Assignment 7: Clustering

Low, Daniel Mark (S3120155)
Petre, Bogdan (S3480941)
Xu, Teng Andrea (S3548120)

Group 7

October 22, 2017

1 Pen-and-paper question

Table 1: Distance matrix step 0

	p1	p2	p3	p4	p5
p1	0.0	0.9	0.6	0.5	0.7
p2	0.9	0.0	0.4	0.6	0.1
p3	0.6	0.4	0.0	0.6	0.2
p4	0.5	0.6	0.6	0.0	0.3
p5	0.7	0.1	0.2	0.3	0.0

1.1 Single linkage clustering

Shortest distance is 0.1 between p2 and p5. We merge p2 and p5 and recompute the proximity matrix. We get Table 2. The distance between 2 clusters is the distance between closest elements in those clusters. For example: d((p2,p5),p3) = min (d(p2,p3), d(p5,p3)) = min (0.4, 0.2) = 0.2.

Table 2: Distance matrix step 1 (single link)

	p1	(p2,p5)	p3	p4
p1	0.0	0.7	0.6	0.5
(p2,p5)	0.7	0.0	0.2	0.3
p3	0.6	0.2	0.0	0.6
p4	0.5	0.3	0.6	0.0

Shortest distance is 0.2 between (p2,p5) and p3. We merge p3 and (p2,p5) and recompute the proximity matrix. We get Table 3.

Table 3: Distance matrix step 2 (single link)

	p1	(p2,p3,p5)	p4
p1	0.0	0.6	0.5
(p2,p3,p5)	0.6	0.0	0.3
p4	0.5	0.3	0.0

Shortest distance is 0.3 between (p2,p3,p5) and p4. We merge p4 and (p2,p3,p5) and recompute the proximity matrix. We get Table 4.

Table 4: Distance matrix step 3 (single link)

	p1	(p2,p3,p4,p5)
p1	0.0	0.5
(p2,p3,p4,p5)	0.5	0.0

We merge the 2 remaining clusters. Then, we construct the dendrogram from Figure 1.

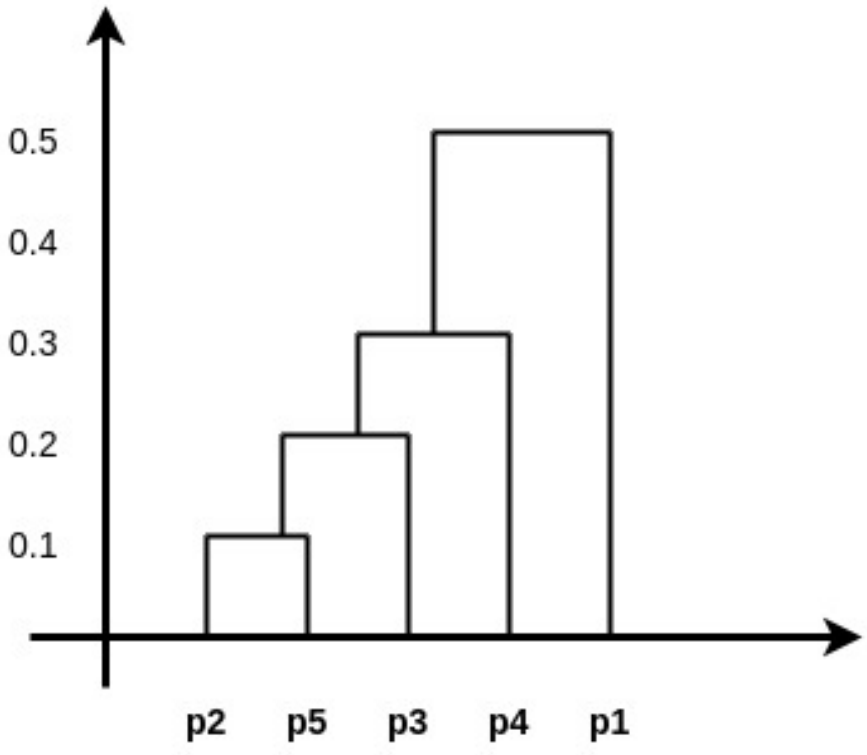

Figure 1: Dendrogram 1

1.2 Average linkage clustering

Shortest distance is 0.1 between p2 and p5. We merge p2 and p5 and recompute the proximity matrix. We get Table 5. The distance between 2 clusters is the average of all pairwise distances in those clusters. For example: d((p2,p5),p3) = (d(p2,p3) + d(p5,p3))/2 = (0.4 + 0.2)/2 = 0.3 .

Table 5: Distance matrix step 1 (average link)

	p1	(p2,p5)	p3	p4
p1	0.0	0.8	0.6	0.5
(p2,p5)	0.8	0.0	0.3	0.45
p3	0.6	0.3	0.0	0.6
p4	0.5	0.45	0.6	0.0

Shortest distance is 0.3 between (p2,p5) and p3. We merge p3 and (p2,p5) and recompute the proximity matrix. We get Table 6.

Table 6: Distance matrix step 2 (average link)

	p1	(p2,p3,p5)	p4
p1	0.0	0.73	0.5
(p2,p3,p5)	0.73	0.0	0.5
p4	0.5	0.5	0.0

Shortest distance is 0.5 between both (p2,p3,p5) and p4 and also between p1 and p4. We can now choose which clusters to merge. We merge p1 and p4 in order to reduce the *chaining* effect and recompute the proximity matrix. We get Table 7.

Table 7: Distance matrix step 3 (average link)

	(p1,p4)	(p2,p3,p5)
(p1,p4)	0.0	0.61
(p2,p3,p5)	0.61	0.0

We merge the 2 remaining clusters. Then, we construct the dendrogram from Figure 2.

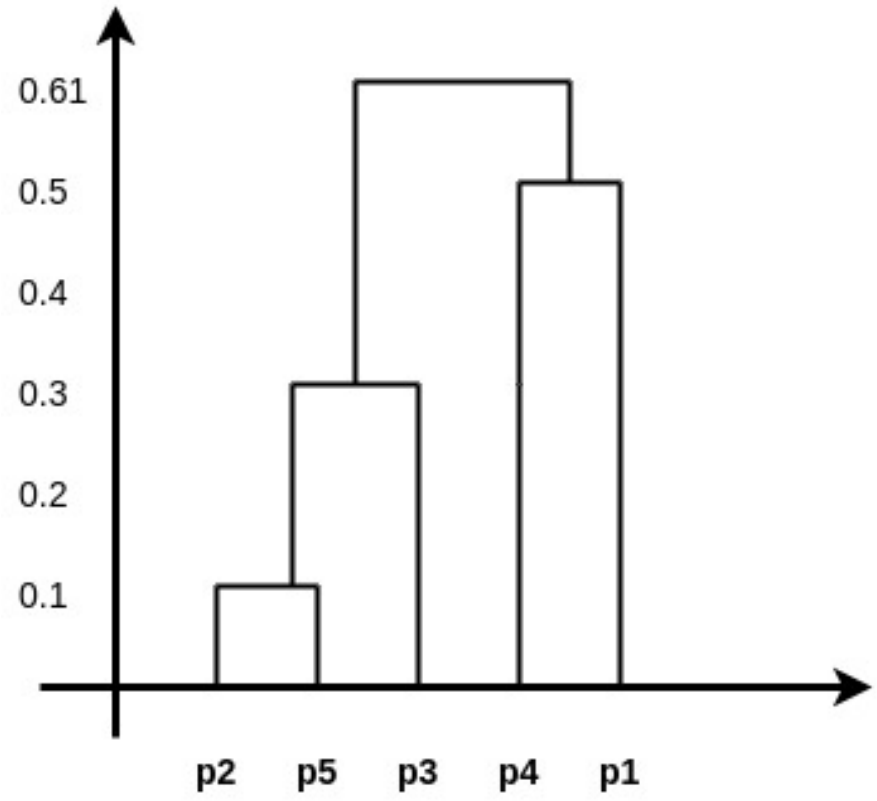

Figure 2: Dendrogram 2

2 Descriptive Analysis and Agglomerative Clustering (20 pts)

2.1 Which difficulties do you encounter?

Code in: **2_1_descriptive.py**
First we wanted to visualize the data. So we reduced each dataset to 2 dimensions with PCA and plotted. The problem is there are 280 thousand data points, which is to large to compute a

covariance matrix fast and unnecessary for plotting purposes. So we randomly selected 1% of the data (n=2800 data points). Another problem is that the dimensions in the datasets are in different magnitudes or even different scales (e.g., space vs. energy). So we z-transformed and normalized each dimension.
The first two components of the PCA of 3D data set explain 85% and 16% of the variance, respectively, whereas they explain 39% 27% for the 6D dataset. We plotted both PCAs in Figure 3 and Figure 4. We see the components do not distribute normally for the 3D dataset but do for the 6D dataset. For the 3D dataset, most datapoints are centered around (-1,0) and there seems to be a second cluster around (4,0). This could be useful for clustering. In the 6D dataset visualization, it is hard to find two clusters by just visual inspection, but one could assume that since these two dimensions are explaining most of the variance, they should be able to separate data in two different parameters quite well. Summing up, the difficulties consist in that it is a very large dataset (solved with randomly subsampling) and that the scales of the different dimensions are quite different (solved by z-transformation and normalization).

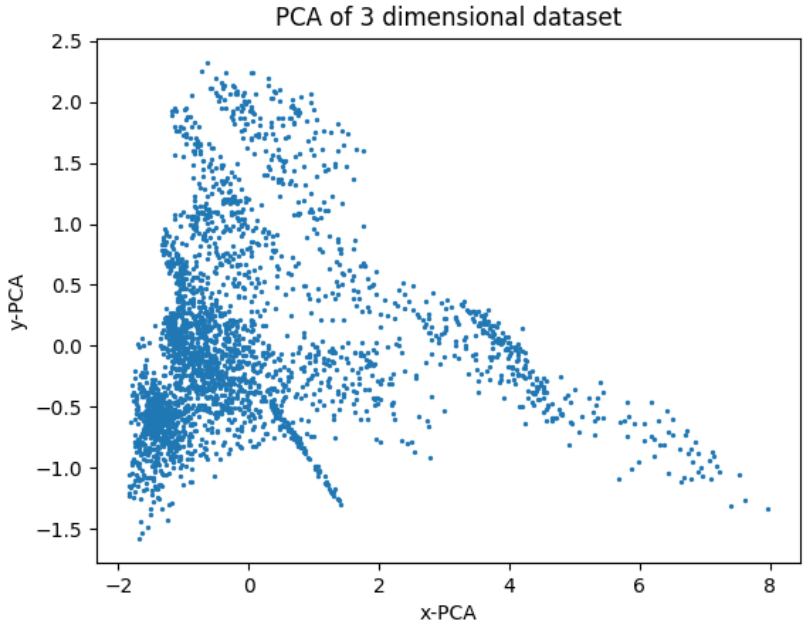

Figure 3: PCA 3D dataset

2.2 What differences are there between the 3d and 6d representation of the data?

Code in **2.py**
We used **agglomerative clustering technique with Ward method** to cluster the data. We then used **Cophenetic Correlation Coefficient** to measure the clustering. This coefficient correlates the pairwise distances of all the samples to those implied by the hierarchical clustering. The closer the value is to 1, the better the clustering preserves the original distances, which in our case was **0.60** for the 3D database –which is not too great– and **0.35** for the 6D database –which is pretty bad. So the 3D database clusters better (at least using this technique), which was somewhat visible from the PCA visualizations from 2.1 (the 3D dataset seemed to have at least two clusters).

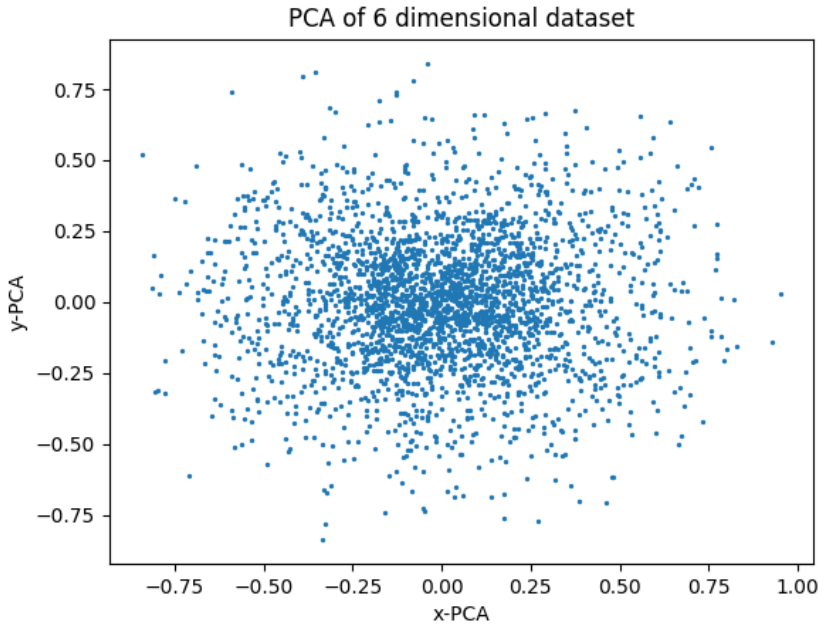

Figure 4: PCA 6D dataset

2.3 Plot the dendrograms

Code in **2.py**
We plot the dendrograms using **Ward** method in Figure 5 and Figure 6. The 3D dataset clustered into two clusters as we have been seeing from the PCA on. The 6D dataset clusters into 4 clusters. We will further analyze the dendrograms along with the other ones using different .

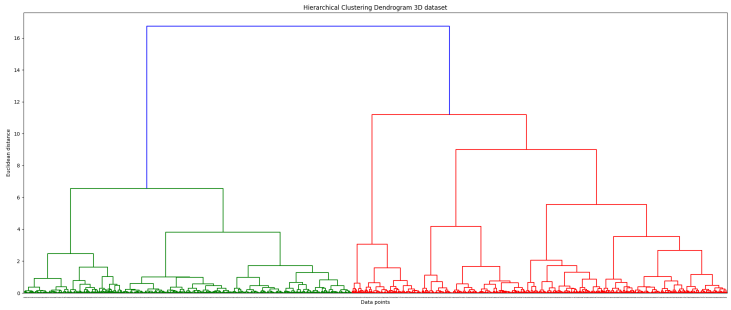

Figure 5: Dendrogram 3D dataset Ward method

2.4 How do the clusterings change when different linkages are considered? Which linkage strategy do you think is best for this data? Argue why.

In Figure 7 we compare the **ward** method with **single**, **complete**, and **average** in the 3D dataset. The single-link and average-link methods seem to be unbalanced in that they assign a vast amount of data points to a single cluster. The complete and Ward methods are more balanced. The Ward, as we have seen, assigns two main clusters and the complete spots an additional one.
In complete-link clustering, the similarity of three clusters is the similarity of their most dissimilar members. It has the benefit of using information from all the data points (in comparison to

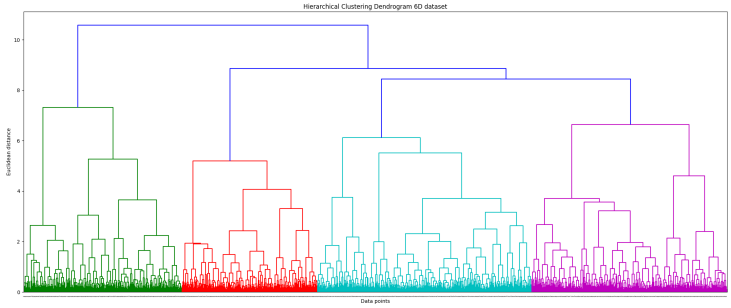

Figure 6: Dendrogram 6D dataset Ward method

single-link clustering), which creates a preference for compact clusters with small diameters over long clusters, but also causes sensitivity to outliers.
Ward may be the best because it can be more accurate than complete when discovering uneven clusters with different variances, which we should have here since each galaxy should have different variance. In Figure 8 we compare the same four methos in the 6D dataset. Here again complete-link and Ward are the most balanced and best options.

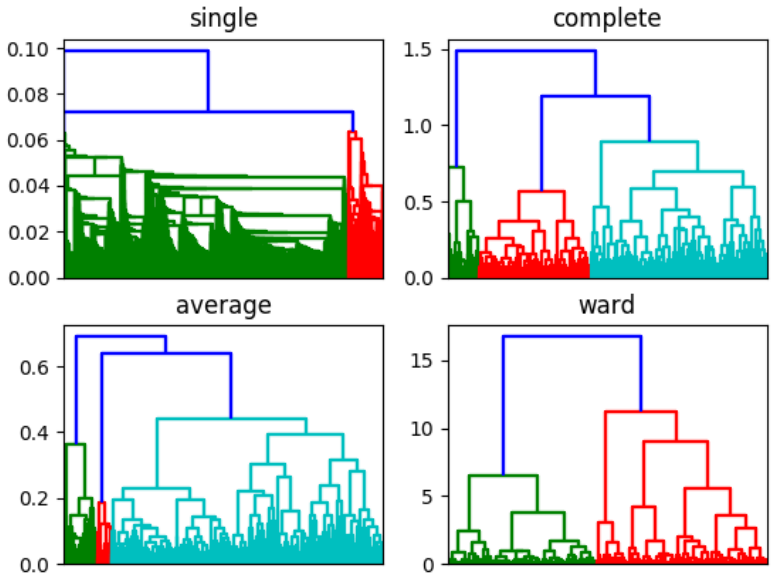

Figure 7: Dendrograms 3D dataset Single, Complete, Average and Ward methods

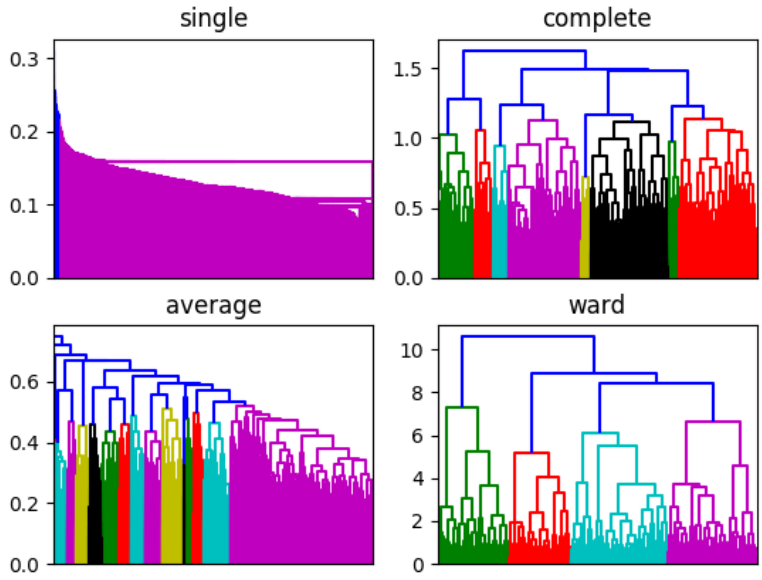

Figure 8: Dendrograms 6D dataset Single, Complete, Average and Ward methods

3 K-Means Clustering

3.1 Differences between 3D and 6D data

The following are the main differences discovered when running the code:

- using euclidean distance, the avg silhouette of the clustering for each k in 3D data is more like to be close to 0.6 showing that the 3D achieves a better clustering than using 6D data with or without PCA decomposition.
- they differ in the best k(following section)
- using cosine distance with 6D brings to immediate drop out after the second cluster, instead for 3D data brings to an not usual curve, because for some k the absolute value of dispersion raises again instead of going constantly close to zero.

3.2 What is the best number of clusters k? Perform an empirical experiment (cross-validation), and plot a relevant metric vs. k (keyword "ellbow"). This could be internal evaluation measures, like the SSE and Silhouette as mentioned in the slides. (In general it is advisable to compare with different measures)

The code for this exercise in in 3_2_elbow.py and 3_2_silhouette.py.
In 3_2_silhouette:
We used the v-1 cross validation technique and for each fold we run the k mean for each value of k *in* [2,25] calculating the silhouette avg for each cluster and then obtaining the average for each value of k. Despite a strange high value of the silhouette coefficient for k=2, **the best K** obtained for 3D k=4 (but also k=5 obtained a good score) and for 6D is k=3 (but also k=4 and k=5 have good score), both have a close value and the elbow starts from here.

In the following a nice plot using PCA for both, the code is taken from the official site of sklearn[1] and modified as needed.

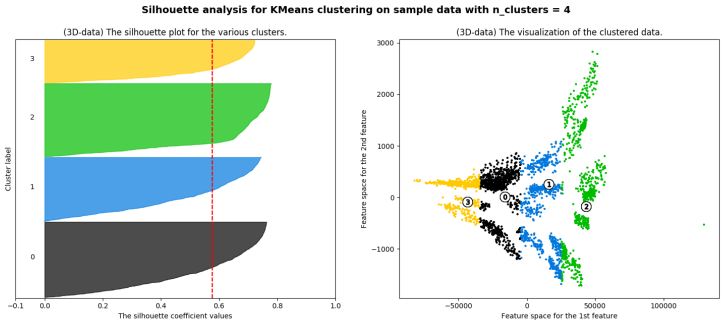

Figure 9: Visualization of data(3D). sx silhouette average for each cluster, dx clusters

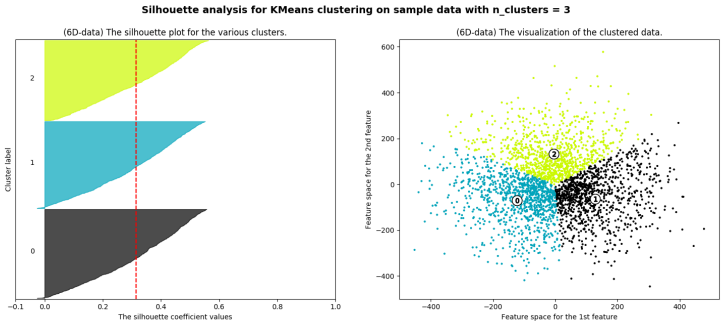

Figure 10: Visualization of data(6D). sx silhouette average for each cluster, dx clusters

In 3_2_elbow:
The algorithm is quite simple, for the same range of K as above we obtained a score evaluation of clustering. This time we used the "dispersion" metric[2]. In the following the plot of metrix vs k:

Using 3D data we could confirm the best k is around 4 or 5, instead the elbow method do not help us much for since decrease almost constantly without any sudden decrease.

3.3 What happens when you use different distance measures for the clustering, e.g. Man- hattan?

The code for this exercise is in 3_3_Manhattan.py
Here there were a lot of problems since the cluster function KMeans of numpy library only has the euclidean distance as distance metric.
So a rough implementation of k means was implemented here. Using Manhattan distance achieves do not improve so much the final clustering evaluation, because for what concerns the silhouette coefficient both for 3D and 6D we have a decrease of 0.1. Furthermore the elbow plot of 6D still do not have any sudden decrease but goes slowly down close to zero.

[1] http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html
[2] http://kldavenport.com/the-cost-function-of-k-means/

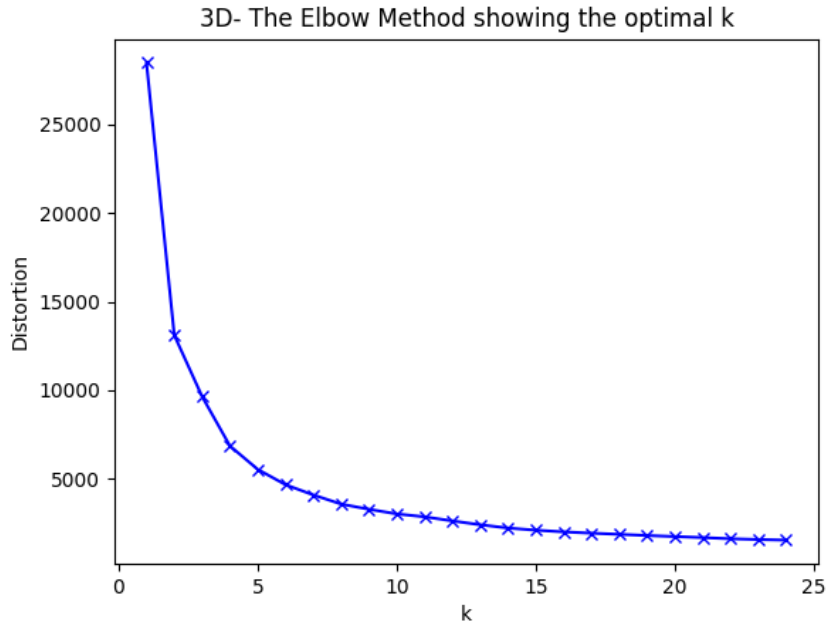

Figure 11: Elbow plot for 3D data

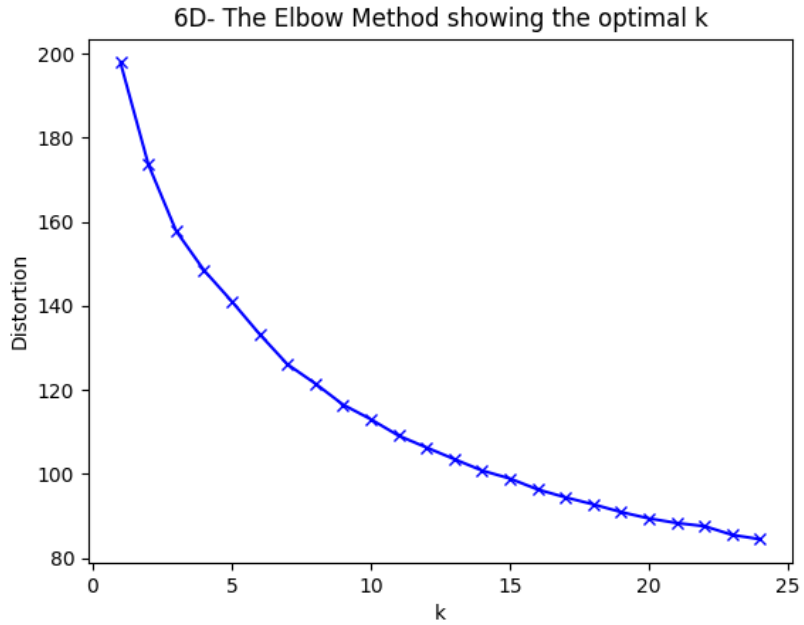

Figure 12: Elbow plot for 6D data

3.4 Consider: how do you initialize the clusters? Do you observe differences in the evaluation measures dependent on the strategy of initialization?

The code is in 3_4.py. In every script we **initialized** the clusters **randomly**. **Trying** with chosen cluster centroids do not bring to any improvement in cluster evaluation, since in k means in every iteration the centroid is recomputed and each point re-assigned. This is the property of k means that always converges. The only gain that we can obtain is that the runtime is lower than

randomized centroids because it stops in few steps, we can see it easily running the script 3_4.py.

4 Gaussian Mixture Model

4.1 Pseudo-code

Notations:

- mean: μ
- covariance matrix: Σ
- mixture weight: w
- unknown parameters: $\theta = (w, \{\mu_k\}, \{\Sigma_k\})$
- gaussian: N
- log likelihood function: $log(p(x|\theta)) = \sum_n log(\sum_k w_k N(x_n|\mu_k, \Sigma_k))$
- $z^{(m)}$: value of variable z at iteration m

The pseudo-code can be described as follows:

Data: $\{x_i\}_{i=1}^n \in \mathbb{R}^d$, number of clusters k.

1. Initialize K Gaussians with mean $\mu_1, ..., \mu_k$, covariance matrix $\Sigma_1, ..., \Sigma_k$ and mixture weights $w_1, ..., w_k$
2. repeat:
3. for each Gaussian k:
4. for each x_i:
5. $\gamma_{i,k}^{(m)} = \frac{w_k^{(m)} N(x_i|\mu_k, \Sigma_k)}{\sum_{l=1}^k w_l^{(m)} N(x_i|\mu_l, \Sigma_l)}$ responsibility of Gaussian k for sample x_i
6. $\mu_k^{(m+1)} = \frac{1}{N_k} \sum_n \gamma_{nk} x_n$ with $N_k = \sum_n \gamma_{nk}$
7. $\Sigma_k^{(m+1)} = \frac{1}{N_k} \sum_n \gamma_{nk} (x_n - \mu_k^{(m+1)})(x_n - \mu_k^{(m+1)})$
8. $w_k^{(m+1)} = \frac{N_K}{N}$
9. evaluate log likelihood function
10. until log likelihood function does not change (much)

4.2 GMM on the datasets

The code for this exercise can be found in **4_2.py**.

We used the implementation of GMM provided by **sklearn.mixture.GaussianMixture**. We initialized the covariance matrix Σ_k as a diagonal matrix, instead of a general matrix, because it provides a good compromise between quality and model size. For the means μ_k we used both k-means initialization, which uses the K-means method to get the initial means, and random initialization.

Figure 13 shows the values of the log likelihood function for the GMM method with both k-means and random initialization and for both the 3- and 6-feature datasets. They are obtained for a number of clusters ranging from 1 to 24. We can observe that better results (i.e. higher log likelihood values) are obtained for both datasets when the k-means initialization is used.

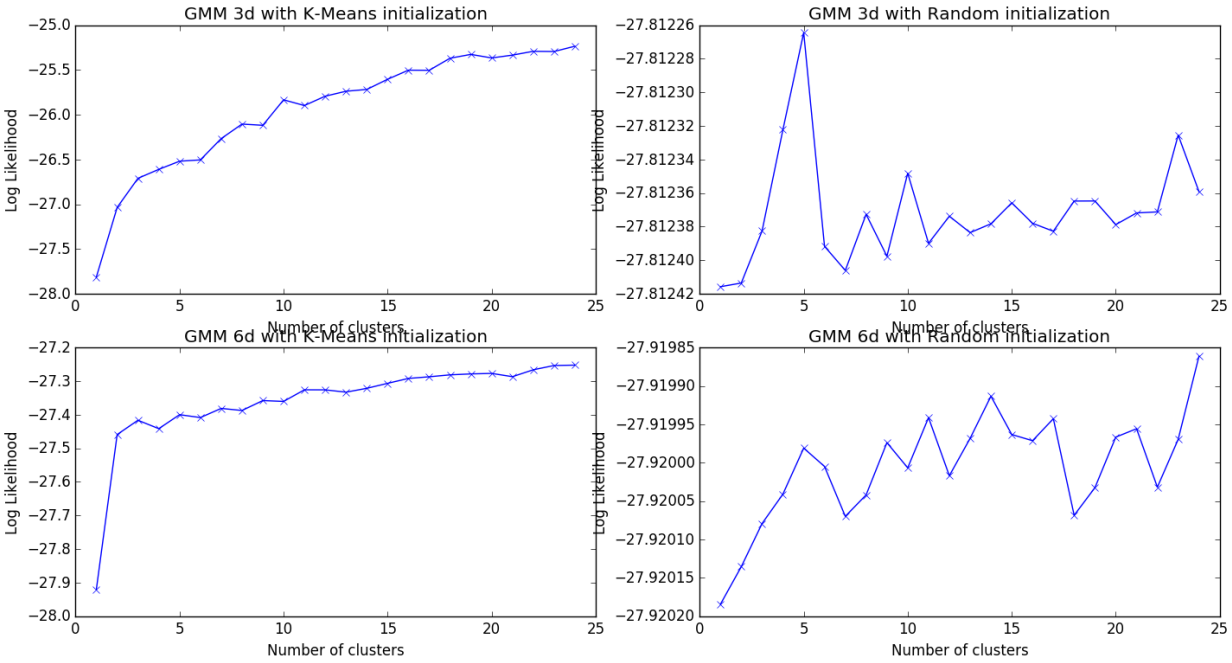

Figure 13: GMM applied to the dataset

4.3 Cluster evaluation

The code for this exercise is in 4_3.py.
In this exercise we used again the silhouette coefficient and from sklearn metrics we used the calinski harabaz score [3]. In the end we run the algorithm for range k *in* [2,25] and found that for k=10 we had the best tuple calinski and silhouette then we predicted for GMM with k = 10.

[3] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.calinski_harabaz_score.html