CS5691: Pattern Recognition and Machine Learning
Assignment #2

Topics: LDA, GMM, DBSCAN **Deadline:** 28 April 2023, 11:55 PM

Teammate 1: (Bhavadeep) (50% of contribution) **Roll number:** CS20B015
Teammate 2: (Vaastav) (50% of contribution) **Roll number:** CS20B060

- **For any doubts regarding questions 1 and 2**, you can mail cs22s013@smail.iitm.ac.in and cs21s043@smail.iitm.ac.in
- **For any doubts regarding question 3**, you can mail cs21d015@smail.iitm.ac.in and cs22s015@smail.iitm.ac.in
- Please refer to the **Additional Resources** tab on the Course webpage for basic programming instructions.
- This assignment has to be completed in teams of 2. Collaborations outside the team are strictly prohibited.
- Any kind of plagiarism will be dealt with severely. These include copying text or code from any online sources. These will lead to disciplinary actions according to institute guidelines. Acknowledge any and every resource used.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- You should submit a zip file titled **'rollnumber1_rollnumber2.zip'** on Moodle where rollnumber1 and rollnumber2 are your institute roll numbers. Your assignment will **NOT** be graded if it does not contain all of the following:
    1. Type your solutions in the provided LaTeX template file and title this file as **'Report.pdf'**. **State your respective contributions in terms of percentage at the beginning of the report clearly.** Also, embed the result figures in your LaTeX solutions.
    2. Clearly name your source code for all the programs in **individual Google Colab files**. Please submit your code only as Google Colab file (.ipynb format). Also, embed the result figures in your Colab code files.
- We highly recommend using `Python 3.6+` and standard libraries like `NumPy, Matplotlib, Pandas, Seaborn`. Please use `Python 3.6+` as the only standard programming language to code your assignments. Please note: the TAs will only be able to assist you with doubts related to Python.
- You are expected to code all algorithms from scratch. **You cannot use standard inbuilt libraries for algorithms until and unless asked explicitly**.
- **Any graph that you plot is unacceptable for grading unless it labels the x-axis and y-axis clearly.**

- Please note that the TAs will **only** clarify doubts regarding problem statements. The TAs won't discuss any prospective solution or verify your solution or give hints.
- Please refer to the CS5691 PRML course handout for the late penalty instruction guidelines.

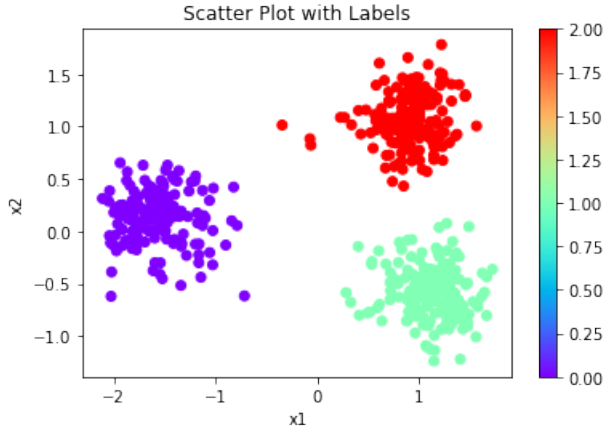
---

1. **[Linear Discriminant Analysis (LDA), Principal Component Analysis (PCA)**] You will implement dimensionality reduction techniques (LDA, PCA) as part of this question for the dataset1 provided here.

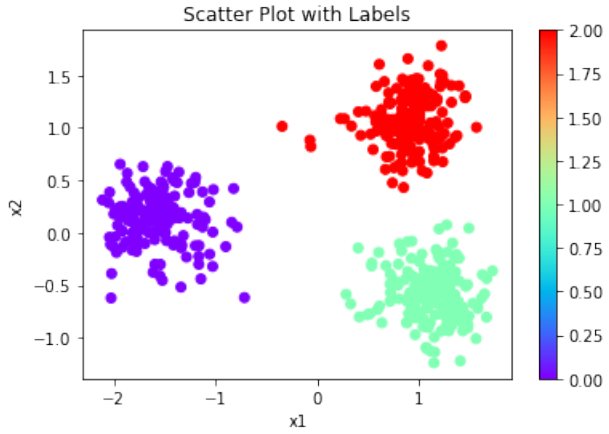
   Note that you have to implement **LDA from scratch** without using any predefined libraries (i.e. sklearn, scipy) . However, you can use **predefined libraries to implement PCA.**

   (a) (2 marks) Use Linear Discriminant analysis (LDA) to convert dataset1 into the two-dimensional dataset and then visualize the obtained dataset. Also, perform an analysis on how results will change if we perform normalization (i.e., zero mean, unit variance normalization) on the initial dataset before applying LDA.

Solution:
PART A) There was a dataset of 65 columns and the last column was the labels. Now in the dataset, there were some columns with every value being the same. So in the preprocessing step, we can reduce the data directly. so now we are left with about 57 columns. For LDA (linear discriminant analysis)



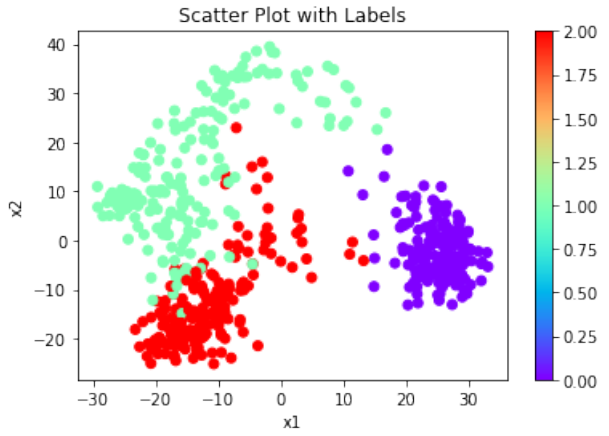
Here x1 and x2 are columns after reduction using LDA. Now that we have used numpy to calculate eigen values, we get vector forms/ complex forms. And we take top 2 values where the j part is zero and then compute the dataset.
After Normalizing
After we normalize and use all the steps mentioned above we get the following graph.



We don't see any difference because:
As we can see, the only difference between the charts is the scaling. This demonstrates how normalisation has no effect on the LDA plots and how the data is projected onto the same 2-dimensional plane in both the normalised and unnormalized cases. Consequently, the plots are almost identical.

(b) (1.5 marks) Use PCA to convert dataset1 into two-dimensional data and then visualize the obtained dataset. Now, compare and contrast the visualizations of the final datasets obtained using LDA and PCA.

Solution:
When I used PCA (principal component analysis), Which is also a dimensionality reduction technique, We got the graph shown below.



Here x1 and x2 are the new columns we get after the reduction using PCA.
PCA and LDA :
Unlike LDA PCA need to be normalized first.
PCA is a statistical technique that is used to reduce the dimensionality of the dataset

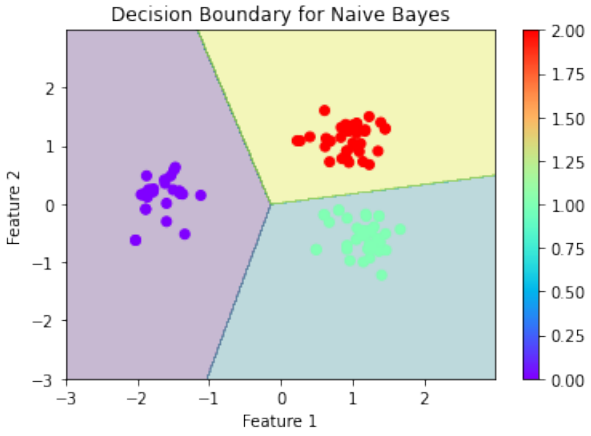
by identifying most important features or variables that contribute to the data variation. On the other hand we see that LDA is a probabilistic model where it tries to find the latent topics in the data by modeling the underlying probability distribution of the data.
We see that the main goal of PCA is dimensionality reduction and LDA tries to find underlying topics.
While doing LDA we made separate categories and then tried to analyze them. So we kind of searched for some inbuilt dependencies for the labels and then found out the most effective data from it. In PCA it does not happen so we see some difference in the graph we obtained above.
The main difference is PCA is unsupervised and LDA is used as a main cause of Supervised learning. So PCA does not take initial label assignments into consideration and then analyze data, unlike LDA where it considers labels.

(c) (1.5 marks) Randomly shuffle and split the obtained dataset from part (a) into a training set (80%) and testing set (20%). Now build the Bayes classifier using the training set and report the following:

- Accuracy on both train and test data.
- Plot of the test data along with your classification boundary.
- confusion matrices on both train and test data.

Solution:
As we implemented a naive Bayes algorithm in the assignment, we used the same algorithm here. Now that we shuffled and slipt data into 80:20 ratio, below are the results.
Accuracy on train data:
Accuracy: 1.0
Accuracy on test data:
Accuracy: 1.0

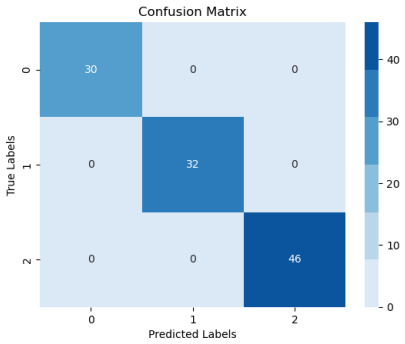
Plot of test data with decision boundary is as follows:



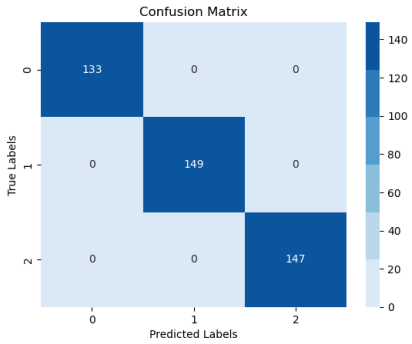
. Labels are mentioned as scale.

Confusion Matrices

For Test data



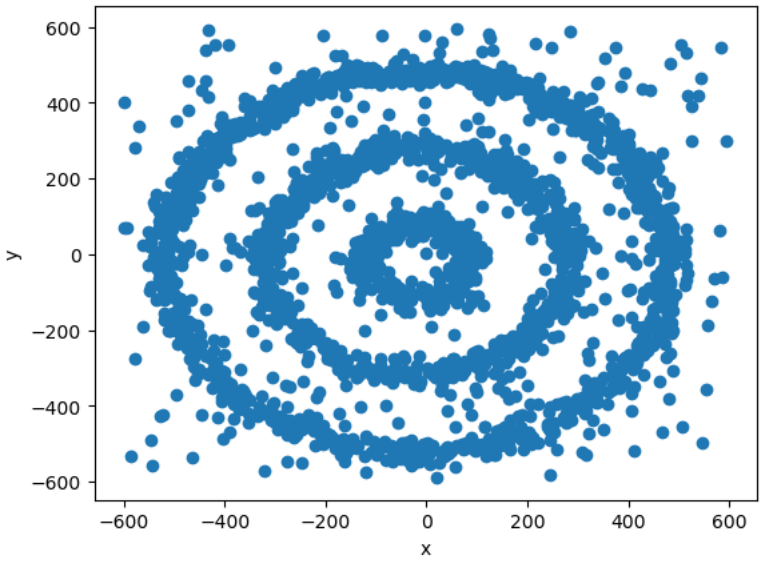
For train data



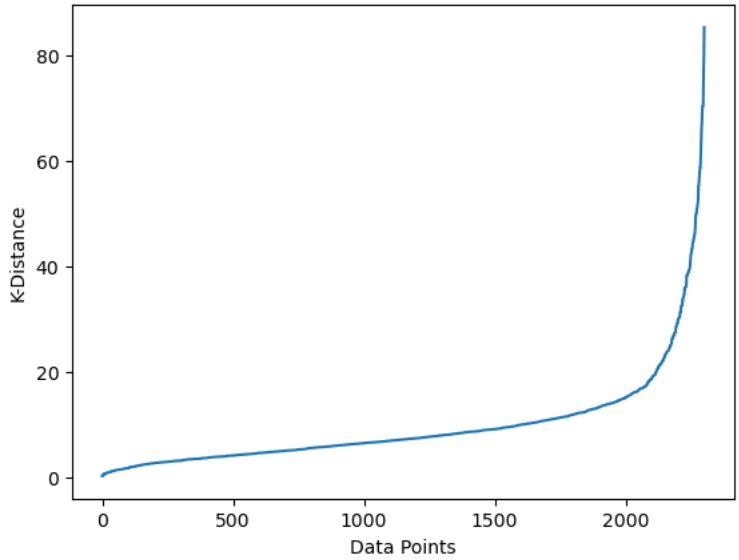
2. [**DBSCAN**] In this Question, you are supposed to implement **DBSCAN algorithm from scratch** on dataset2 provided here and dataset3 provided here. You also need to compare and contrast your observations from above with K-Means applied on both datasets. **However, you can use predefined libraries to implement K-means.**

   (a) (1 mark) Visualize the data in dataset2. Then, find a suitable **range of values for epsilon** (a hyperparameter in DBSCAN algorithm) by using the 'Elbow Curve' of Datapoints plotted between K-Distance vs Epsilon. For simplicity, take only integer values for epsilon. **You can use predefined libraries to implement K-distance.**

Solution:
Visualisation of the data in dataset2 is as given below :-



The Elbow curve of the datapoints plotted with K-Distance is as follows :-



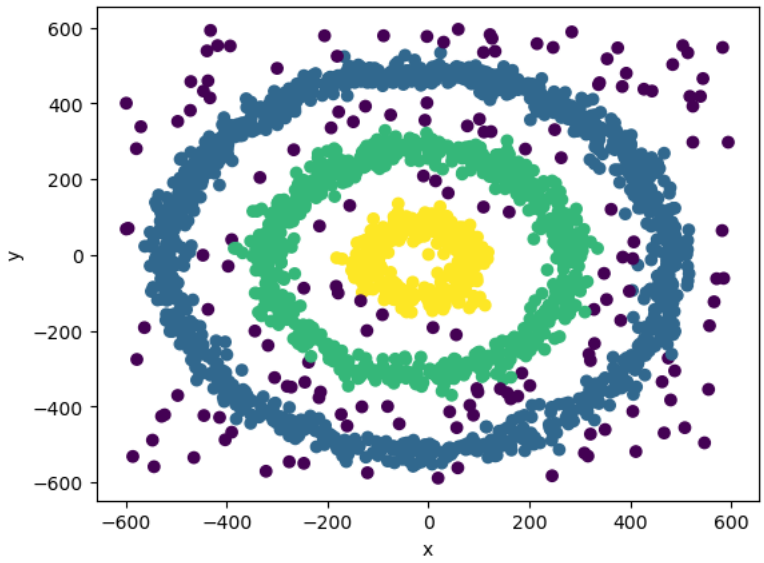
We can clearly see from the elbow curve that the suitable range of values for Epsilon are somewhere in between 20 to 40.

(b) (2 marks) Implement DBSCAN with the above suitable range of values of epsilon and detect the optimal value of epsilon, which gives the best clustering visually on the dataset. Show a visualization of the clusters formed for the best value of epsilon.

Solution:
After implementing DBSCAN for the above range of integer Epsilon values between

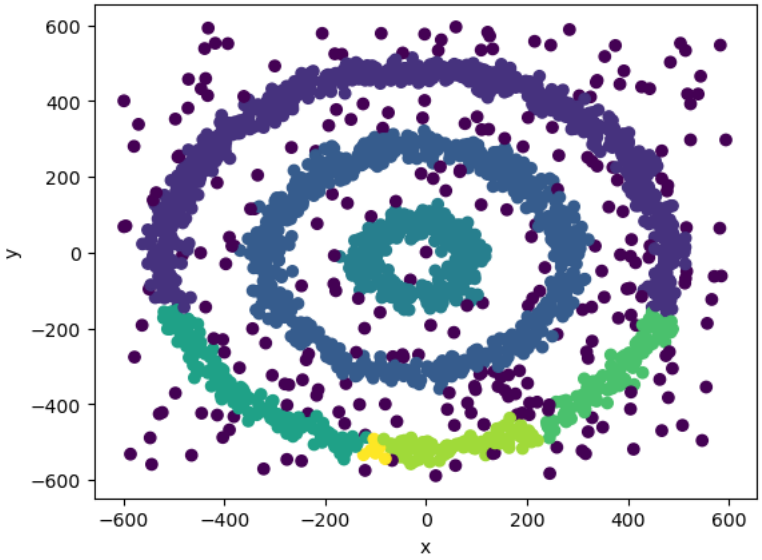
20 to 40 in and with minpts ranging from 4 to 10 , visually we can observe that best clustering for the dataset 2 is obtained at the Epsilon value of 33 with minpts = 6.

The visualization of the clusters formed for the best value of epsilon that is 33 and minpts=6 is as shown below



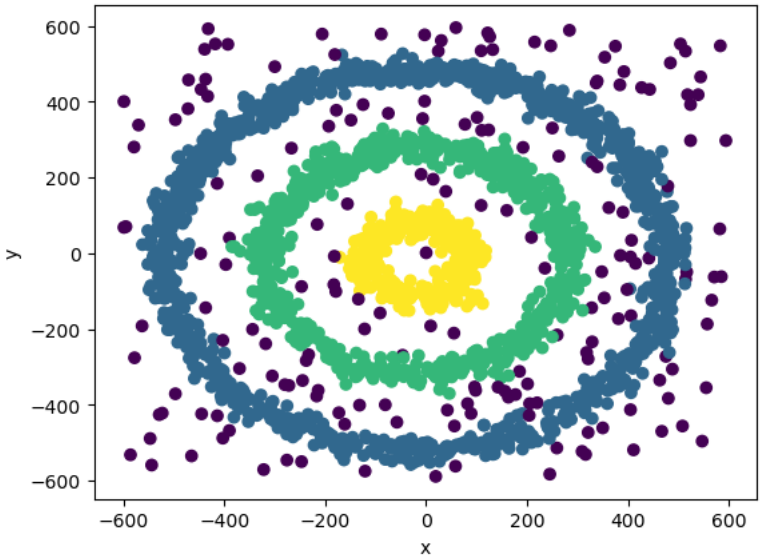
To give emphasis over why we choose eps=33 as optimal we will view few clustering visualisations for the nearest eps values.

For eps=25 :-



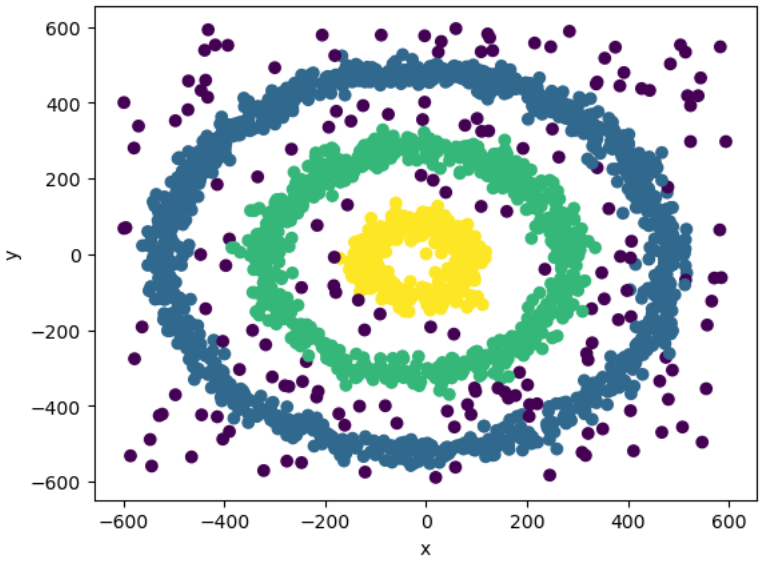
Here we can observe clearly that though outer most ring seems to be a single cluster the algorithm splits them into different clusters.

For eps=30 :-



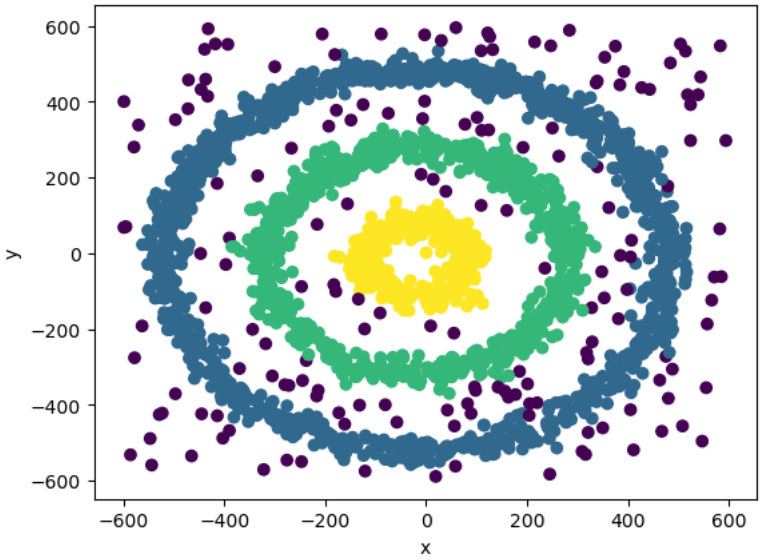
Here we can observe that there is a point inside the inner most circle that doesnt seem like an outlier but is labeled.

For eps=31 :-



The observations from the above visualisation of above clustering we can see that in the outer most circle near (200,-400) region we have few points that would likely belong to the same cluster as the outermost circle rather than being an outlier or a noise point.

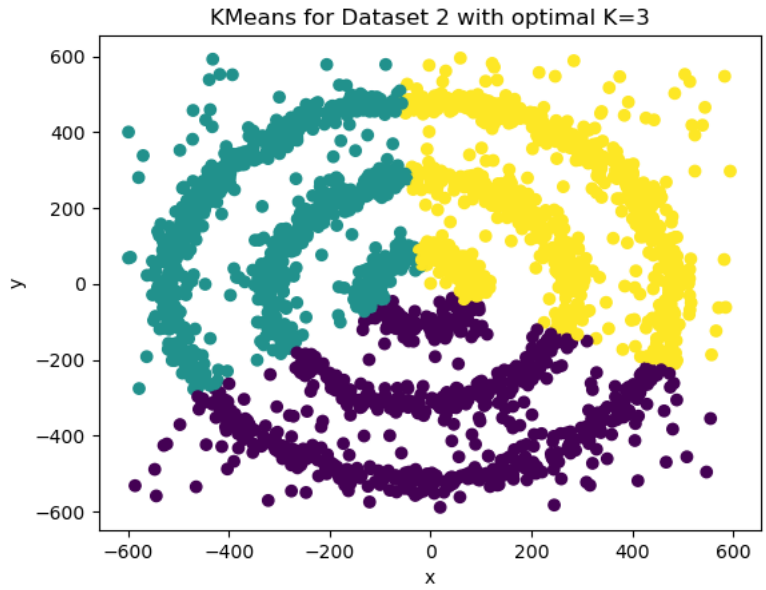
For eps=32 :-



Same issue as in the above clustering continues here as well which is further resolved in the next integer eps value that is eps=33. Thus we chose 33 as the optimal eps value.

(c) (1.5 marks) Implement K-Means and use it on dataset2 with value of K (number of clusters) set to the optimum number of clusters that you get from (b) above. Suggest various techniques to improve the clustering by KMeans in this case.

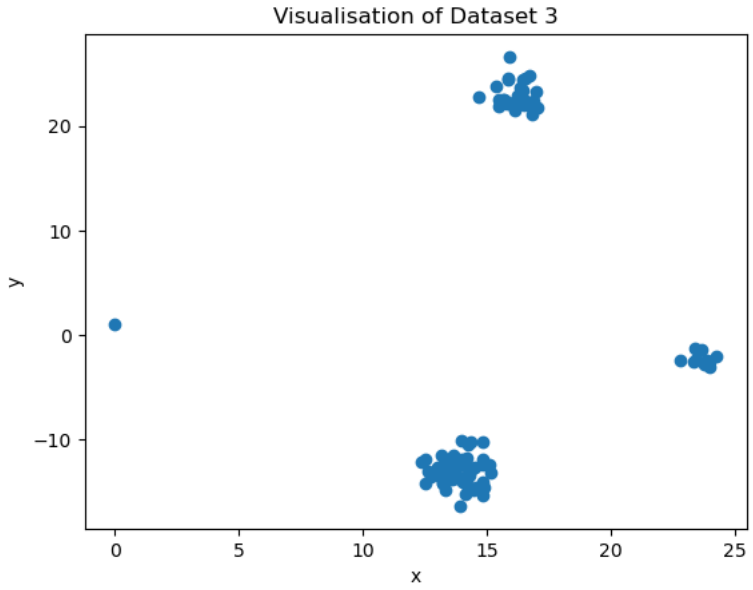
Solution:
The optimal number of clusters obtained from the above model is 3. Now the visualisation of the K-Means(K=3) algorithm applied on the dataset 2 is as follows:-



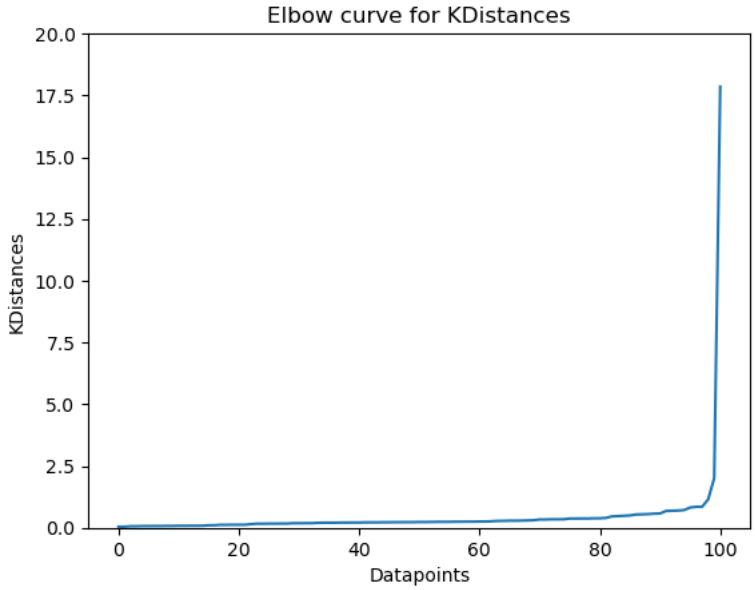
Techniques to improve KMeans :-
1)In this case the clusters are spherical and isotropic so use polar coordinates: Instead of using the raw x and y coordinates, it may be beneficial to transform the data into polar coordinates, with radius and angle. This can help to reduce the effect of the circular shape of the dataset.
2)Use a different distance metric: The Euclidean distance metric used in KMeans may not be the best choice for the concentric circle dataset. Instead, consider using a different distance metric, such as the cosine distance or the Manhattan distance.
3)Feature engineering: Consider creating new features that better capture the circular nature of the data. For example, one could create a feature that represents the distance of each point from the center of the circles.

(d) (1.5 marks) Show a visualization of the data in dataset3. Use your implementation of DBSCAN with `minPts=15` on dataset3. Plot 'Elbow curve' to get an optimal range of values for `eps`. Detect the optimal value of epsilon which gives the best clustering visually on the dataset. Show a visualization of the clusters formed for the best value of epsilon.

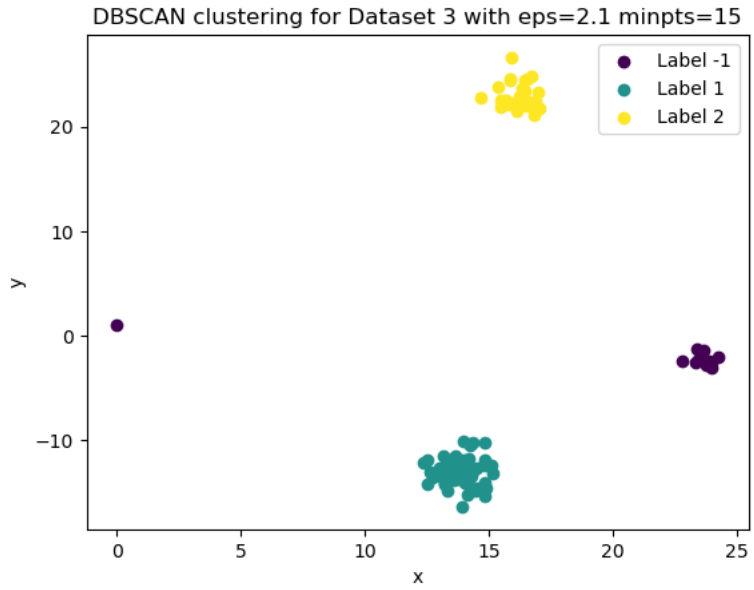
Solution:
Visualisation of dataset 3 :-



Elbow curve plot using KDistances for determining optimal range of values for eps :-

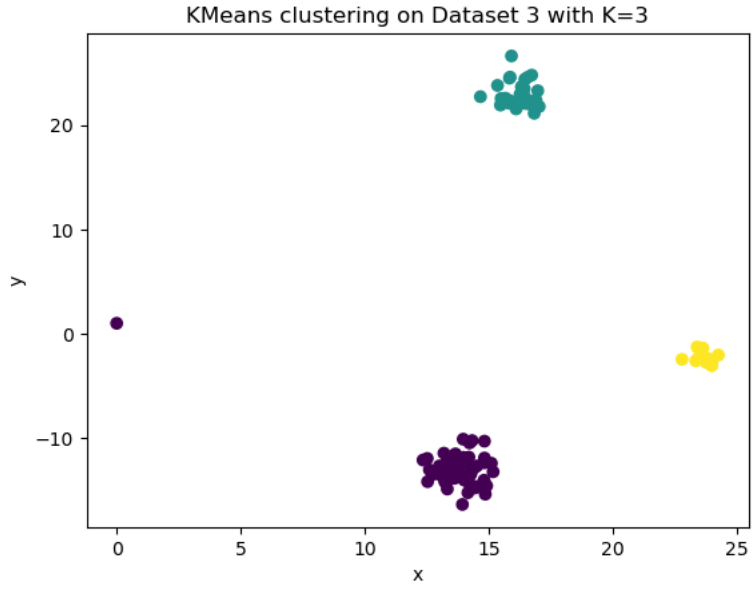


The optimal range from the above curve is between 1 and 3.
After clustering the dataset with the eps values in the range the optimal eps value is determined to be 2.1 when minpts=15.
Visualisation of the cluster for the best value of epsilon(refer the collab file for other clustering visualisations as well):-



(e) (1 mark) Now perform KMeans with K=3. Write your observations for obtained results in (d) and (e). Did we give you bad initialization values?

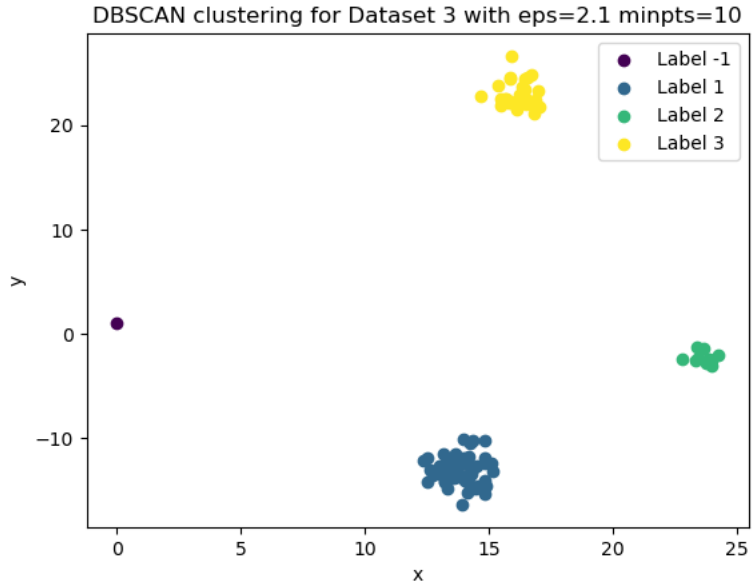
Solution:
Clustering Visualisation for KMeans with K=3 for the dataset3 :-



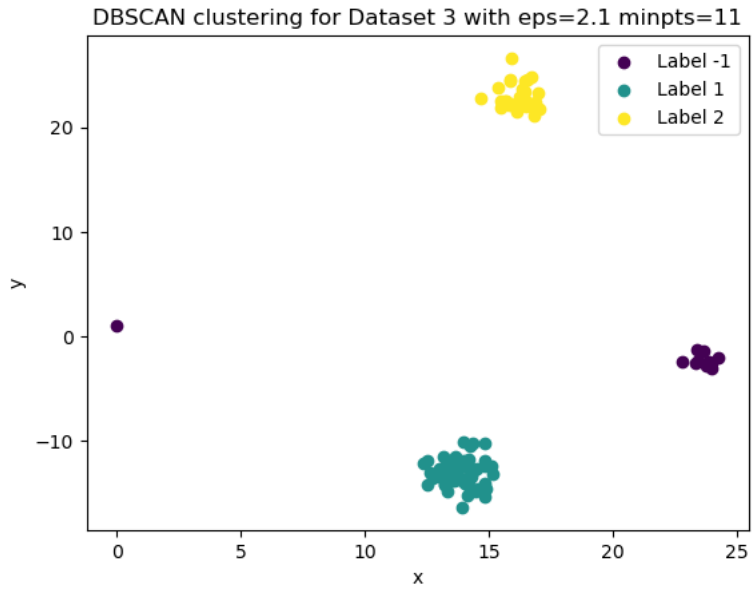
Observations:-
1) KMeans gives us a better clustering in this case than the DBSCAN with the above intialisation. Because in dataset3 we can observe a clear cluster formed to the right-

most section of the plot but in the DBSCAN implemented in d we got that to be considered as noise rather a cluster, which is not desired.
2)Unlike DBSCAN, KMeans was not able to identify the outlier/noise point that is present in the dataset. KMeans included it in the closest cluster.
3) When we had the concentric circular cluster problem we faced issues with KMeans not giving desired results as it split the same circular cluster segment into several clusters. Whereas in case of a sparse dataset like the dataset3 we have KMeans relatively giving better results under the given intialisations for dbscan.

Remarks :-
Yes the given intialization values for dbscan, i.e; minpts=15 is bad. Because due to the high minpts value even though there is a dense cluster to the right of the plot it is considered as noise because it just doesnt have 15 points in it. And thus it is clustering that particular set of datapoints as noise, which doesnt seem justifiable. Whereas if we consider reducing the minpts to a less value say 2 we can observe the dbscan algorithm to detect that cluster and label it rather than intrepreting it as noise. Until we consider the minpts=10 we will cluster that particular set of datapoints. But as there are only fewer points there for all minpts values above 10 it is considered as noise. The issue is though the eps neighbourhood for the obtained optimal eps cover all the points inside a common label as they dont satisfy minpts condition they are all labeled to -1(noise).
DBSCAN Clustering for minpts=10:



DBSCAN Clustering for minpts=11:



The above clustering models explain the remarks made and also refer to the collab files for more clustering plots.

(f) (1 mark) Based on all your learnings from this question, state the relative pros and cons of KMeans vs DBSCAN.

Solution:

KMeans
Pros:
1)Simple to understand and implement: KMeans is a relatively simple algorithm that is easy to understand and implement. It is based on the principle of minimizing the sum of squared distances between each point and its assigned centroid, making it a popular choice for clustering tasks.
2)Can be computationally efficient for small datasets: For small datasets, KMeans can be computationally efficient and scale well.
3)Performs well when the clusters are clearly separated and distinct: KMeans performs best when the clusters are well separated and distinct from each other. In this case, the algorithm can accurately group the data points into their respective clusters.
4)Generally works well with high-dimensional data: KMeans can work well with high-dimensional data, making it a popular choice for clustering tasks involving large datasets.

Cons:
1)The number of clusters must be specified in advance: One of the biggest limitations of KMeans is that the number of clusters must be specified in advance. This can be difficult to determine, especially when working with complex datasets.
2)The algorithm can be sensitive to the initial placement of centroids: KMeans is sensitive to the initial placement of centroids, which can lead to different results when the algorithm is run multiple times with different initializations.
3)Can be sensitive to outliers and noise: KMeans can be sensitive to outliers and noise, which can result in inaccurate clustering results.
4)Cannot handle irregular shapes and sizes of clusters: KMeans cannot handle clusters that have irregular shapes and sizes, as it assumes that all clusters are of equal size and shape.

DBSCAN
Pros:
1)Can handle arbitrary shapes and sizes of clusters: DBSCAN can handle arbitrary shapes and sizes of clusters, making it a more versatile algorithm than KMeans.
2)Does not require the number of clusters to be specified in advance: Unlike KMeans, DBSCAN does not require the number of clusters to be specified in advance, making it a more flexible algorithm for clustering tasks.
3)Robust to outliers and noise: DBSCAN is robust to outliers and noise, making it suitable for datasets with noise and outliers.
4)Performs well on large datasets: DBSCAN can perform well on large datasets, making it a popular choice for clustering tasks involving large datasets.

Cons:
1)Can be computationally expensive for large datasets: For very large datasets, DBSCAN can be computationally expensive and may require the use of approximation

methods.
2)Can struggle with high-dimensional data: DBSCAN can struggle with high-dimensional data, as the distance metric becomes less meaningful in high-dimensional spaces.
3)The performance of the algorithm is sensitive to the choice of distance metric and parameters: The performance of DBSCAN is sensitive to the choice of distance metric and parameters, which can make it difficult to choose the optimal values for these parameters.
4)It may not be suitable for datasets with varying densities: DBSCAN is not well-suited for datasets with varying densities, as it can result in clusters

3. **[GMM]** In this question, you are supposed to implement the Expectation-Maximization algorithm for Gaussian mixture models on the given dataset4. The data can be found here.

   (a) (3 marks) Implement EM for GMM and plot the log-likelihood as a function of iterations.

Solution: A Gaussian mixture model (GMM) is a probabilistic model that assumes that the data is generated by a mixture of several Gaussian distributions. In other words, it assumes that there are several subpopulations within the data, and each subpopulation is represented by a Gaussian distribution.
It is an expectation maximization problem. Here are the formulas used in the execution:

Univariate Gaussian Distribution

$$G(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

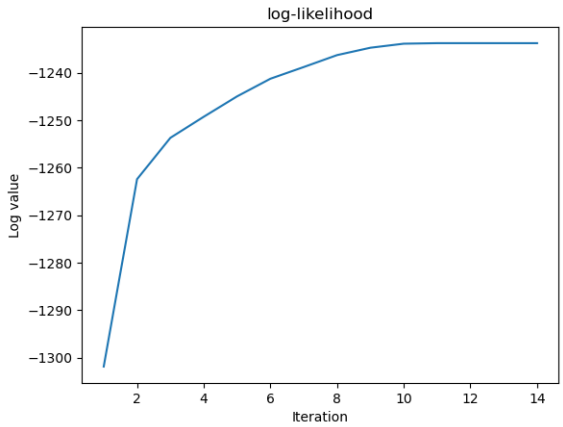
mean variance

Multi-Variate Gaussian Distribution

$$\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi|\Sigma|)^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}$$

mean covariance

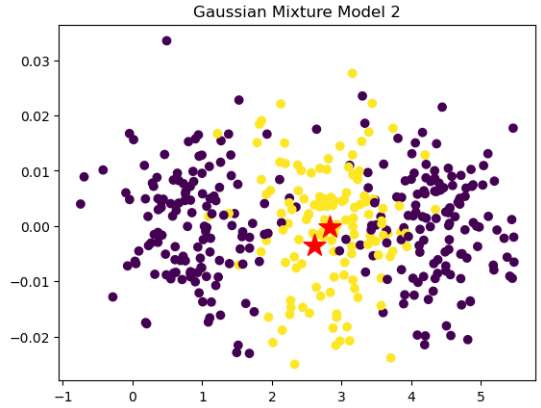
Implemented code is submitted in the folder.
LogLikelihood Vs Iterations :



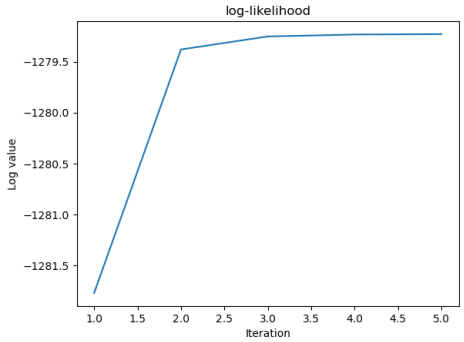
We can see that the likelihood graph is kind of going to saturation and as we go we see very minimal changes.

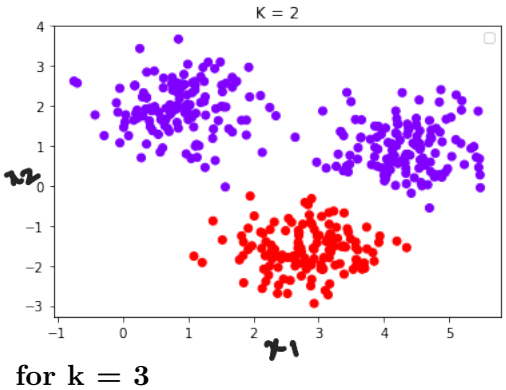
(b) (2 marks) Run EM for different numbers of Gaussians (k)(Try 2,3,4,5,6). Plot figures that can help in visualization and also log-likelihood as a function of iteration for different values of k. Report the observations.

Solution:
for k = 2
— GMM



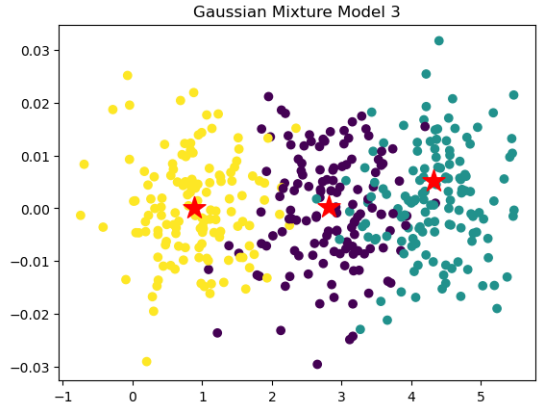
LogLikelihood

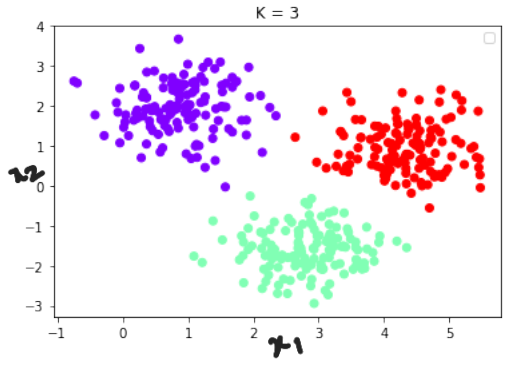




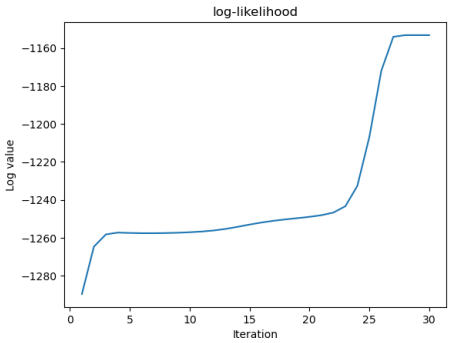
for k = 3

— GMM



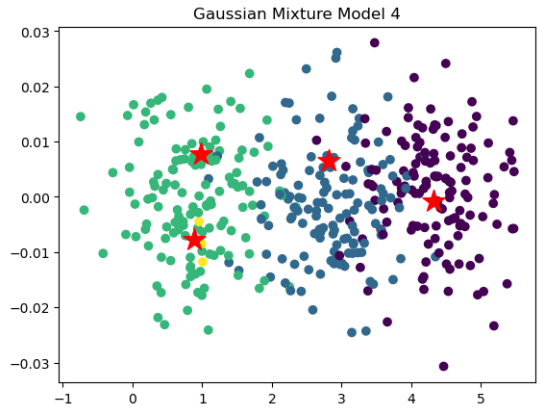


LogLikelihood

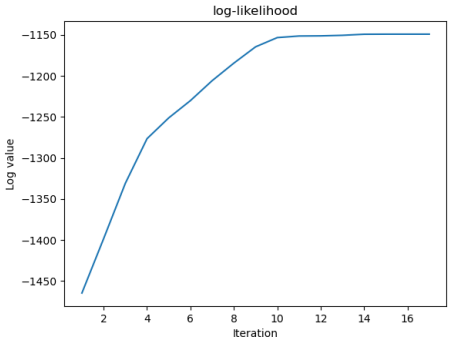


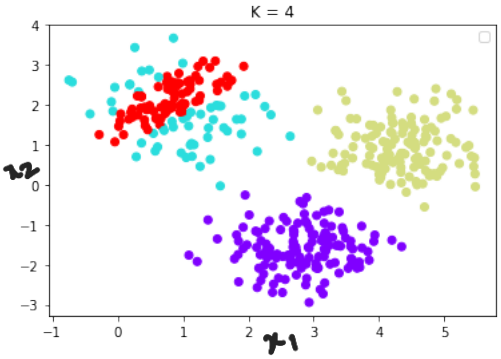
for k = 4

— GMM



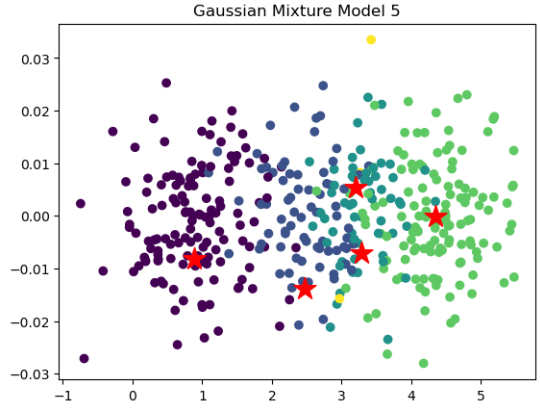
LogLikelihood



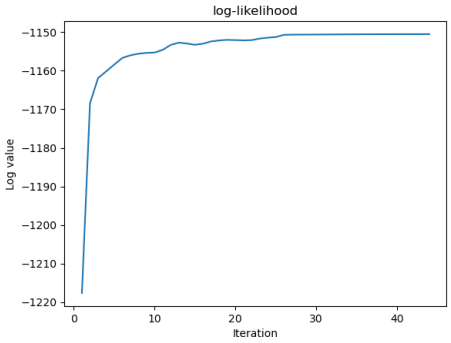


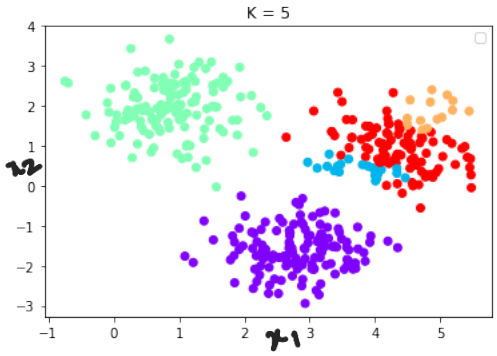
for k = 5

— GMM



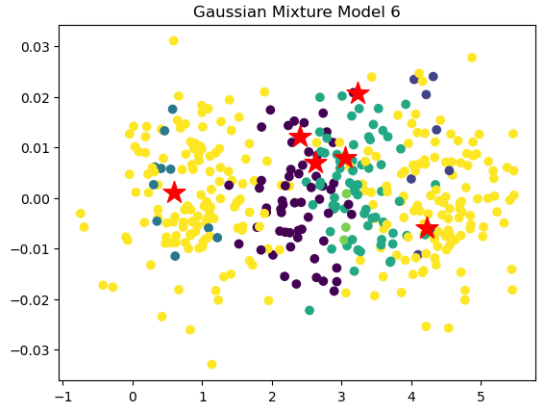
LogLikelihood



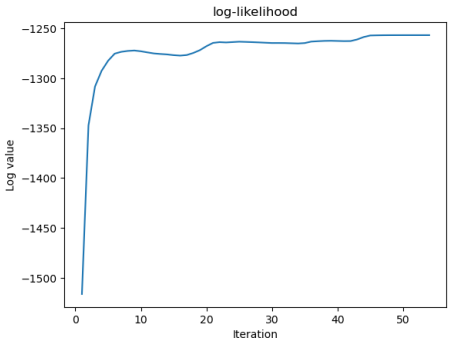


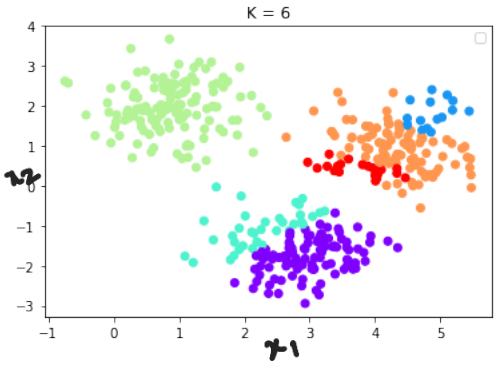
for k = 6

— GMM



LogLikelihood





Observations:

Here in each Gaussian mixture model, every color corresponds to one unique cluster in that particular graph.

When we initialize we take some random points from the given data and declare that those are initial means of the clusters. So we can't guarantee the same cluster every time.

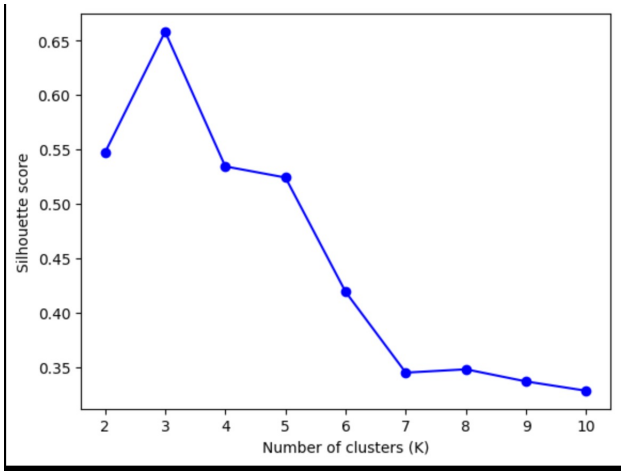
Every time the mean gets initialized randomly, sometimes we may reach good clusters and sometimes we may end up getting bad clusters.

Now that we have some measures like log-likelihood, silhouette score, and all we can

conclude to one optimal cluster according to their math and properties.
.
But initially, when we plot we could see that the there are 3 clusters as we are also trained to see some differences in the space defined.

(c) (2 marks) Find the optimal k. There are several metrics like Silhouette score, Distance between GMMs, and Bayesian information criterion (BIC), or even you can use log-likelihood from the last question to infer. Give a clear explanation for your decision.
Note: **You can use third-party libraries - sklearn or any other only in this subsection.**

Solution: We have used the Silhouette score to find the optimal K.
We have used different values of k and plot it in a graph as shown below.



We can see that we get optimal (which is defined as the max silhouette score) value of K is **3**.
The silhouette score is a measure of how well-separated the clusters are in a clustering solution, taking into account both the distance between points within a cluster and the distance between points in different clusters.

We first calculate the average distance between a point and every other point in the same cluster, which we label as a(i), and then use that value to get the silhouette score for that particular clustering solution. We can determine how well the point fits into its own cluster using this. The average distance between a point and every other point in the closest neighboring cluster is then calculated; this cluster is known as b(i), and it is the one with the least average distance to the point that is not also the cluster of the point. This allows us to gauge how well the point fits into the cluster next to it.
Finally, we compute the silhouette score for the clustering solution as the average silhouette score across all points:

s(i) = (b(i) - a(i)) / max(a(i), b(i))

.

When clusters are clearly defined and well-separated, the overall silhouette score for a clustering solution is greater; when clusters are overlapping or poorly separated, the score is lower. Therefore, we can assess the effectiveness of various clustering solutions using the silhouette score and select the number of clusters that best separates the data points.

Explanation for the decision

Now getting a maximum separation is a good measure as we are trying to form clusters that are kind of differentiable so we can see the max separation measure using silhouette score.

From this we can take k = 3 and from plot we can see that there are 3 differentiated clusters from our eye.