Problem 1:

Problem 2:

a. Compute the embedding vector for a school that is similar to Stanford but located in San Diego.

Embedding Vector for a school similar to Stanford but located in San Diego
= Stanford + San Diego - San Francisco
= [0.4, 0.7, -0.6, 0.4] + [0.1, 0.3, 0.8, 0.7] - [0.1, 0.2, 0.9, 0.6]
= [0.4, 0.8, -0.7, 0.5]

b. Which school is the most similar to MIT among the ones in San Francisco?

Using Cosine Similarity between MIT and Berkely; and MIT and Stanford since we know that Berkely and Stanford are in San Francisco.

$$Cosine_{Sim}(MIT,\ Berkeley) = \frac{[0.3, 0.5, -0.8, 0.4]*[0.4, 0.6, -0.7, 0.3]}{\sqrt{0.3^2+0.5^2+0.8^2+0.4^2}*\sqrt{0.4^2+0.6^2+0.7^2+0.3^2}} = 0.9822994$$

$$Cosine_{Sim}(MIT,\ Stanford) = \frac{[0.3, 0.5, -0.8, 0.4]*[0.4, 0.7, -0.6, 0.4]}{\sqrt{0.3^2+0.5^2+0.8^2+0.4^2}*\sqrt{0.4^2+0.7^2+0.6^2+0.4^2}} = 0.96$$

So Berkeley is more similar to MIT than Stanford.

Problem 3:

a. Construct a complete-link hierarchical clustering for the points in the dataset. Show your full dendrogram, and illustrate how the points in the dataset will be grouped into two clusters

According to the given dataPoints, we need first to get the distances between all six points.

Using the Euclidean Distance $D(X,Y) = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

	P1	P2	P3	P4	P5	P6
P1	**0**					
P2	**0.23573**	**0**				
P3	**0.22187**	**0.148347**	**0**			
P4	**0.3688**	**0.204217**	**0.151294**	**0**		
P5	**0342119**	**0.138856**	**0.284333**	**0.293197**	**0**	
P6	**0.234766**	**0.254012**	**0.10992**	**0.221595**	**0.392145**	**0**

Now, the shortest distance is D(P3,P6).

Now, we need to get the new distance between (P3,P6) and other points using **Complete Linkage.**

Mathematically,

$D(\{P3, P6\},\ P_i) = max(\ D(P3, P1)\ ,\ D(P6, P_i)),\ where\ i = 1, 2, 4, 5$

More Formally, using the equation posted in the slides:

$$dist(c_1, c_2) = max_{x_i \in c_1, x_j \in c_2}\, dist(x_i, x_j)$$

The New Table:

	P1	P2	{P3,P6}	P4	P5
P1	**0**				

P2	**0.23573**	**0**			
{P3,P6}	**0.234766**	**0.254012**	**0**		
P4	**0.3688**	**0.204217**	**0.221595**	**0**	
P5	**0.342119**	**0.138856**	**0.392145**	**0.293197**	**0**

We can See that the minimum Distance is between **P5 and P2.**

Now, we need to get the distance between {P5,P2} and other points. Using the complete Linkage as mentioned in its equation above, we will get this new table.

	P1	**{P2,P5}**	**{P3,P6}**	**P4**
P1	**0**			
{P2,P5}	**0.342119**	**0**		
{P3,P6}	**0.234766**	**0.392145**	**0**	
P4	**0.3688**	**0.293197**	**0.221595**	**0**

Again, the minimum distance is between {P3,P6} and P4.

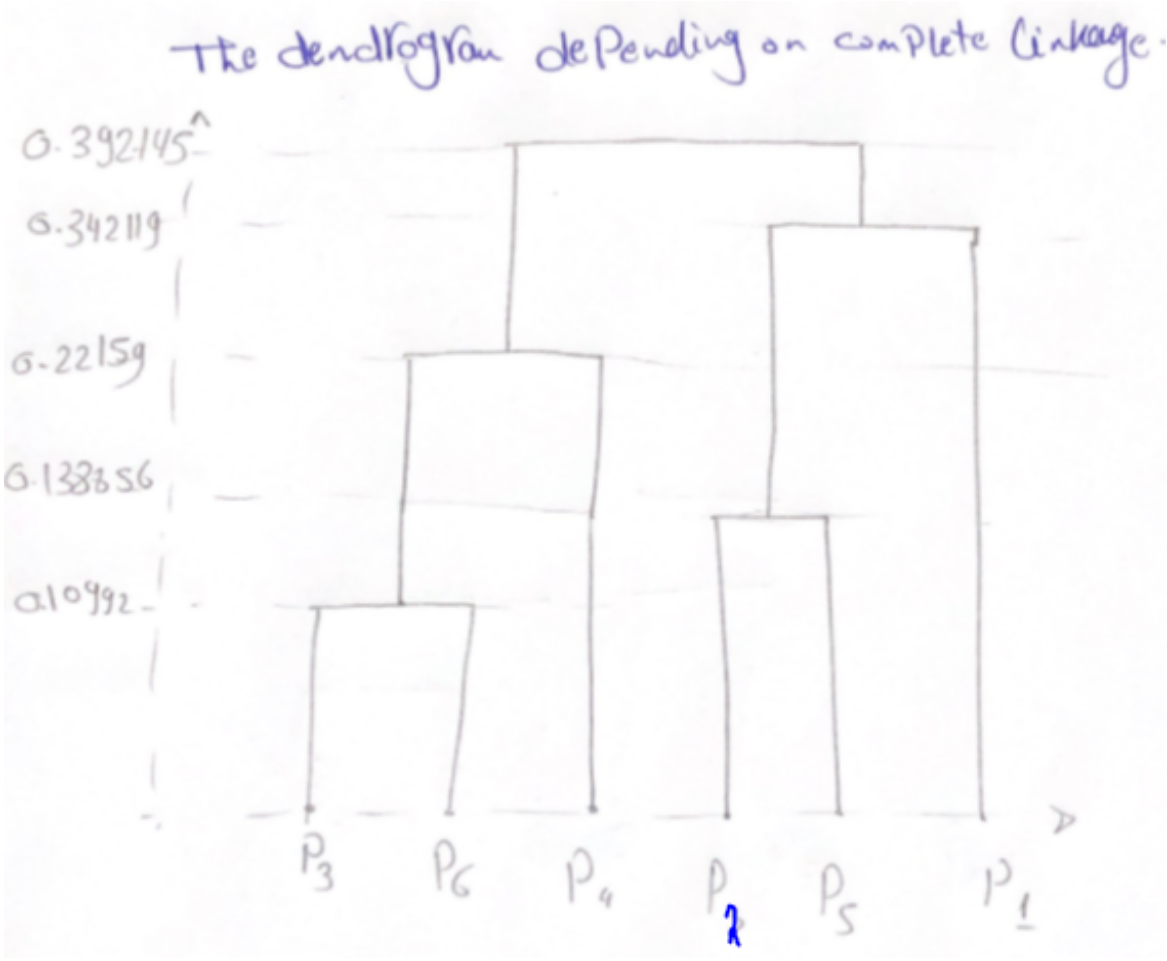
Now, we need to get the whole distance difference between {{p3,P6},P4} and other points using Complete Linkage.

	P1	**{P2,P5}**	**{{P3,P6},P4}**
P1	**0**		
{P2,P5}	**0.342119**	**0**	
{{P3,P6},P4}	**0.3688**	**0.392145**	**0**

Now, The new smallest distance is the distance between {**P1} and {P2,P5}. Finally,** we need to get the final distance between {{P1,{P2,P5}} with the rest.

	{{P1,{P2,P5}}	{{P3,P6},P4}
{{P1,{P2,P5}}	0	
{{P3,P6},P4}	0.392145	0

The Tree of the Dendrogram is :



b. Assuming we start from cluster centroids p1 & p2, trace k-means clustering with k=2 for 5 iterations and show how the points in the dataset are assigned to the two clusters and how the centroids change

Taking P1 and P2 as the Centroid for the 2 Clusters. We need to get all Distance from P1 to all other points. Similarly, we need to do with P2. I will bruising Euclidean Distance whose Formula is: $D(X,Y) = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

D1 is the distance between P1 and Other Points

D2 is the distance between P2 and Other Points

C is which Point is included to which Cluster.

First Iteration:

#	X	Y	C	D1	D2
P1	**0.4005**	**0.5306**	1	0	**0.23573**
P2	**0.2148**	**0.3854**	2	**0.23573**	0
P3	**0.3457**	**0.3156**	2	**0.22187**	**0.148347**
P4	**0.2652**	**0.1875**	2	**0.3688**	**0.204217**
P5	**0.0789**	**0.4139**	2	**0342119**	**0.138856**
P6	**0.4548**	**0.3022**	1	**0.234766**	**0.254012**

First, we need to get the new mean of the clustered points.

$\mu_1 = (\frac{0.4005+0.4548}{2}, \frac{0.5306+0.3022}{2}) = (0.42765,\ 0.4164)$

$\mu_2 = (\frac{0.2148+0.3457+0.2652+0.0789}{4}, \frac{0.3854+0.3156+0.1875+0.4139}{4}) = (0.22615, 0.3256)$

Second Iteration:

Using the μ_1 *and* μ_2 *as the new centroids,*

#	X	Y	C	D1	D2

P1	**0.4005**	**0.5306**	1	**0.117383**	**0.269115**
P2	**0.2148**	**0.3854**	2	**0.215096**	**0.0608676**
P3	**0.3457**	**0.3156**	2	**0.122909**	**0.119968**
P4	**0.2652**	**0.1875**	2	**0.280687**	**0.143515**
P5	**0.0789**	**0.4139**	2	**0.348759**	**0.171696**
P6	**0.4548**	**0.3022**	1	**0.117383**	**0.229844**

$$\mu_1 = \left(\frac{0.4005+0.0.4548}{2}, \frac{0.5306+0.3022}{2}\right) = (0.42765,\ 0.4164)$$

$$\mu_2 = \left(\frac{0.2148+0.3457+0.2652+0.0789}{4}, \frac{0.3854+0.3156+0.1875+0.4139}{4}\right) = (0.22615, 0.32561)$$

We can see that the number of clustering has not changed yet. It means that there is no need for more iterations, and so terminate. However, according to the question, I will be doing all of 5 iterations.

Third Iteration:

#	X	Y	C	D1	D2
P1	**0.4005**	**0.5306**	1	**0.11738297**	**0.269115**
P2	**0.2148**	**0.3854**	2	**0.21509561**	**0.0608676**
P3	**0.3457**	**0.3156**	2	**0.12990936**	**0.119968**
P4	**0.2652**	**0.1875**	2	**0.28068703**	**0.143515**
P5	**0.0789**	**0.4139**	2	**0.34875896**	**0.171696**
P6	**0.4548**	**0.3022**	1	**0.11738297**	**0.229844**

$$\mu_1 = \left(\frac{0.4005+0.0.4548}{2}, \frac{0.5306+0.3022}{2}\right) = (0.42765,\ 0.4164)$$

$$\mu_2 = \left(\frac{0.2148+0.3457+0.2652+0.0789}{4}, \frac{0.3854+0.3156+0.1875+0.4139}{4}\right) = (0.22615, 0.32561)$$

Still No Change!

Fourth Iteration:

#	X	Y	C	D1	D2
P1	**0.4005**	**0.5306**	1	**0.11738297**	**0.269115**
P2	**0.2148**	**0.3854**	2	**0.21509561**	**0.0608676**
P3	**0.3457**	**0.3156**	2	**0.12990936**	**0.119968**
P4	**0.2652**	**0.1875**	2	**0.28068703**	**0.143515**
P5	**0.0789**	**0.4139**	2	**0.34875896**	**0.171696**
P6	**0.4548**	**0.3022**	1	**0.11738297**	**0.229844**

$$\mu_1 = \left(\frac{0.4005+0.0.4548}{2}, \frac{0.5306+0.3022}{2}\right) = (0.42765,\ 0.4164)$$

$$\mu_2 = \left(\frac{0.2148+0.3457+0.2652+0.0789}{4}, \frac{0.3854+0.3156+0.1875+0.4139}{4}\right) = (0.22615, 0.32561)$$

Fifth iteration:

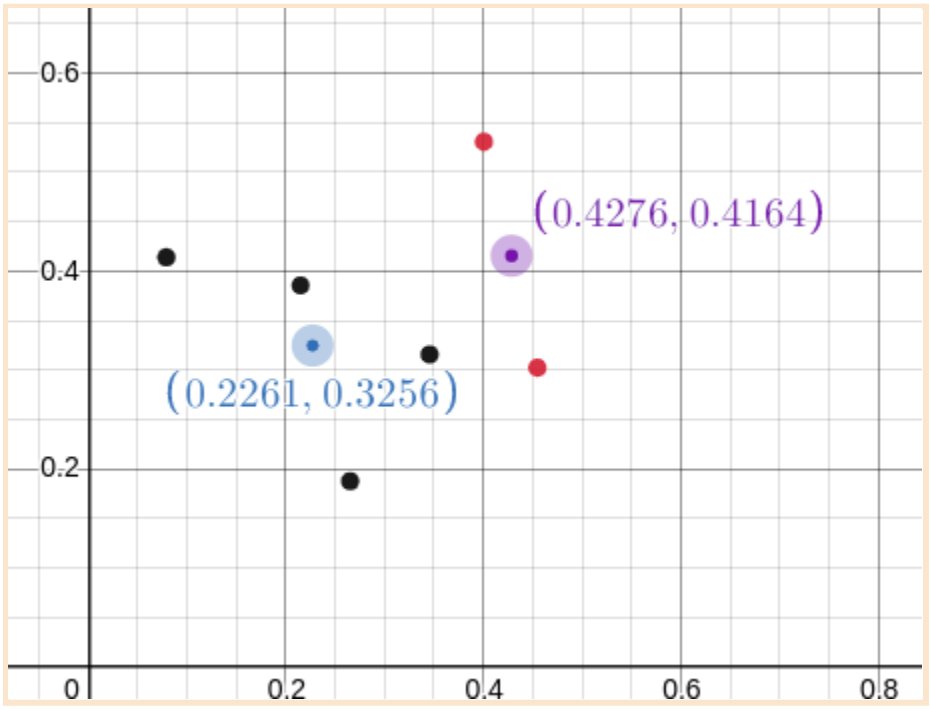
#	X	Y	C	D1	D2
P1	**0.4005**	**0.5306**	1	**0.11738297**	**0.269115**
P2	**0.2148**	**0.3854**	2	**0.21509561**	**0.0608676**
P3	**0.3457**	**0.3156**	2	**0.12990936**	**0.119968**
P4	**0.2652**	**0.1875**	2	**0.28068703**	**0.143515**
P5	**0.0789**	**0.4139**	2	**0.34875896**	**0.171696**
P6	**0.4548**	**0.3022**	1	**0.11738297**	**0.229844**

$$\mu_1 = \left(\frac{0.4005+0.0.4548}{2}, \frac{0.5306+0.3022}{2}\right) = (0.42765,\ 0.4164)$$

$$\mu_2 = \left(\frac{0.2148+0.3457+0.2652+0.0789}{4}, \frac{0.3854+0.3156+0.1875+0.4139}{4}\right) = (0.22615, 0.32561)$$

As we can See from the five iterations above, it converges in the second one with 2 clusters whose Centroid Points are $(0.42765,\ 0.4164)$ and $(0.22615, 0.32561)$ respectively.

From Desmos, The Two Clusters:



c. Assuming that the results from hierarchical clustering are the true assignments of the points (their true clusters). Compute the precision and recall for the k-means assignment with respect to the assumed true classes of the points.

The True Results from the hierarchical clustering:

Class A = [P3, P6, P4]
Class B = [P2, P5, P1]

The Predicted Results from the k-mean clustering:

Class A = [P1, P6]
Class B = [P2, P3, P4, P5]

First, I will do the normal one. Then, I found from the link a different solution to avoid overestimation or underestimation to the clusters.

Normal **Solution:**

Assume that cluster 1 is **negative** and cluster 2 is **positive**.

	Predicted Class 2 (Positive)	Predicted Class 1 (Negative)
True Class 2 (+ve)	2 {P5,P2}	1 {P1}
True Class 1 (-ve)	2 {P3,P4}	1 {P6}

Recall = $\frac{TP}{TP+FN} = \frac{2}{2+2} = 0.5$

Precision= $\frac{TP}{TP+FN} = \frac{2}{2+1} = 0.75$

Assume that cluster 2 is **Positive** and cluster 2 is **Negative**.

	Predicted Class 1 (Positive)	Predicted Class 2 (Negative)
True Class 1(+ve)	1 {P6}	2 {P3,P4}
True Class 2 (-ve)	1 {P1}	2 {P5, P2}

Recall = $\frac{TP}{TP+FN} = \frac{1}{2} = 0.5$

Precision= $\frac{TP}{TP+FN} = \frac{1}{2+1} = 0.33$

Following the link I used to get the Precision and Recall for each assignment: https://towardsdatascience.com/evaluating-clustering-results-f13552ee7603

The reason for not using normal criteria is that we might overestimate or underestimate the cluster. We do not know how perfect it is. In this sense, we will be using different thing to calculate the precision and recall.

First, we know that the number of classes = the number of Clusters with assumption that the hierarchical clustering is the true classes.

To get the **precision**, "we sum the maximum number of objects for each cluster and divide it by the total number of clustered objects".

$$Precision = \frac{\sum_k max_s\{a_{ks}\}}{\sum_k \sum_s a_{ks}}$$

To get recall, "for each gold standard class, we obtain the cluster with the maximum number of objects assigned. Then, we sum the maximum number of objects for each gold standard class and divide it by the total number of clustered objects and unclustered objects". This is the **Recall**

$$Recall = \frac{\sum_s max_k\{a_{ks}\}}{(\sum_k \sum_s a_{ks} + U)}$$

	True Cluster 1	True Cluster 2
Predicted Cluster 1	1 {P6}	1 {P1}
Predicted Cluster2	2 {P3, P4}	2 {P2,P5}

Don't Forget the **U** is the Unclustered Object = 0.

Precision = $\frac{1+2}{1+1+2+2} = 0.5$

Recall $= \frac{4}{6} = 0.667$

d. Using affinity propagation, go through 5 iterations of computing the availability and responsibility matrices for this data set. Discuss the impact of the different choices of the self-similarity values on the projected clustering output (hint: you can write a small script to compute the matrices and try different self similarity values for the discussion - Make sure to illustrate at least one iteration by hand)

Keep in mind that I followed this link for understanding more how to write the code and avoid the number of oscillations during optimization (convergence): https://nl.hideproxy.me/go.php?u=Gtem05IjgLnIAOW8P8VVTrxHWndYW4F7zCO4E%2FmesT2r7HBmxc9Ft0VbQivrYxF%2Bok4pZgiqgebiQykTOlBa7zGHlh9Pr3StTLYzhS4Ot20hJuE41CShc2T0fpOBEEj0Bfb2Q5ot&b=5

First, the Similarity distance is calculated as

$$S(i,j) = -||xi-xj||^2$$

	P1	P2	P3	P4	P5	P6
P1	**0**	**-0.055568**	**-0.049228**	**-0.136024**	**-0.117045**	**-0.055115**
P2	**-0.055568**	**0**	**-0.022007**	**-0.041705**	**-0.019281**	**-0.064522**
P3	**-0.049228**	**-0.022007**	**0**	**-0.022890**	**-0.080845**	**-0.012082**
P4	**-0.136024**	**-0.041705**	**-0.022890**	**0**	**-0.085965**	**-0.049104**
P5	**-0.117045**	**-0.019281**	**-0.080845**	**-0.085965**	**0**	**-0.153778**
P6	**-0.055115**	**-0.064522**	**-0.012082**	**-0.049104**	**-0.153778**	**0**

Now, for the sake of time, I will do one iteration by hand. Then, I Implemented a code to give us the Responsibility and Availability for each iteration.

- We need to take into **consideration** the **Values** of diagonals:
  - Initializing it to a value close to the minimum similarity produces fewer classes
  - Initializing it to a value close to the maximum similarity produces many classes

In this sense, I start by giving the values with Minimum values and will test our code if we increase the values of diagonals compared to the number of clusters.

The min. Values for the diagonal is preferred to be the minimum value in the whole Table, which is

-0.153778

	P1	P2	P3	P4	P5	P6
P1	**-0.153778**	**-0.055568**	**-0.049228**	**-0.136024**	**-0.117045**	**-0.055115**
P2	**-0.055568**	**-0.153778**	**-0.022007**	**-0.041705**	**-0.019281**	**-0.064522**
P3	**-0.049228**	**-0.022007**	**-0.153778**	**-0.022890**	**-0.080845**	**-0.012082**
P4	**-0.136024**	**-0.041705**	**-0.022890**	**-0.153778**	**-0.085965**	**-0.049104**
P5	**-0.117045**	**-0.019281**	**-0.080845**	**-0.085965**	**-0.153778**	**-0.153778**
P6	**-0.055115**	**-0.064522**	**-0.012082**	**-0.049104**	**-0.153778**	**-0.153778**

First Iteration:

Responsibility Calculation:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} \{a(i,k') + s(i,k')\}$$

$$s(i,k) = -\left\| x_i - x_k \right\|^2$$ similarity between i and k

Intuitively, it is the Similarity (i,k) - The max(row without i = k)

	P1	P2	P3	P4	P5	P6

P1	**-0.104550**	**-0.006339**	**0.005887**	**-0.086796**	**-0.067817**	**-0.005887**
P2	**-0.036286**	**-0.134497**	**-0.002726**	**-0.022424**	**0.002726**	**-0.045241**
P3	**-0.037146**	**-0.009924**	**-0.141695**	**-0.010807**	**-0.068763**	**0.009924**
P4	**-0.113134**	**-0.018815**	**0.018815**	**-0.130888**	**-0.063075**	**-0.026214**
P5	**-0.097764**	**0.061564**	**-0.061564**	**-0.066684**	**-0.063075**	**-0.134497**
P6	**-0.043033**	**-0.052440**	**0.037022**	**-0.037022**	**-0.063075**	**-0.141695**

Then, we need to get the Availability in all points:

Mathematically, I used the following equations

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \neq i,k} \max\{0, r(i',k)\}\right\}$$

$$a(k,k) \leftarrow \sum_{i' \neq k} \max\{0, r(i',k)\}$$

For simplicity,

$a\,(P_1,\ P_1)\ =\ the\ Max\ Column\ of\ P1\ without\ Point\ (P1, P1)$.

More Formally A(P1,P1) =

$$\sum_{i=2}^{6} max(P1, P_i)\ =\ Sum(max([0\ 0\ 0\ 0\ 0], [-\ 0.10455\ -\ 0.10455\ -\ 0.10455\ -\ 0.10455\ -\ 0.1045$$

$$=\ 0.$$

Now with a(P1,P2)

I will get the maximum between Each Row with respect to the P2 column value except the Row which has P1 and 0.

$$P(P1, P2)\ =\ min(0,\ r(P1, P2)\ +\ sum(max([0\ 0\ 0\ 0\ 0],$$

$-\ 0.134497\ -\ 0.009924\ -\ 0.018815\ 0.061564\ -\ 0.052440))$
= min(0, -0.134497 +0.06156) = -0.10455
Similarly, I have done as below.

	P1	P2	P3	P4	P5	P6
P1	**0**	**-0.072933**	**-0.085859**	**-0.130888**	**-0.131771**	**-0.131771**
P2	**-0.10455**	**0.061564**	**-0.079972**	**-0.130888**	**-0.134497**	**-0.131771**
P3	**-0.10455**	**-0.072933**	**0.061724**	**-0.130888**	**-0.131771**	**-0.141695**
P4	**-0.10455**	**-0.072933**	**-0.098786**	**0**	**-0.131771**	**-0.131771**
P5	**-0.10455**	**-0.134497**	**-0.079972**	**-0.130888**	**0.002726**	**-0.131771**
P6	**-0.10455**	**-0.072933**	**-0.116994**	**-0.130888**	**-0.131771**	**0.009924**

So, Cretiran Matrix:
Using the function below,

$$c(i,k) \leftarrow r(i,k) + a(i,k).$$

	0	1	2	3	4	5
0	-0.104550	-0.079272	-0.079972	-0.217684	-0.199589	-0.137658
1	-0.140836	-0.072933	-0.082698	-0.153312	-0.131771	-0.177012
2	-0.141696	-0.082857	-0.079972	-0.141696	-0.200534	-0.131771
3	-0.217684	-0.091748	-0.079972	-0.130888	-0.194846	-0.157986
4	-0.202314	-0.072933	-0.141536	-0.197572	-0.131771	-0.266268
5	-0.147583	-0.125373	-0.079972	-0.167910	-0.273466	-0.131771

After some research, I have found there is a damping factor needed to be calculated to avoid any problem with oscillations during optimization.

Updating the scores: how clusters are formed

Now that we know about the formulae for responsibility and availability, let's take a look at how scores are updated after every iteration (Scikit-learn, n.d.):

$$r_{t+1}(i,k) = \lambda \cdot r_t(i,k) + (1-\lambda) \cdot r_{t+1}(i,k)$$

$$a_{t+1}(i,k) = \lambda \cdot a_t(i,k) + (1-\lambda) \cdot a_{t+1}(i,k)$$

Now, Let's Get the responsibility, Availability, and Criteria for each Iteration (You can see the code for more detail).

Iteration 2:

Responsibility:

:

	0	1	2	3	4	5
0	-0.088695	0.010832	0.020564	-0.070941	-0.051963	0.009967
1	-0.021700	-0.117957	0.011861	-0.007837	0.016767	-0.030655
2	-0.021151	0.006070	-0.125124	0.005187	-0.052768	0.023934
3	-0.094784	0.000943	0.033401	-0.112539	-0.044725	-0.007865
4	-0.071410	0.075605	-0.035210	-0.040329	-0.107598	-0.108142
5	-0.019634	-0.029041	0.054692	-0.013623	-0.118297	-0.118297

Availability:

	0	1	2	3	4	5
0	0.037488	-0.058346	-0.068687	-0.104711	-0.105417	-0.105417
1	-0.083640	0.134424	-0.063978	-0.104711	-0.107598	-0.105417
2	-0.083640	-0.058346	0.136740	-0.104711	-0.105417	-0.113357
3	-0.083640	-0.058346	-0.079029	0.052947	-0.105417	-0.105417
4	-0.083640	-0.107598	-0.063978	-0.104711	0.027035	-0.105417
5	-0.083640	-0.058346	-0.093595	-0.104711	-0.105417	0.057256

Criterion:

	0	1	2	3	4	5
0	-0.051207	-0.047514	-0.048123	-0.175652	-0.157380	-0.095449
1	-0.105340	0.016467	-0.052117	-0.112547	-0.090830	-0.136072
2	-0.104791	-0.052276	0.011616	-0.099524	-0.158185	-0.089423
3	-0.178424	-0.057404	-0.045628	-0.059592	-0.150142	-0.113282
4	-0.155050	-0.031992	-0.099187	-0.145040	-0.080563	-0.213559
5	-0.103274	-0.087387	-0.038903	-0.118334	-0.223713	-0.061041

Iteration 3:

Responsibility:

	0	1	2	3	4	5
0	-0.078929	0.020810	0.029388	-0.061175	-0.042197	0.019734
1	-0.024603	-0.107925	0.008958	-0.010740	0.013428	-0.033557
2	-0.023359	0.003862	-0.114784	0.002979	-0.054976	0.020138
3	-0.083022	0.012579	0.042153	-0.100776	-0.032963	0.003897
4	-0.055189	0.081977	-0.018988	-0.024108	-0.091458	-0.091921
5	-0.007426	-0.016833	0.060642	-0.001415	-0.106088	-0.104258

Availability:

	0	1	2	3	4	5
0	0.040211	-0.046677	-0.054950	-0.083768	-0.086882	-0.084334
1	-0.066912	0.159402	-0.051182	-0.083768	-0.088641	-0.084334
2	-0.066912	-0.046677	0.163829	-0.083768	-0.086882	-0.090685
3	-0.066912	-0.046677	-0.063224	0.059997	-0.089699	-0.084334
4	-0.066912	-0.086078	-0.051182	-0.085030	0.024459	-0.084334
5	-0.072945	-0.046677	-0.074876	-0.086358	-0.086882	0.068745

Criterion:

	0	1	2	3	4	5
0	-0.038718	-0.025867	-0.025561	-0.144943	-0.129078	-0.064600
1	-0.091515	0.051477	-0.042224	-0.094508	-0.075212	-0.117891
2	-0.090271	-0.042815	0.049045	-0.080789	-0.141858	-0.070547
3	-0.149934	-0.034098	-0.021070	-0.040780	-0.122662	-0.080436
4	-0.122101	-0.004101	-0.070170	-0.109138	-0.066998	-0.176254
5	-0.080371	-0.063510	-0.014235	-0.087773	-0.192970	-0.035513

Iteration 4:

Responsibility:

	0	1	2	3	4	5
0	-0.073450	0.026370	0.034114	-0.055696	-0.036718	0.025213
1	-0.031920	-0.102457	0.001640	-0.018057	0.005762	-0.040875
2	-0.030543	-0.003322	-0.108846	-0.004205	-0.062160	0.011684
3	-0.076400	0.018945	0.046821	-0.094154	-0.026341	0.010520
4	-0.046488	0.087589	-0.010288	-0.015407	-0.082850	-0.083220
5	0.000043	-0.009364	0.063103	0.006054	-0.098620	-0.096770

Availability:

	0	1	2	3	4	5
0	0.038153	-0.037342	-0.043960	-0.069483	-0.079159	-0.073429
1	-0.057853	0.172235	-0.040946	-0.069483	-0.080567	-0.067467
2	-0.057853	-0.037342	0.174258	-0.069483	-0.079159	-0.072548
3	-0.057853	-0.037342	-0.050579	0.059062	-0.081443	-0.071405
4	-0.057853	-0.068862	-0.040946	-0.074371	0.019597	-0.067467
5	-0.068663	-0.037342	-0.059901	-0.078741	-0.079159	0.071824

Criterion:

	0	1	2	3	4	5
0	-0.035297	-0.010971	-0.009846	-0.125179	-0.115877	-0.048216
1	-0.089773	0.069777	-0.039305	-0.087540	-0.074805	-0.108342
2	-0.088396	-0.040664	0.065412	-0.073688	-0.141319	-0.060864
3	-0.134252	-0.018396	-0.003758	-0.035092	-0.107783	-0.060885
4	-0.104341	0.018727	-0.051233	-0.089779	-0.063253	-0.150687
5	-0.068620	-0.046706	0.003202	-0.072687	-0.177779	-0.024946

Iteration 5:

Responsibility:

	0	1	2	3	4	5
0	-0.070934	0.028620	0.036027	-0.053180	-0.034201	0.027729
1	-0.040341	-0.100131	-0.006780	-0.026478	-0.002938	-0.049296
2	-0.038376	-0.011155	-0.105963	-0.012038	-0.069993	0.002835
3	-0.073631	0.021509	0.048688	-0.091385	-0.023572	0.013289
4	-0.042971	0.090573	-0.006771	-0.011890	-0.079407	-0.079703
5	0.003408	-0.005999	0.064457	0.009419	-0.095255	-0.093775

Availability:

	0	1	2	3	4	5
0	0.033896	-0.029873	-0.035168	-0.066637	-0.076454	-0.070229
1	-0.055082	0.173659	-0.032757	-0.066637	-0.077580	-0.057901
2	-0.055082	-0.029873	0.174808	-0.066637	-0.076454	-0.061966
3	-0.055082	-0.029873	-0.040463	0.052261	-0.078281	-0.065924
4	-0.055082	-0.057886	-0.032757	-0.070983	0.015678	-0.057901
5	-0.067104	-0.029873	-0.047921	-0.078619	-0.076454	0.069891

Criterion:

4]:

	0	1	2	3	4	5
0	-0.037038	-0.001253	0.000860	-0.119816	-0.110656	-0.042500
1	-0.095423	0.073528	-0.039537	-0.093115	-0.080518	-0.107197
2	-0.093458	-0.041028	0.068845	-0.078675	-0.146448	-0.059131
3	-0.128713	-0.008364	0.008225	-0.039124	-0.101853	-0.052635
4	-0.098053	0.032687	-0.039527	-0.082873	-0.063729	-0.137604
5	-0.063696	-0.035872	0.016536	-0.069200	-0.171709	-0.023884

Now, we have 2 Clusters at Point 1 and Point 2 whose values are as above.

```
In [423]: ClusterPoints = np.zeros(len(c))
          for row in range(len(c)):
              ClusterPoints[row] = np.argmax(c[row,])
```

```
In [424]: ClusterPoints
Out[424]: array([2., 1., 2., 2., 1., 2.])
```

I have done the damping factors since without it, the results were completely illogical and led to a complete chaos. If we do not take into consideration the damping factor to avoid oscillations, we will have the following results in iteration number 5:

Responsibility:

	0	1	2	3	4	5
0	-0.104550	-0.239971	-0.233632	-0.320428	-0.301449	-0.239519
1	-0.467597	-0.134497	-0.434037	-0.453734	-0.431311	-0.476552
2	-0.467061	-0.439840	-0.141696	-0.440723	-0.498678	-0.429915
3	-0.397720	-0.303401	-0.284586	-0.130888	-0.347661	-0.310801
4	-0.257515	-0.159751	-0.221315	-0.226434	-0.134497	-0.294247
5	-0.317897	-0.327304	-0.274864	-0.311886	-0.416560	-0.141696

Availability:

	0	1	2	3	4	5
0	0.00000	-0.134497	-0.141696	-0.130888	-0.134497	-0.141696
1	-0.10455	0.000000	-0.141696	-0.130888	-0.134497	-0.141696
2	-0.10455	-0.134497	0.000000	-0.130888	-0.134497	-0.141696
3	-0.10455	-0.134497	-0.141696	0.000000	-0.134497	-0.141696
4	-0.10455	-0.134497	-0.141696	-0.130888	0.000000	-0.141696
5	-0.10455	-0.134497	-0.141696	-0.130888	-0.134497	0.000000

Criterion:

	0	1	2	3	4	5
0	-0.104550	-0.374468	-0.375328	-0.451316	-0.435946	-0.381215
1	-0.572147	-0.134497	-0.575732	-0.584622	-0.565808	-0.618248
2	-0.571611	-0.574336	-0.141696	-0.571611	-0.633175	-0.571611
3	-0.502270	-0.437898	-0.426282	-0.130888	-0.482158	-0.452497
4	-0.362065	-0.294248	-0.363010	-0.357322	-0.134497	-0.435943
5	-0.422447	-0.461801	-0.416560	-0.442774	-0.551057	-0.141696

As you can see below that it creates for us 5 clusters for 5 points.

```
In [430]: ClusterPoints = np.zeros(len(c))
          for row in range(len(c)):
              ClusterPoints[row] = np.argmax(c[row,])
          ClusterPoints
Out[430]: array([0., 1., 2., 3., 4., 5.])
```