CM146, Winter 2020
Problem Set 4

1 Problem 1

(a) **Solution:**

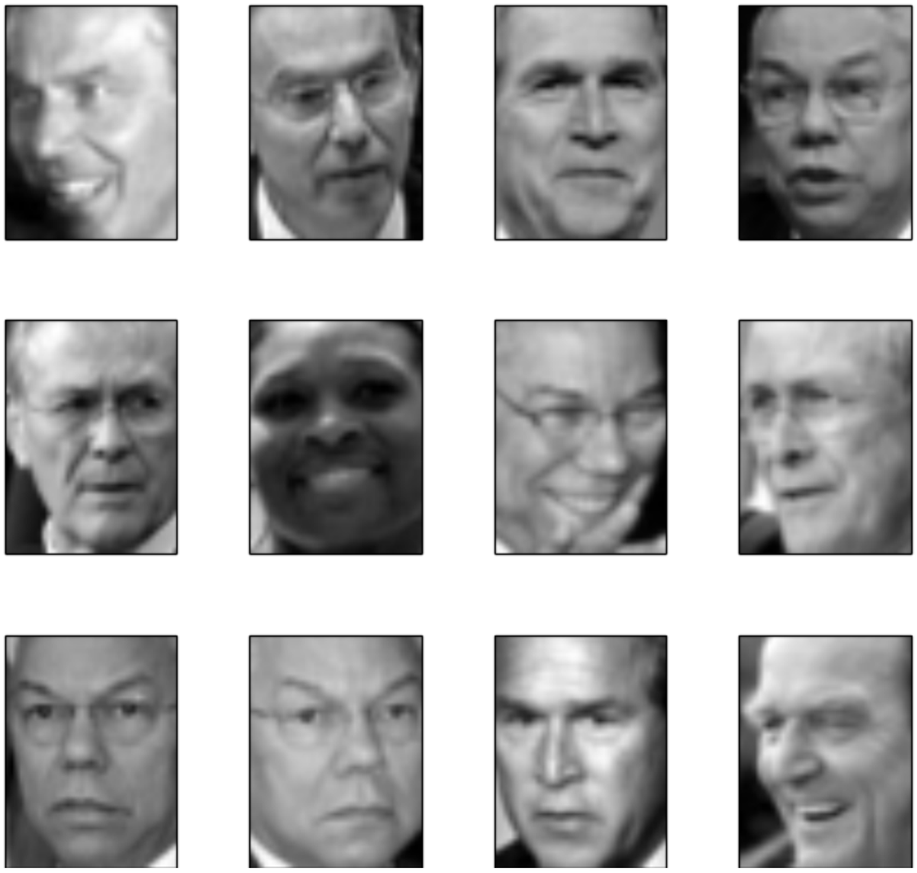


Figure 1: OG Images

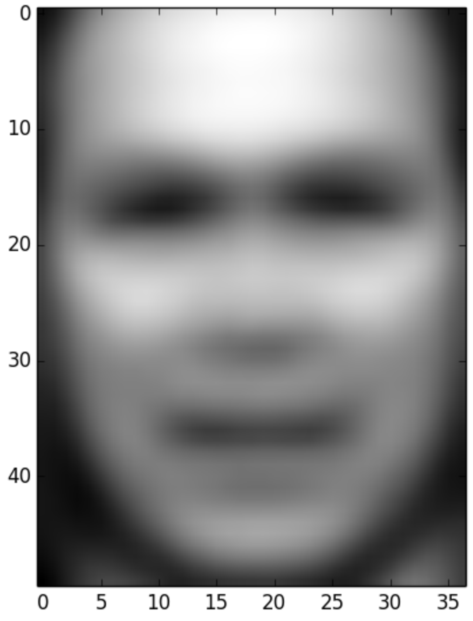


Figure 2: Average Face

In addition to being terrifying, this average face is essentially the average of all pixels in the dataset. This tells us which pixels are closer to max (ie 255) and which are closer to min (ie 0) in terms of brightness/whiteness of the pixel. Rather than giving us details about the face(s) itself, this gives us a more general idea of the image itself, and what pixels are brighter/darker, which can indicate details about the orientation of the face for example!

(b) **Solution:**

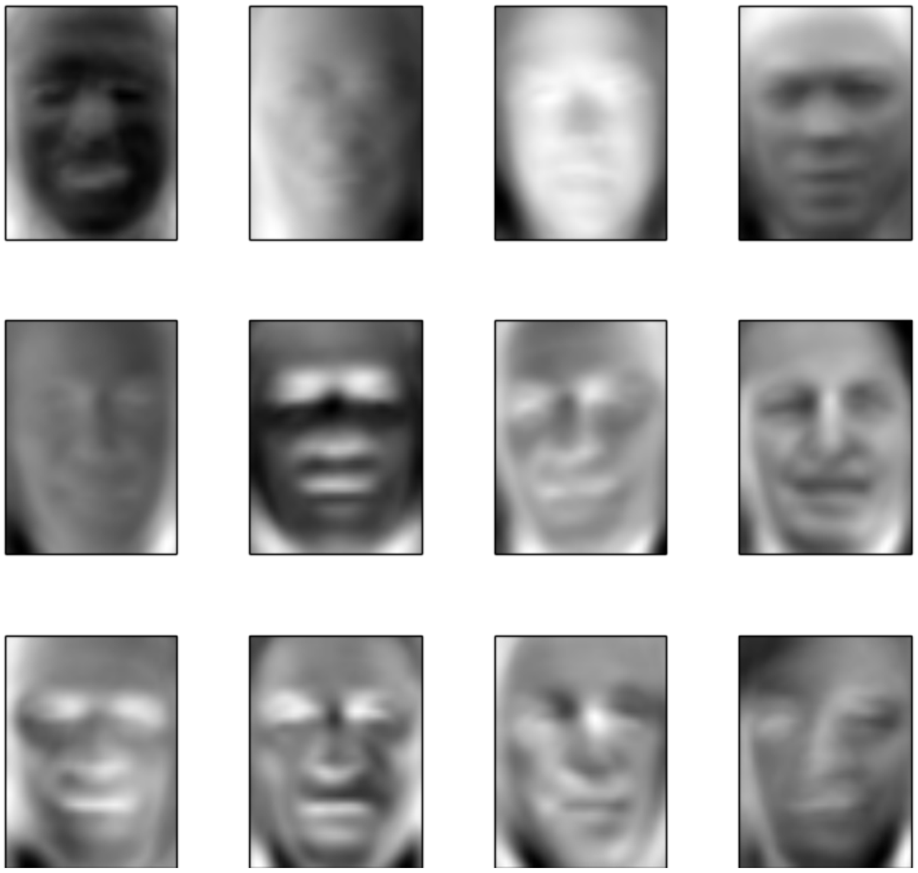


Figure 3: Principal Components / Eigenfaces

The main reason these faces were chosen is because they were likely the ones with the closest resemblance to a face, ie, the only ones who, when mapped down to lower dimensions/had the features reduced, still maintain a face-like quality. A mathematical view of this is that each image captures the most variance of the dataset (from lighting, facial structure, etc...). Intuitively, this means that they maintain the most information about these faces, and not only that, will likely have the lowest reconstruction error.

(c) **Solution:**

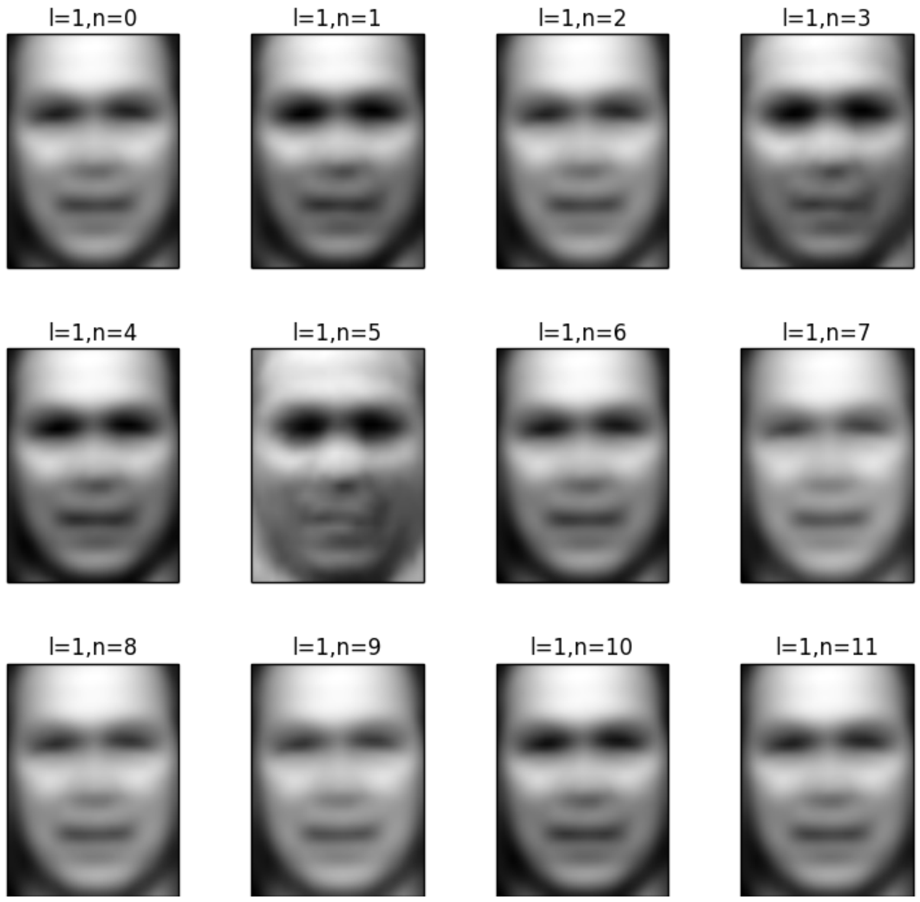


Figure 4: Using $l = 1$, ie using first principal components

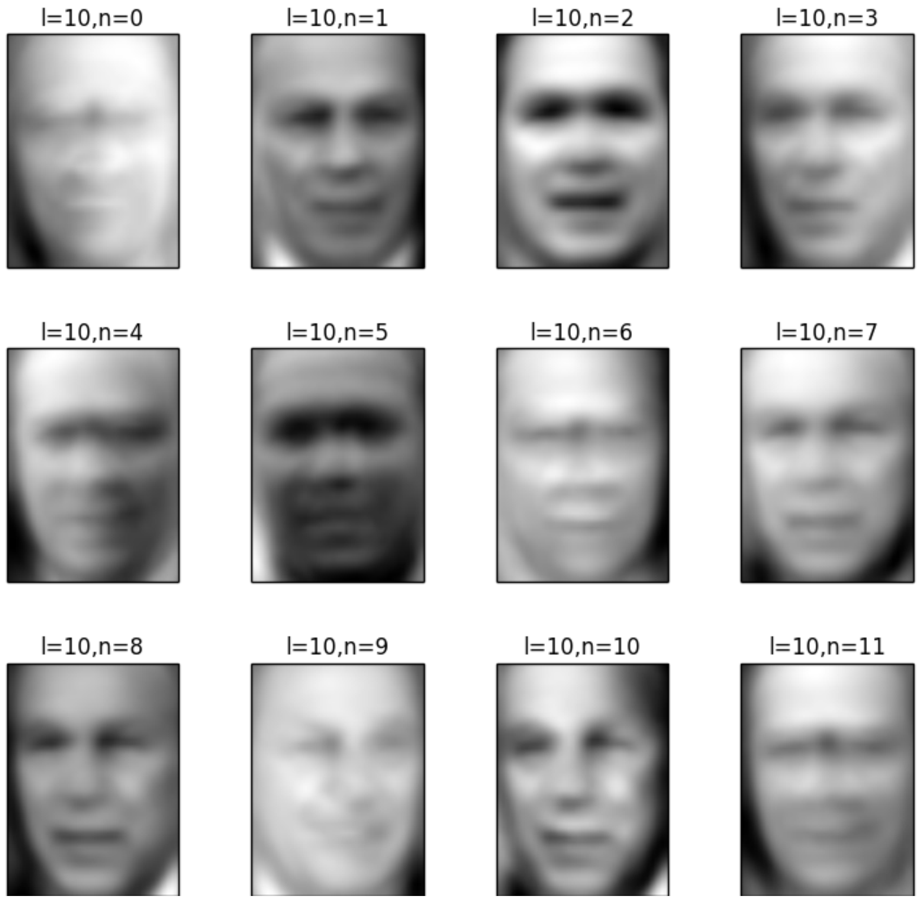


Figure 5: Using $l = 10$, ie using first 10 principal components

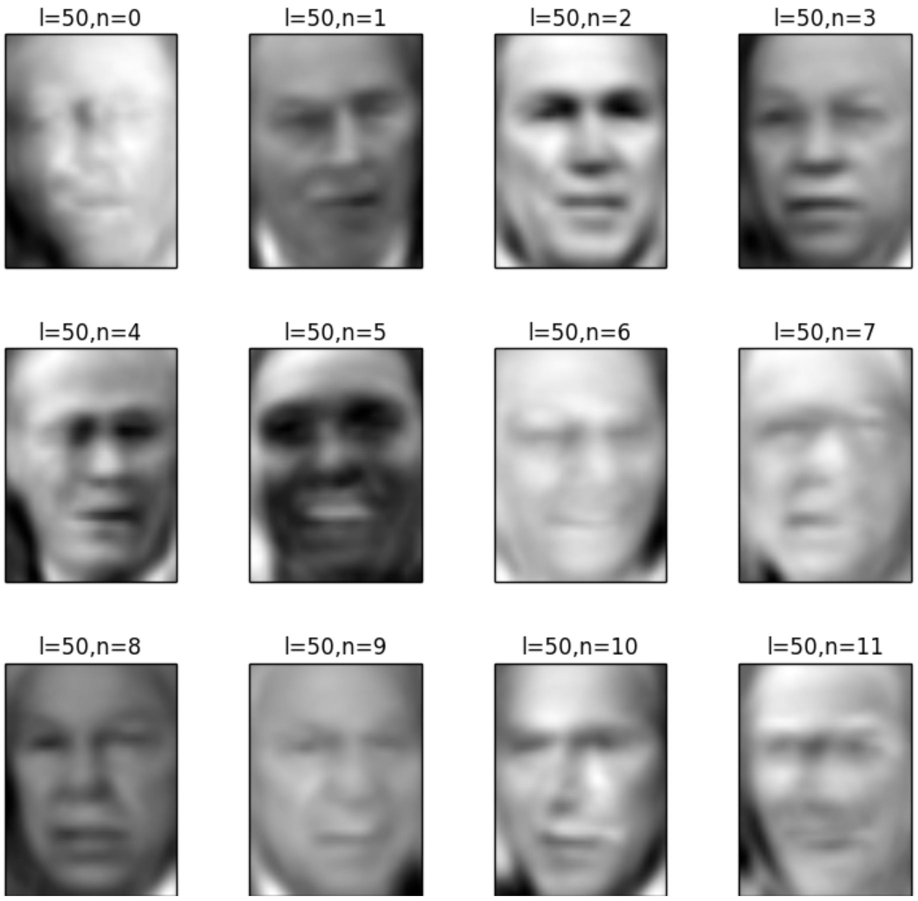


Figure 6: Using $l = 50$, ie using first 50 principal components

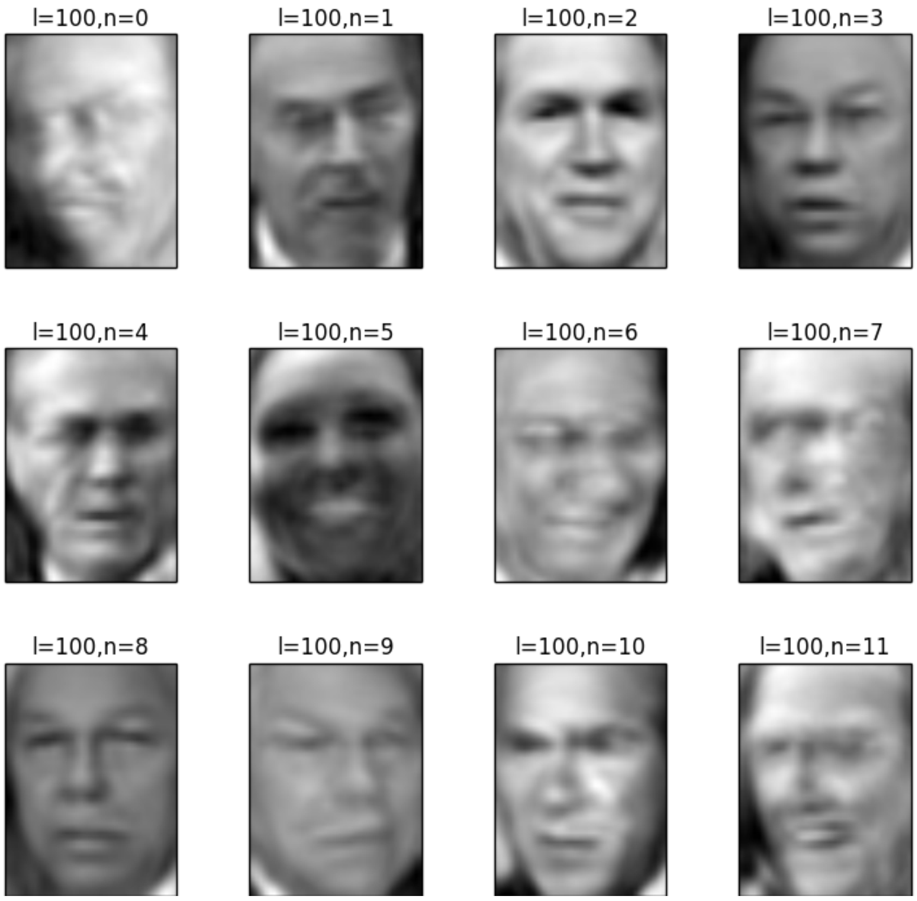


Figure 7: Using $l = 100$, ie using first 100 principal components

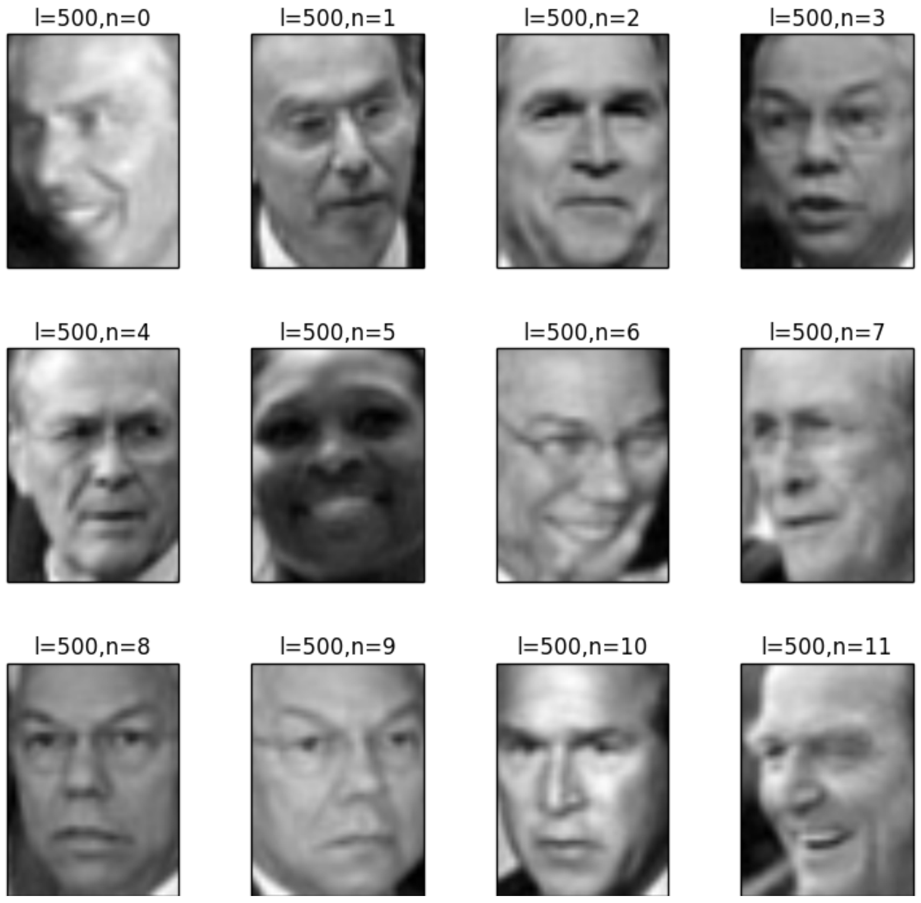


Figure 8: Using $l = 500$, ie using first 500 principal components

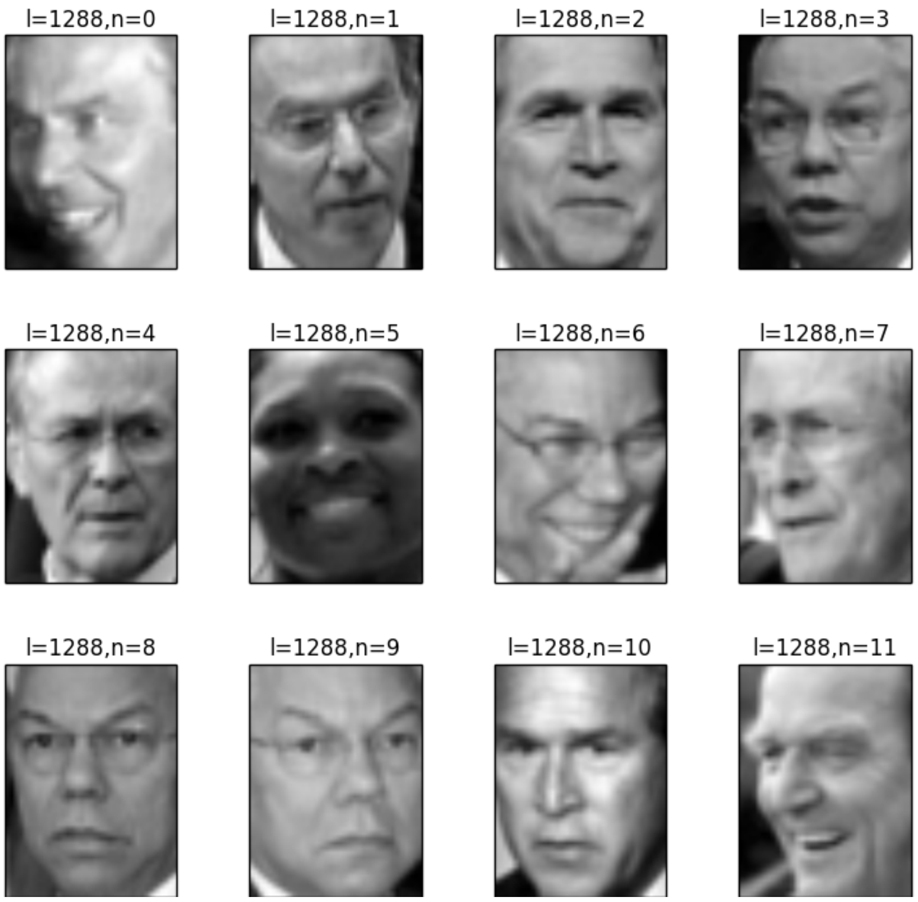


Figure 9: Using $l = 1288$, ie using first 1288 principal components

l represents the number of features we are scaling down to (I think originally it was ≈ 1850 features), so it makes sense that as we increase the l, we get more and more accurate values as we scale the number of eigenfaces we use to represent this data, eventually getting passable results at $l = 500$.

2 Problem 2

(a) **Solution:**
In that situation, where you are minimizing over c, μ, k, the cost function $J(c, \mu, k) = 0$, as we can specify each data point to be its own centroid (we have $\mu_i -> x_i$, and so $c = x_i$, making the cost 0 as described).

(b) d) **Solution:**
These series of graphs represent doing kmeans with random init:

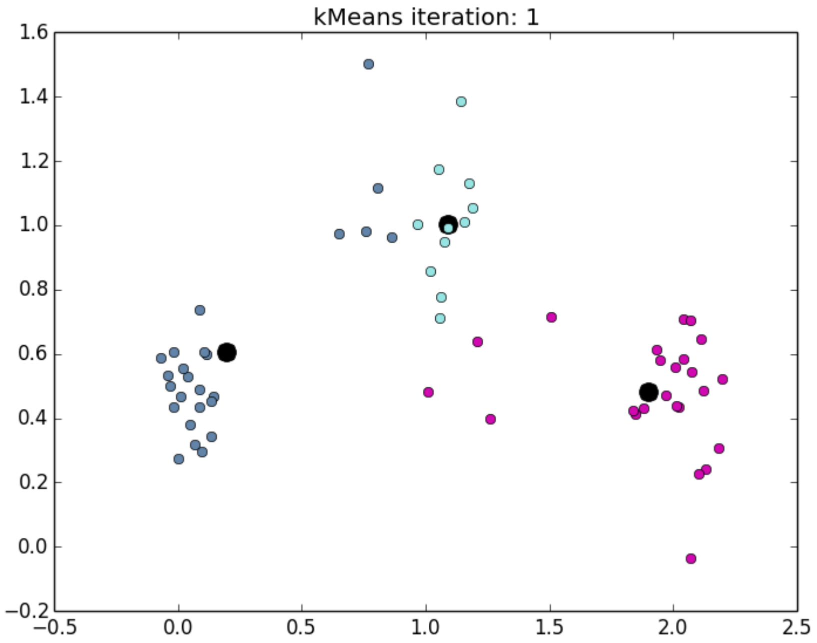


Figure 10:

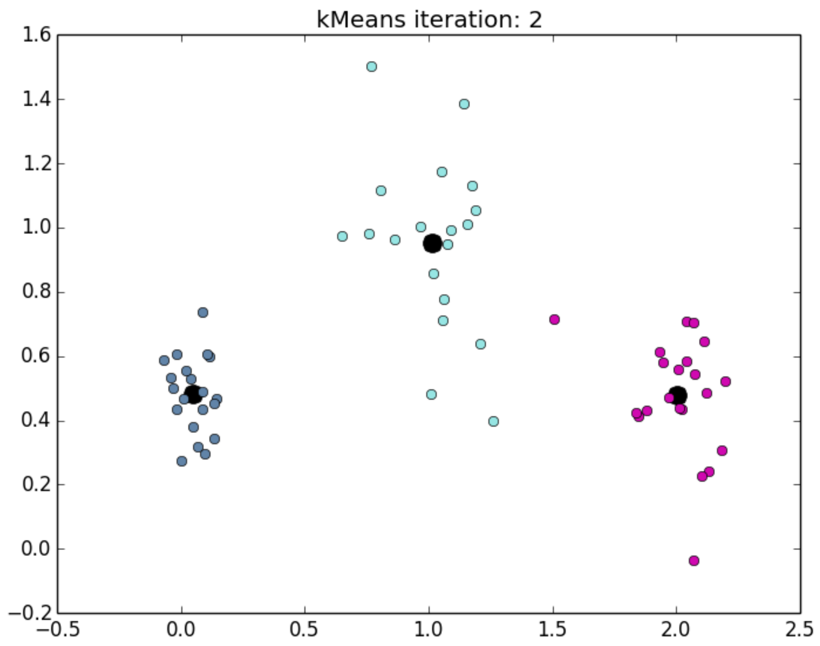

kMeans iteration: 2
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Figure 11:

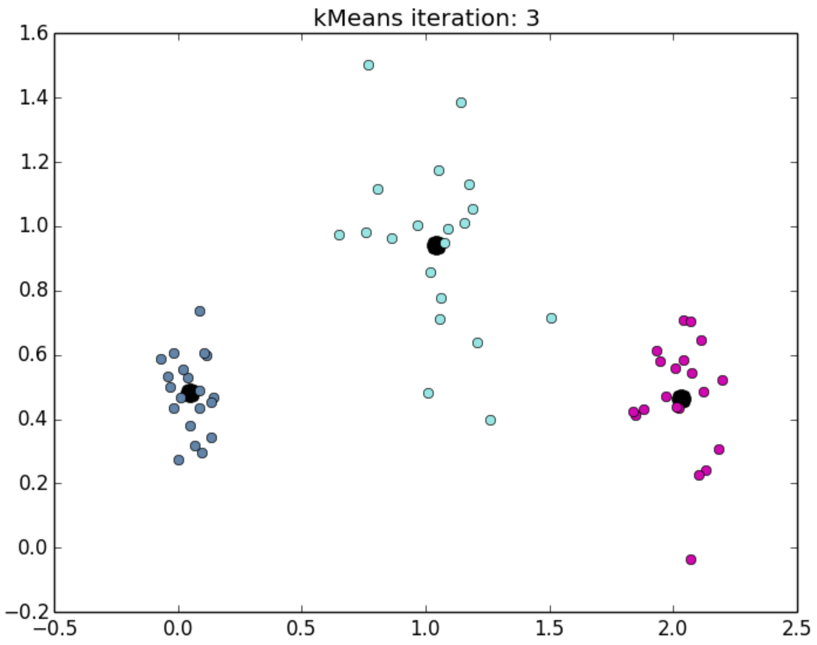

kMeans iteration: 3
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Figure 12:

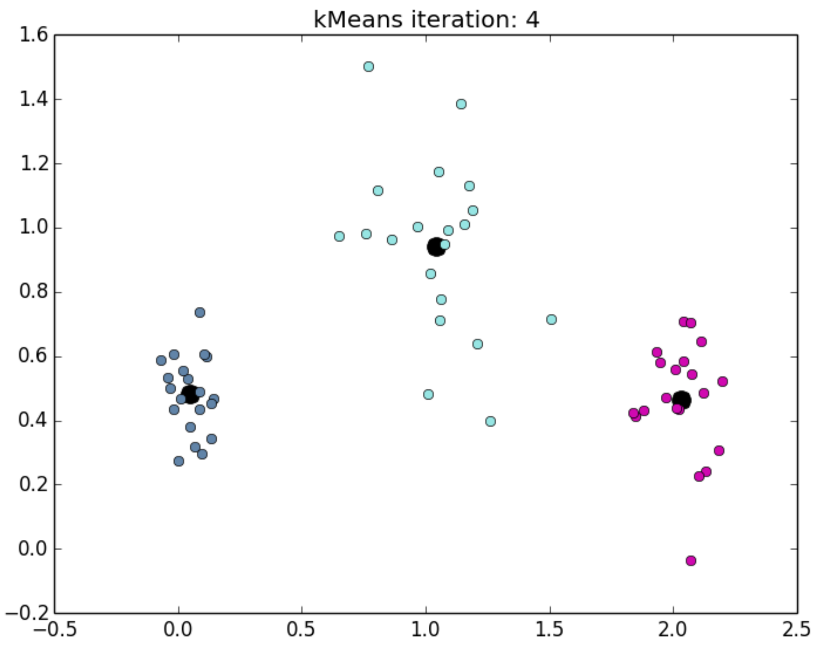
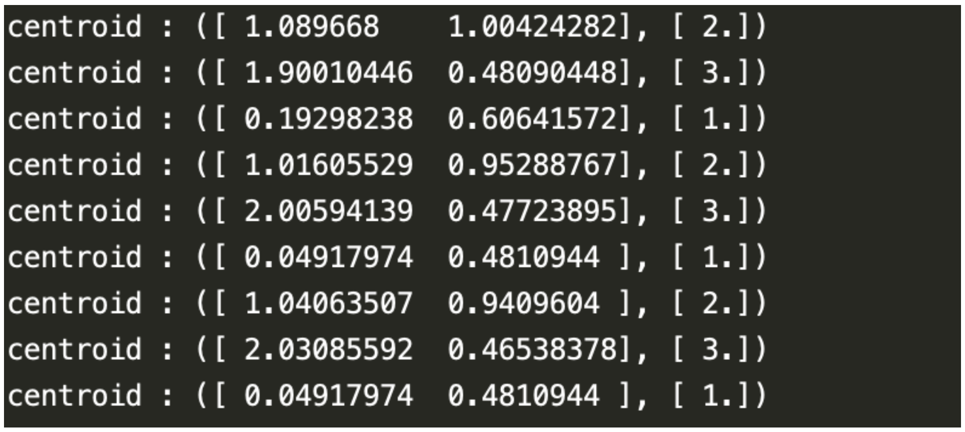


Figure 13:

And here are the centroid values:



```
centroid : ([ 1.089668    1.00424282], [ 2.])
centroid : ([ 1.90010446  0.48090448], [ 3.])
centroid : ([ 0.19298238  0.60641572], [ 1.])
centroid : ([ 1.01605529  0.95288767], [ 2.])
centroid : ([ 2.00594139  0.47723895], [ 3.])
centroid : ([ 0.04917974  0.4810944 ], [ 1.])
centroid : ([ 1.04063507  0.9409604 ], [ 2.])
centroid : ([ 2.03085592  0.46538378], [ 3.])
centroid : ([ 0.04917974  0.4810944 ], [ 1.])
```

Figure 14:

(c) e) **Solution:**

These series of graphs represent doing kmedoids with random init:

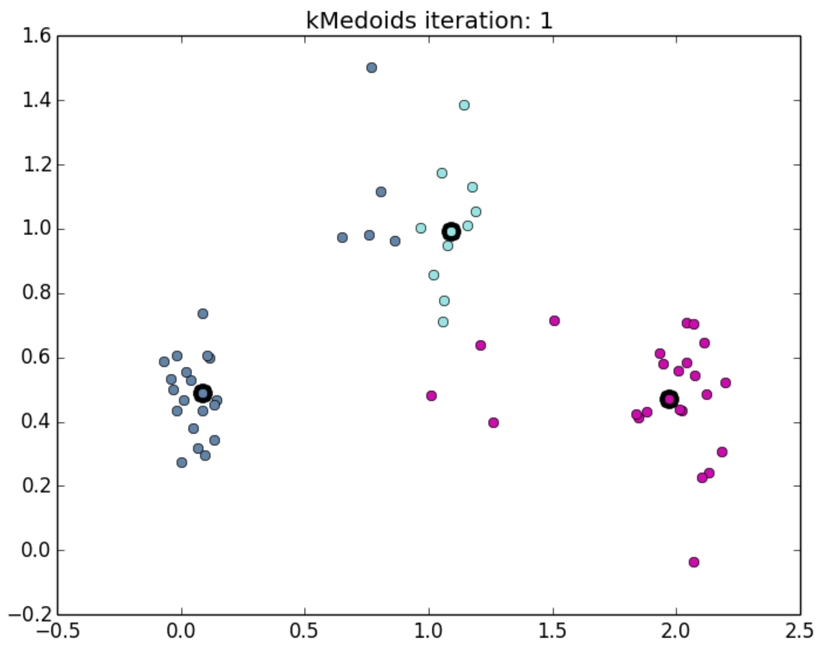


Figure 15:

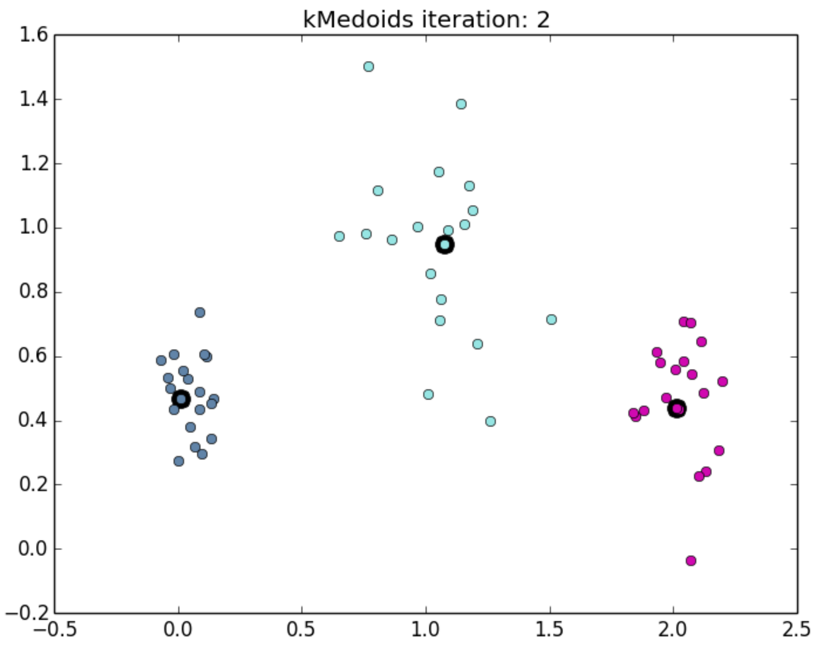

kMedoids iteration: 2
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Figure 16:

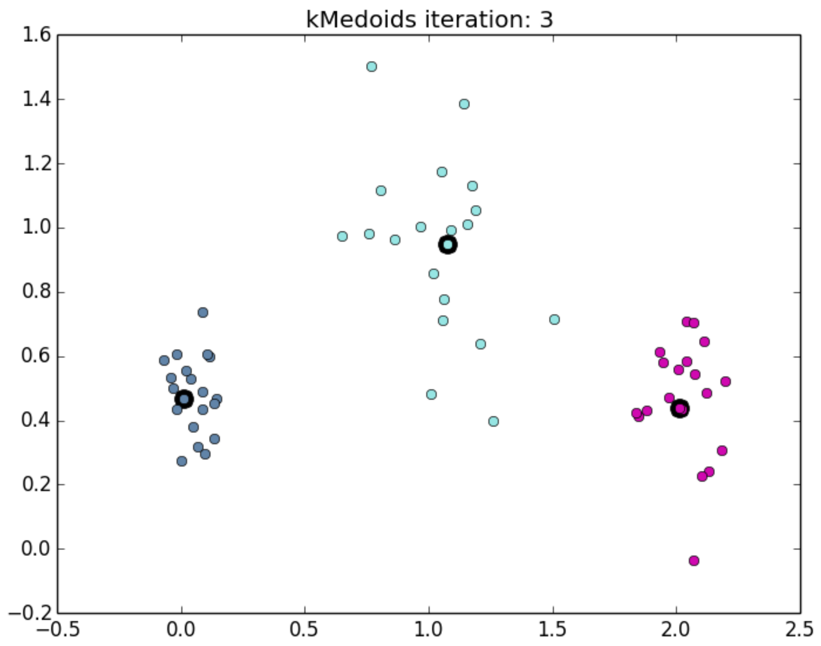

kMedoids iteration: 3
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Figure 17:

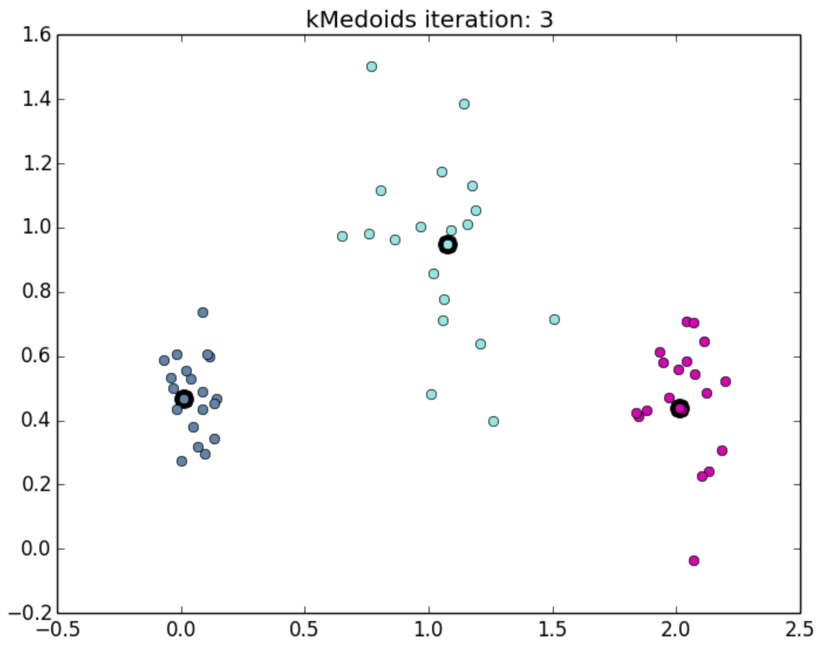


Figure 18:

(d) f) **Solution:**
Now we get to cheat! Not in the course, but in the initialization of points! For kmeans:

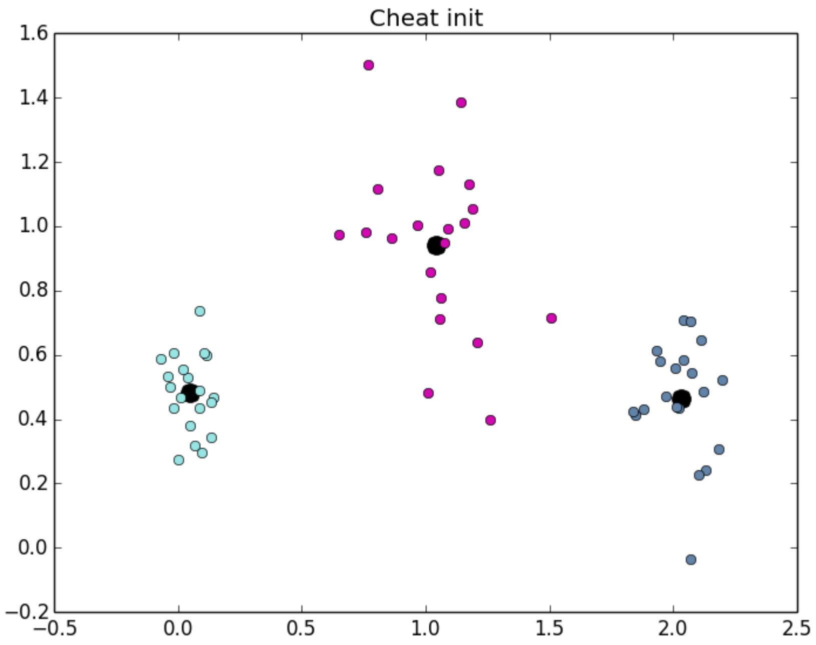


Figure 19:

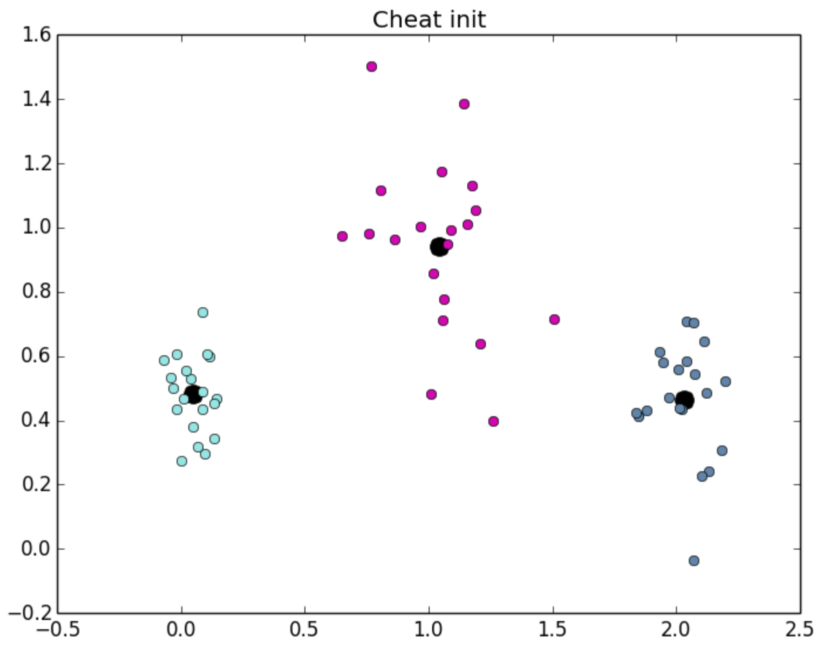


Figure 20:

As you can see, it doesn't even need more than 2 iterations, or one plot, to get to the "correct" end result!

For kmedoids:

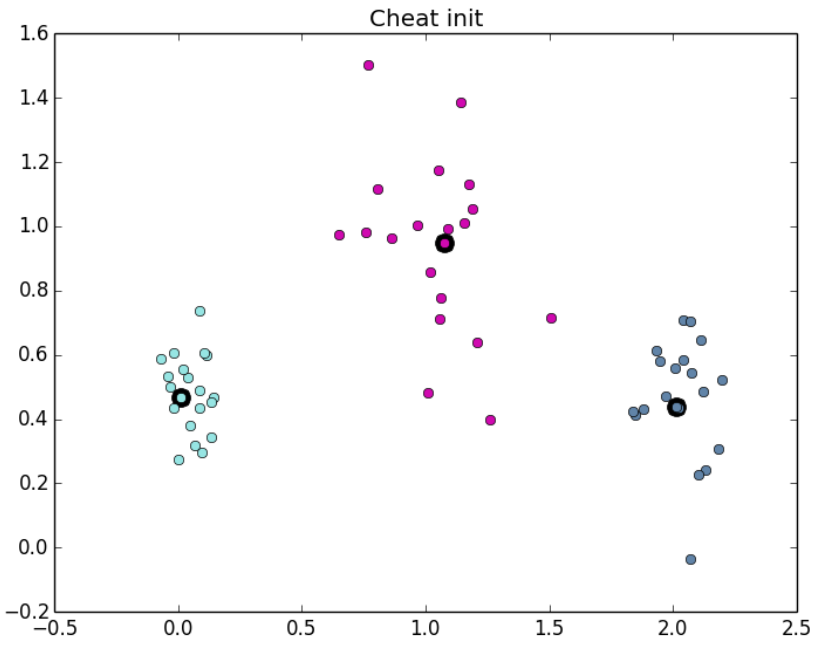


Figure 21:

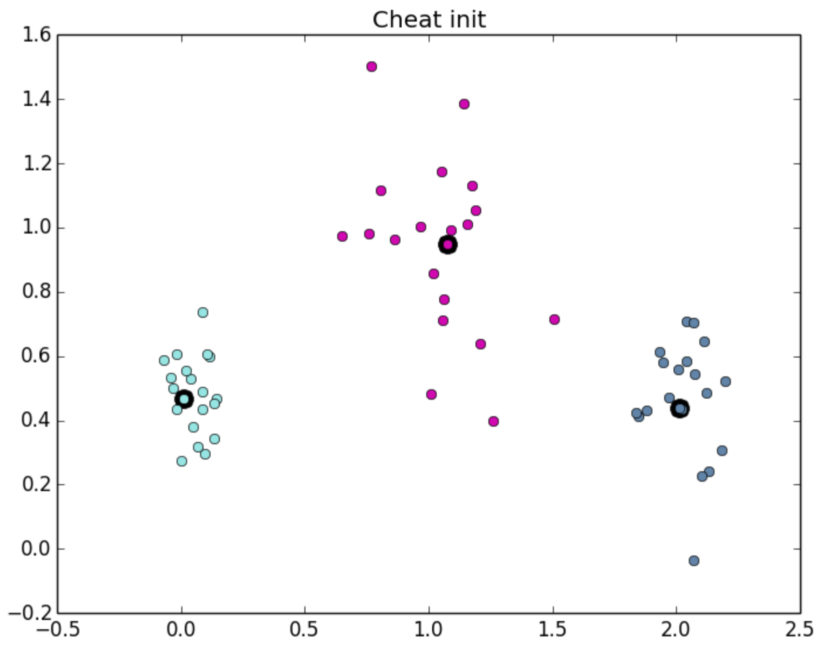
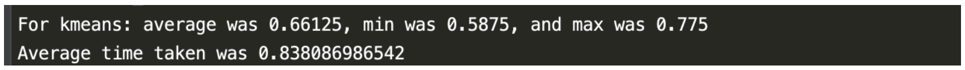


Figure 22:

Similar to the above, it takes one plot to get to where we want!

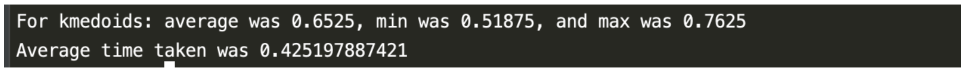
3 Problem 3

(a) **Solution:**



```
For kmeans: average was 0.66125, min was 0.5875, and max was 0.775
Average time taken was 0.838086986542
```

Figure 23:



```
For kmedoids: average was 0.6525, min was 0.51875, and max was 0.7625
Average time taken was 0.425197887421
```

Figure 24:

Generally speaking, kmedoids appears more accurate and also faster, making it the likely ideal choice.

(b) **Solution:**

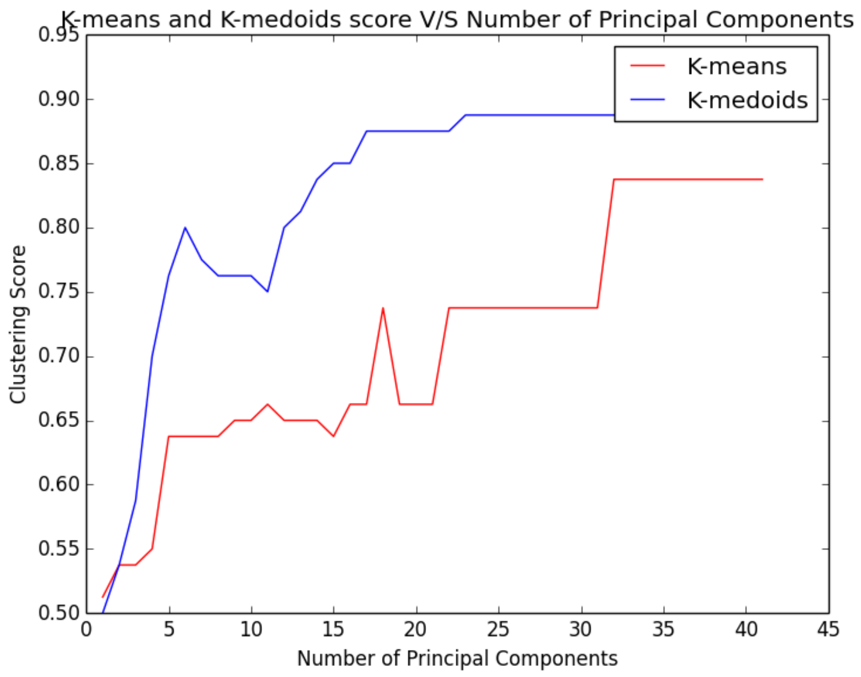


Figure 25:

kMedoids has a higher clustering score than kmeans when the number of features become less trivial. As our principal components increase, this means that the reconstructed dataset becomes closer to our original one. AS shown before with PCA, once we go past a certain number of components, our reconstruction accuracy goes up, which generalyl improves clustering. And from this graph, it seems that medoids is a more accurate method by which to do our clustering on these features.

(c) **Solution:**
With 40 pics of each person, and differentiating into 2 clusters for the people, I checked every pair of images between 2 people (40*2). After converting into points, I used both kmeans and kmedoids to compare the two.
Kmeans most similar:

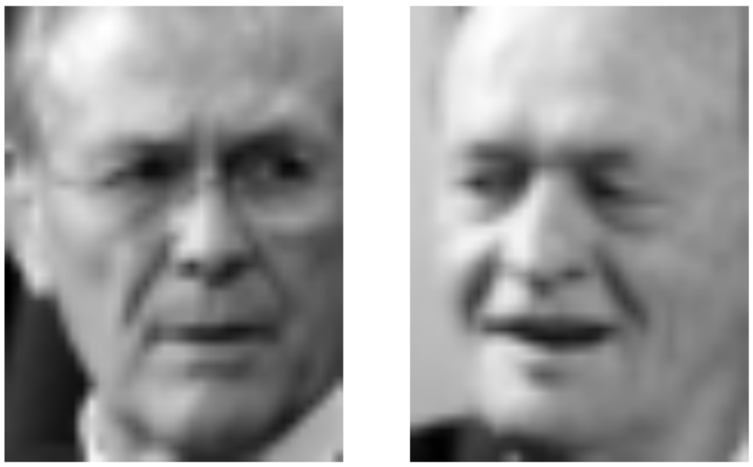


Figure 26:

Kmeans most different:

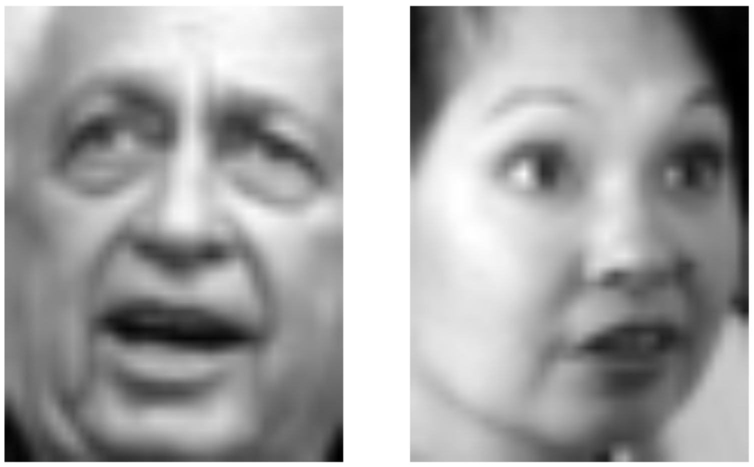


Figure 27:

VS
Kmedoids most similar:

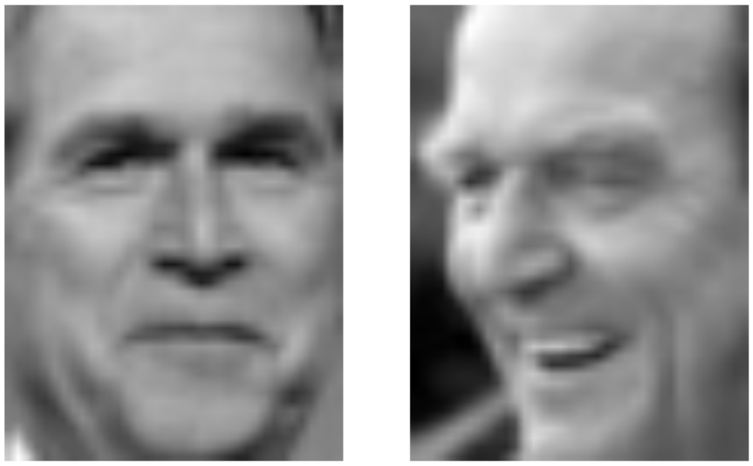


Figure 28:

Kmedoids most different:

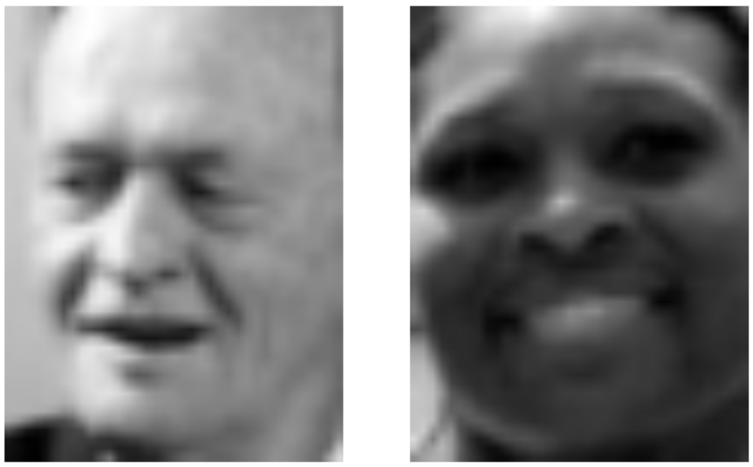


Figure 29:

Or, for the numbers:
Medoids:
(0.98750000000000004, 9, 16) :=> max
(0.51249999999999996, 4, 5) :=> min

Means:
(0.98750000000000004, 0, 6) :=> max
(0.5, 3, 9) :=> min

This tells us a few things! kmedoids makes images 4 and 5 out to be very similar, and images 9 and 16 to be very different. Whereas means notes 0 and 6 to be most different and 3 and 9 to be most similar. The contributing factors would be things like lighting, skin tone, contrast, etc.... The lower score indicates that the model could not differentiate the 2 faces well, whereas the higher score indicates that the model could very easily compare/separate the 2.