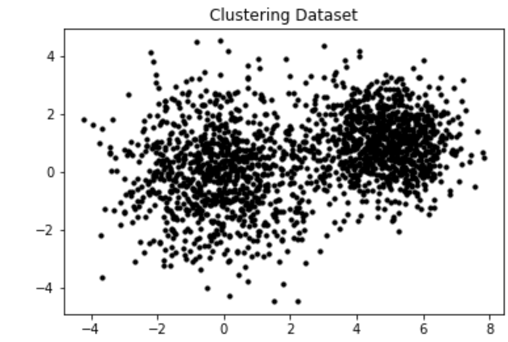
Computation Assignment Report - 2

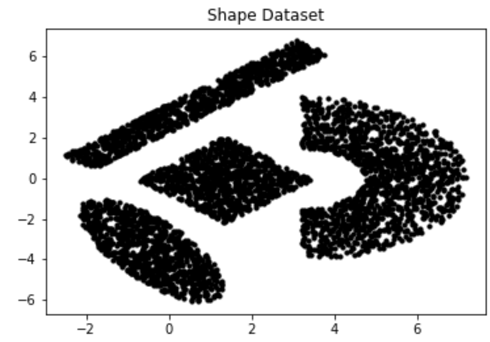
IE 529 - Stats of Big Data and Clustering
Wan-Yi Shen
December 15, 2021

I. Clustering Analysis:

In the report, I'm going to compare each clustering algorithm (K-means/Greedy K-centers/Single Swap/Spectral Clustering) with two kinds of datasets. The distribution of two datasets are shown below (Figure 1).



(a) Clustering.csv



(b)Shape.csv

Figure 1 Data Distribution

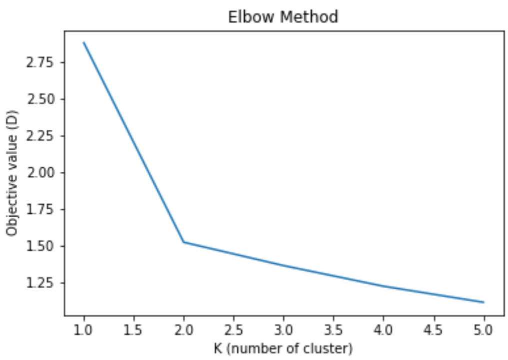
1.1 Lloyd's (K-means) Algorithm

In k-means algorithm, our objective is to minimize the average distance from each data point to their cluster centroids (D) with different convergence criteria (*tol*). We choose $\|Y_{p+1} - Y_p\|$ to be our convergence criteria (stopping criteria, *tol*), which means when the 1-norm distance between the last centers and the new centers we obtained is less than *tol*, we will stop iterating and return the current objective value (D).

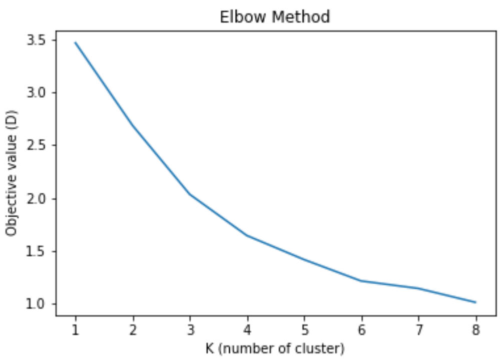
$$D = \sum_{x_i \in X} \frac{\min_{c_j \in Q} \|x_i - c_j\|_2}{N}$$

1.1.1 Convergence Criteria (*tol=10*$^{-5}$)

The results for *tol=10*$^{-5}$ for each dataset are shown below:



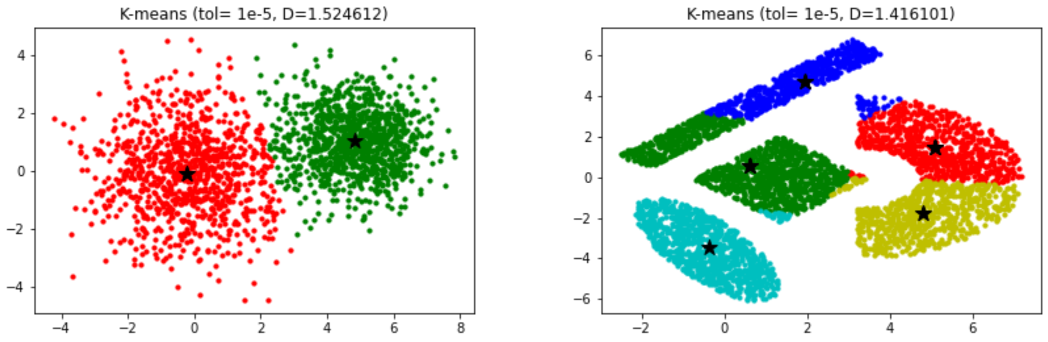
(a) Custering.csv (k=1~5)



(b) Shpae.csv (k=1~8)

Figure 2 Elbow Plot (D vs. K)

We obtain the result by iterating the number of clusters (K) and calculate the values of D for each number of cluster to get the elbow plot (Figure 2). We further determined the best number of clusters for each dataset by select the value of K at the point of the "elbow". Thus for the given dataset, we can conclude that the optimal number of clusters for *Clustering* dataset is "2" and the optimal number of clusters for *Shape* dataset is approximately "4 to 5" (see Figure 3). Their objective value of D are 1.5246 and 1.4161 respectively.

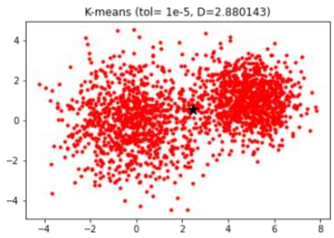
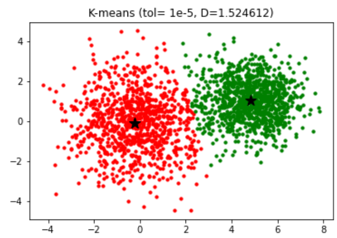
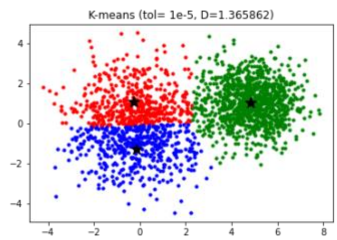


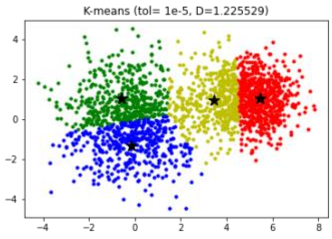
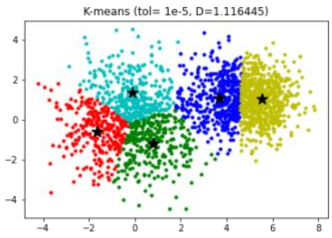
(a) Custering.csv (k=2) *(b) Shpae.csv (k=5)*

Figure 3 K-means Clustering Result

The results and outputs of each clustering value are shown below for *Clustering* dataset (Table 1). C vector represent the clustering label for each data point and D is the objective distance.

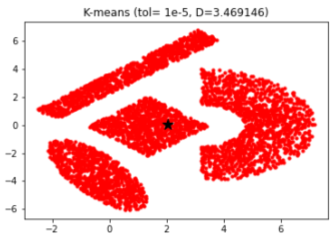
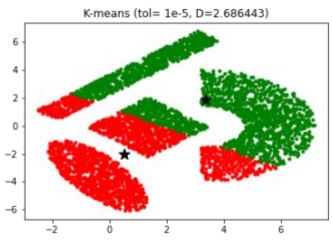
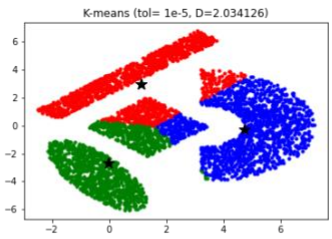
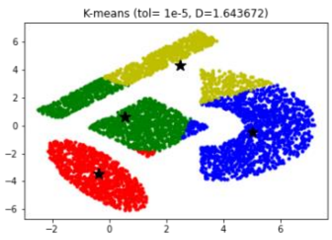
Table 1 Clustering Result (Clustering.csv)

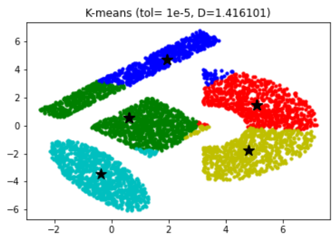
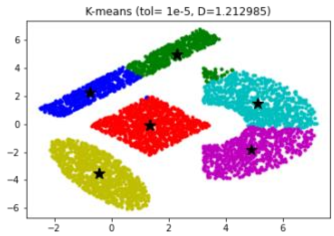
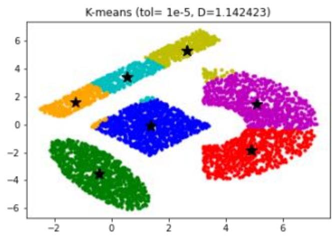
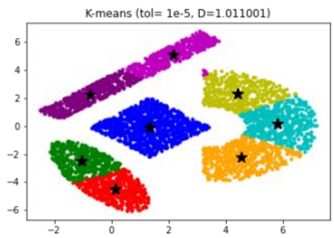
Data: Clustering	**Center**	**C (cluster index)**	**D**	**Plot**
K=1	[[2.44441693,0.52838422]]	[0. 0. 0. ... 0. 0. 0.]	2.8801	
K=2 *optimal	[[-0.2159331,-0.0629825] [4.80833513,1.05385739]]	[0. 0. 0. ... 1. 1. 1.]	1.5246	
K=3	[[-0.25259373,1.09896645] [4.82582500,1.04521043] [-0.14106817,-1.2486810]]	[2. 2. 0. ... 1. 1. 1.]	1.3659	

K=4	[[5.47443283,1.05226643] [-0.58976107,1.04789157] [-0.13630645,-1.30372475] [3.45225033,0.97551474]]	[2. 2. 1. ... 0. 3. 3.]	1.2255	
K=5	[[-1.6357134,-0.56098921] [0.78405923,-1.17619595] [3.66952087,1.08868157] [5.53615955,1.0285038] [-0.10786425,1.3732315]]	[1. 0. 0. ... 3. 2. 2.]	1.1164	

The results and outputs of each clustering value are shown below for *Shape* dataset (Table 2). C vector represent the clustering label for each data point and D is the objective distance.

Table 2 Clustering Result (Shape.csv)

Data: Shape	**center**	**C**	**D**	**Plot**
K=1	[[2.03627747,0.06476151]]	[0. 0. 0. ... 0. 0. 0.]	3.4691	
K=2	[[0.53202036,1.99986491] [3.36760224,1.89203443]]	[0. 1. 0. ... 1. 1. 0.]	2.6864	
K=3	[[1.1158781,2.94638044] [-0.01660614,-2.69452927] [4.72410242,-0.2501095]]	[1. 2. 2. ... 2. 2. 1.]	2.0341	
K=4 *optimal	[[-0.38157179,-3.480831] [0.55367684,0.62047415] [5.01488842,-0.459872]]	[1. 2. 2. ... 2. 2. 0.]	1.6437	

K=5 *optimal	[[5.07768529,1.47412081] [0.60249701,0.56684975] [1.93811992,4.71284379] [4.81122732,-1.7996012] [-0.3829894,-3.4826704]]	[1. 3. 3. ... 0. 0. 4.]	1.2129	
K=6	[[1.35366772, -0.06444487] [2.28532931, 4.98457166] [-0.74026185, 2.27485299] [-0.43282777, -3.531065] [5.10592807, 1.48083517] [4.88075265, -1.86457129]]	[0. 5. 5. ... 4. 4. 3.]	1.1424	
K=7	[[4.8812859,-1.85966862] [-0.4328277,-3.531065] [1.3831844,-0.09305032] [2.63418, 5.30156193] [0.5378219,3.42072794] [5.0968926,1.49284143] [-1.2647724,1.6314094]]	[2. 0. 0. ... 5. 5. 1.]	1.1424	
K=8	[[0.12864448,-4.5190796] [-1.03302224,-2.4749114] [1.35353385,-0.0624556] [4.40602577,2.31565452] [5.82568268,0.17297719] [2.18091815,5.09926319] [4.54109821,-2.2455013] [-0.74646629,2.2720129]]	[2. 6. 6. ... 3. 3. 0.]	1.011	

1.1.2 Convergence Criteria Analysis

In order to test the which value of *tol* will be a best stopping criteria for each dataset, we compare the objective value of each dataset by iterating *tol* = [10^{-7},10^{-6},10^{-5},10^{-4},10^{-3},10^{-2}, 10^{-1},1]. We use the optimal number of clusters for each dataset. The comparison results of objective values can be obtained below. (Table 3)

Table 3 Convergence with different tol with k-means algorithm (D)

Data \ tol	1e-7	1e-6	1e-5	1e-4	1e-3	1e-2	1e-1	1
Clustering (k=2)	1.5246	1.5246	1.5246	1.5246	1.5246	1.5247	1.5241	1.619
Shape (k=5)	1.4431	1.4159	1.4161	1.4431	1.4431	1.416	1.4402	1.5869

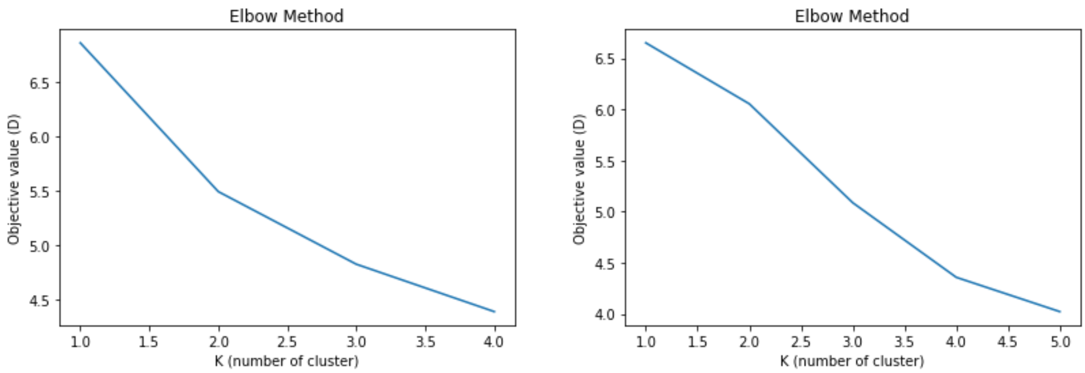
By observing the results in Table 3, we notice that the objective value (D) for *Clustering* dataset become stable around *tol*=10^{-5}~10^{-3} and the objective value (D) for *Shape* dataset become stable around *tol*=10^{-5}. The optimal tol value of *Clustering* data is slightly bigger than the tol value of *Shape* data because the sparsity of the distribution of each dataset is different. Thus, we can conclude that for different dataset the value of *tol* should be customized.

1.2 Greedy K-centers Algorithm

In k-center algorithm, our objective (D) is to minimize the maximum distance between any observation x_i and its closest center $c_j \in Q$. We perform the k-center algorithm for 20 iterations and obtained the minimum result.

$$D = \min_{Q \subset X,\ |Q|=K} \left(\max_{x_i \in X} \left(\min_{c_j \in Q} \left\| x_i - c_j \right\|_2 \right) \right)$$

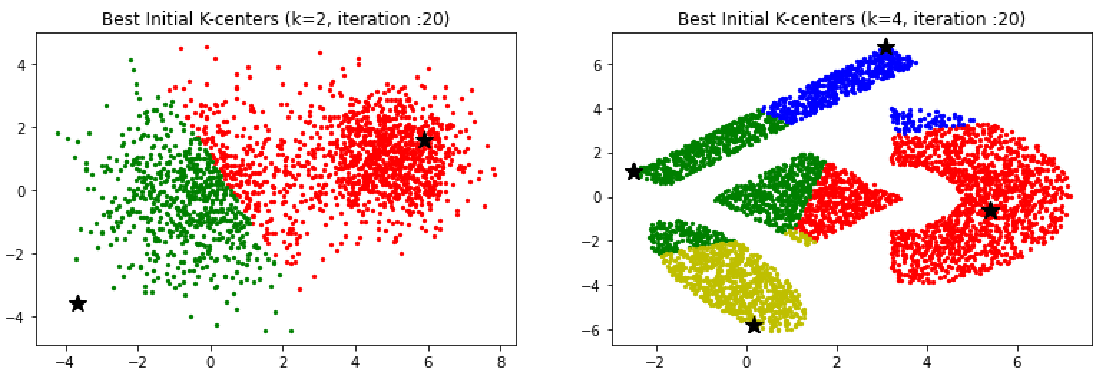
The comparison of D with different number of clusters are shown below (Figure 4):



(a) Custering.csv (k=1~4) *(b) Shpae.csv (k=1~5)*

Figure 4 Elbow Plot (D vs. K)

We obtain the result by iterating the number of clusters (K) and calculate the values of D for each number of cluster to get the elbow plot (Figure 4). We further determined the best number of clusters for each dataset by select the value of K at the point of the "elbow". Thus for the given dataset, we can conclude that the optimal number of clusters for *Clustering* dataset is approximately "2 to 3" and the optimal number of clusters for *Shape* dataset is "4". Their objective value of D is 5.4903 and 4.3568 respectively.



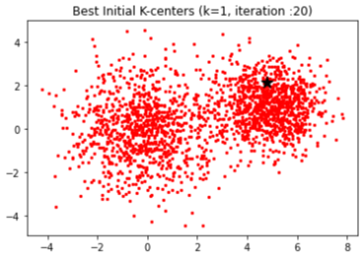
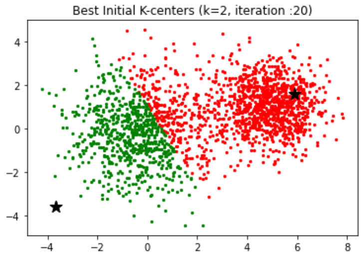
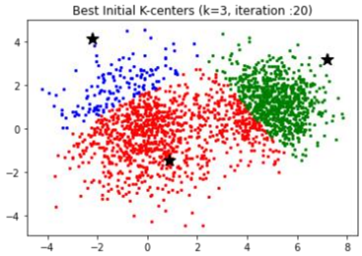
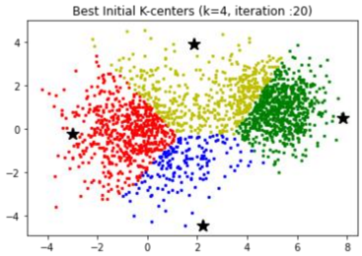
(a) Custering.csv (k=2) *(b) Shpae.csv (k=4)*

Figure 5 K-centers Clustering Result

However, by visualizing the clustered dataset (see Figure 5), we noticed that the result for k-center clustering does not fully match with the nature group distribution. The reason for this poor clustering might due to the rule for k-centers. Since k-centers devoted to set the centers for each cluster as far away from each other to avoid a set of bad starting centroid for performing clustering algorithm. In order to optimize the clustering result, we decide to implement k-median's single swap heuristic algorithm. We will discuss this algorithm in the next section.

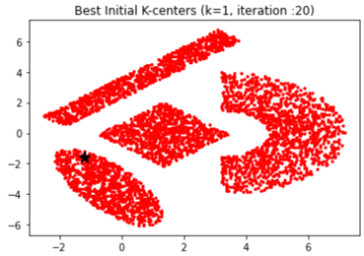
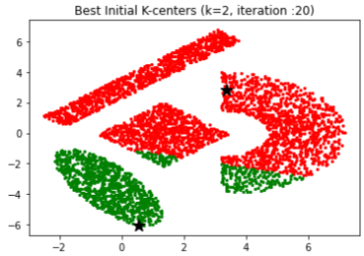
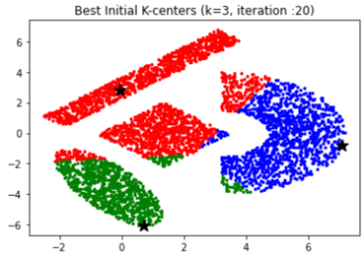
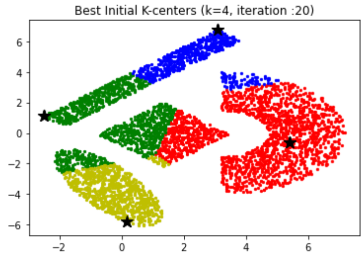
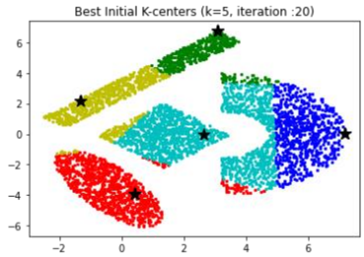
The results and outputs of each clustering value are shown below for *Clustering* dataset (Table 4). C vector represent the clustering label for each data point and D is the objective distance.

Table 4 K-centers Clustering Result (Clustering.csv)

Data: Shape	Center	D	Plot
K=1	[[4.7744 2.1152]]	6.8562	
K=2 ***Optimal**	[[5.8796 1.5898] [-3.6773 -3.5965]]	5.4903	
K=3	[[0.88748 -1.4478] [7.1976 3.1657] [-2.1987 4.1088]]	4.8238	
K=4	[[-2.9996 -0.23488] [7.8132 0.51561] [2.2232 -4.4286] [1.8645 3.882]]	4.3884	

The results and outputs of each clustering value are shown below for *Shape* dataset (Table 5). C vector represent the clustering label for each data point and D is the objective distance.

Table 5 K-centers Clustering Result (Shape.csv)

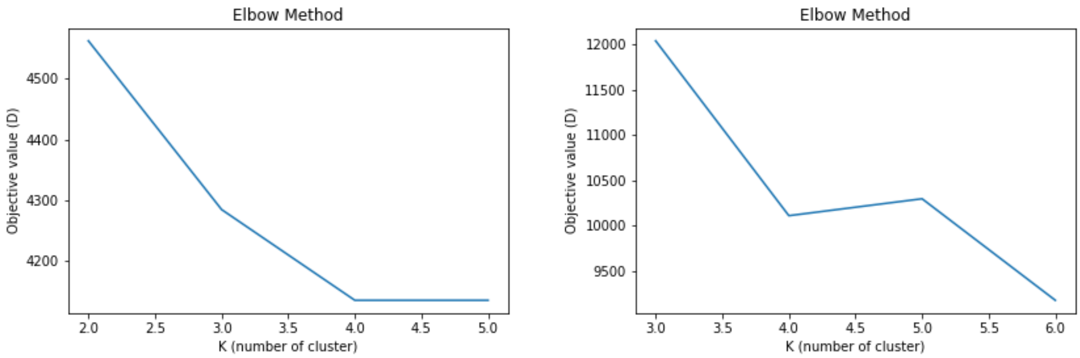
Data: Shape	Center	D	Plot
K=1	[[-1.19 -1.6139]]	6.6502	
K=2	[[3.3819 2.8053] [0.54923 -6.0252]]	6.0536	
K=3	[[-0.06601 2.8207] [0.71578 -6.0515] [7.0904 -0.82791]]	5.0861	
K=4 ***Optimal**	[[5.3926 -0.60165] [-2.4871 1.1324] [3.0811 6.7565] [0.16325 -5.806]]	4.3568	
K=5	[[0.42116 -3.9318] [3.0811 6.7565] [7.165 0.066071] [-1.3173 2.1785] [2.6574 -0.011731]]	4.0214	

1.3 K-median's Single Swap Heuristic Algorithm

In this part, we implement k-median's single swap heuristic algorithm to optimize the initial clustering generated by k-center algorithm. In k-medians, the objective (D) is to minimize the total distance between each data points and their corresponding medoid. We set a value $\gamma = 0.05$ as our swapping criteria and denoted to find a new medoid by iterating through data points in each cluster that can reduce the objective value D by $(1 - \gamma)$.

$$D = \sum_{x_i \in X} \min_{c_j \in Q} \left\| x_i - c_j \right\|_2$$

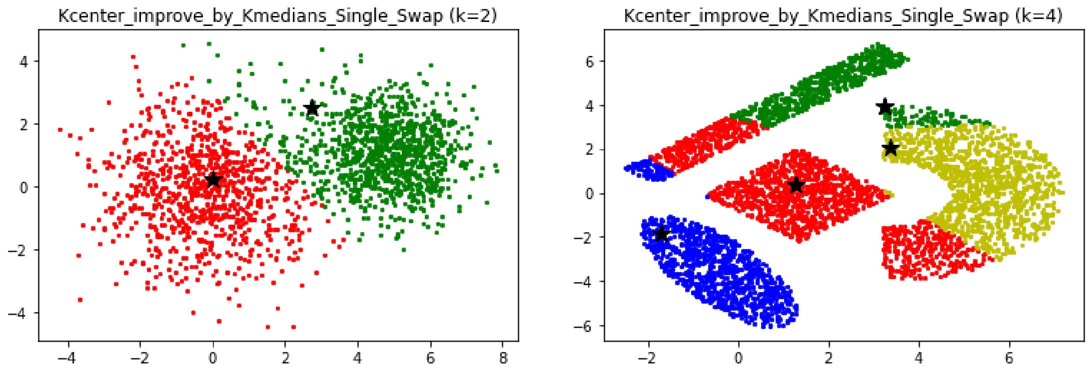
The comparison of D with different number of clusters are shown below (Figure 6):



(a) Custering.csv (k=1~4) *(b) Shpae.csv (k=3~6)*

Figure 6 Elbow Plot (D vs. K)

We obtain the result by iterating the number of clusters (K) and calculate the values of D for each number of cluster to get the elbow plot (Figure 6). We further determined the best number of clusters for each dataset by select the value of K at the point of the "elbow". Thus for the given dataset, we can conclude that the optimal number of clusters for *Clustering* dataset is "2", since we ignore the k=1 cluster. And the optimal number of clusters for *Shape* dataset is "4" as well. Their objective value of D is 4562.72 and 10110.93 respectively.

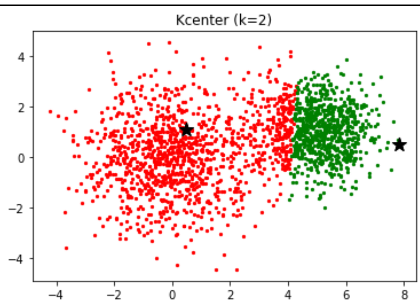
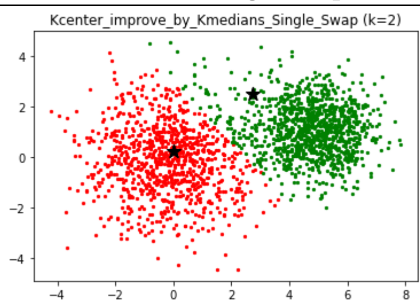


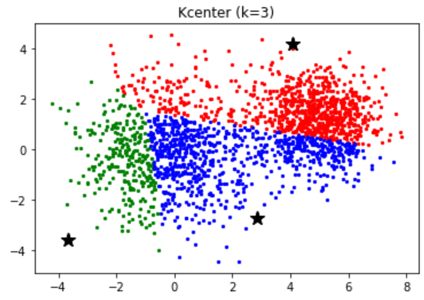
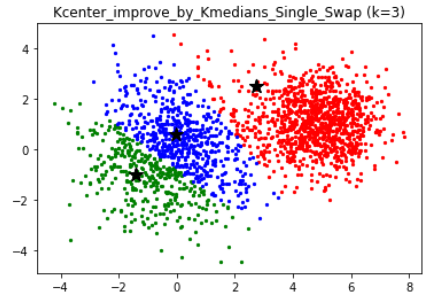
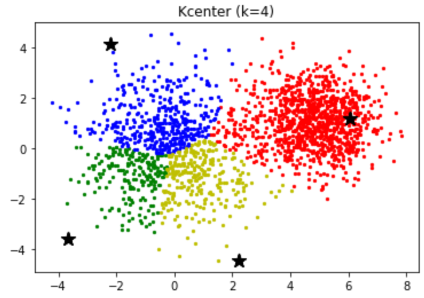
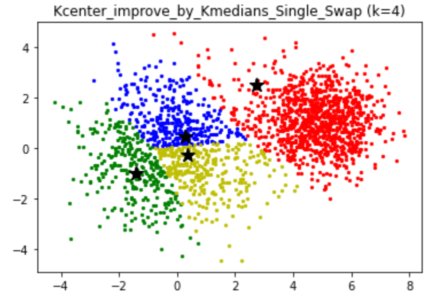
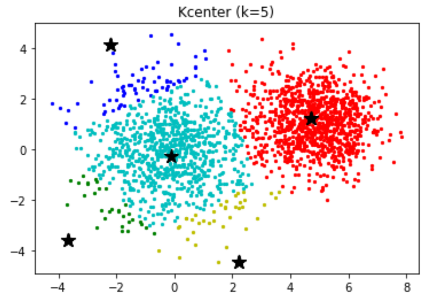
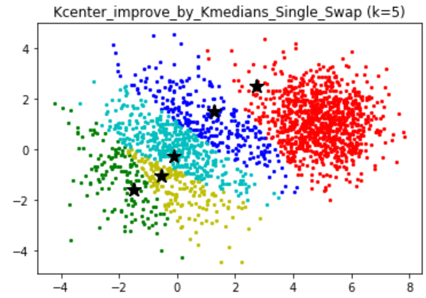
(a) Custering.csv (k=2) *(b) Shpae.csv (k=4)*

Figure 7 K-medians Single Swap Clustering Result

The comparison between k-center and single swap's clustering results of each clustering value are shown below for *Clustering* dataset (Table 6).

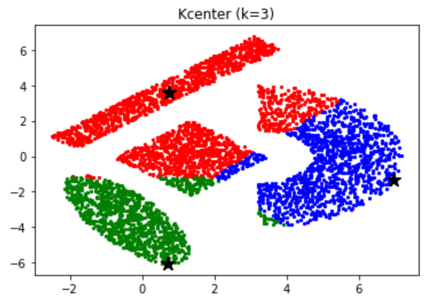
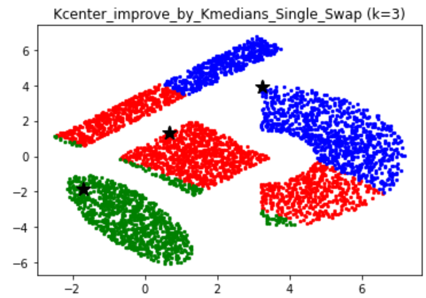
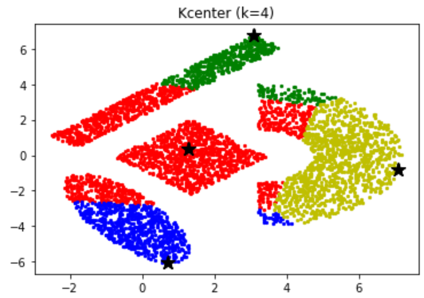
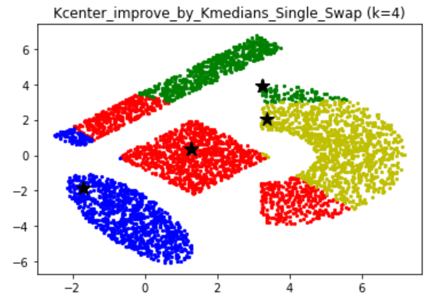
Table 6 Clustering Result: K-center vs. Single Swap

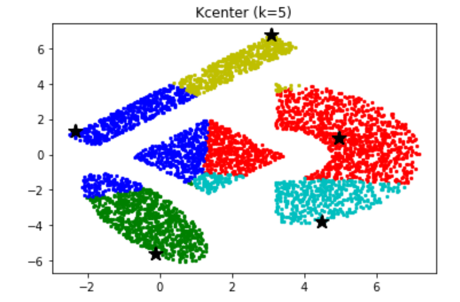
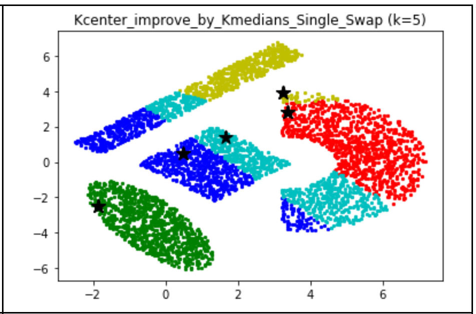
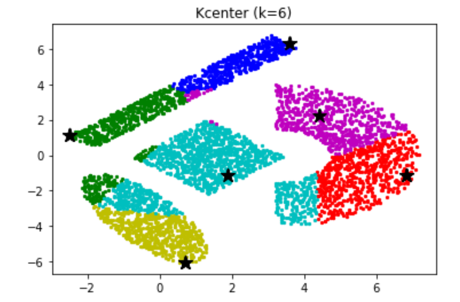
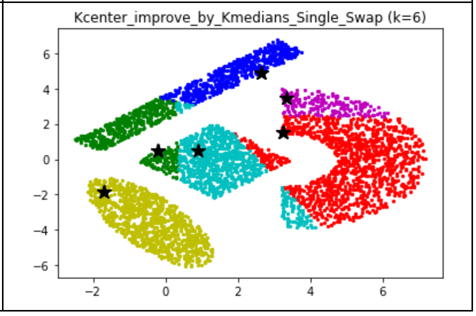
Data: Clustering	Initial K-centers	K-medians Single Swap
K=2 ***Optimal**		

K=3		
K=4		
K=5		

The comparison between k-center and single swap's clustering results of each clustering value are shown below for *Shape* dataset (Table 7).

Table 7 Clustering Result: K-center vs. Single Swap

Data: Shape	**Initial K-centers**	**K-medians Single Swap**
K=3		
K=4 ***Optimal**		

K=5		
K=6		

1.4 Spectral Clustering Algorithm

In spectral clustering, we choose the best average distance as our objective function (Dist.). We use Euclidean distance along with Gaussian similarity function between each point to construct similarity matrix (S) and combined with the adjacency matrix (A) we obtained by k-nearest neighborhood structure to get the weighted adjacency matrix (W). Also, by summing values of each row in adjacency matrix, we got the diagnose matrix (D).

$$S(x_i, x_j) = e^{-\left(\|x_i - x_j\|_2 / 2\sigma^2\right)}$$

σ controls the width of the neighborhoods.

Then we can obtain the Laplacians matrix L = D-W, after we obtain the L matrix we need to calculate its eigenvalues and eigenvectors. By sorting the eigenvalues of L, we could know the optimal number of clusters the subjected dataset is distributed. And the eigenvectors represent the grouping connection.

In this part, I have come up with the algorithm of the spectral clustering. However, I not able to put the outcomes of the spectral clustering into k-means algorithm to obtain the final clustering assignment.

Although I not able to plot the final clustering assignment for each data set, I know that spectral clustering will be the best clustering method for *Shape* data set. Because spectral clustering make no assumption on the shape of cluster. It is able to obtain a good result for the data set whose distribution has obvious patterns.

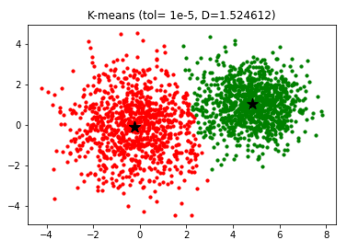
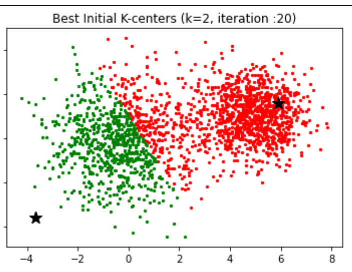
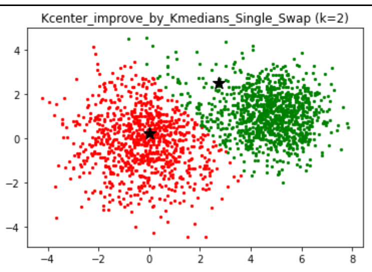
II. Discussion:

We have successfully come up with the algorithm for each data set, and now it’s time to discuss the strength and weakness for each algorithm.

2.1 Algorithm Comparison – Clustering Dataset

Comparing the clustering result for k=2, we noticed that K-means is the one who can obtained the best clustering solution for *clustering* dataset. Since the clustering dataset has a shape of filled circle, k-means is able to found the centroid of the clustered data by the mean value (Table 8) .

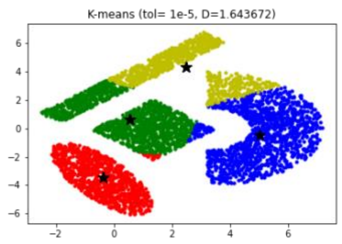
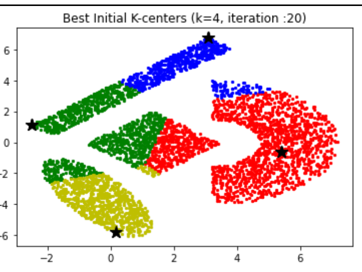
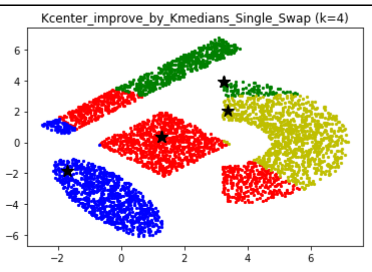
Table 8 Comparison Between Algorithms

Algorithm/Clustering Plot		
K-means	K-centers	K-medians (Single Swap)
		

2.2 Algorithm Comparison – Shape Dataset

Comparing the clustering result for k=4, we noticed that K-medians is the one who can obtained the better clustering solution for *shape* dataset. Since the distribution of *shape* dataset has an obvious pattern with high density, k-medians is able to find the medoid of the clustered data by the minimize the dissimilarity of each data point (Table 9).

Table 9 Comparison Between Algorithms

Algorithm/Clustering Plot		
K-means	K-centers	K-medians (Single Swap)
		

2.3 Strength and Weakness

The differences and strength and weakness of each algorithm is provided below. (Table 10)

Table 10 Algorithm Comparison

Algorithm	**K-means**	**K-centers**	**K-medians (Single Swap)**
Strength	1. The center of the cluster is not necessarily one of the input data points. 2. It is a simple and good method for normal datasets.	1. The first set of assigned centers will be sparse enough to avoid poor clustering result.	1. More flexible, it consider the dissimilarity of data set. 2. More robust and immune to noise and outlier.

Weakness	1. The result of k-means algorithm greatly relies on the first set of randomly assigned centroids. 2. Not immune to noise and outliers.	1. Centers might be too distributed.	1. Not able to deal with shape data.

References

[1] Sharma, P. (2021). *K Means Clustering: K Means Clustering Algorithm in Python*. Analytics Vidhya. Retrieve from https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

[2] Scikit Learn (2021). Clustering. Retrieve from https://scikit-learn.org/stable/modules/clustering.html

[3] Geeks4Geeks. (2021). *K-centers Problem | Set 1.Greedy Approximate Algorithm*. Geeks4Geeks. Retrieve from https://www.geeksforgeeks.org/k-centers-problem-set-1-greedy-approximate-algorithm/

[4] NumPy-norm (2021). Retrieve from https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

[5] Tyagi, H. (2020). *Calculating Vector P-Norms — Linear Algebra for Data Science - IV*. Towards Data Science. Retrieve from https://towardsdatascience.com/calculating-vector-p-norms-linear-algebra-for-data-science-iv-400511cffcf0

[6] Sahani, G.R. (2020). *Euclidean and Manhattan distance metrics in Machine Learning*. Medium. Retrieve from https://medium.com/analytics-vidhya/euclidean-and-manhattan-distance-metrics-in-machine-learning-a5942a8c9f2f

[7] Nguyen, T. (2019). *K-Medoids Clustering on Iris Data Se*t. Towards Data Science. Retrieve from https://towardsdatascience.com/k-medoids-clustering-on-iris-data-set-1931bf781e05

[8] Lucińska, M., & Wierzchoń, S. T. (2012, September). *"Spectral clustering based on k-nearest neighbor graph"*. In IFIP International Conference on Computer Information Systems and Industrial Management (pp. 254-265). Springer, Berlin, Heidelberg. https://dl.ifip.org/db/conf/cisim/cisim2012/LucinskaW12.pdf

[9] Sacconi, D. (2017). *Gaussian similarity kernel*. Stack Exchange. Retrieve from https://datascience.stackexchange.com/questions/25604/how-do-you-set-sigma-for-the-gaussian-similarity-kernel

[10] Hupphurr. (2020). *Finding the K-smallest values of a NumPy array*. Geeks4Geeks. Retrieve from https://www.geeksforgeeks.org/finding-the-k-smallest-values-of-a-numpy-array/

[11] Fleshman, W. (2019). *Spectral Clustering. Towards Data Science*. Retrieve from https://towardsdatascience.com/spectral-clustering-aba2640c0d5b

[12] Von Luxburg, U. (2007). *"A tutorial on spectral clustering"*. Statistics and computing, 17(4), 395-416.

[13] Gupta, A., Majumder, D. (2021), *"Elbow Method for optimal value of k in K-Means"*. GeeksforGeeks. Retrieve from https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

Appendix

I. K-means

```
def dist(a,b): #ord_norm: none,1,'fro'
    return np.linalg.norm(a - b)
---------------------------------------------------------------------------------------------------
def k_means_tol_1(X,K): # X: d-dimensional obersvation, K: number of clusters
    index = np.zeros(len(X)) # index matrix
    phi = np.zeros(len(X))

    tol= 10**(-5) #convergence threshold
    Y = np.array(random.sample(list(X),K)) # randomly select K oberservation
from dataset to set as our initial centers
    Y_old = np.zeros(Y.shape) # new centers awaiting assignment
    con = np.linalg.norm(Y-Y_old,ord=1) # 1-norm btw Y and Y_new
    avg_D=0 # Obj value
    colors = ['r', 'g', 'b','y', 'c', 'm','orange','purple'] # color pallete for plotting cluster

    while con > tol:

        """
        Assign data to the closest center (with label)
        """
        for i in range(len(X)): # for each observation
            min=9999
            for j in range(0,K): # compare distance with each center
                distances = dist(X[i], Y[j])
                if distances < min:
                    min=distances
                    phi[i] = distances # store the distance to the closest center for each
observation
                    index[i] = j # store the index (label) for each observation to locate its
closest center

        Y_old = deepcopy(Y)

        """
        Compute cluster mean and assign it as the new center
        """
        for j in range (K):
            group_data_point=[X[i] for i in range (len(index)) if index[i]==j]
            Y[j] = sum(group_data_point)/len(group_data_point)
            con = np.linalg.norm(Y-Y_old,ord=1)
```

```
        #print(con)

      """
      D (objective value)
      """
      avg_D = sum(phi)/len(phi)

   """
   Plot with color by clusters
   """
   for z in range (K):
      data=np.array([X[i] for i in range (len(index)) if index[i]==z])
      plt.scatter(data[:, 0], data[:, 1],s=10, c=colors[z])
      plt.scatter(Y[:, 0], Y[:, 1], marker='*', s=150, c='k')
      plt.title('K-means (tol= 1e-5, D=%f)'%(avg_D))
      plt.savefig('K-means (tol= 1e-5, K=%d, D=%f).png'%(K,avg_D))

   plt.show()

   return Y,index,avg_D
```

II. K-centers

```
def dist(a,b): #ord_norm: none,1,'fro'
   return np.linalg.norm(a - b)

def InitialCenters(X,K):

   Y=[] #list for storing centers
   first_c_position= np.random.randint(X.shape[0]) # randomly select a position
within obersavation
   Y.append(X[first_c_position]) # add the randomly selected observation to
center list
   label = np.zeros(len(X)) # index matrix
   phi = np.zeros(len(X))

   """
   search for another (k-1) centers from observation
   """
   for i in range(K-1):

      distance=[]
      #phi=0
```

```
        """
        1. find the nearest center for each observation
        2. calculate d(xi,cj)
        """
        for i in range(len(X)): # for each observation
            min=9999
            for j in range(len(Y)): # compare distance with each center
                d = dist(X[i], Y[j])
                if d < min:
                    min=d
                    #phi[i] = distances # store the distance to the closest center for each
observation
                    #index[i] = j
            distance.append(min)  # store the distance to the closest center for each
observation (by order)

        """
        select the observation with the biggest distance as the next center
        """
        max_dist = max(distance)
        for n in range(len(distance)):
            if distance[n]==max_dist:
                next_c_position = n
        Y.append(X[next_c_position])

        distance=[]

    """
    Grouping
    """
    for i in range(len(X)): # for each observation
        min_val=9999
        for j in range(0,K): # compare distance with each center
            distances = dist(X[i], Y[j])
            if distances < min_val:
                min_val=distances
                phi[i] = min_val # store the distance to the closest center for each
observation
                label[i] = j
    obj=max(phi)
    #max_dist

    return Y, label, obj
```

```
def Greedy_K_centers (X,K):

    iteration = 20
    Y_store=[0]*iteration
    obj_store=[0]*iteration
    label_store=[0]*iteration
    colors = ['r', 'g', 'b','y', 'c', 'm','orange','purple']

    for i in range (0,iteration):
        d=999
        Y,label,obj=InitialCenters(X,K)

        Y_store[i]=Y
        label_store[i]=label
        obj_store[i]=obj



    min_d=9999
    index=0
    for j in range(len(obj_store)):
        if obj_store[j]< min_d:
            min_d = obj_store[j]
            index=j

    Y_best=np.array(Y_store[j])
    label_best=np.array(label_store[j])

    """
    plt.scatter(X[:, 0], X[:, 1], c='b')
    plt.scatter(Y_best[:, 0], Y_best[:, 1], marker='*', s=150, c='k')
    plt.title('Best Initial K-centers (k=%d, iteration :20)'%(len(Y)))
    plt.show()
    """

    for z in range (K):
        data=np.array([X[i] for i in range (len(label_best)) if label_best[i]==z])
        plt.scatter(data[:, 0], data[:, 1],s=5, c=colors[z])
        plt.scatter(Y_best[:, 0], Y_best[:, 1], marker='*', s=150, c='k')
        plt.title('Best Initial K-centers (k=%d, iteration :20)'%(len(Y)))
        plt.savefig('K-centers (k=%d, iteration :25)'%(len(Y)))
    plt.show()


    return Y_best, min_d
```

III. K-medians Single Swap

```
def dist(a,b): #ord_norm: none,1,'fro'
    return np.linalg.norm(a - b)

def InitialKCenters(X,K):

    Y=[] #list for storing centers
    first_c_position= np.random.randint(X.shape[0]) # randomly select a position
within obersavation
    Y.append(X[first_c_position]) # add the randomly selected observation to
center list
    label = np.zeros(len(X)) # index matrix
    phi = np.zeros(len(X))

    """
    search for another (k-1) centers from observation
    """
    for i in range(K-1):

        distance=[]
        #phi=0

        """
        1. find the nearest center for each observation
        2. calculate d(xi,cj)
        """
        for i in range(len(X)): # for each observation
            min=9999
            for j in range(len(Y)): # compare distance with each center
                d = dist(X[i], Y[j])
                if d < min:
                    min=d
                    #phi[i] = distances # store the distance to the closest center for each
observation
                    #index[i] = j
            distance.append(min)  # store the distance to the closest center for each
observation (by order)

        """
        select the observation with the biggest distance as the next center
        """
        max_dist = max(distance)
```

```
        for n in range(len(distance)):
            if distance[n]==max_dist:
                next_c_position = n
        Y.append(X[next_c_position])

        distance=[]

    """
    Grouping
    """
    for i in range(len(X)): # for each observation
        min_val=9999
        for j in range(0,K): # compare distance with each center
            distances = dist(X[i], Y[j])
            if distances < min_val:
                min_val=distances
                phi[i] = min_val # store the distance to the closest center for each
observation
                label[i] = j
    obj=max(phi)
    #max_dist

    #return Y, label, obj

    return Y, label

def all_cost(X,Q): # X:data, Q:medians

    N,d = X.shape
    n_k = np.array(Q).shape[0]

    D = np.zeros((N,n_k))

    for i in range (X.shape[0]):
        for j in range (len(Q)):
            cost= dist(X[i],Q[j])
            D[i][j]=cost

    return D

def Label(D): # input: the output of cost function

    N,kk = D.shape
    labels=np.zeros(N)
```

```
    obj_cost=[] #

    for i in range (N):
        min=999
        for j in range (kk):
            c= D[i][j]
            if c < min:
                min = c
            #labels[i] = j
                labels[i] = j
        obj_cost.append(min) #

    sum_cost= sum(obj_cost) #

    return labels, sum_cost

def single_swap_debug(X,Q,tau):

    """
    Assign each data point to the new cluster
    - calculating the cost(distance) for <all observations>
    - reassign labels
    """
    D = all_cost(X,Q)
    label,sum_cost = Label(D) #

    new_Q = Q

    for i in range(len(Q)):
        # collect data points in cluster(i) as a new list
        cluster_data_0 = np.array([X[j] for j in range (len(label)) if label[j]==i])

        # calculate the cost matrix for each cluster (median)
        cluster_cost = all_cost(cluster_data_0,Q[i])
        cost_Qi = np.sum(cluster_cost)


        """
        Calculate the new cost:
        by assigning each data points in cluster (i) other than median (i) as the new
median
        """
        for a in cluster_data_0: # grab each data points in cluster (i) by iteration
            new_Qi = a
```

```
        new_cluster_cost = all_cost(cluster_data_0,a) ## ??? how to exclude [a] in
[cluster_data]
        new_cost_Qi = np.sum(new_cluster_cost)

        """
        Compare new cost with the old one:
        if it reduce by (1-tau), update median(i)
        """
        if new_cost_Qi <= (1-tau)*cost_Qi :
          cost_Qi =  new_cost_Qi

          new_Q[i] = a # update median(i)

        #else:
        #    new_Q[i] = Q[i]

  new_D = all_cost(X,new_Q)
  new_label, new_sum_cost = Label(new_D) #

  return new_Q, new_label, new_sum_cost

def convg_optimal(Q,new_Q):

  #use set() to hash the tuple and sort it
  Q_val = set([tuple(c) for c in Q])
  new_Q_val = set([tuple(c) for c in new_Q])

  """
  True: medians does not move after swaps
  False: medians moved
  """
  return Q_val == new_Q_val

def Kcenter_improve_by_Kmedians_Single_Swap(X,K,tau,max_same): #debug
(test which swap is correct)

  colors = ['r', 'g', 'b','y', 'c', 'm','orange','purple']

  """
  Generate the initial k-centers
  """
  first_Q,first_label = InitialKCenters(X,K)
```

```
    for f in range (K):
        data=np.array([X[i] for i in range (len(first_label)) if first_label[i]==f])
        plt.scatter(data[:, 0], data[:, 1],s=5, c=colors[f])
        plt.scatter(np.array(first_Q)[:, 0], np.array(first_Q)[:, 1], marker='*', s=150,
c='k')
        plt.title('Kcenter (k=%d)'%(len(first_Q)))
        #plt.savefig('K-centers (k=%d, iteration :25)'%(len(Y)))
    plt.show()

    new_Q = first_Q.copy()

    Stop = False
    i=0

    #colors = ['r', 'g', 'b','y', 'c', 'm','orange','purple']

    # Stop swapping if the medians did not move for [max_same] iterations
    while (not Stop) and (i <= max_same):

        Q = new_Q.copy()

        """
        Perform single swap heuristic
        """
        new_Q, new_label, new_sum_cost = single_swap_debug(X,new_Q,tau)


        """
        True: medians does not move after swaps
        False: medians moved
        """
        Stop = convg_optimal(Q,new_Q)

        i+=1 # count how many times does medians remain the same




    for z in range (K):
        data=np.array([X[i] for i in range (len(new_label)) if new_label[i]==z])
        plt.scatter(data[:, 0], data[:, 1],s=5, c=colors[z])
        plt.scatter(np.array(new_Q)[:, 0], np.array(new_Q)[:, 1], marker='*', s=150,
c='k')
```

```
        plt.title('Kcenter_improve_by_Kmedians_Single_Swap
(k=%d)'%(len(new_Q)))
        #plt.savefig('K-centers (k=%d, iteration :25)'%(len(Y)))
    plt.show()

    return new_Q, new_label, new_sum_cost, first_Q, first_label
```

IV. Spectral Clustering

```
def dist(a,b):
    return np.linalg.norm(a-b)

# Gaussian Similarity Function
def Gaussian_Similarity(x1,x2,sigma):
    return math.exp(-(dist(x1,x2))/(2*(sigma**2)))

# Similarity Matrix
def Similarity_Matrix(X,sigma):

    N,d = X.shape
    S = np.zeros((N,N))

    for i in range (N):
        for j in range (i,N):
            s = Gaussian_Similarity(X[i],X[j],sigma)
            S[i][j]=s
            S[j][i]=s

    return S

def Spectral_Clustering(X,K,sigma):

    N,d = X.shape
    #S = np.zeros((N,N)) # Gaussian Similarity matrix (euclidean)
    W = np.zeros((N,N)) # adjacency matrix
    C = np.zeros(N)
    #Y =

    """
    S: Similarity matrix: store the gaussian similarity value between each data point
    """
    S=Similarity_Matrix(X,sigma) # Gaussian Similarity matrix (euclidean)

    """
```

```
    A: K-nearest neighborhood structure (knn), in order to determine if there is an
edge between nodes or not
    """
    # W = kneighbors_graph(X, n_neighbors=5, metric=S).toarray()
    nrst_neigh = NearestNeighbors(n_neighbors = int(N/K), algorithm = 'ball_tree')
    nrst_neigh.fit(X)
    A = nrst_neigh.kneighbors_graph(X).toarray()


    """
    W: Weighted Adjacency Matrix [N*N] --> combine S & A
    """
    for i in range(N):
        for j in range (i,N):
            if A[i][j]== 1: #there is an edge between node i and j
                W[i][j]= S[i][j]
                W[j][i]= W[i][j]
            else:
                W[i][j]= 0
                W[j][i]= W[i][j]

    """
    D matrix [N*N], degreeness
    """
    diag = np.sum(W, axis=1) #sum each value of each row in adjacency matrix
    D = np.diag(diag)

    """
    Liplacian matrix [N*N]
    """
    L = D-W

    """
    U: first K eigenvectors [N*K dimension]
    """
    eivals, U = np.linalg.eigh(L)
    U=U[:,-K:]

    input_data = np.array(U)

    #Y,C,D = k_means_tol_1(input_data,K,X)
    #return S,U,Y,C,D

    return W,U
```