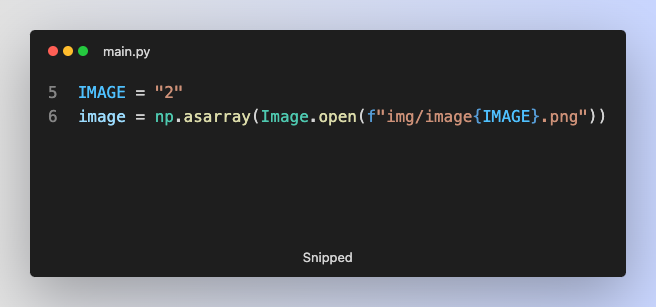
**Report**

1. **Code Explanation**

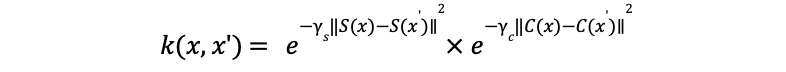
* **Kernel K-Means**
* **Load Image**

I read image with Pillow package and convert it into a numpy array.

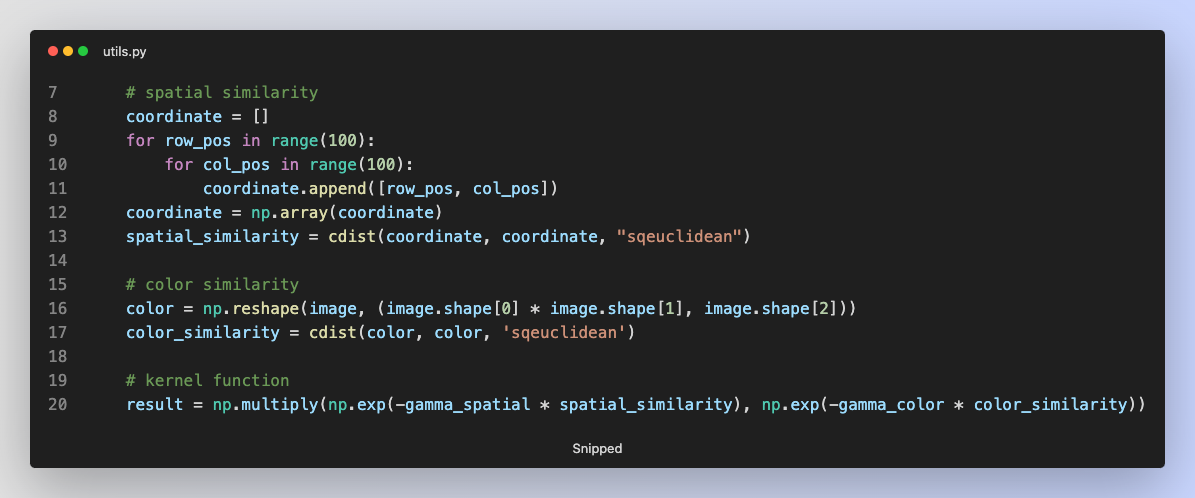
* **Big Picture**

The kernel k-means can be broken down into three steps. The first step is computing gram matrix, which is defined in compute\_kernel function. The second step is initializing the center of all clusters, which is defined in initialize\_clustering function. The last step is executing kernel k-means algorithm, which is defined in clustering function.

* **Compute Gram Matrix**



The computation of gram matrix is based on the above formula. The first part is RBF kernel of spatial information, and the second part is RBF kernel of color information.

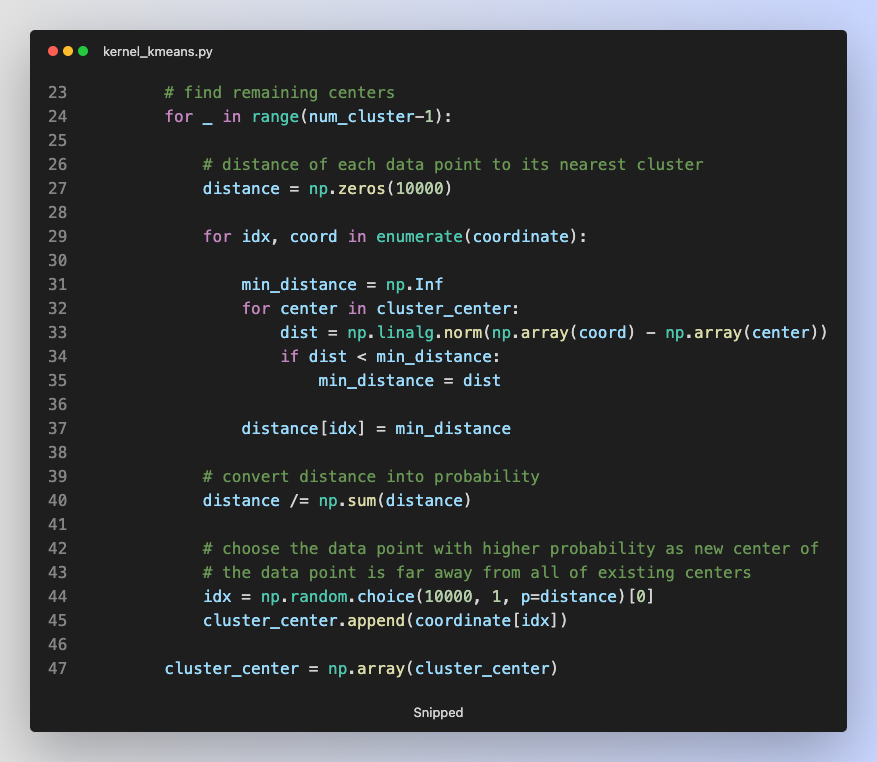


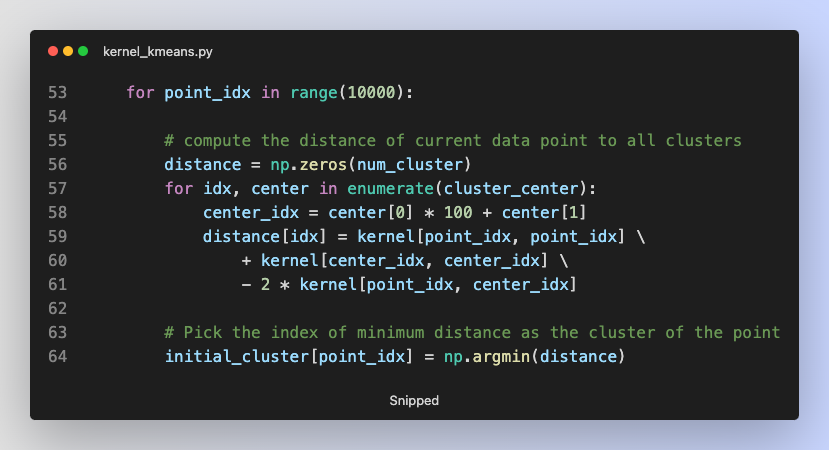
Therefore, in the above code implementation, I store the coordinates and pixel values of all data points, calculate similarity (distance), and multiply two RBF kernels to get gram matrix.

* **Initialize Cluster (Random or K-Means++)**

There are two ways to initialize center clusters, random initialization or k-means++. In k-means++, the center of first cluster is randomly initialized.

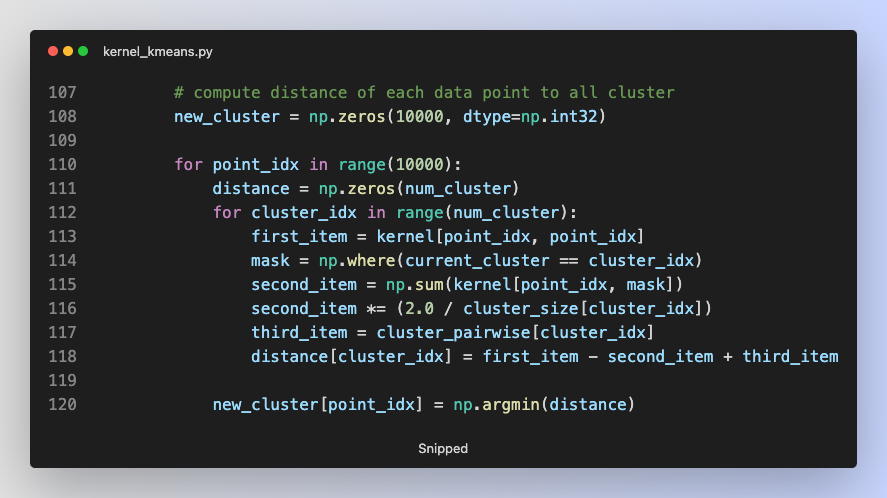


After that, the remaining center of clusters will be determined based on the distance of each data points to its nearest cluster. If the data point is far away from the existing clusters, it is likely to selected as center of new cluster.

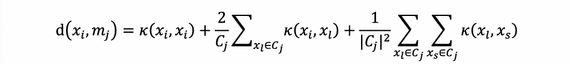
After determining the center of all clusters, each data point will be assigned to nearest cluster.

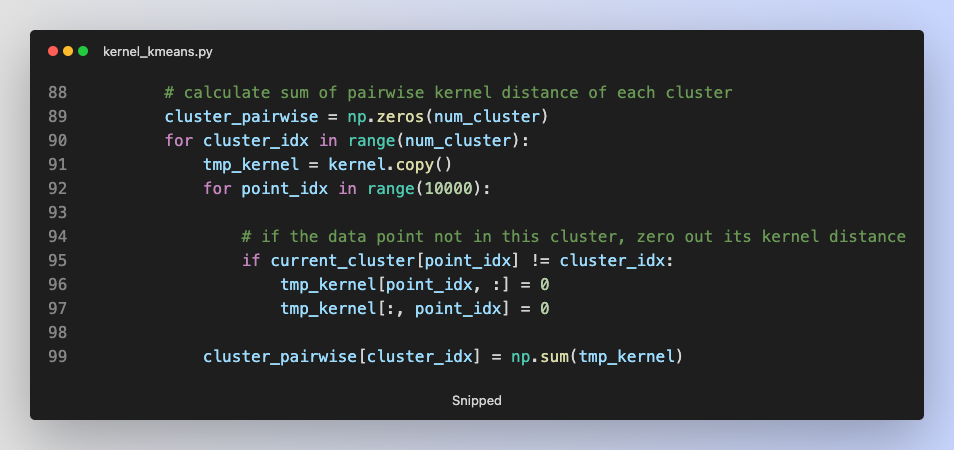
The distance of data point between center of cluster in feature space follows this formula, ||φ(Xn) − φ(μn)|| = k(xn, xn) + k(μk, μk) − 2k(xn, μk).

* **Core Algorithm**

In the core part of kernel k-means algorithm, the clustering process is executed up to 100 iterations. In each iteration, I calculate the distance between each data points and all center of clusters, and the data points are assigned to its nearest cluster.

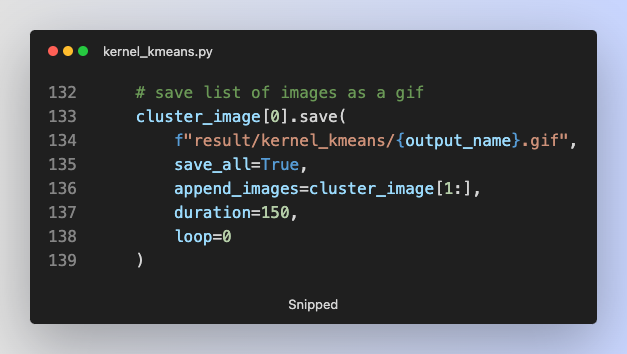
The distance follows this formula:



The third item is the sum of distance between all data points in the cluster.

* **Convert Cluster Information into Image**

In each iteration of kernel k-means, I save the cluster information as a single image. Each data point has its own color based on its cluster.

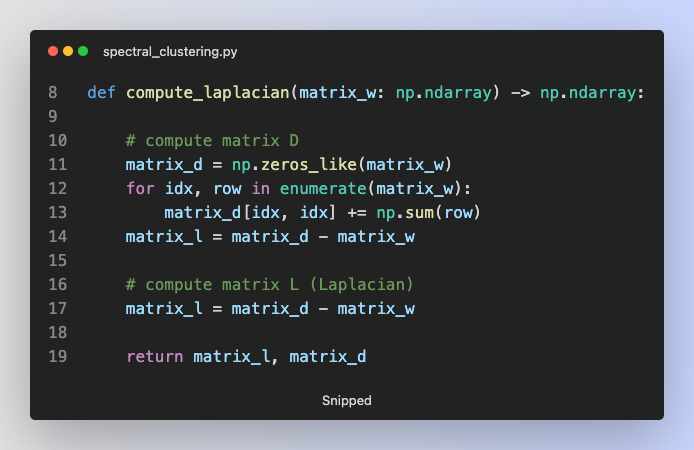
* **Convert List of Images into GIF**

With Pillow package, it is easy to convert list of images into a single gif file.

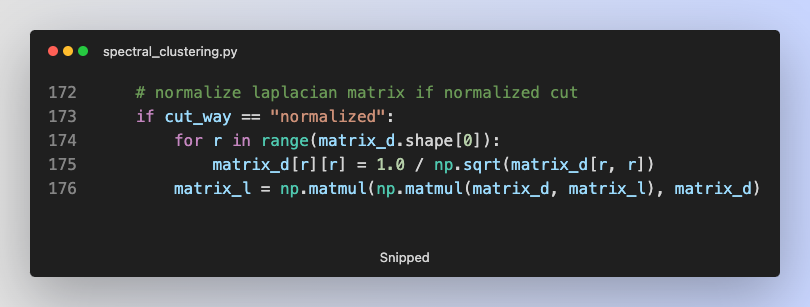
* **Spectral Clustering**
* **Compute Gram Matrix**

The approach to computing gram matrix in spectral clustering is same as it in kernel k-means.

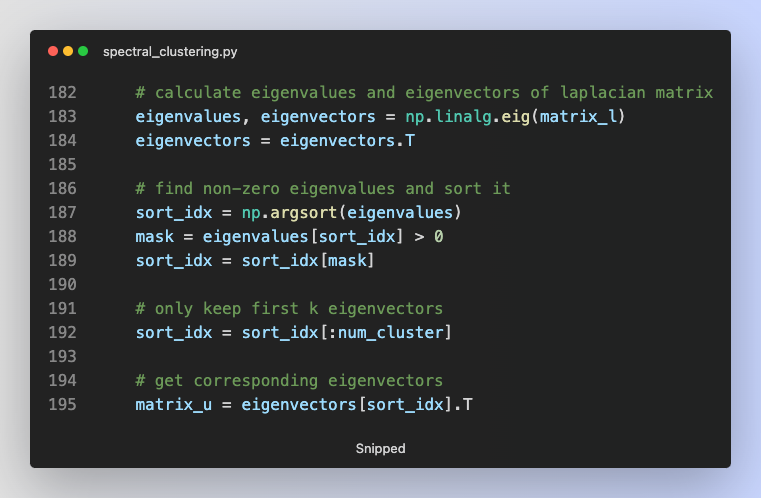
* **Compute Laplacian Matrix**

****

With the weight matrix, I am able to compute diagonal matrix and Laplacian matrix.

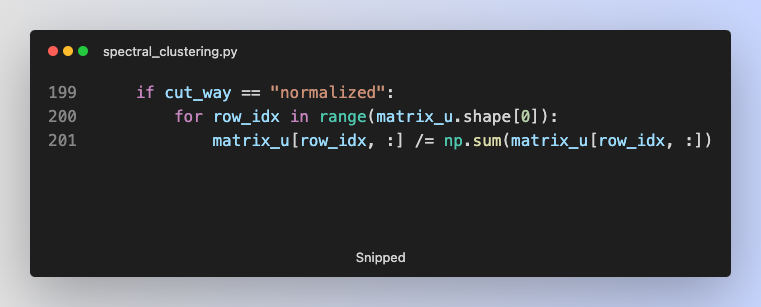
* **Normalize Laplacian Matrix (Normalized Cut Only)**

It must be noted that if we use normalized cut in spectral clustering, we have to normalize Laplacian matrix. Therefore, I calculate the diagonal matrix to the power of -1/2, and multiply it with Laplacian matrix.

* **Eigen-Decomposition of Laplacian Matrix**

I use numpy package to get eigenvalues and eigenvectors from Laplacian matrix, and only keep the first k non-zero eigenvalues with corresponding eigenvectors.

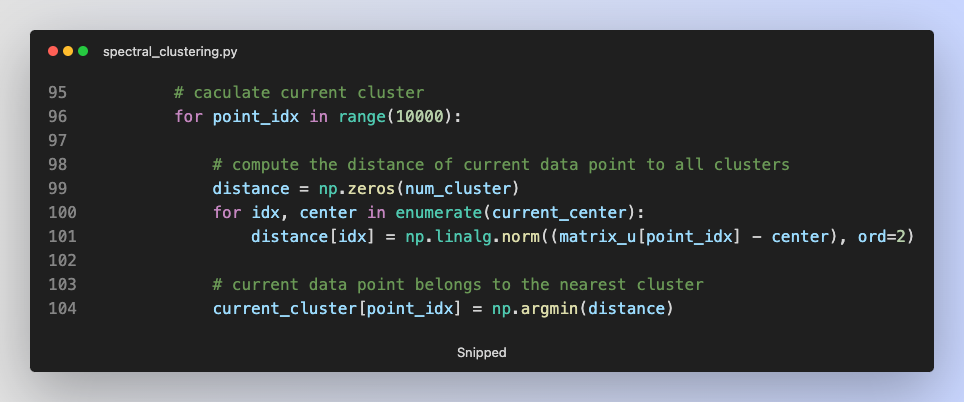
* **Normalize Eigenvectors (Normalized Cut Only)**

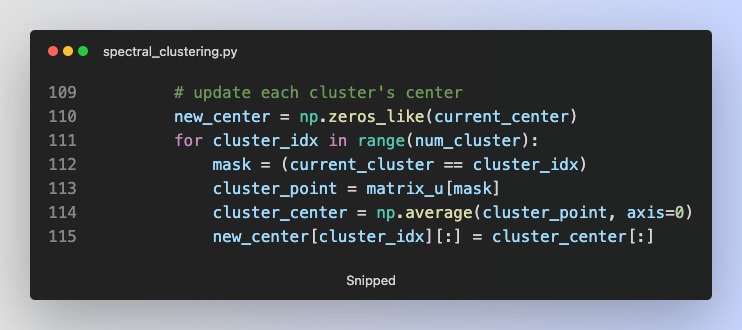
After the previous step, the matrix U will contain eigenvectors of Laplacian matrix. If we use normalized cut in spectral clustering, we have to normalize each row in matrix U.

* **Find Initial Center of Clusters**

There are two approaches to find the initial center of clusters, random and k-means++. The implementation of these two methods is same as it in kernel k-means. However, instead of returning the coordinate information, I transform the coordinate to row index, and return specific rows of matrix U (eigenspace).

* **Core Algorithm**

In the core part of spectral clustering, the clustering process is executed up to 100 iterations. In each iteration, I calculate the distance between each data points and all center of clusters, and the data points are assigned to its nearest cluster.

After that, I calculate new center of all clusters.

If the centers do not change a lot, the clustering process ends in this iteration.

* **Convert Cluster Information into Image**

In each iteration of spectral clustering, I save the cluster information as a single image. Each data point has its own color based on its cluster. The implementation is same as it in kernel k-means.

* **Convert List of Images into a Single GIF**

The implementation is same in kernel k-means.

1. **Experiment Result and Discussion**

* **Kernel K-Means with 2 Clusters (Random vs K-Means++)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Random | K-Means++ |
| Image 1 | Iteration 0 |  |  |
| Final |  |  |
| Image 2 | Iteration 0 |  |  |
| Final |  |  |

From the experiment, I find something interesting:

* Random initialization needs more iterations than kernel k-means++
* Random initialization tends to converge to bad result like the example in image 2
* **Kernel K-Means with 3 Clusters (Random vs K-Means++)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Random | K-Means++ |
| Image 1 | Iteration 0 |  |  |
| Final |  |  |
| Image 2 | Iteration 0 |  |  |
| Final |  |  |

From the experiment, I find that when the number of clusters increases, the advantage of k-means++ is more significant. For example, k-means++ initialization make the model converge in better result.

* **Spectral Clustering with 2 Cluster, Ratio Cut, Random Initialization**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Clustering Result | Eigenspace |
| Image 1 | Final |  |  |
| Image 2 | Final |  |  |

* **Spectral Clustering with 2 Cluster, Ratio Cut, K-Means++ Initialization**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Clustering Result | Eigenspace |
| Image 1 | Final |  | Same as Above |
| Image 2 | Final |  | Same as Above |

When using spectral clustering with 2 clusters and ratio cut, I find that the k-means initialization is also better than random.

* **Spectral Clustering with 2 Cluster, Normalized Cut, Random Initialization**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Clustering Result | Eigenspace |
| Image 1 | Final |  |  |
| Image 2 | Final |  |  |

* **Spectral Clustering with 2 Cluster, Normalized Cut, K-Means++ Initialization**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Clustering Result | Eigenspace |
| Image 1 | Final |  | Same as Above |
| Image 2 | Final |  | Same as Above |

When using spectral clustering with 2 clusters and normalized cut, I find that the k-means initialization is still better than random. However, in terms of convergence, I think ratio cut is better that normalized cut in this case because normalized cut tends to classify all many data points in one same cluster.

* **Spectral Clustering with 3 Cluster, Ratio Cut, K-Means++ Initialization**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Clustering Result | Eigenspace |
| Image 1 | Final |  | Only for 2 Clsuters |
| Image 2 | Final |  | Only for 2 Clsuters |

* **Spectral Clustering with 3 Cluster, Normalized Cut, K-Means++ Initialization**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Clustering Result | Eigenspace |
| Image 1 | Final |  | Only for 2 Clsuters |
| Image 2 | Final |  | Only for 2 Clsuters |

1. **Experiment Result and Discussion**

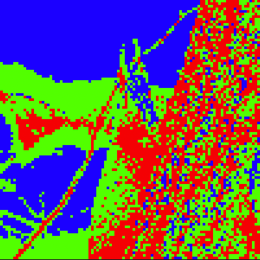
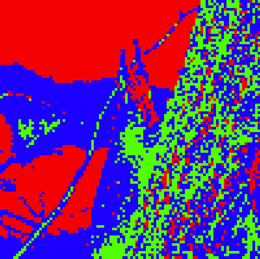
* **Performance: Initialization Method**

From experiments, I discover that the initialization method plays an important role in both kernel k-means and spectral clustering. For example, in the experiment, Kernel K-Means with 2 Clusters (Random vs K-Means++),k-means++ has better result than random initialization on image 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Random | K-Means++ |
| Image 2 | Iteration 0 |  |  |
| Final |  |  |

* **Performance: Clustering Algorithm**

The left image is result of kernel k-means on image 2, and the right image is result of spectral clustering with ratio cut on same image. Both uses k-means++ as initialization method.



I find that it is hard to determine one approach outperforms the other because sometimes one way is better and sometimes the other way is better.

* **Performance: Different Cut in Spectral Clustering**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Ratio Cut | Normalized Cut |
| Image 1 | Final |  |  |
| Image 2 | Final |  |  |

In terms of different cut in spectral clustering, I find that ratio cut usually outperforms normalized cut because normalized cut is prone to generate “big” cluster. In other words, normalized cut usually includes more data points in one cluster.

* **Execution Time of Different Setting**

From above experiments and implementation, I find:

* Initialization: random < k-means++ (because k-means++ needs additional operations)
* Clustering algorithm: kernel k-means < spectral clustering (because eigen-decomposition takes a lot of time)
* Cut Method in Spectral Clustering: ratio cut < normalized cut (because normalized cut needs additional operations)