Shrikanth Mahale (smahale6)

Solution 1(a)

Beginning with dissimilarity function $|x_n - \mu^k|^2$ in the distortion function below

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r^{nk}|x_n - \mu^k|^2$$

The aim is to reduce J and find out the ideal values of r^{nk} and μ^k. We can do this in the following way.

1. With μ^k fixed we can reduce J with respect to r^{nk}
2. With r^{nk} fixed we can reduce J with respect to μ^k
   Take derivative of J with respect to μ^k while minimizing

$$r^{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin} \left\|x^n - u^k\right\|^2 \\ 0 & \text{otherwise} \end{cases}$$

To minimize first we take the derivative of J with respect to μ^k then set the gradient to 0

$$\frac{dJ}{d\mu^k} = 2\sum_{n=1}^{N} r_{nk}(x_n - \mu^k) = 0$$

$$\sum_{n=1}^{N} r_{nk}(x_n - \mu^k) = 0$$

$$\sum_{n=1}^{N} r_{nk}x_n = \sum_{n=1}^{N} r_{nk}\mu^k$$

$$\sum_{n=1}^{N} r_{nk}\mu^k = \sum_{n=1}^{N} r_{nk}x_n$$

μ^k is the center of k^{th} cluster since μ^k is the mean of all points x_n assigned to cluster k. Since μ^k is not dependent we take it out of the summarization and hence get below and proof.

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$

Solution 1(b)

We need to calculate the cluster centers and then ensure that data points are re assigned to the clusters. This is repeated till there are no changes in the cluster centers any longer or the maximum number of iterations is reached. Each repetition reduces the outcome of distortion function so that the convergence is assured. Since the values of r^{nk} and μ^k are fixed at every step, optimal results are not necessarily assured.

Assuming the algorithm progresses from iteration *t* to *t+1* . No cluster can be repeated `since the number of possible clusters is finite k^n .Hence the algorithm must terminate.

$C^0 \ - \ Initial\ Clustering$

$Sum\ of\ squared\ error\ (C^{t+1}|, \mu^{t+1}) \leq Sum\ of\ squared\ error(C^{t+1}, \mu^t)$

$\sum_{i=1}^{n} |x^i - \mu^t_{C^{t+1}(i)}|^2 < \sum_{i=1}^{n} |x^i - \mu^t_{C^t(i)}|^2$

Solution 1(c)

Any given two moons will be successfully separated by single linkage. By assigning each data point to its own cluster and combining the closest clusters and reaching the top cluster, bottom up hierarchical clustering is ensured. As is evident from the given figure, distance between any given pair of close points in a moon is lesser than the distance between the closest set of points from both moons. Thus all data points on a given moon are clustered first by the method of single linkage. Thereafter any given point from the other moon can be clubbed with the same cluster.

Solution 2

The algorithm for k-means and k-medoids is written in python and the code is as below:

"kmeans_kmedoids.py" – This script contains two different functions for k-means and k-medoids which utilize Euclidean and Manhattan distance metrics respectively.

"process_image.py" – This script imports the k-means and k-medoids functions from the kmeans_kmedoids.py file and processes them with the butterfly image and k value. The final line in the script may be edited to test different images and k values. for example: process(butterfly.bmp',3) will process the butterfly.bmp image placed in the same directory as the script with a k values of 3.

1. ***Within the K-medoids framework, you have several choices for detailed implementation. Explain how you designed and implemented details of your K-medoids algorithm, including (but not limited to) how you chose representatives of each cluster, what distance measures you tried and chose one, or when you stopped iteration.***

   For each cluster, the following steps were performed
   - Select k data points in a random fashion
   - Calculate the Manhatten distance between each data point and the k –medoids and assign the data point to the nearest medoid.
   - Determine the geometric center of each cluster
   - Assign new medoids by calculating the nearest data point to the center of said cluster

   Of the three validated distance measures, (Manhattan, Minkowski and Euclidean) Manhattan was used in the algorithm. Any of the distance measures can be used with similar results. Although the maximum number of iterations possible in the code is 200, roughly 4-8 iterations were utilized.

2. ***Attach a picture of your own. We recommend size of 320 x 240 or smaller.***

   I used an image called "butterfly.bmp" with a resolution of 320 x 240. The image is submitted along with the code.

3. ***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?***

- When number of iterations are fewer usually lesser than 10, smaller K values run faster and for higher number of iterations usually greater than 25, larger K values run slower. For smaller K values with usually less than 10 iterations, image gets highly pixelated and for larger K values usually greater than 25 iterations, image is less pixelated and of better quality
- Larger K values converge within 20 to 25 seconds and Smaller K values converge within 4 to 8 seconds.

4. ***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.)***

   The first centroid assignment is randomized through the code. This leads to variations in the convergence times. With a poor assignment, I could not find a significant difference on run times with K-medoids in most cases. There were major deviations in some cases but I could not find any patterns.

5. ***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?***

   It was observed that K-medoids was 4 to 6 times faster than K-means in most of the runs. The processing time for K-medoids is much faster than K-means as it requires fewer iterations.
   When K values are same for both K-means and K-medoids, then for smaller values of K, the quality of the images of K-medoids tend to be slightly better than K-means.

   When K values are same for both K-means and K-medoids, then for larger values of K, there is no noticeable difference in output quality or robustness.

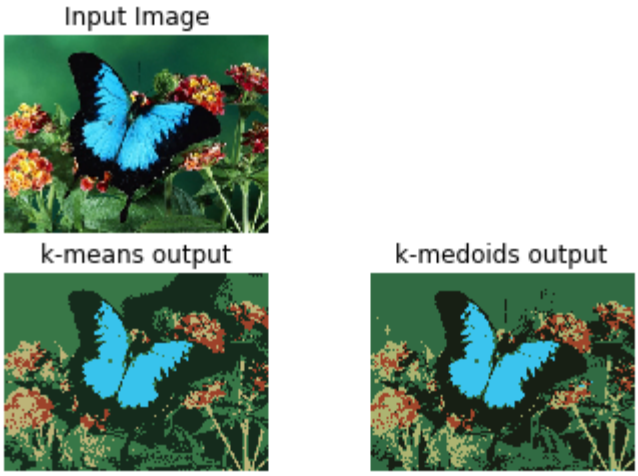
Below are some outputs for illustration:

Case 1: Lower value of K
K = 5

Code:
process_image('butterfly.bmp',5)

Output:

```
In [25]: process_image('butterfly.bmp',5)
Number of k-means iteration: 34
Number of k-medoids iteration: 8
```



Case 2: Higher value of K
K = 32

Code:
process_image('butterfly.bmp',32)

Output:

```
In [27]: process_image('butterfly.bmp',32)
Number of k-means iteration: 201
Number of k-medoids iteration: 11
```