# INF 552: Assignment 2

# Clustering

*Priyambada Jain (priyambj@usc.edu)*
*Sai Sree Kamineni (skaminen@usc.edu)*
*Varad Kulkarni (vdkulkar@usc.edu)*

## Part 1: Implementation

### 1) K-means:

- **Data Structures used: Numpy array and Pandas Data Frames**

- **Results:**

  ('Mean of Cluster1:', [3.083182557032258, 1.7762137380322585])
  ('Mean of Cluster2:', [-0.97476571808235302, -0.68419304117647095])
  ('Mean of Cluster3:', [5.6201657349705876, 5.0262263441764716])

Plotting of the results obtained using K-means Clustering algorithm:

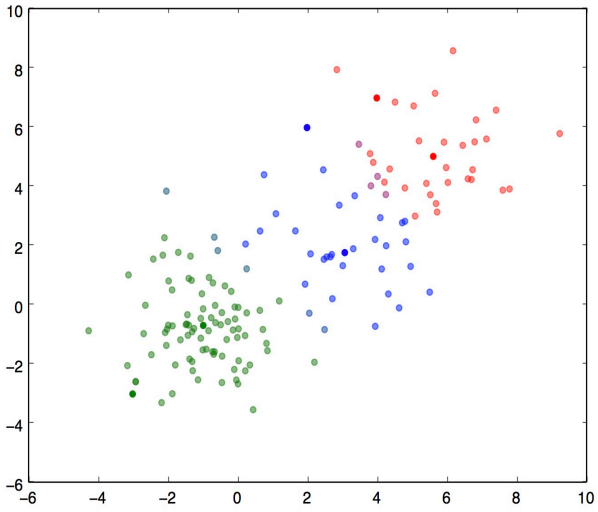


*Figure 1*

- **<u>Concept Involved</u>:**

  *k*-means clustering aims to partition the *n* observations into *k* (≤ *n*)(here which is 3 for our case) sets **S** = {$S_1$, $S_2$, …, $S_k$} so as to minimize the within-cluster sum of squares (WCSS).Formally, the objective is to find:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

where $\left\| x_i^{(j)} - c_j \right\|^2$ is a chosen distance measure between a data point $x_i^{(j)}$ and the cluster centre $c_j$, is an indicator of the distance of the *n* data points from their respective cluster centres.

  - The algorithm is composed of the following steps:

1. Assign three random points represent initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the K centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

- **<u>Challenges faced</u>**: Approach to assign the initial centroids should that we just random or we should process and analyze the distribution of data to do that.

**2) GMM:**

- **<u>Data Structures used</u>: Numpy array and python dictionary**

- **<u>Results</u>:**

  a) Mean of the Gaussians:

  ('Mean of Gaussian', 1, array ([ 1.54401653, 1.30214638]))
  ('Mean of Gaussian', 2, array ([ 1.42531497, 1.0985425]))
  ('Mean of Gaussian', 3, array ([ 1.13198182, 0.97108193]))

b) Amplitude of the Gaussians:
('Amplitude of Gaussian', 1, 0.3226041868117297)
('Amplitude of Gaussian', 2, 0.31984829965646111)
('Amplitude of Gaussian', 3, 0.35754751353180914)

c) Covariance Matrix for Gaussian distribution:
('Covariance of Gaussian:', 1) [[ 9.34568574 6.29367376]
[ 6.29367376 7.40844345]]

('Covariance of Gaussian:', 2) [[ 9.4185478 5.98321294]
[ 5.98321294 6.70085124]]

('Covariance of Gaussian:', 3) [[ 9.27494141 6.29134028]
[ 6.29134028 7.62548818]]

Plotting of the results obtained using GMM Clustering algorithm:

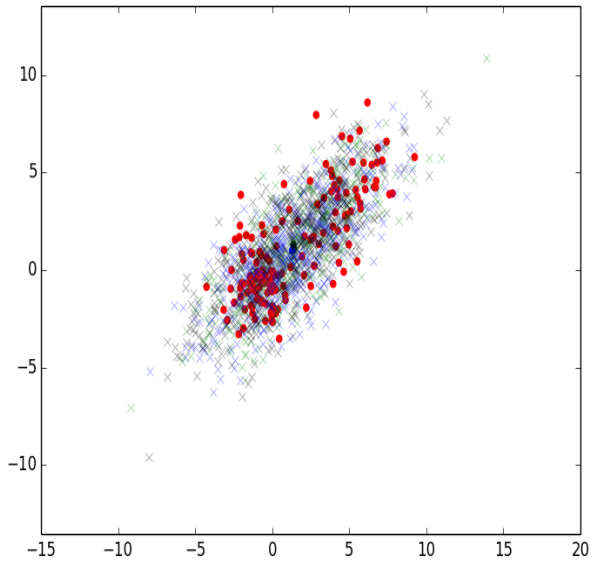


*Figure2*

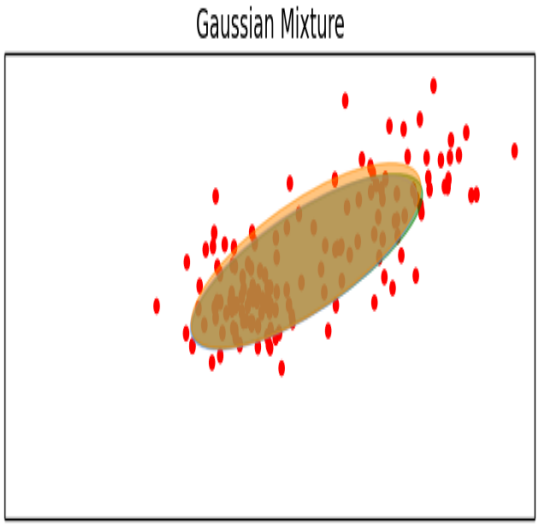


*Figure3*

- **Concept Involved:**

  For k dimensions, the Gaussian distribution of a vector x = (x1, x2, …,xd)T is defined by:

$$p(\mathbf{x}|\mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^d \det \Sigma_k}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu_k)^\top \Sigma_k^{-1}(\mathbf{x}-\mu_k)\right].$$

  where μ is the mean and Σ is the covariance matrix of the Gaussian.

  The probability of a data point given in a mixture of K Gaussians is:

$$p(\mathbf{x}|\Theta) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}_i|\mu_k, \Sigma_k),$$

The Expectation-Maximization Step for GMM:

a) E-step: for each point, estimate the probability/soft membership that each Gaussian generated it.
b) M-step: modify the parameters according to the soft membership to maximize the likelihood of the data (and the soft membership).
c) Convergence Step: **Log likelihood**

$$\ell(\Theta; X) = E_{\mathbf{x}\sim\hat{p}}[\log p(\mathbf{x}|\Theta)] = \frac{1}{n}\sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k p(\mathbf{x}_i|\mu_k, \Sigma_k)$$

- **Challenge Faced:**

1) *Initialization of soft membership* for each data point, we had to different choices one by randomly assigning the soft membership for each data point other by initializing the soft membership by using K-means algorithm.
2) *Convergence of the algorithm* we used the log likelihood in this algorithm.

**Comparison between K-means and Gaussian Mixture Model Clustering:**

| ***Context*** | ***Algorithm*** | ***K-means Clustering*** | ***GMM Clustering*** |
|---|---|---|---|
| Membership of Data points | | **We assume that the data point belongs to only one cluster at a time** | **The data point can have soft membership that means can belong to more than one cluster at a time.** |
| Minimization Function | | $(x-\mu k)^2$ | $(x-\mu k)^2/\sigma^2$ |

| | | |
|---|---|---|
| Measurement Consideration | **conventional Euclidean distance i.e. only distance** | **variance into consideration i.e. weighted distance** |
| Clusters Structure | **Spherical clusters only due to hard membership.** | **Works well with nonlinear geometrical structures as well.** |

K-means is a special case of Mixture of Gaussian, and Mixture of Gaussian is a special case of Expectation-Maximization.

It can be easily shown that in case of hard clustering Mixture of Gaussian is K-means.

## Part 2: Software Familiarization

We have used **Python sklearn** library to compare against our implementation for K-means and GMM (Gaussian Mixture Model) and following are the results obtained:

1) sklearn KMeans library function:

   Parameters used: Number of clusters = 3

   Cluster Centroids:

   ```
   [-0.97476572 -0.68419304]
    [ 3.08318256  1.77621374]
    [ 5.62016573  5.02622634]
   ```

   Plotting of the results obtained using sklearn KMeans Clustering algorithm:

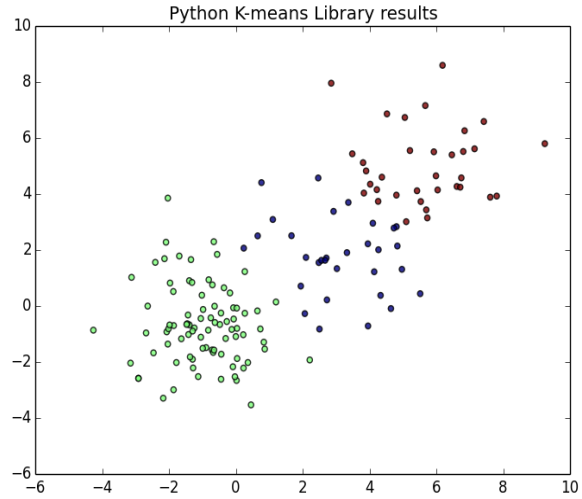


Figure 4

2) sklearn GaussianMixture library function:

Parameters Used:

- **n_components** : int, set to 3.
- **covariance_type** : set to full : each component has its own general covariance matrix.
- **init_params** : 'random' for initialing the soft membership for each data point

Means :

[ 1.147001  0.96959838]
[ 1.52213969  1.20056878]
[ 1.41132411  1.18851795]

**Here we analyzed that the initializing plays a pivotal role in creating the gmm model as explained earlier as well if we initialize the data point's membership using Kmeans the means obtained are pretty close to the ones in pure K-means as it becomes a special case in gmm where every data point is having hard membership belonging to only and only one cluster at a time.**

## Part 3: Applications

**a) K means Applications:**

- **Pricing Segmentation**: E-retailers or e-commerce companies have taken the retail industry by storm. They are offering luring offers and discounts. They aim to move from discount led to convenience or differentiation led offering over a period in time. Some of them have been forced to start the journey.

  One of the large retailer wanted to segment the customers based on customer spend patterns and understand price sensitivity of the customers. Some of the segmentation variables considered are – total spend, value of discounts, % discounts across transactions, number of items bought on discounts and used k means clustering to find the discount orientation of the customers.

- **Loyalty Segmentation**: Every organization wants their customers to be loyal but some time defining loyalty could be a tricky. Some of the common expectations are frequent spend, higher spend, higher ticket size and diversified spend. A large Indian retailer was looking to understand customers from these 4 dimensions and wanted to develop segments for building a focus engagement strategy for each of the target segment.

- **Healthcare Fraud Detection Segmentation**: For an Indian healthcare provider, aim was to find claims which could be fraudulent. K Means clustering method was used for anomaly detection and claim routing to right claim adjudicator. Some of the K Means clustering dimensions or variables used were
  - Dieses category
  - Claim Type
  - Age Group

- **Cluster analysis**
  In cluster analysis, the *k*-means algorithm can be used to partition the input data set into *k* partitions (clusters).
  However, the pure *k*-means algorithm is not very flexible, and as such is of limited use (except for when vector quantization as above is actually the desired use case!). In particular, the parameter *k* is known to be hard to choose (as discussed above) when not given by external constraints. Another limitation of the algorithm is that it cannot be used with arbitrary distance functions or on non-numerical data. For these use cases, many other algorithms have been developed since.

**b) GMM Applications:**

- GMMs have been used recently for **feature extraction** from speech data for use in speech recognition systems. They have also been used extensively in object tracking of multiple objects, where the number of mixture components and their means predict object locations at each frame in a video sequence. The EM algorithm is used to update the component means over time as the video frames update, allowing object tracking.
- Audio analysis for surveillance applications adaptively learns a Gaussian mixture model (GMM) to model the background sounds and updates the model incrementally as new audio data arrives

**References:**


1) https://en.wikipedia.org/wiki/K-means_clustering
2) https://www.mathworks.com/help/stats/clustering-using-gaussian-mixture-models.html?requestedDomain=www.mathworks.com
3) https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html
4) https://brilliant.org/wiki/gaussian-mixture-model/
5) https://page-one.live.cf.public.springer.com/pdf/preview/10.1007/11494683_2
6) https://brilliant.org/wiki/gaussian-mixture-model/#citation-5
7) https://metacademy.org/graphs/concepts/gaussian_mixtures_vs_k_means
8) https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4433949/
9) http://www.rmki.kfki.hu/~banmi/elte/bishop_em.pdf
10) http://statweb.stanford.edu/~tibs/stat315a/LECTURES/em.pdf


**Appendix:**

**Kmeans.py:** Implementation for K-means. Command to run: *python Kmeans.py*

**gmm.py:** Implementation for GMM. Command to run*: python gmm.py*

**Plots:** This folder contains the plots obtained in part 1 and part 2.

**Cluster.txt :**File for input data points