Assignment06_20133096_HyunjaeLee

May 5, 2019

1 20133096 Hyunjae Lee

[K-means clustering on the spatial domain]

Apply K-means algorithm to the regular grid of a spatial domain in two dimension with varying number of clusters.

The spatial domain can be represented by two matrices where one matrix represents the horizontal index and the other matrix represents the vertical index.

Define a distance between each spatial point (x_i, y_i) and a centroid (c_x^k, c_y^k) for cluster k using L2-norm square and L1-norm.

Visualize the result using color coding scheme that distinguishes different clusters.

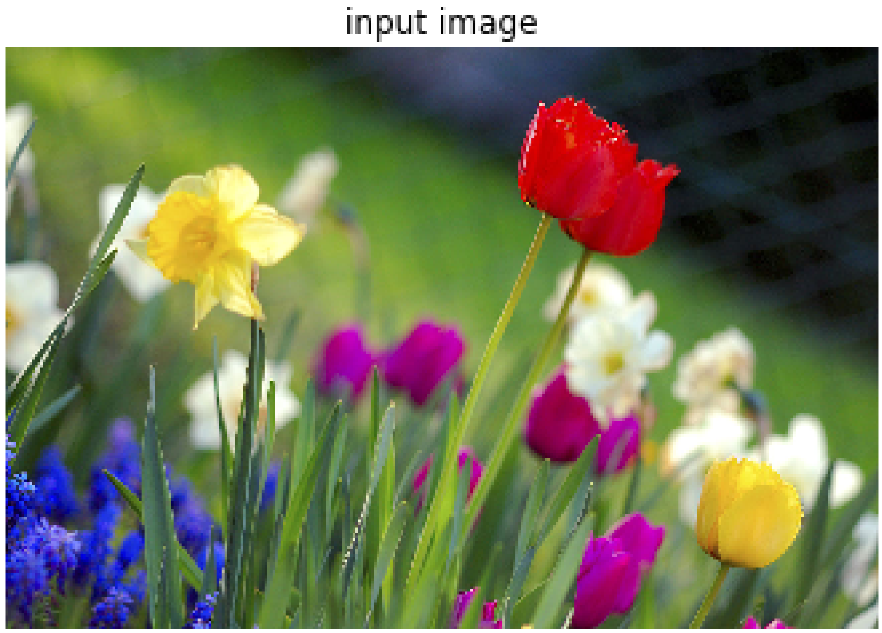
Observe the trajectory of centroid during the optimization and the shape of the clusters depending on the distance.

2 1. Set-up

Input image and import libraries

```
In [104]: import matplotlib.pyplot as plt
          from statistics import median
          import numpy as np
          import random
          import math
          from scipy import signal
          from skimage import io, color
          from skimage import exposure
          import sys

          file_image = 'img.jpg'
          im_color = io.imread(file_image)
          plt.title('input image')
          plt.imshow(im_color)
          plt.axis('off')
          plt.show()
```



3 1.1 Define functions

```
In [102]: def initialize(labelsize, k):
              return np.random.randint(k, size = labelsize)

          def Average(lst):
              return sum(lst) / len(lst)

          def Centroid(X_matrix, Y_matrix, centroid_list, count_list,
                       Label_Array, K, width, height):
              for row in range(height):
                  for col in range(width):
                      for k in range(K):
                          if Label_Array[row][col] == k:
                              # X_centroid
                              centroid_list[k][0] += X_matrix[row][col]
                              count_list[k][0] += 1

                              # Y_centroid
                              centroid_list[k][1] += Y_matrix[row][col]
                              count_list[k][1] += 1
              return centroid_list / count_list

          def Median(X_matrix, Y_matrix, median_list, count_list,
           Label_Array, K, width, height):


    for k in range(K):
        tempX = []
        tempY = []
        for row in range(height):
            for col in range(width):
                if Label_Array[row][col] == k:
                    # X_Median
                    tempX.append(X_matrix[row][col])
                    # Y_Median
                    tempY.append(Y_matrix[row][col])
        median_list[k][0]=median(tempX)
        median_list[k][1]=median(tempY)

    return median_list

def Labeling(X_matrix, Y_matrix, avg, Label_Array, K, width, height):

    energy = 0

    for row in range(height):
        for col in range(width):
            temp = []

            for k in range(K):
                inputNum = math.sqrt((X_matrix[row][col] - avg[k][0])**2
                                     + (Y_matrix[row][col] - avg[k][1])**2)
                temp.append(inputNum)

            Label_Array[row][col] = np.argmin(temp)

    # Calculate Energy
    for row in range(height):
        for col in range(width):
            for k in range(K):
                if Label_Array[row][col] == k:
                    energy += math.sqrt((X_matrix[row][col] - avg[k][0])**2
                                        + (Y_matrix[row][col] - avg[k][1])**2)

    energy /= (row*col)

    return Label_Array, energy

def Labeling_L1(X_matrix, Y_matrix, avg, Label_Array, K, width, height):

    energy = 0
```

```
                 for row in range(height):
                     for col in range(width):
                         temp = []
                         #avg_temp = np.zeros(K)

                         for k in range(K):
                             inputNum = abs(X_matrix[row][col] - avg[k][0])
                             + abs(Y_matrix[row][col] - avg[k][1])
                             temp.append(inputNum)

                         Label_Array[row][col] = np.argmin(temp)

                 # Calculate Energy
                 for row in range(height):
                     for col in range(width):
                         for k in range(K):
                             if Label_Array[row][col] == k:
                                 energy += abs(X_matrix[row][col] - avg[k][0])
                                 + abs(Y_matrix[row][col] - avg[k][1])

                 energy /= (row*col)

                 return Label_Array, energy

             def Initialize():
                 # Two matrices where one matrix represents the horizontal index
                 # and the other matrix represents the vertical index.
                 X_axis_matrix = np.zeros((height,width))
                 Y_axis_matrix = np.zeros((height,width))
                 temp = 0
                 # Assigne each matrix
                 ## X_axis
                 for eachX in X_axis_matrix:
                     for eachIndex in range(0, width):
                         eachX[eachIndex] = temp
                     temp += 1

                 ## Y_axis
                 for num in range(0, width):
                     for eachY in Y_axis_matrix:
                         eachY[num] = num

In [84]: # Width = 1200 pixel , Height = 798
         Function_X = np.array(im_color) # 798, 1200, 3
         Function_X_Size = list(Function_X.shape)
         width = Function_X_Size[1]
         height = Function_X_Size[0]
```

4 2. Labeling and Centroid list

X matrix and Y matrix should be labelled same

```
In [111]: Label_Array_K4_copy = initialize((height, width), 4)
          Label_Array_K10_copy = initialize((height, width), 10)
          Label_Array_K20_copy = initialize((height, width), 20)
          Energy = []
          threshold = 0.5
```

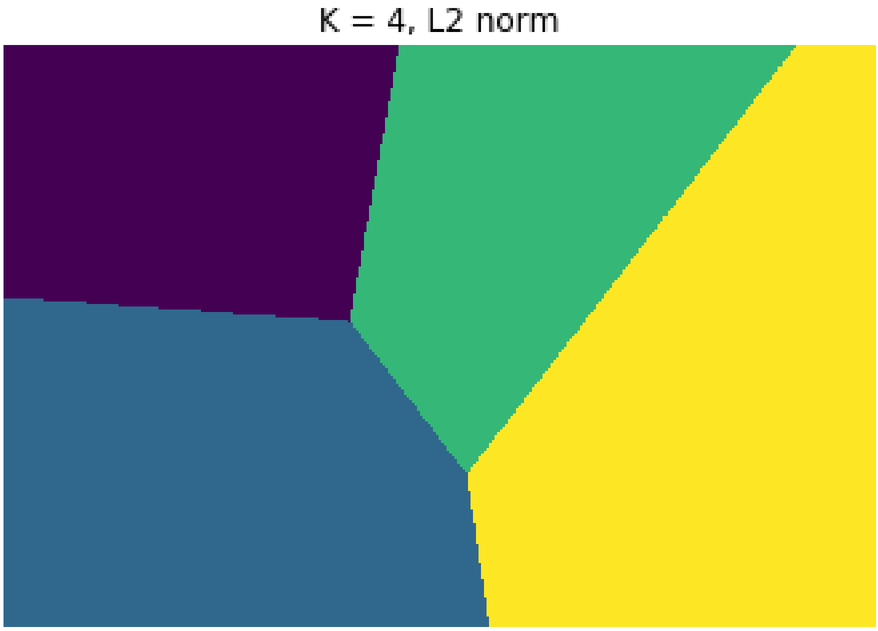
5 3. K=4, L2 norm and L1 norm

```
In [113]: k = 4
          Initialize()
          Label_Array_K4 = Label_Array_K4_copy
          Centroid_list_K4 = np.zeros((k,2))
          Count_list_K4 = np.zeros((k,2))
          Energy = []
          Energy.append(999999)
          while True:
              average = Centroid(X_axis_matrix, Y_axis_matrix, Centroid_list_K4,
                                  Count_list_K4, Label_Array_K4, k, width, height)
              Label_Array_K4 , energy = Labeling(X_axis_matrix, Y_axis_matrix,
                                                  average, Label_Array_K4, k, width, height)
              Energy.append(energy)

              if Energy[-2] - Energy[-1] < threshold:
                  print('done')
                  break
```

```
done
```

```
In [114]: plt.figure(1)
          plt.title('K = 4, L2 norm')
          plt.imshow(Label_Array_K4.astype(np.uint8))
          plt.axis('off')
          plt.show()
```



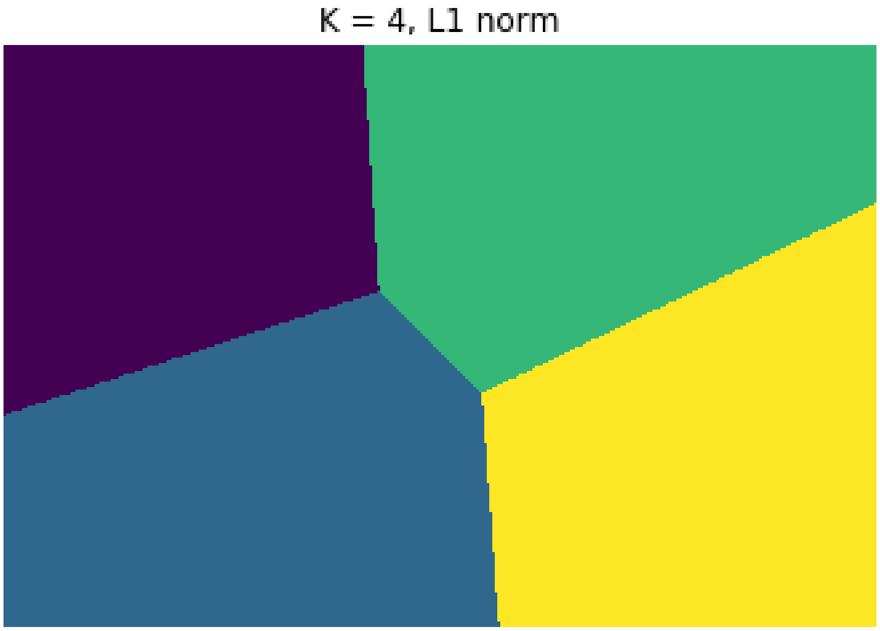
K = 4, L2 norm

```
In [115]: k = 4
          Initialize()
          Label_Array_K4 = Label_Array_K4_copy
          Median_list_K4 = np.zeros((k,2))
          Count_list_K4 = np.zeros((k,2))
          Energy = []
          Energy.append(999999)
          while True:
              average = Median(X_axis_matrix, Y_axis_matrix, Median_list_K4,
                                 Count_list_K4, Label_Array_K4, k, width, height)
              Label_Array_K4 , energy = Labeling(X_axis_matrix, Y_axis_matrix,
                                                    average, Label_Array_K4, k, width, height)
              Energy.append(energy)

              if Energy[-2] - Energy[-1] < threshold:
                  print('done')
                  break
```

```
done
```

```
In [116]: plt.figure(2)
          plt.title('K = 4, L1 norm')
          plt.imshow(Label_Array_K4.astype(np.uint8))
          plt.axis('off')
          plt.show()
```



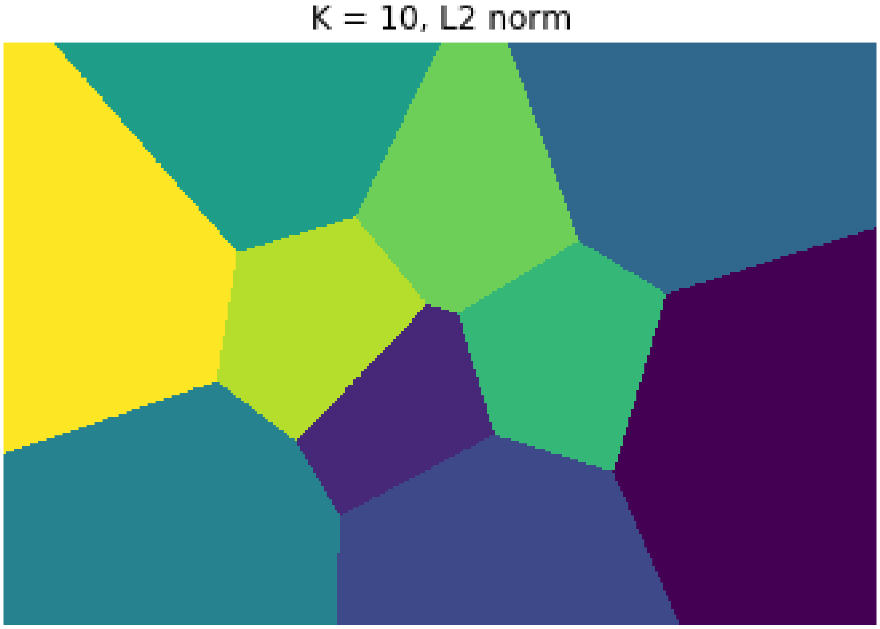
6 4. K=10, L2 norm and L1 norm

```
In [119]: k = 10
          Initialize()
          Label_Array_K10 = Label_Array_K10_copy
          Centroid_list_K10 = np.zeros((k,2))
          Count_list_K10 = np.zeros((k,2))
          Energy = []
          Energy.append(999999)
          while True:
              average = Centroid(X_axis_matrix, Y_axis_matrix, Centroid_list_K10,
                                 Count_list_K10, Label_Array_K10, k, width, height)
              Label_Array_K10 , energy = Labeling(X_axis_matrix, Y_axis_matrix,
                                                  average, Label_Array_K10, k, width, height)
              Energy.append(energy)

              if Energy[-2] - Energy[-1] < threshold:
                  print('done')
                  break
```

```
done
```

```
In [120]: plt.figure(3)
          plt.title('K = 10, L2 norm')
plt.imshow(Label_Array_K10.astype(np.uint8))
plt.axis('off')
plt.show()
```

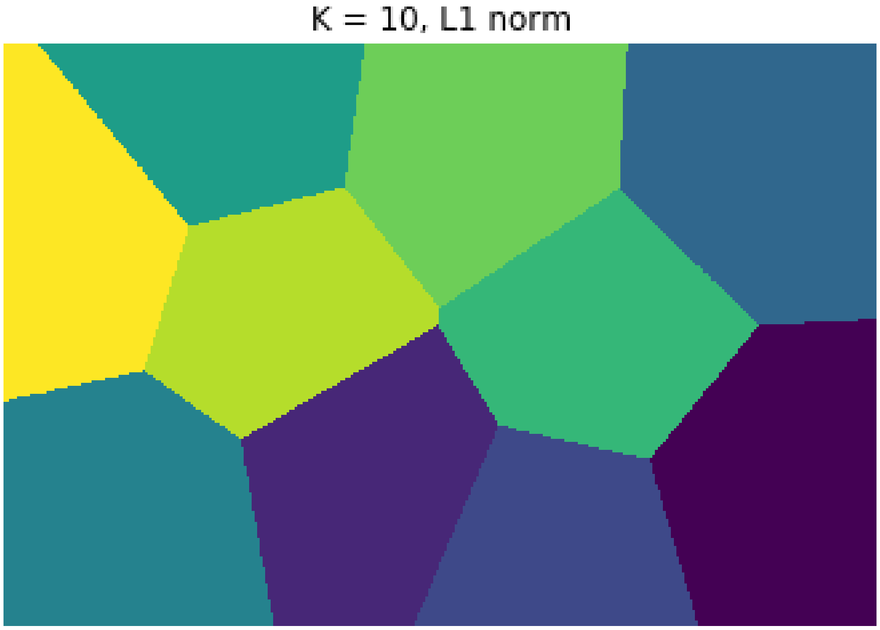


```
In [121]: k = 10
          Initialize()
          Label_Array_K10 = Label_Array_K10_copy
          Median_list_K10 = np.zeros((k,2))
          Count_list_K10 = np.zeros((k,2))
          Energy = []
          Energy.append(999999)
          while True:
              average = Median(X_axis_matrix, Y_axis_matrix, Median_list_K10,
                               Count_list_K10, Label_Array_K10, k, width, height)
              Label_Array_K10 , energy = Labeling(X_axis_matrix, Y_axis_matrix,
                                                  average, Label_Array_K10, k, width, height)
              Energy.append(energy)

              if Energy[-2] - Energy[-1] < threshold:
                  print('done')
                  break

done
```

```
In [122]: plt.figure(4)
          plt.title('K = 10, L1 norm')
          plt.imshow(Label_Array_K10.astype(np.uint8))
          plt.axis('off')
          plt.show()
```



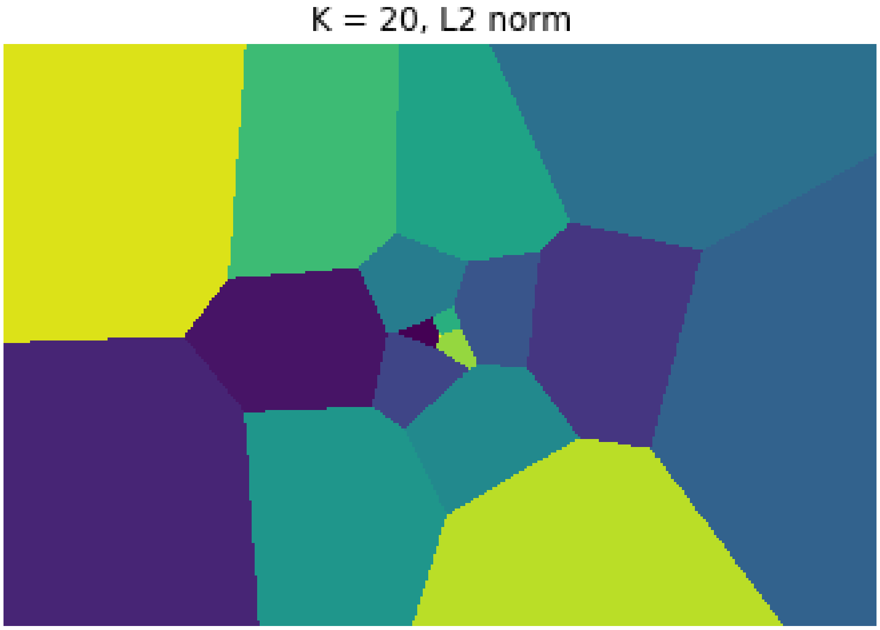
7 5. K=20, L2 norm and L1 norm

```
In [123]: k = 20
          Initialize()
          Label_Array_K20 = Label_Array_K20_copy
          Centroid_list_K20 = np.zeros((k,2))
          Count_list_K20 = np.zeros((k,2))
          Energy = []
          Energy.append(999999)
          while True:
              average = Centroid(X_axis_matrix, Y_axis_matrix, Centroid_list_K20,
                                 Count_list_K20, Label_Array_K20, k, width, height)
              Label_Array_K20 , energy = Labeling(X_axis_matrix, Y_axis_matrix,
                                                  average, Label_Array_K20, k, width, height)
              Energy.append(energy)

              if Energy[-2] - Energy[-1] < threshold:
                print('done')
                break
```

```
done
```

```
In [124]: plt.figure(5)
          plt.title('K = 20, L2 norm')
          plt.imshow(Label_Array_K20.astype(np.uint8))
          plt.axis('off')
          plt.show()
```

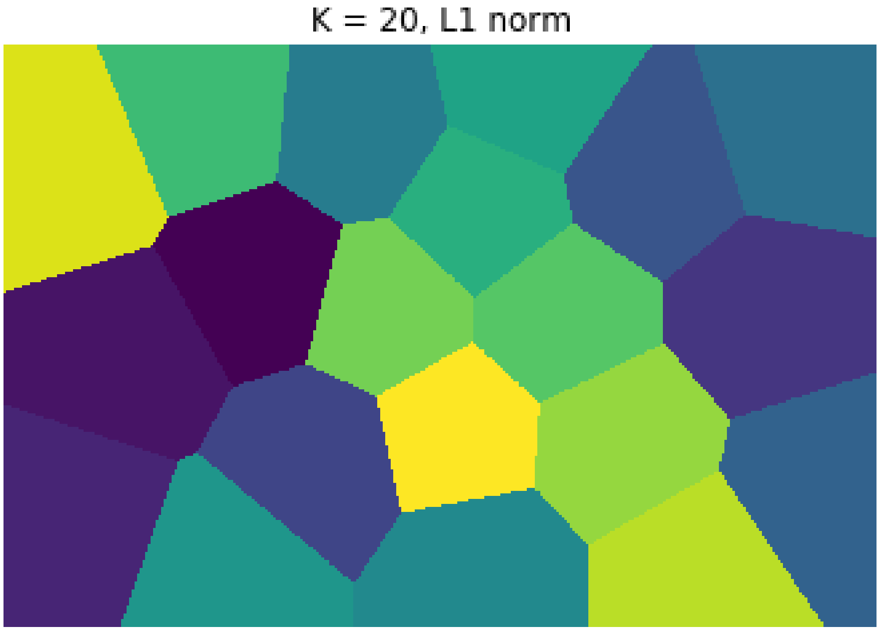


```
In [125]: k = 20
          Initialize()
          Label_Array_K20 = Label_Array_K20_copy
          Median_list_K20 = np.zeros((k,2))
          Count_list_K20 = np.zeros((k,2))
          Energy = []
          Energy.append(999999)
          while True:
              average = Median(X_axis_matrix, Y_axis_matrix, Median_list_K20,
                                 Count_list_K20, Label_Array_K20, k, width, height)
              Label_Array_K20 , energy = Labeling(X_axis_matrix, Y_axis_matrix,
                                                    average, Label_Array_K20, k, width, height)
              Energy.append(energy)
```

```
            if Energy[-2] - Energy[-1] < threshold:
                print('done')
                break
```

```
done
```

```
In [126]: plt.figure(6)
          plt.title('K = 20, L1 norm')
          plt.imshow(Label_Array_K20.astype(np.uint8))
          plt.axis('off')
          plt.show()
```



```
In [ ]:
```