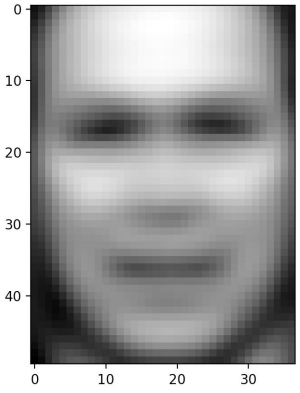
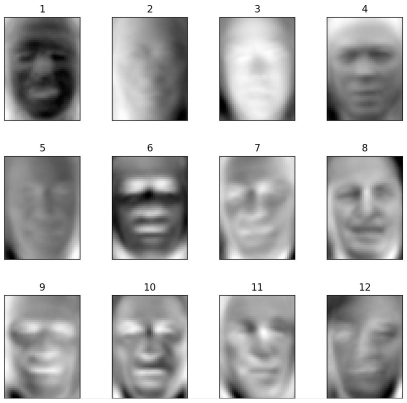
Homework 4 (CS M146) Saman Hashemipour (904903562)

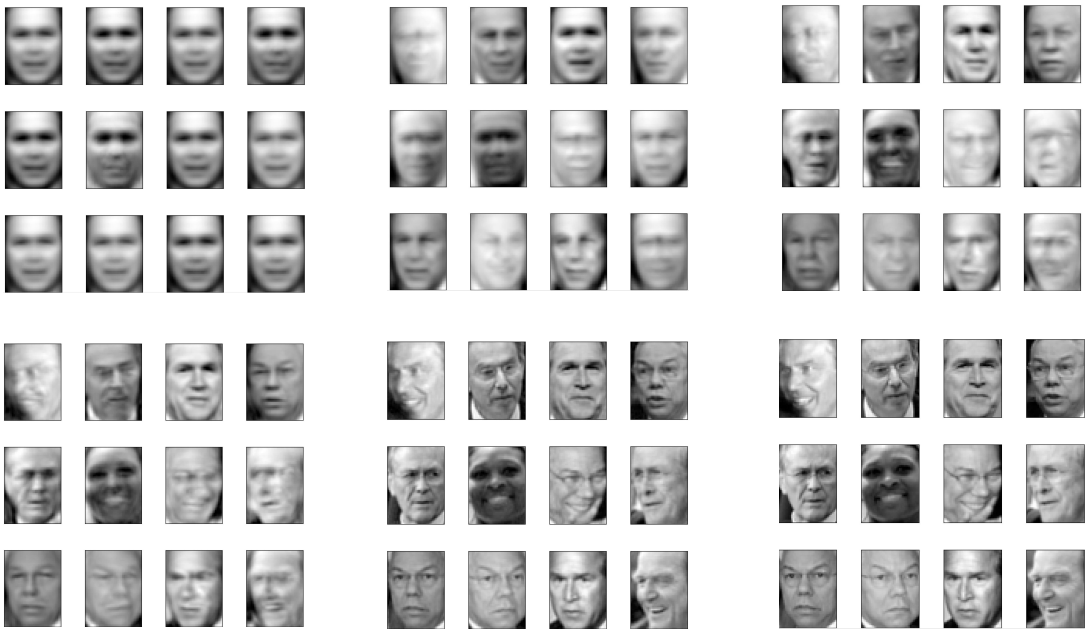




1a) I plotted 12 random faces just to get a feel for the data that I would be working with. The average face was the combination of all the image vectors in the set added together and then divided by the number of samples. The “average face” looks almost creepy with dark circles in the eyes, mouth and nostril areas. This face seems almost perfectly symmetrical.



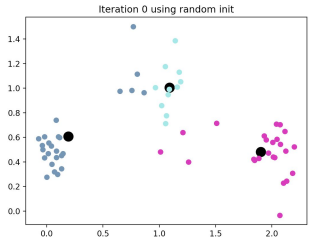

1b) The figure on the left shows the top 12 eigenfaces of our data set. These are the top 12 eigenfaces because they contain some of the most important features of each value in the data set. This means that we can use these to approximate (pretty well) the other faces in the data set. We can see that the faces are also varied in the important features that we would expect to look at in a face: the angle, the lighting, the skin tone, and facial structure. Variation is good when we want to approximate what a random face looks like.

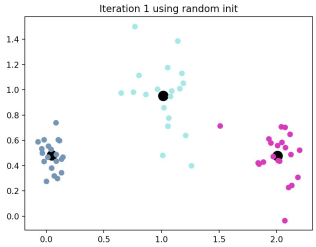
1c) As *l* increases the quality of the faces goes up. This makes sense since the more *l* increases the number of features that we have also goes up. Using less *l* features makes the images more blurry but easier to generalized given a new face. The high resolution and high *l* means that we can see the faces better but it won’t be able to generalize a new face as well. (L-Top Row: left: l=1, middle: l=10, right: l=50) (L-Bottom Row: left: l=100, middle: l=500, right: l=1288)



2a) If we don't hold the cluster value fixed and we attempt to minimize the the objective function we see that every point is a cluster center. This is due to that fact that the closest point to each point is itself. Which would make the values up to n be 0 and the sum of that would result in 0. This would result in overfitting and by doing this we essentially defeat the purpose of clustering.

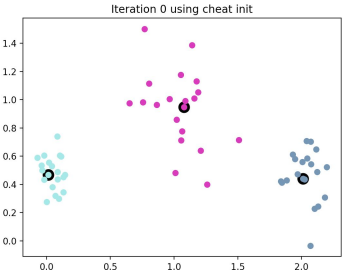
2d)
(k-mean)

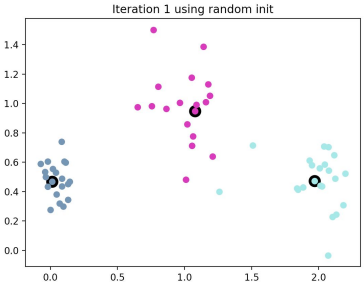


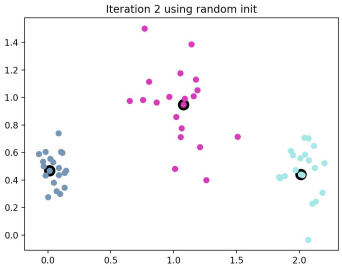





2e)
(k-medoids)

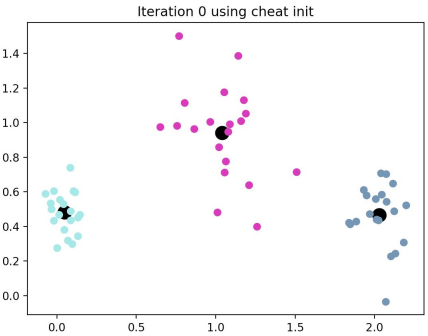


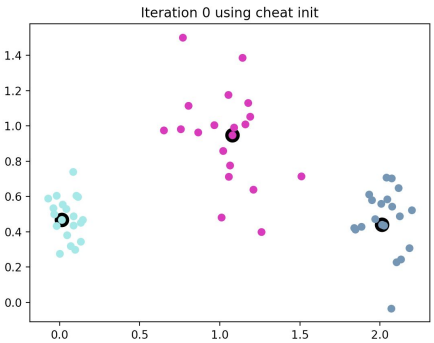




2f) Setting Init to "cheat" ensures that the values are at their optimal positions. Knowing this, it should be obvious why the two converged immediately (because they were already optimal).

(k-mean)




(k-medoid)

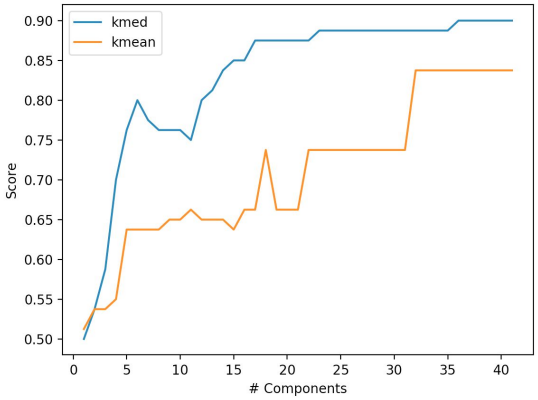


3a) We see that kmeans has a lower average than kmedoids but it has more drastic max and min values for the scores. While the runtime for kmeans is constantly faster than kemedoids for the average maximum and minimum times.

| Score | ave | max | min |
|---|---|---|---|
| Kmeans | 0.6174 | 0.775 | 0.55 |
| Kmedoids | 0.6325 | 0.725 | 0.575 |

| Runtime | ave | max | min |
|---|---|---|---|
| Kmeans | 0.2527 | 0.5226 | 0.1076 |
| Kmedoids | 0.3397 | 0.8656 | 0.1538 |

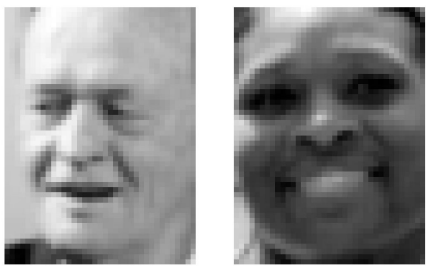
3b) In the graph below we see that kmeans has consistently lower scores than kmedoids. This makes sense given the information in part 3a. We can also see that with a low number of components we don't have a very high score making it hard for us to differentiate faces. There also seems to be a drop off for both kmed and kmean around the 20 components mark. This shows that after a certain amount of components our score won't increase that much.




3c) To determine these images I compared each of the 19 people to all the other people with kmedoid and then found the lowest and highest score for easiest and hardest to differentiate, respectively. I stored the minimum score and then then max score and I saved the number for these images in a tuple labeled max faces and min faces. I then displayed the values of the people in the tuple using the plot_representative_images function and I got the faces shown below. These are faces 9 and 16 (on the left, easiest to differentiate) and faces 4 and 5 (on the right, hardest to differentiate)

**Easiest to Differentiate:**

Person 9: Person 16:



**Hardest to Differentiate:**

Person 4: Person 5: