Grenoble INP - Ense3

AI and Autonomous systems

Lab 6 - M2-MARS

clustering

Author :
Souhaiel BEN SALEM

16 November 2021

Lab Objectives: clustering, is an unsupervised machine learning task. It involves automatically discovering natural grouping in data. The objective of this lab is to illustrate the basic concepts of clustering, its use cases, its different variations and compare it to the EM algorithm. The whole process will be carried through application of the proposed methods to different datasets.

Introduction:

Clustering is the assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis used in many fields.
K-means clustering is an algorithm to classify or to group objects based on attributes/features into K number of group where K is positive integer number.
Clustering algorithms are a powerful technique for machine learning on unsupervised data.

1 Notebook 1: KMEANS BASICS

1.1 Implementation of Kmean on a simple case

Exercise:

1. **Explain/ comment the code below**
   The code below represents the steps of implementing the Kmeans algorithm on a 1-dimentional dataset:

   - We start by setting the number of desired clusters $K = 2$ (i.e we want out data points to be grouped in two different groups/clusters) and the number of features $p = 1$ (1D problem).
   - The next step is to choice two random points to be the centroids of the two clusters.
   - Assign all the points to the closest cluster centroid.
   - Recompute the centroids of newly formed clusters.
   - Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

   The approach kmeans follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.
   Here is the code fully commented:

```
# N = 20 (the number of elements in the matrix X)
N=X.size

# This variable will be used to store the index of the points having the minimum distance from the centroid
idx=np.zeros((N,1))

# This 2x1 array is used to store the centroids of the two clusters.
# This array will be updated each time we find new centroids minimizing the distance between the points of
# a given cluster
muvec=np.zeros((K,1))

#the vaible that will be used to stop the algorithm
#this variable will tell us if the centroids do change or not after each iteration
#the algorithm will be stopped when the centroids are fixed
change = True

#Generating two random indices in the range [0,20] to be used for the centroids of the two clusters
perm=np.random.permutation(N)[0:2]

#fixing the two inial centroids of each cluster
for k in range (0,K):
    muvec[k] = X[perm[k],:]

# Compute the sum of the squared distance between data points and the two centroids.
# and assign each point to a cluster (based on its distance from the closest centroid)
for i in range (0,N):
    d=(X[i] - muvec )**2
    idx[i]=np.where(d==d.min())[0]


#update centroids until there is no change (i.e until we reach the two Centroids of newly formed clusters do not change
while change:
    change=False
    #update the centroids
    for k in range (0,K):
        muvec[k]= np.mean( X[idx == k] )
    #prediction : assign each data point to the new formed clusters
    for i in range (0,N):
        d=(X[i] - muvec )**2
        index=np.where(d==d.min())[0]
        if index != idx[i]:
            change=True
            idx[i]=index

# plotting the histogram and the data points after clustering ( red points belong to cluster 0 and green poitnts to cluster 1)
X0=X[idx==0]
X1=X[idx==1]
bins=np.arange(np.min(X)-1,np.max(X)+2,1)
n,bin_edges,patches=plt.hist(x=X,bins=bins, color='blue',histtype='step')
plt.scatter(X0,np.zeros_like(X0)+.5,c='red',marker='+', label='class 0')
plt.scatter(X1,np.zeros_like(X1)+.5,c='green',marker='+',label='class 1')
plt.legend()
h=plt.gcf()
```

Figure 1: the implementation of the Kmeans algorithm fully explained

2. **What is the main problem left aside by this code?**
   The main problem of the previous code is the random assignment of the centroids of each cluster. Thus, the final result (i.e the two final clusters) highly depends on selection of initial centroids. k-Means algorithm is sensitive to initial centroids so proper selection of initial centroids is necessary.

1.2 sklearn implementation

Exercise:

1. **Compare the results obtained with the simple code above**
   The scikit-learn implementation uses the k-means++ which ensures a smarter initialization of the centroids and improves the quality of the clustering. Apart from initialization, the rest of the

algorithm is the same as the standard K-means algorithm.
We notice that the two algorithms yield the same clustering result, with the only difference being the name of the class/cluster which is not important in this case because we are dealing with unlabeled data.
We probably got the same result because the problem is simple and the k-means++ does not make a difference in the case.

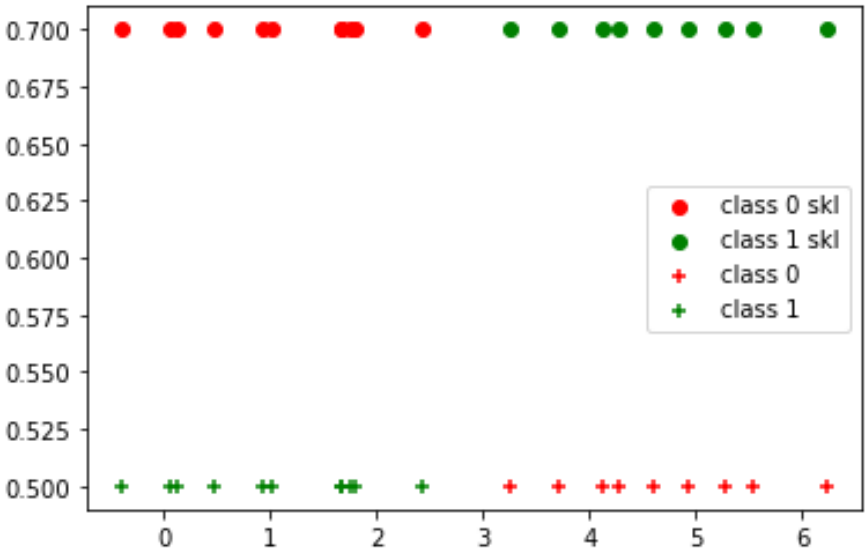

Figure 2: the results given by the two implementations

2. **Comment and explain the role of the input parameters used in this implementation**
   The parameters of KMeans:
   - **n_clusters**:The number of clusters to form which also serves as the number of centroids to generate.
   - **init**: The method used for initialization, in this case we use the k-means++ algorithm.
   - **max_iter**: The maximum number of iterations of the k-means algorithm for a single run.
   - **n_init**: The number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
   - **random_state**: Determines random number generation for centroid initialization. Here we used an integer (0) to make the randomness deterministic.

2 Notebook 2: IRIS DATA : KMEANS

2.1 Exercise:

1. **Comment the choice of Kmeans input parameters used above**
   In this example we used K-Means in a loop to compare the result we would get for different numbers of clusters. The parameters are:
   - **n_clusters**:The number of clusters to form which also serves as the number of centroids to generate. The number of clusters in this case is variable (i) ranging from 1 to 10.
   - **init**: The method used for initialization, in this case we use the k-means++ algorithm.
   - **max_iter**: The maximum number of iterations of the k-means algorithm for a single run. Here, specified that the algorithm must stop after 300 iterations.
   - **n_init**: The number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia. In this case we chose n_init=10 .
   - **random_state**: Determines random number generation for centroid initialization. Here we used an integer (0) to make the randomness deterministic.
2. **'The elbow method' from the above graph : find the optimum number of clusters by observing the within cluster sum of squares (WCSS). Explain the shape of the curve WCSS=f(nb of clusters)**
   By observing the elbow graph, we can deduce that the optimal K value is $k = 3$ (the value on the elbow).
   the shape of this grapth is explained by the fact that each iteration we increase the number of clusters, which means that we get more cluster with lesser points and as result the within cluster sum of squared distances keep decreasing.
   The wcss decreases dratically at first due to the new centroids being not the optimal ones, howevwr with each iteration and as we get more clusters, the new generated centroids would not change as much and eventually we would have reached the final result with fixed centroids. That is why the wcss decreases slowly after a certain point (the elbow point)after which the distortion/inertia start decreasing in a linear fashion.

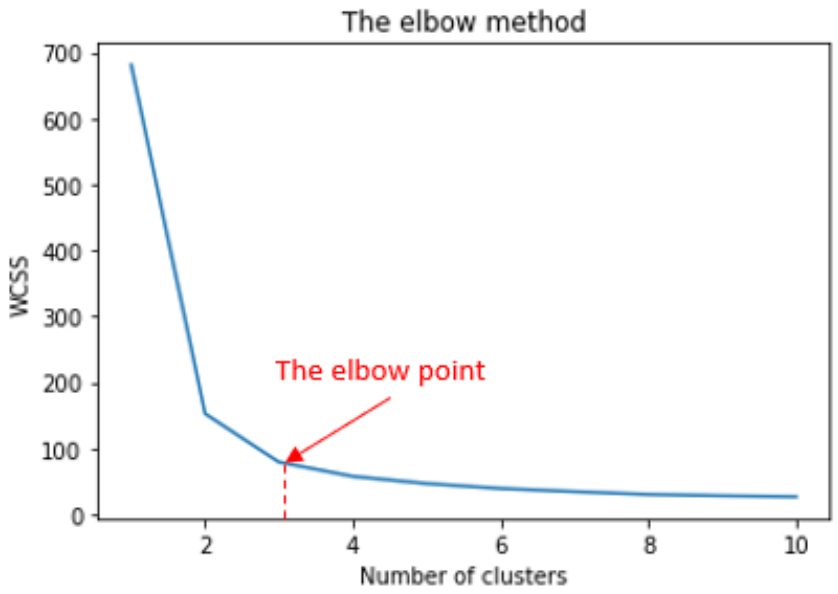

Figure 3: the results given by the two implementations

3. **What is the asymptotic value of WCSS when the number of clusters approaches N (nb of points)?**
   By inspecting the graph, we can assume the the asymptotic value of WCSS when the number of clusters approaches N=10 is around 25.
   We can get the exact value by using **np.min(wcss)** which yields a value of 26.534529220779234 .
4. **Explain why the curve doesn't decrease significantly with every iteration**
   the curve doesn't decrease significantly with every iteration because the number of clusters increases and thus there is less points per cluster and the new generated centroids are better positioned than their predecessors. These are the two reasons that the WCSS does not decrease as dramatically after each iteration.

Exercise:

1. **remind the reasons why the clusters formed by KMeans algorithm are included in Voronoï cells associated to the centroïds**
   The K-Means clusters and Voronoi diagrams are closely related by concept. For example, take any set of k points in a convex bounded region R of space. Then the Voronoi diagram consists of k regions such that region R_x is the set of points in R closer to the point x than to any other of the $k-1$ points. K-means is an algorithm which tries to find a set of k points in R such that the Voronoi diagram has an additional property that each of the k points x is the centroid of R_x and that is why the clusters resulting of the K-means algorithm are the Voronoi diagrams associated to the centroids.

2. **Comment the shape of the obtained clusters represented in the figure above**
   We notice that the blue cluster's variance is mostly explained by the vertical axis whereas for the two others (red and green), the variance is explained by the horizontal axis. We also notice the the green and red cluster are overlapping.
3. **How would you check that enough iterations were performed?**
   To check whether enough iterations were performed or not, we just have to keep track of the distortion/inertia. We can use the WCSS metric to keep track of the inertia. The stopping criterion would be when the inertia start to decrease slowly or not decrease at all.

Exercise:

Propose a measure of the goodness of clustering, associated to this problem (implementation is not required).
To evaluate the goodness of clustering we can use a lot of metrics such as the Silhouette Coefficient/score which is used to evaluate the quality of clusters created using clustering algorithms such as K-Means in terms of how well samples are clustered with other samples that are similar to each other. The Silhouette score is calculated for each sample of different clusters. To calculate the Silhouette score for each observation/data point, the following distances need to be found out for each observations belonging to all the clusters:

- Mean distance between the observation and all other data points in the same cluster. This distance can also be called a mean intra-cluster distance. The mean distance is denoted by a.
- Mean distance between the observation and all other data points of the next nearest cluster. This distance can also be called a mean nearest-cluster distance. The mean distance is denoted by b.

The Silhouette score, S, for each sample is calculated using the following formula:

$$S = \frac{(b - a)}{max(a, b)}$$

The score is the range of [-1,1] and can be interpreted as follows:

- 1: Means clusters are well apart from each other and clearly distinguished.
- 0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.
- -1: Means clusters are assigned in the wrong way.

The implementation yiels a score of 0.553.

How could the cost-complexity tradeoff be tackled?
to stop our model from getting too complicated (which will lead to underfitting and high computational cost) we can use cross-validation to choose the best number of clusters k.

```
#Visualising the clusters x1, x3
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 2], s = 25, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 2], s = 25, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 2], s = 25, c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,2], s = 25, c = 'yellow', label = 'Centroids')

plt.legend();
score = silhouette_score(x, y_kmeans, metric='euclidean')
print('Silhouetter Score: %.3f' % score)
```

```
Silhouetter Score: 0.553
```

Figure 4: Implementing the Silhouette score metric

3 Notebook 3: Kernel Kmeans

Questions:

1. **Briefly explain why usual Kmeans algorithm will fail to detect the classes above**
   The two concetric circles (i.e classes) are non-linearly separable. Also, keeping in mind that k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, we can see why the K-Means algorithm would fail to separate these classes as the two circular classes would have a very similar mean mean.

2. **Is the Kernel approach the only possibily for this kind of clustering problem?**
   Kernel k-means clustering is a powerful tool for unsupervised learning of non-linearly separable data. This method would allow us to find clusters of arbitrary shapes using polynomial representations and thus the problem proposed earlier would be perfectly solvable.
   We can also convert our Cartesian (x, y) coordinates to polar coordinates, and use only the radius r for clustering, as the angle θ doesn't matter.

3. **Propose a change of representation space to allow successfull Kmeans clustering in a 1D space. Implement it (use Kmeans**$_{b}asic.ipynbexample$**)**
   As proposed earlier, we can work with the polar coordinates to solve the problem of k-means failing to cluster non-linearly correlated data in 1D.
   We can also project our data into a higher dimensional space to make the resulting clusters linearly separable (use of a Kernel trick).

4. **Explain the role, then change the parameter values of 'gamma' and 'pos' in Kernel Kmeans code below.**
   In this example we are using an RBF kernel. **Gamma** is the sensitivity to differences in feature vectors whereas the **pos** variable specifies The starting position (point) of the kernel (i.e where the kernel must be centered at the start). Different results obtained after altering gamma and pos:

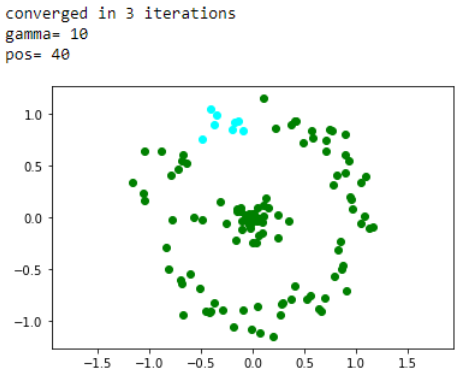

Figure 5: result for gamma=10 and pos=40

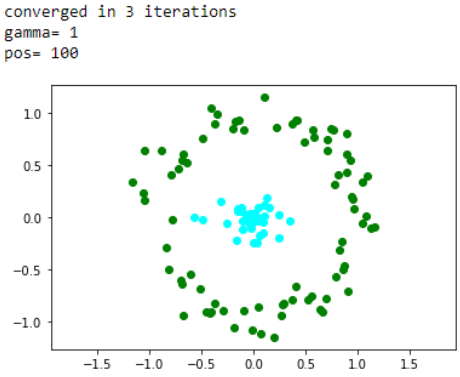

Figure 6: result for gamma=1 and pos=100

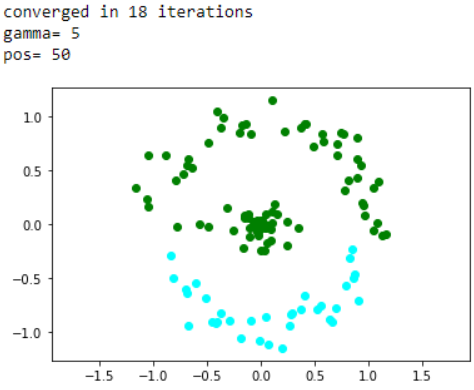

Figure 7: result for gamma=5 and pos=50

5. **Comment your findings**
   We notic that Kernel K-means is very sensitive to initial conditions. If the inial conditions are badly stated, the algorithm will yield an unuseful result.

4 Notebook 4: EM basic example

Exercise:

1. The arrays used in the code can be identified as follows:
   - **pivec**: the mixing distribution $\pi = (\pi_1, \pi_2..\pi_k)$
   - **muvec**: the set of means $\mu = (\mu_1, \mu_2..\mu_k)$
   - **sigvec**: the set of covariance matrices $\Sigma = (\Sigma_!..\Sigma_k)$
   - **postpr**: the probability distributions of samples (posterior distribution of the latent variables) $p = p_1, p_2...p_n$
2. **Explain why different means are initialized, whereas same variances may be used**
   The EM algorithm is a local optimization method, and hence particularly sensitive to the initialization of the model. The simplest way to initiate the GMM is to pick numClusters data points at random as mode means, initialize the individual covariances as the covariance of the data, and assign equa prior probabilities to the modes.
3. **Comment the line codes below, briefly**
   These lines of code are the implementation of the GMM algorithm on our dataset. The GMM algorithm consists of two steps: the E-step and the M-step. Firstly, the model parameters and the $z^(i)$ can be randomly initialized. In the E-step, the algorithm tries to guess the value of $z^(i)$ based on the parameters, while in the M-step, the algorithm updates the value of the model parameters based on the guess of $z^(i)$ of the E-step. These two steps are repeated until convergence is reached.The algorithm in GMM is repeat until convergence.

```
#A posteriori Proba to be in a class


for t in range (0,MaxIter):
    #E-Step
    # Computing the conditional distribution of the latent vriables given the data samples
    for i in range(0,N):
        px = 0
        for k in range (0,K):
            px=px + pivec[k] * stats.norm.pdf( X[i], muvec[k,:],  np.sqrt( sigvec[k,:] ) )

        for k in range (0,K):
            postpr[i,k]=pivec[k]* stats.norm.pdf( X[i], muvec[k,:],  np.sqrt( sigvec[k,:] ) ) / px

 #M-step
 # Updating the parameters of the model based on the results obtained during the E step
    for k in range (0,K):
        pivec[k,:]=np.mean( postpr[:,k] )
        muvec[k,:]= np.sum( np.reshape(postpr[:,k],(N,p)) * X ) / np.sum(postpr[:,k]) # a*b = a.*b matlab
        sigvec[k,:]=np.sum( np.reshape(postpr[:,k],(N,p))* (X-muvec[k,:])**2 ) / np.sum(postpr[:,k])


print("muvec={}".format(muvec))
print("sigvec={}".format(sigvec))
print("pivec={}".format(pivec))
```

Figure 8: The implementation of the EM algorithm

Exercise:

1. **Below, the plot diaplays the 2 Gaussian pdfs involved in the mixture; add the mixture distribution to that plot.**
   the resulting pdf (Gaussian mixture) is the weighted sum of the two Gaussians pdfs yielded by the model. The weights are the elements of the mixing distribution contained in the **pivec** array as :

$$pdf = \sum_{k=1}^{2} \pi_k * \mathcal{N}(\mu, \sigma^2)$$

```
Xt=np.linspace(-2,8,1000)
g0=stats.norm.pdf(Xt,muvec[0],np.sqrt(sigvec[0]))
g1=stats.norm.pdf(Xt,muvec[1],np.sqrt(sigvec[1]))
g_mix=pivec[0][0]*g0+ pivec[1][0]*g1
# g=??
plt.plot(Xt,g0, label='g0')
plt.plot(Xt,g1,label='g1')
plt.plot(Xt,g_mix,label='g_mix')
# plt.plot(Xt,g,label='g')
plt.legend()
plt.xlabel('X');
```

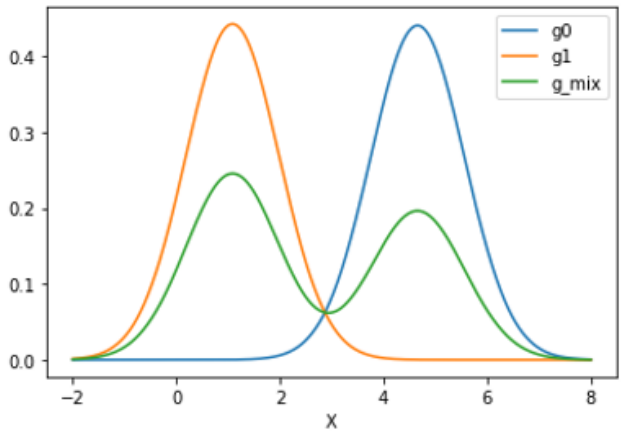

Figure 9: The mixture distribution

2. **compare the results obtained with Sklearn with the previously obtained results.Comments?**
   The results obtained by the scikit-learn implementation are almost identical to those obtained by manual implementation.
3. **Add the mixture pdf to the plot**
   The mixture pdf is as follows:

```
Xt=np.linspace(-2,8,1000)
g0sklearn=stats.norm.pdf(Xt,np.squeeze(est.means_[0]),
                  np.sqrt(np.squeeze(est.covariances_[0])))

g1sklearn=stats.norm.pdf(Xt,np.squeeze(est.means_[1]),
                  np.sqrt(np.squeeze(est.covariances_[1])))


gsklearn=est.weights_[0]*g0sklearn+est.weights_[1]*g1sklearn

plt.plot(Xt,g0sklearn, label='g0sklearn')
plt.plot(Xt,g1sklearn,label='g1sklearn')
plt.plot(Xt,gsklearn,label='gsklearn')
# plt.plot(Xt,gsklearn,label='gsklearn')
plt.legend()
plt.xlabel('X')
```

```
Text(0.5, 0, 'X')
```

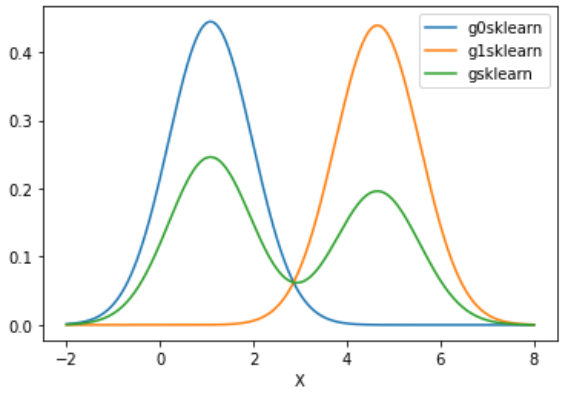

Figure 10: Sklearn mixture distribution

Exercise11:

1. EM and K-means are similar in the sense that they allow model refining of an iterative process to find the best congestion. However, the K-means algorithm differs in the method used for calculating the Euclidean distance while calculating the distance between each of two data items; and EM uses statistical methods. The K-Means algorithm yields the cluster whereas the EM algorithm yields the pdf of samples over the k clusters.

2. **interpret the responsibility functions resp0 and resp1**.
   These two functions represent the responsibilities of components for data items. These function (resp0 and resp1) are derived by the Bayes rule such that:

   $$resp_{ik} = P(data\ item\ i\ came\ from\ component\ \mathrm{k}|x_i) = \frac{\prod_{k=1} \pi_k * \mathcal{N}(x_i|\mu_k, \sigma_k^2}{\sum_{k'} \prod -k' \pi_k * \mathcal{N}(x_i|\mu_{k'}, \sigma_{k'}^2}$$

3. **Why is EM sometimes interpreted as a ”soft-Kmeans” algorithm? What would be the responsibility curves for Kmeans?**
   EM is interpreted as ”soft K-means” because unlike the K-Means algorithm which assign a point to only one cluster (**every point must be assigned to a cluster and if its assigned to one it is considered to be a member of that cluster with a certain probability =1**), it assign each data point a ”responsibility” value for each cluster where higher values correspond to a stronger cluster membership.
   the responsibility curves for Kmeans would be either 0 or 1 with no value in between: resp0 + resp 1= 1. In other words, the responsibility of class 0 would be 1 for every element in cluster0 and the responsibility of class1 would be 0 and vice-versa.

Conclusion:

Clustering is an unsupervised learning method that constitutes a cornerstone of an intelligent data analysis process. It is used for the exploration of inter-relationships among a collection of patterns, by organizing them into homogeneous clusters.
During this lab, we explored a wide range of clustering techniques like K-Means , kernel K-Means, EM and GMM algorithms through a variety of application.