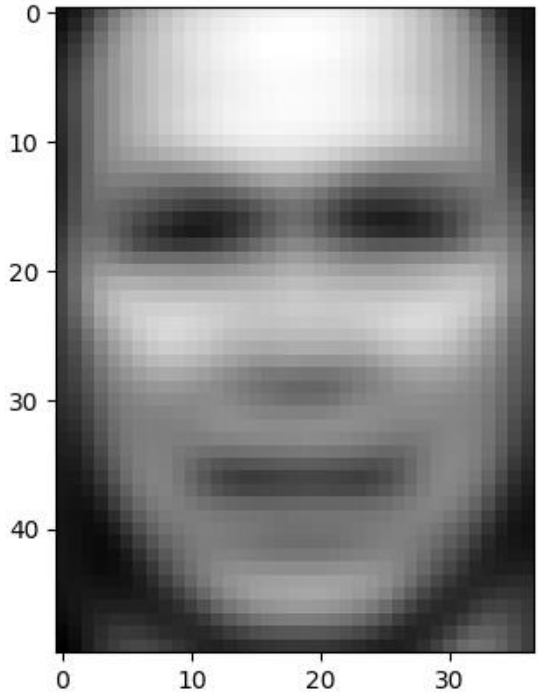
Danning Yu, 305087992
CS M146 Discussion 1A
PS4
3/8/2020

1. (a) This average face looks very generic (and even a bit creepy), as it contains two dark blobs for where the eyes should go, some contours in the middle for a nose, and a dark band in the lower half for a mouth. Because it is an average, the face is centered on the image. The eyes, nose, and mouth are roughly located where you would expect them to if you are look at a person head-on.



(b) From the 12 images shown below, it seems that each one is different from the others in some visible way: the lighting is very bright or very dark, or the face is very well-defined or not very well-defined. Each face itself also looks different. They were most likely selected because these eigenfaces represent the most unique/different representations. As PCA was used, we can interpret it as we are going to categorize the rest of the pictures as being comprised of features/components from these 12 faces.

(c) Based on the pictures below, as l increase, the pictures become more "clear," distinct, and specific, in that that they retain more detail of the original image. This makes sense, because l is a measure of how many components of the original picture we retain. The $l = 1$ images all look very similar to each other and the generic image (see part (a)), as only the most principal component is being retained, while the $l = 500$ and $l = 1288$ images look detailed and can be more confidently attributed to a particular person . Also, it seemed that there was very little difference between the $l = 500$ and $l = 1288$ sets of images, so past $l = 500$, there may be diminishing returns. Thus, for facial recognition purposes, a too low value of l is not helpful, as then all images look the same, but a value of l (such as 1288) that is too high is probably not useful either, as it does not offer much more extra clarity while potentially having much worse performance due to more principal components being retained.

The images are given below.

$l = 1$

$l = 10$

$l = 50$

$l = 100$

$l = 500$

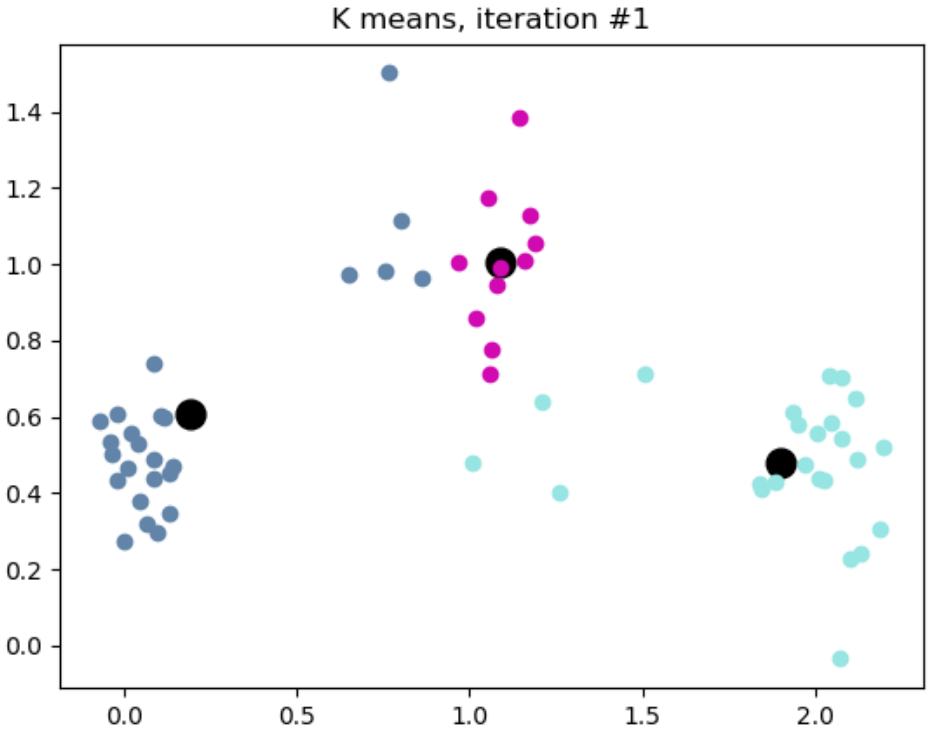
$l = 1288$

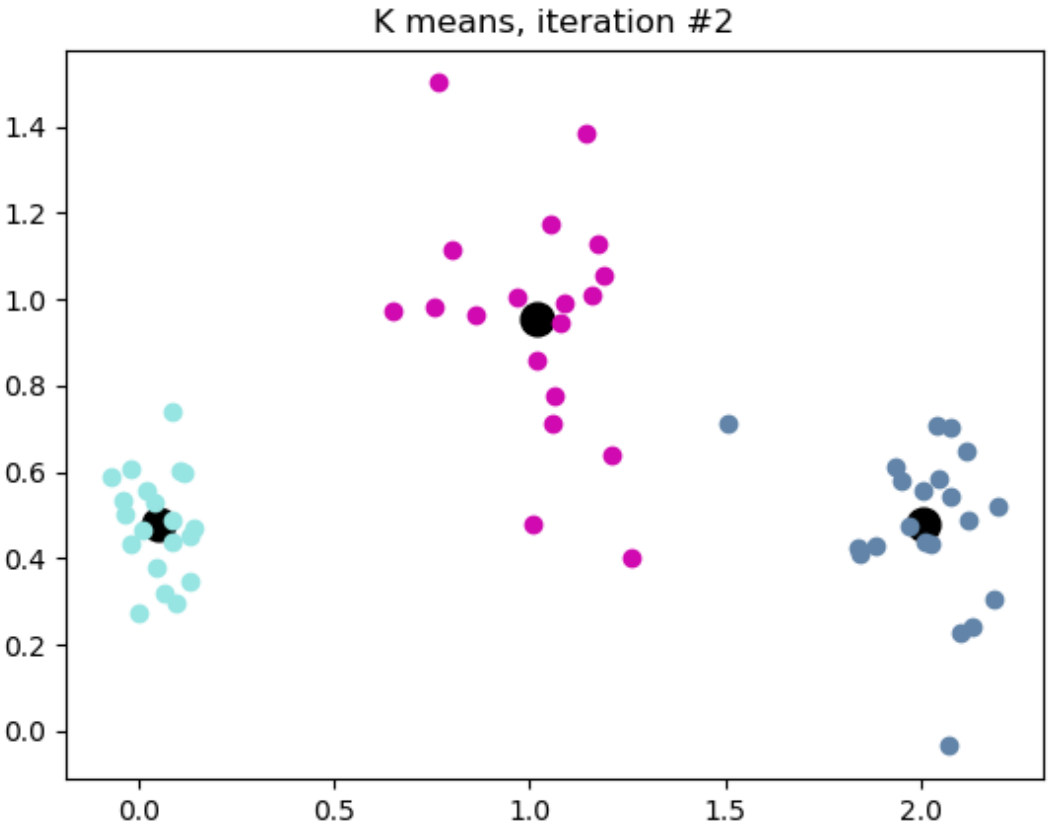
2. (a) It would be a bad idea to optimize over $\boldsymbol{\mu}$, c, and k (and thus not keeping k fixed) because then the optimal solution would just to be let $k = n$. However, having n clusters wouldn't make sense, because then you are not actually clustering any data; rather, you're just artificially making each point the center of its own cluster, giving you no extra new information. For this case of $k = n$, each point is assigned to a cluster centered at itself, and so the resulting cost will be 0. The value of $\boldsymbol{\mu}_i$ in this case would just be $\mathbf{x}_i$, and c would equal i.

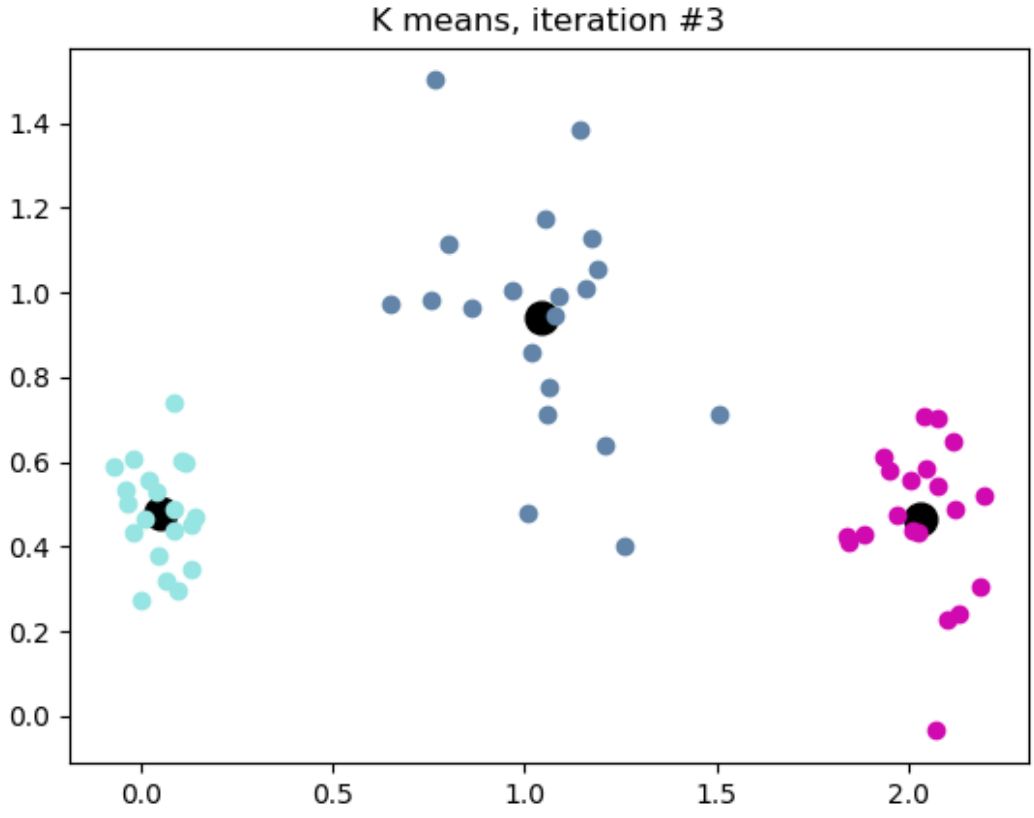
(b) See code.
(c) See code.

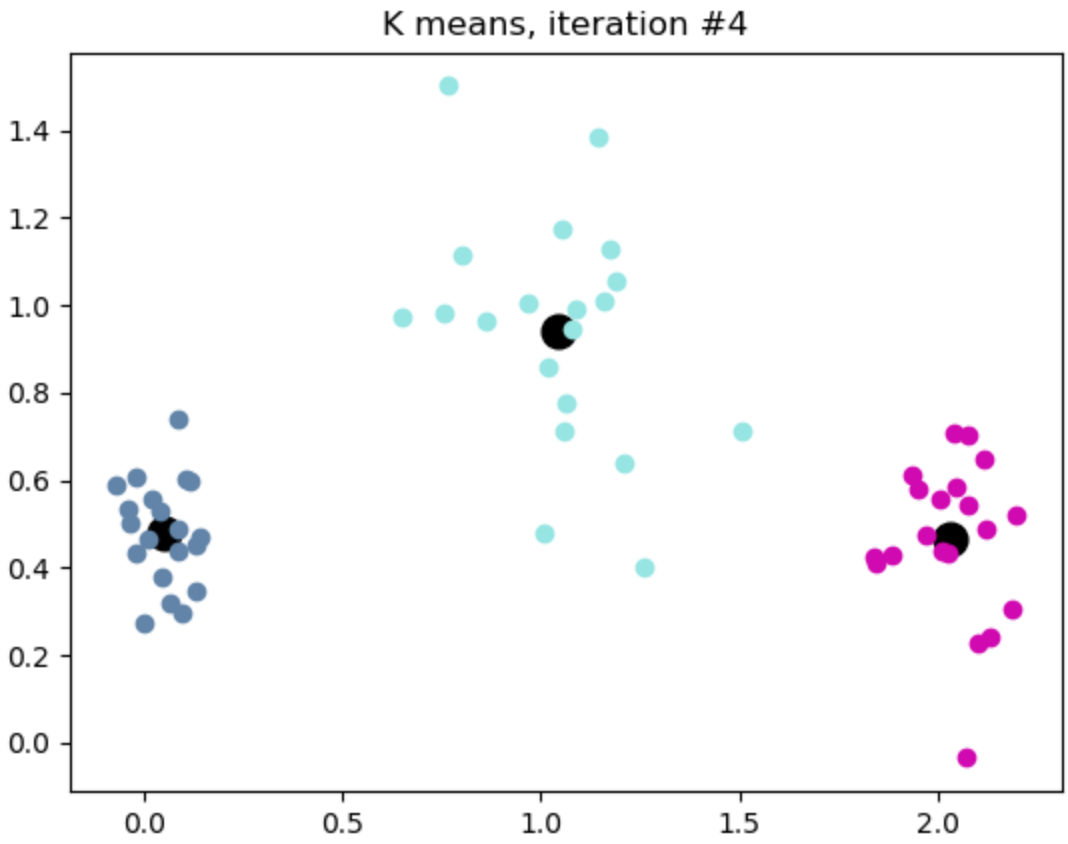
(d) The centroids had the following values. The cluster set converged to a value after 3 iterations: the 4th iteration has the exact same values as the 3rd.

Iteration #	Attribute 1	Attribute 2	Label
1	0.19298238	0.60641572	1
	1.089668	1.00424282	2
	1.90010446	0.48090448	3
2	0.04917974	0.4810944	1
	1.01605529	0.95288767	2
	2.00594139	0.47723895	3
3	0.04917974	0.4810944	1
	1.04063507	0.9409604	2
	2.03085592	0.46538378	3
4	0.04917974	0.4810944	1
	1.04063507	0.9409604	2
	2.03085592	0.46538378	3

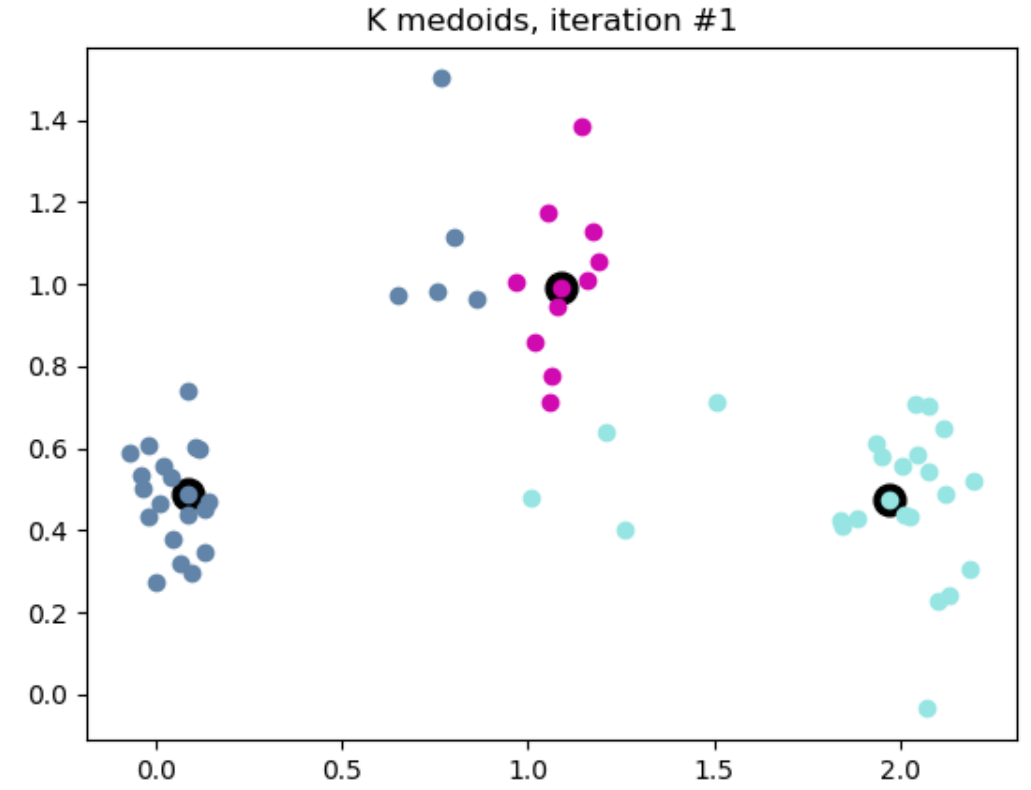


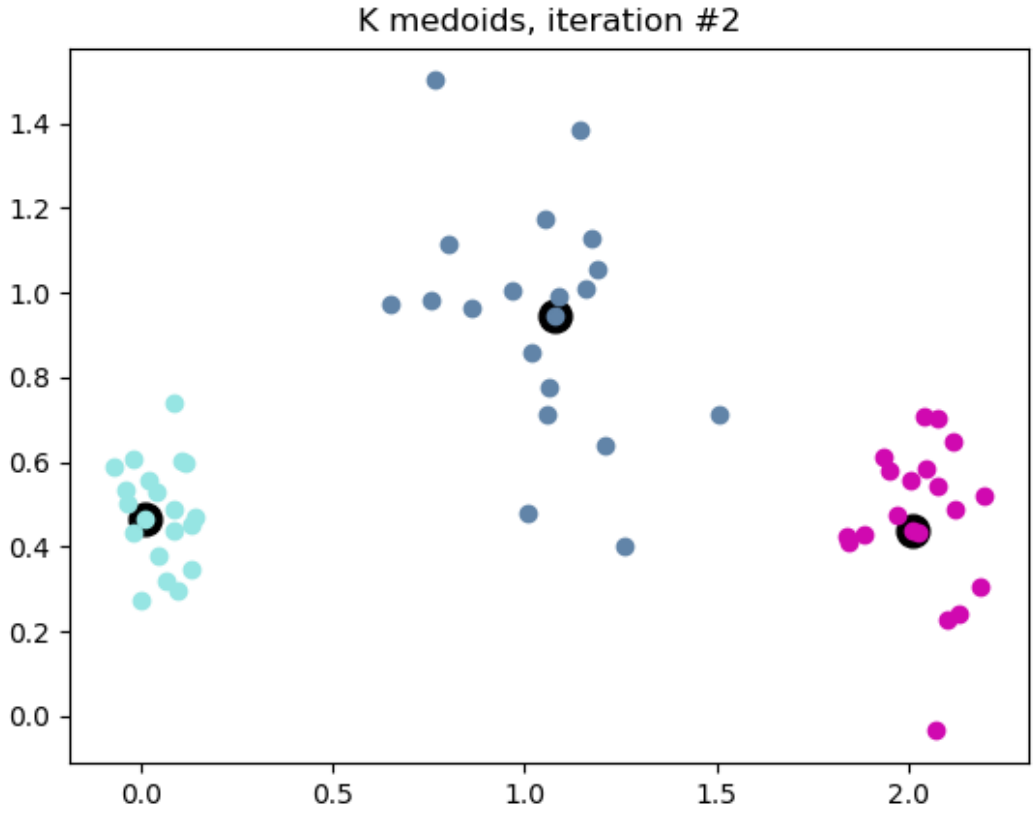

K means, iteration #2
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0

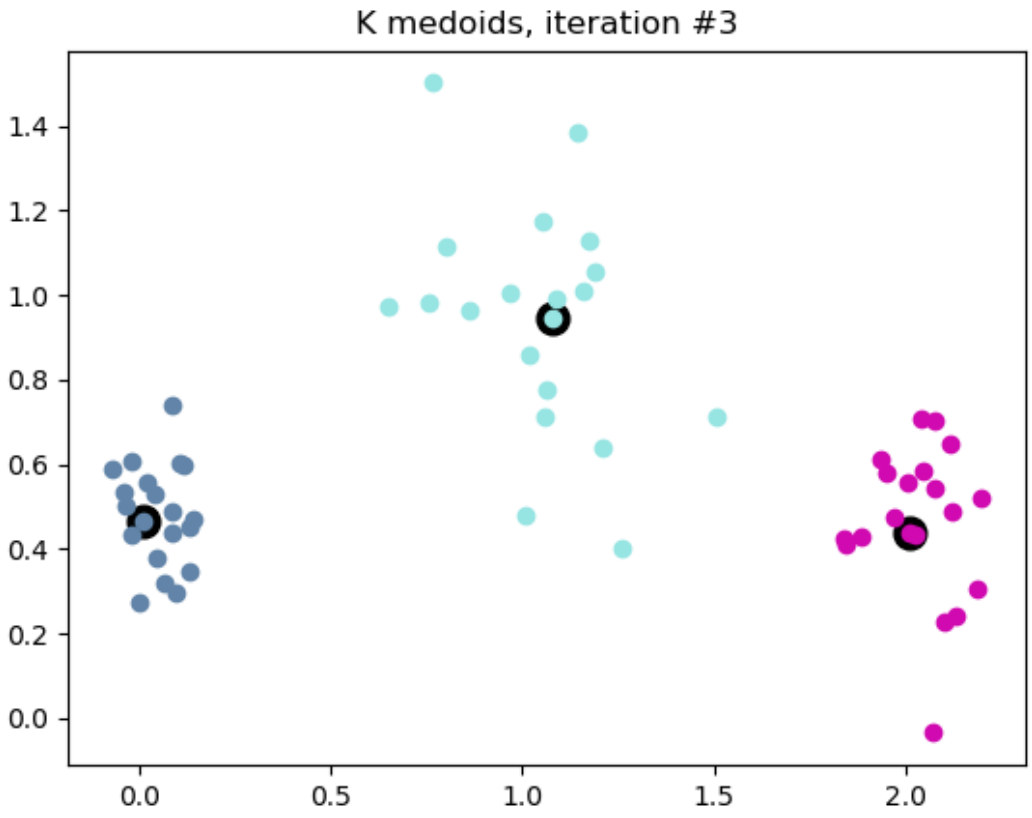

K means, iteration #3
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0


K means, iteration #4
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0

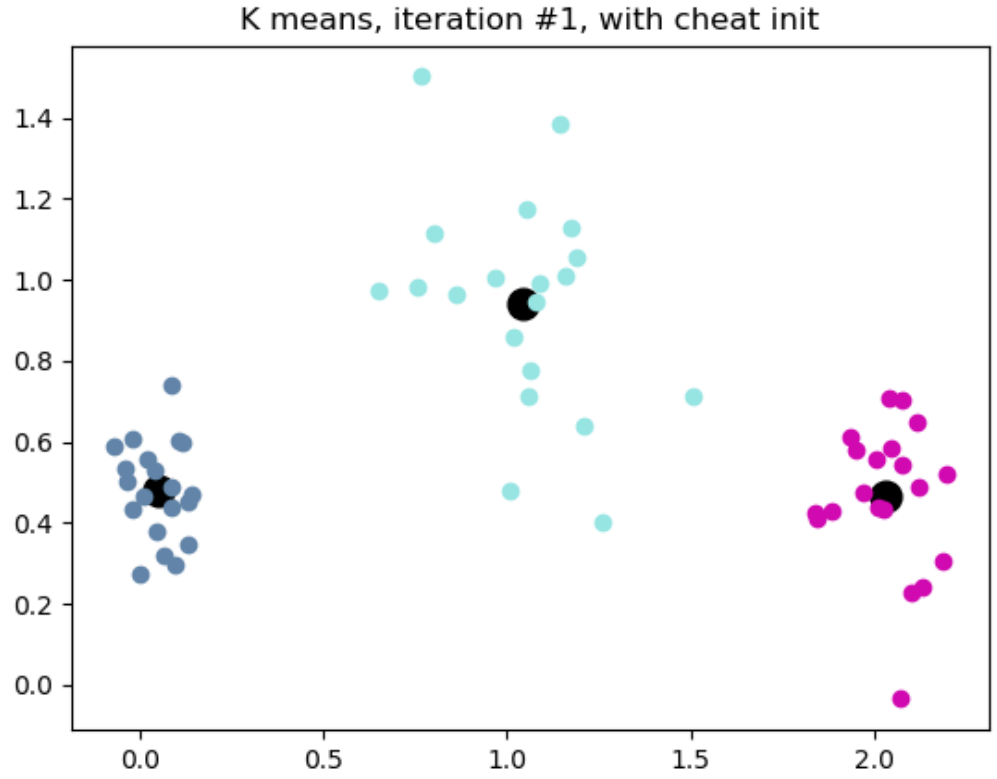
2. (e) The data set converged in 3 iterations this time, with the 2nd and 3rd iterations being the same.

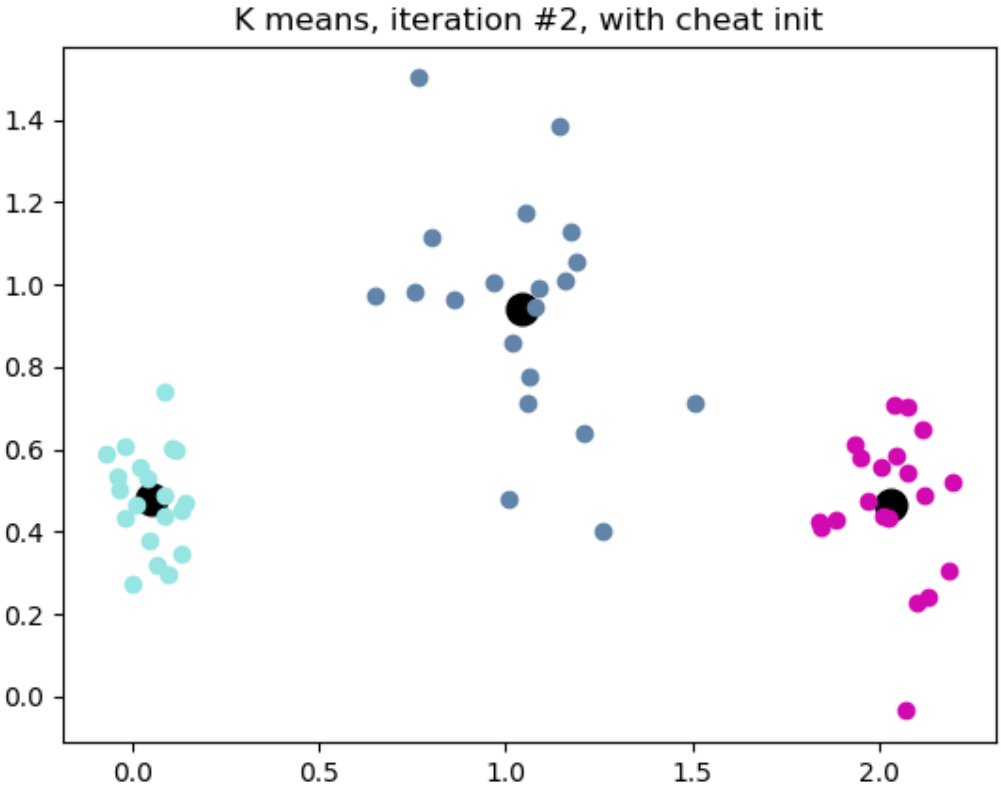




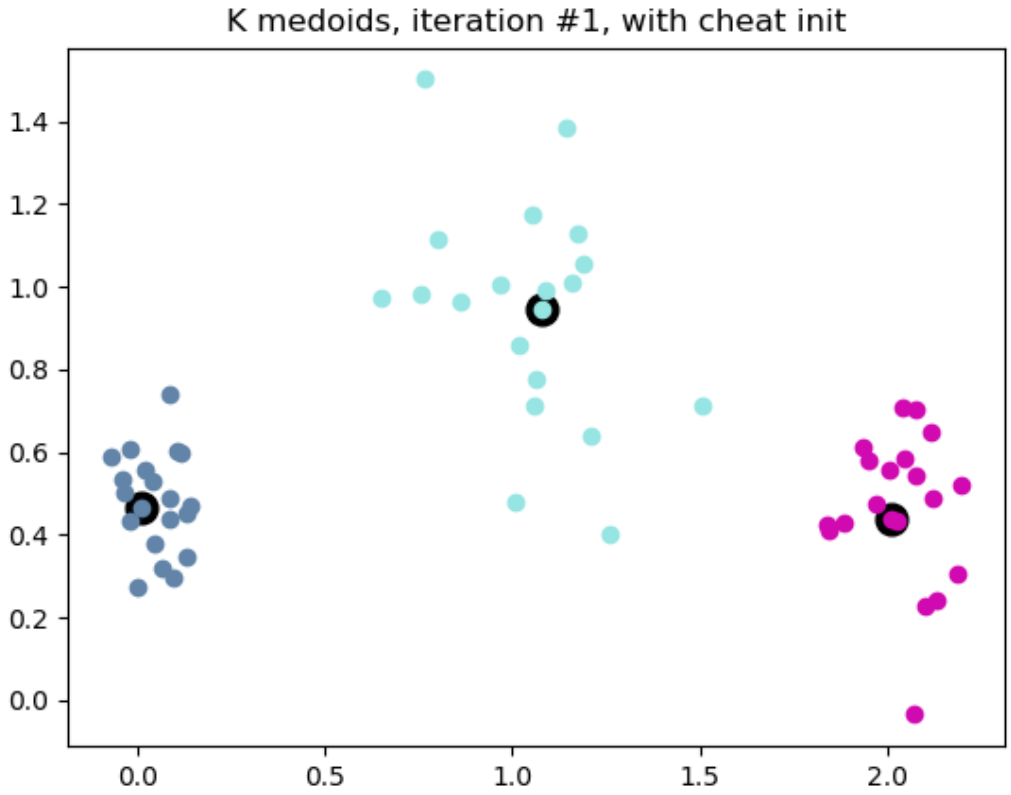

K medoids, iteration #3
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0

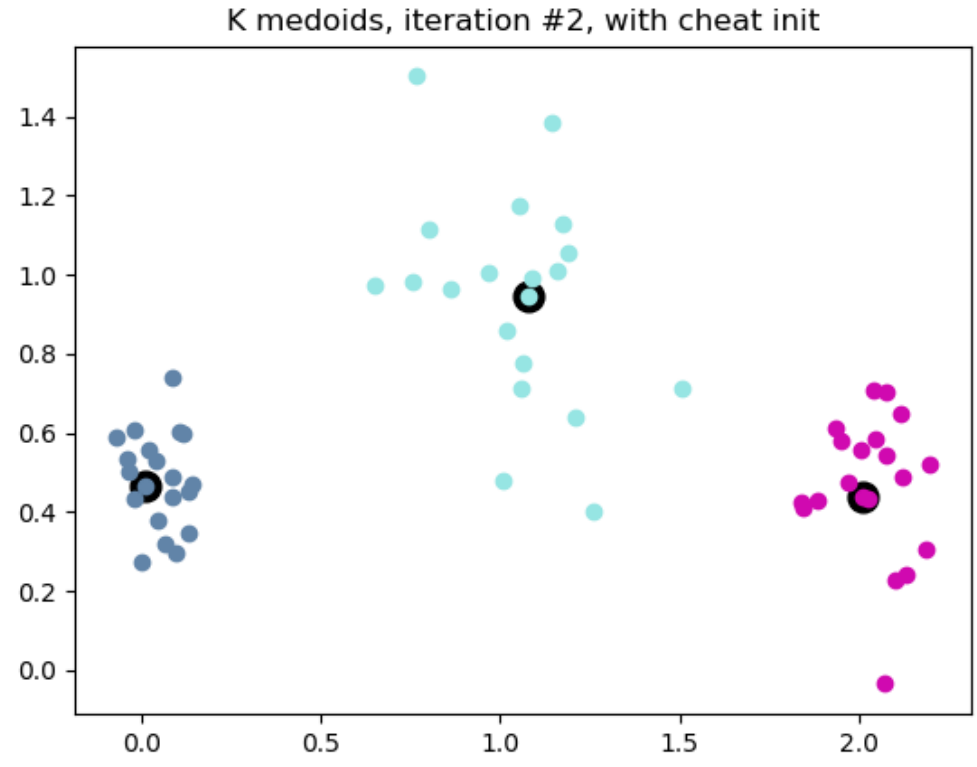
2. (f) The program terminated after 2 iterations for K-means, indicating that the clustering it calculated on the initial iteration was the final value (2nd iteration verifies it is same as the first).





Similarly, it seemed that initializing the centers by cheating, only 1 iteration was needed for K medoids. Again, the program ran for 2 iterations to verify that the results of the 1st and 2nd iteration were the same.





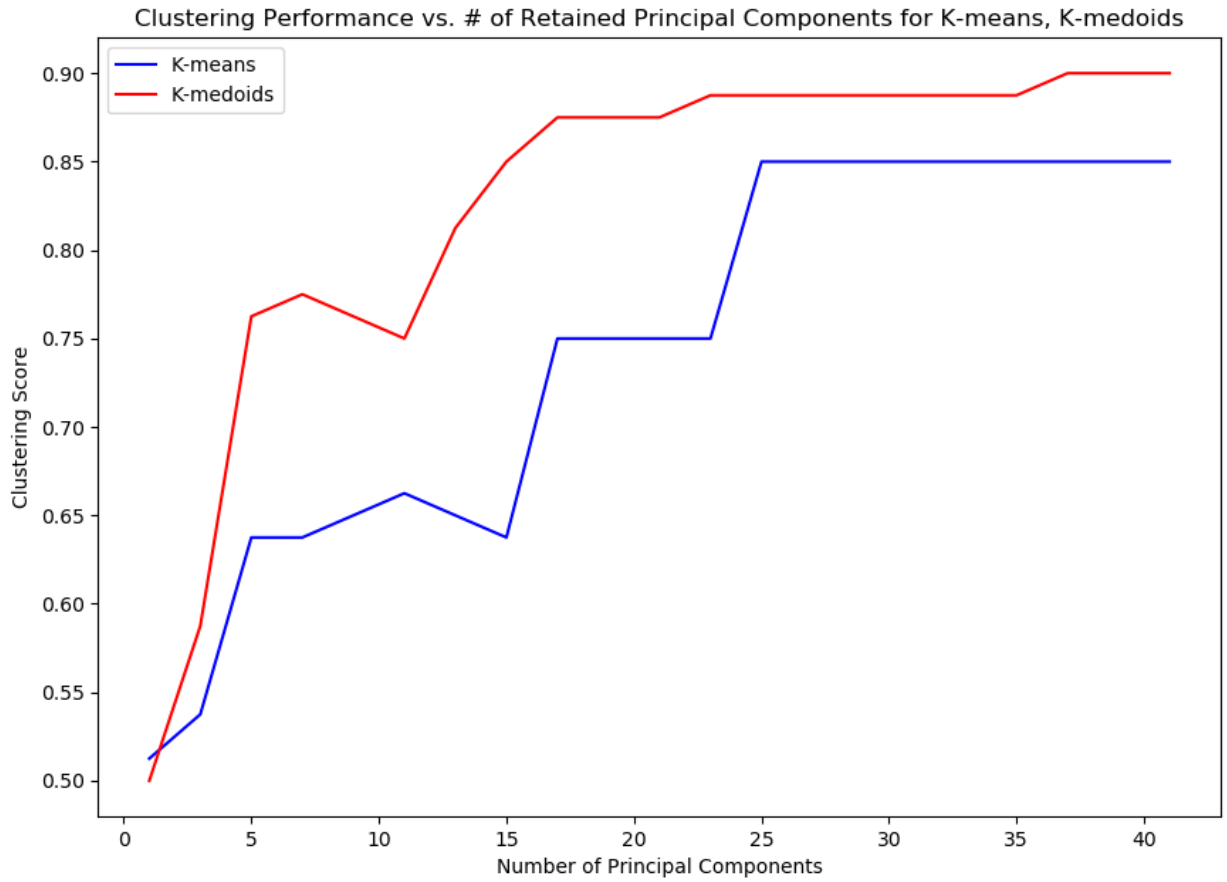
3. (a)

	Average	Min	Max
K-means	0.6231	0.5563	0.7438
K-medoids	0.6594	0.5125	0.75

K-means average run time: 0.5092 s; K-medoids average run time: 0.4674 s

Based on the table above, it seems that K-medoids is better, as on average, it has a slightly higher clustering performance score, a shorter run time, and higher maximum score. However, its minimum is slightly lower than the minimum K-means score, but average score and time is most likely the best measure of performance and runtime here.

(b)



As the number of principal components increases, the performance increases for K-means and K-medoids, which makes sense, as it means we are retaining more information, thus allowing the methods to better find the similarities and differences between data points in the pictures.

Also, we can see that K-medoids almost always does better than K-means, except for the case when only the first principal component is used. Initially, the difference in performance between the two methods is low (for roughly $l < 5$), but then the performance gap quickly widens for values of l between 5 and 25, and then beyond 25, the gap lessens again. I think that K-medoids performed better because it could be that for the problem of facial recognition, it may be more beneficial for the cluster center to be an actual data point rather than an average.

3. (c) My approach was as follows: I selected all possible pairs of classes available out of the 19 classes, and for each pair, I selected 40 images (same as part (b)) from that class, and then built an image from those 40 selected images. Then, I ran it through the K-medoids algorithm with 2 clusters (because we are comparing two images) and cheating initialization so that the initialization would be the same every time. I chose K-medoids over K-means because part (b) showed that K-medoids typically performed better. Then I calculated the clustering score for that particular pair of classes, and after calculating the clustering score for every pair, I found the best and worst scores, as well as the class pair that they corresponded to.

Most different images (highest clustering score): 9 and 16

From this set of images, we can immediately see visible differences in the two pictures: one is lightly colored, while the other is darkly colored; the left image has the face at a slanted angle, while the right image has the face looking head on; one has no teeth displayed while the other does; one is a male while the other is a female (resulting in different facial features). This set of images had the highest clustering score, which makes sense because it is able to assign features from each image to a different cluster.

Most similar images (lowest clustering score): 5 and 17

From this set of images, we can see that unlike the first set above, these two images look quite similar. Both are men, both are looking leftwards, have their mouth open, have similar color and picture brightness, and even have roughly the same wrinkle patterns around their nose and mouth. Thus, it makes sense as to why this would lead to a low clustering score, as K-medoids is unable to effectively distinguish between features from the two image, causing it to have trouble assigning the points to a cluster.