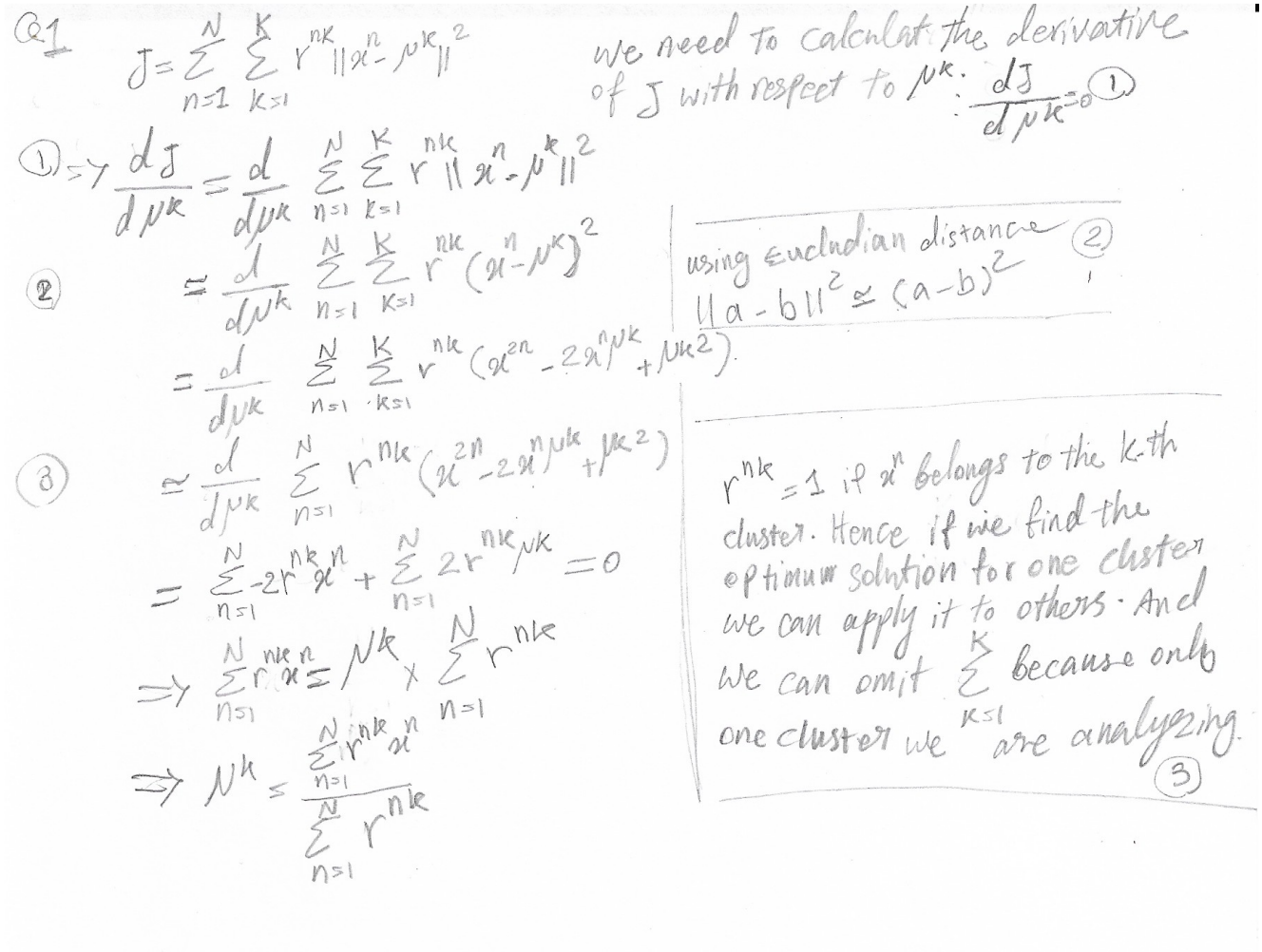
ISYE 6740 Homework 1

Q1 (a)



Q1 (b)

In K-mean algorithm, there is a defined number of iterations in which in each iteration, either

- a new mean is discovered that reduces the J cost function
- or the current mean still is picked because the current cost function is producing the minimum

if we pick the right number iterations, J cost function, considering the first option above, will change and find new mean with less cost and eventually, for a local optimum, it will find the minimum and converges (ends) the algorithm.

Q1 (c)

Looking at the problem from hierarchical tree based clustering problem, single linkage distance metrics will successfully separate the two moons. Firstly, the two moon cluster is a special case (discussed in

various research) in which observation points are tightly clumped in a two trailing clusters and single linkage metrics finds the minimal inter-cluster dissimilarity which fits the best here.

If the hierarchical algorithm starts from any point from any moon, it will identify all the observations tightly clustered (less dissimilar) together and will separate the two moon because observation from one moon are dissimilar enough to not join the other moon (cluster).

Complete linkage in which computes all pairwise dissimilarities between two cluster based on largest distance or average linkage in which computes all pairwise dissimilarities based on average, won't generate similar results.

Q2

My code contains k-means and k-medoids in Jupyter notebook format. The file is also provided in this submission. Please make sure you have the perquisite libraries identified below.
Prerequisite libraries: numpy, skimage
System: Corei7 with 16GB ram and SSD drive

K-mean

I used euclidean distance to calculate the distance between points. All test runs are done in 50 iterations. For details of the results, please refer to the Jupyter notebook 'kmean.ipynb'. The run-time for various K is shown in the table below.

File name	Run time K=20	Run time K=15	Run time K=10	Run time K=5
pic.bmp	19.73	14.48	9.72	5.17
pic.bmp	20.08	14.87	9.79	4.78
pic.bmp	20.46	15.32	9.65	4.72

K-medoid

For K-medoid, I implemented two functions for medoid selection (*adjust_medoids* and *adjust_medoids2*). One used a for loop to go through each node of a cluster to find the minimum dissimilarities. And the other used numpy's function to calculate the pairwise distance (thanks to various discussion on Piazza).

Using numpy's functions (pdis and unique), enhanced the run-time almost 4x. The report below is the run time for using the *adjust_medoids2 function* that used numpy's capabilities for calculating the distance. All test runs are done in 50 iterations. For details of the results, please refer to the Jupyter notebook 'kmedoid.ipynb'

File name	Run time K=20	Run time K=15	Run time K=10	Run time K=5
pic.bmp	23.83	25.38	41.05	92.49
pic.bmp	20.72	28.57	39.36	77.80
pic.bmp	19.41	29.94	37.35	85.39

Answers to questions from the defined task 1- 5

Run your K-medoids implementation with the picture you chose above, with several different K. (e.g, small values like 2 or 3, large values like 16 or 32) What did you observe with different K? How long does it take to converge for each K?

K-medoids's run-time increased for smaller Ks. Due to the nature of algorithm, it makes sens. Smaller K leads to have more points in one cluster and because algorithm requires to check for medoids in each cluster by comparing the distance between all nodes to all other, the run time increases. When K is large, clusters has smaller number of points, and finding the medoids take less time in smaller clusters.

Oppositely, for K-means, with increase in K, run time increases. Larger K leads to more clusters and the algorithm needs to calculate the mean for each of them. Hence, it increases in run time with increase in K. And overall run time of K-means for K in range 5 to 20, was less than K-medoid but I think with larger pictures and larger K, K-medoids may show faster run time as we also witness that with K=20 both algorithms show similar run time. At least for my picture (240 X 320) that was the case.

Run your K-medoids implementation with different initial centroids/representatives. Does it affect final result? Do you see same or different result for each trial with different initial assignments? (We usually randomize initial location of centroids in general. To answer this question, an intentional poor assignment may be useful.)

Both algorithm, sometimes I got blank compressed image. That is due to poor selection of initial centeroid or medoids. This happens for smaller Ks than larger one. And the reason is that if the initial K as color representatives are all from one color, then the all points are assigned to the same color. Hence a blank image emerges as the final result.

Repeat question 2 and 3 with K-means. Do you see significant difference between K-medoids and K-means, in terms of output quality, robustness, or running time?

The output quality were almost similar for various K but K-medoids showed more promising run time performance to K-means with larger K (I have elaborated in question 1). I have not try super large K but I could imagine that K-medoid will show better performance.