**Gaussian Mixture Models (GMM) vs. K-Means**

**1.Introduction**Clustering is one of the core concepts in unsupervised machine learning, and it allows us to group similar data points into clusters based on some features. It is used very widely in many areas such as customer segmentation, image classification, and pattern finding. Two of the most widely used clustering algorithms are K-Means and Gaussian Mixture Models (GMM). Both these models are designed to cluster data, but they differ in the way they decide and assign data points to clusters.

K-Means is a centroid-based approach to cluster data into K groups by assigning each point to the closest centroid. GMM is a probabilistic model which represents data as a mixture of several Gaussian distributions. Therefore, GMM can represent more complex shapes in data distributions. Here, we will explain the differences between K-Means and GMM and show their application to the Iris dataset. We will also compare their performance using measurements like the Silhouette Score and plots

This tutorial is going to cover:

• A brief explanation of K-Means and Gaussian Mixture Models.  
• A step-by-step implementation with Python and the Iris dataset.  
• Plots to demonstrate that both algorithms create clusters on the data.  
• A comparison of the clustering ability of both approaches. **2. Synonyms and Definitions**

**Synonyms:**

• GMM: Gaussian Mixture Model, Mixture Model, Gaussian Mixture Density  
• K-Means: Centroid-based clustering, partitioning method

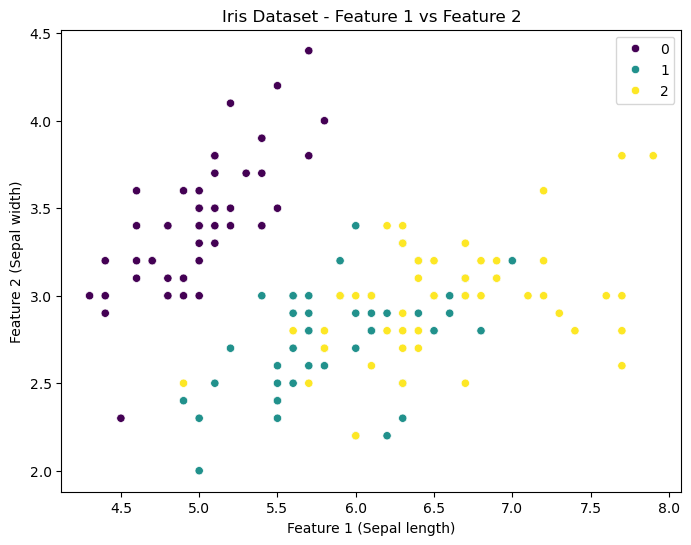
**Definition of GMM:**A **Gaussian Mixture Model (GMM**) is a statistical model which assumes that the data are samples from a mixture of several Gaussians with unknown parameters. Each individual Gaussian distribution has a mean, variance, and the weight it contributes to the total mixture.

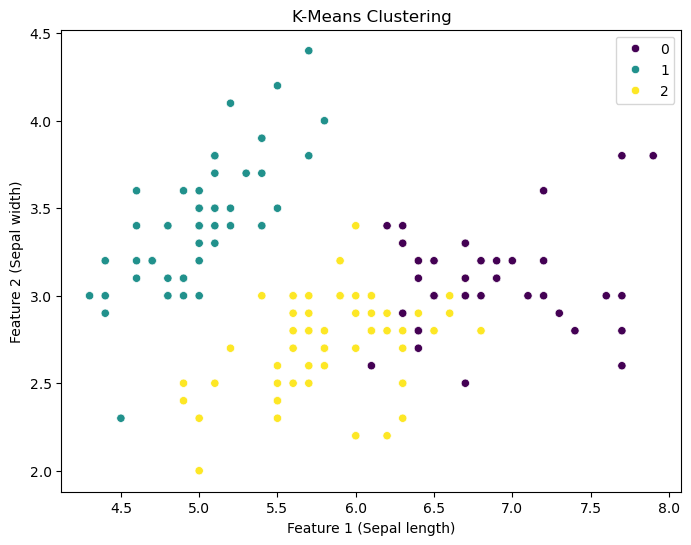
GMM is widely used in clustering, density estimation, and anomaly detection. It is a more general clustering method than K-Means, since it can handle elliptical clusters and provide probabilities for the data points, rather than assigning data points to a cluster with certainty.

In GMM, the EM algorithm performs parameter estimation. The EM algorithm runs in cycles of two steps: computing the Gaussian distribution parameters (Maximization Step) and assigning a data point to the most likely distribution (Expectation Step).  
 **3. What is K-Means?**K-Means is a simple and standard clustering algorithm. K-Means works on the idea of dividing the data into K clusters so that each cluster is represented by the mean of the points of that cluster.  
  
**Steps of K-Means Algorithm:**1.**Initialization**: Choose K random initial centroids.  
2.**Assignment**: Assign each point to the nearest centroid.  
3.**Update**: Update the centroids based on the mean of the assigned points.  
4.**Repetition**: Repeat steps 2 and 3 again and again until centroids do not change much.

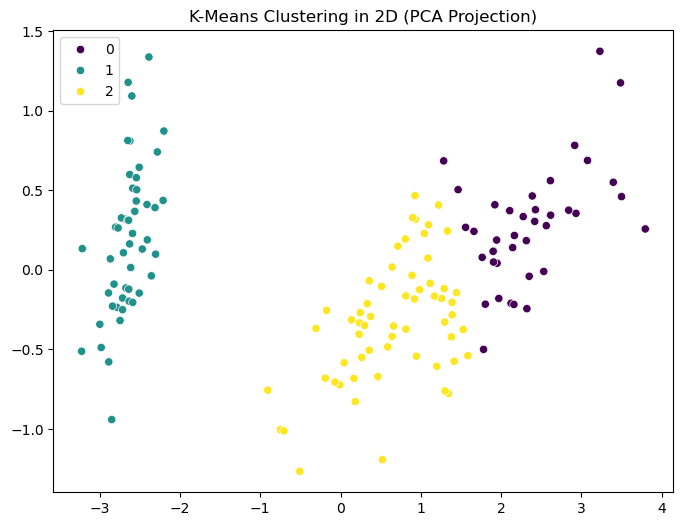
**Disadvantages of K-Means:**  
• **Assumes spherical clusters**: K-Means works on the assumption that the data is of spherical nature, a situation which does not usually dominate in actual data sets.  
• **Hard Clustering:** Any point always belongs to a single cluster though it has varying probabilities with different clusters.  
• **Sensitive to initialization:** K-Means performance can be very sensitive to the initial centroids placement.  
 **4. What is Gaussian Mixture Model (GMM)?**In comparison to K-Means, Gaussian Mixture Models (GMM) provides a probabilistic approach to cluster. GMM does not assign each point to a specific cluster but provides each point's probability of belonging to each cluster. This action allows GMM to more easily manage complex and elliptical clusters and is suitable to apply where K-Means would struggle.

**Steps in GMM Algorithm**1. Initialization: Initialize the parameters (mean, covariance, and weight of each Gaussian component).  
2. Expectation: Compute the probability that each data point is in each cluster.  
3. Maximization: Determine the parameters of the Gaussian distributions to maximize the likelihood of the data.  
4. Repeat: Iterate through steps 2 and 3 until the parameters converge.  
  
GMM uses the Expectation-Maximization (EM) algorithm, which iteratively computes the parameters and the probability of each data point in each cluster.  
  
**Advantages of GMM over K-Means:**  
  
• Soft Clustering: GMM provides soft clustering, i.e., each data point has a possibility of belonging to each cluster.  
• Elliptical Clusters: GMM can represent clusters of any shape (elliptical), while K-Means uses spherical clusters as an assumption.  
• Better for Complex Distributions: GMM is able to model data that is not well separated and possesses complex structures.  
 **5. Implementation**To show the differences between K-Means and GMM, we will use the Iris dataset, which is a common dataset that contains measurements of three iris plant species.  
  
**Dataset: Iris dataset (4 features, 150 samples, 3 classes)**  
  
**Libraries Used:**  
• **Scikit-learn**: To perform K-Means and GMM.  
• **Matplotlib & Seaborn:** For plots.  
• **NumPy**: For data manipulation.  
  
**Step 1: Load the Data**  
from sklearn.datasets import load\_iris  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
**# Load Iris dataset**  
iris = load\_iris()  
X = iris.data # Features  
y = iris.target # True labels

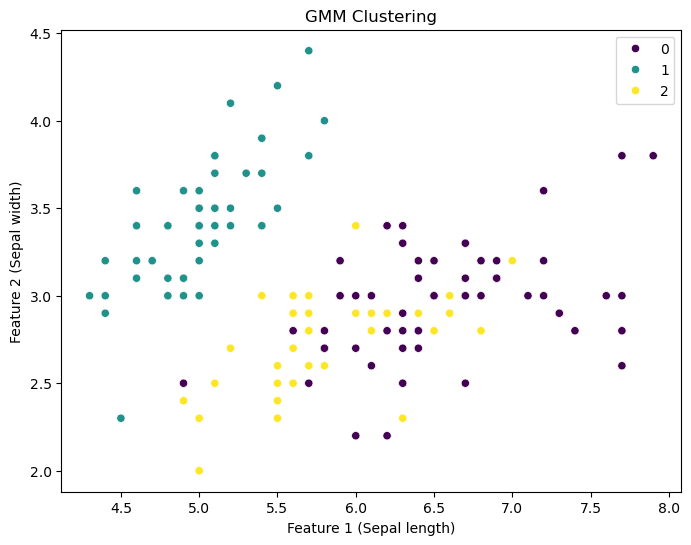
  
**Step 2: Apply K-Means Clustering**  
from sklearn.cluster import KMeans  
 **# Apply K-Means clustering**  
kmeans = KMeans(n\_clusters=3, random\_state=42)  
kmeans\_labels = kmeans.fit\_predict(X)  
Step 3: Apply GMM Clustering  
from sklearn.mixture import GaussianMixture **# Apply GMM clustering**  
gmm = GaussianMixture(n\_components=3, random\_state=42)  
gmm\_labels = gmm.fit\_predict(X)  
Step 4: Visualization  
We will now visualize the K-Means and GMM clustering results.  
K-Means Visualization:  
plt.figure(figsize=(12, 5))  
  
**# Plot K-Means clustering result**  
plt.subplot(1, 2, 1)  
plt.scatter(X[:, 0], X[:, 1], c=kmeans\_labels, cmap='viridis')  
plt.title("K-Means Clustering")



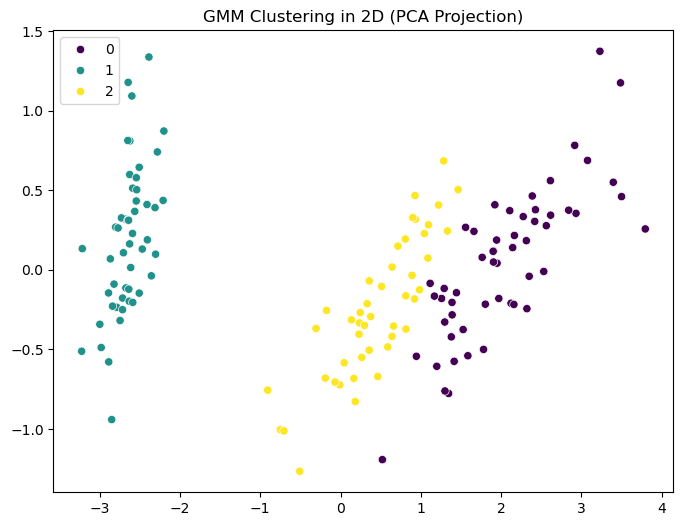
The plot shows the clusters discovered by K-Means. Note that the clusters are hard assignments, and they are colored differently.

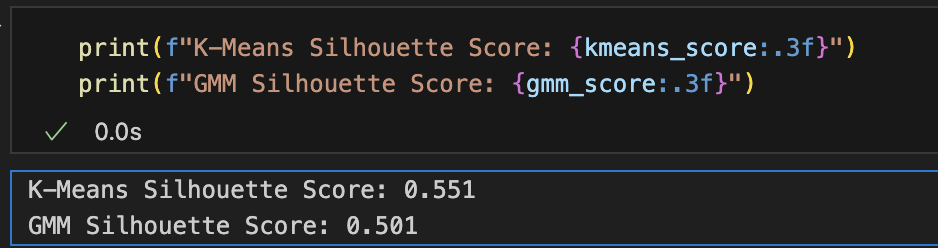


**GMM Visualization:  
# Plot GMM clustering result**  
plt.subplot(1, 2, 2)  
plt.scatter(X[:, 0], X[:, 1], c=gmm\_labels, cmap='viridis')  
plt.title("GMM Clustering")  
plt.show()



The clusters detected by GMM are illustrated in the plot. GMM also gives a probability for every data point, hence the cluster boundaries are less hard compared to K-Means.

**  
Step 5: Silhouette Score evaluation**We can compare the algorithms' performance with the Silhouette Score, which measures how similar an object is to its own cluster compared to other clusters. High values indicate good clustering performance.  
from sklearn.metrics import silhouette\_score  
 **# Compare K-Means and GMM clustering**kmeans\_score = silhouette\_score(X, kmeans\_labels)  
gmm\_score = silhouette\_score(X, gmm\_labels)  
  
print(f"K-Means Silhouette Score: {kmeans\_score}")  
print(f"GMM Silhouette Score: {gmm\_score}")

****

**6. Comparison and Conclusion**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Assumptions** | **Clustering Type** | **Performance** |
| K-Means | Spherical clusters | Hard clustering | Fast but less flexible |
| GMM | Elliptical clusters | Soft clustering | More flexible, better for complex data |

**Summary of Comparison:**

**Key Takeaways:**  
•**K-Means** is simpler and faster but works best when clusters are spherical.  
•**GMM** is more flexible with the ability to create elliptical clusters, but computationally more expensive.  
• **Silhouette Score:** GMM tends to perform well in situations where the clusters are not spherical in nature.

**7. References  
  
1**. Reynolds, Douglas. (2008). Gaussian Mixture Models. Encyclopedia of Biometrics.  
2. Scikit-learn documentation: https://scikit-learn.org  
3. Yu, Dong & Deng, Li. (2015). Gaussian Mixture Models.  
4. Giesen, J., Lucas, M., and others. (2016). Comparison of K-means vs. Gaussian Mixture Models.

**GitHib Repository**

**Github Link:**

**Appendix:**

# Gaussian Mixture Models (GMM) vs. K-Means

# Step 1: Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette\_score

from sklearn.decomposition import PCA

# Step 2: Load Iris Dataset

iris = load\_iris()

X = iris.data # Features

y = iris.target # True labels

# Step 3: Visualize the first two features of the Iris dataset

plt.figure(figsize=(8,6))

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y, palette='viridis', legend='full')

plt.title("Iris Dataset - Feature 1 vs Feature 2")

plt.xlabel("Feature 1 (Sepal length)")

plt.ylabel("Feature 2 (Sepal width)")

plt.show()

# Step 4: Apply K-Means Clustering

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans\_labels = kmeans.fit\_predict(X)

# Step 5: Apply GMM Clustering

gmm = GaussianMixture(n\_components=3, random\_state=42)

gmm\_labels = gmm.fit\_predict(X)

# Step 6: Visualize K-Means Clustering Results

plt.figure(figsize=(8,6))

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=kmeans\_labels, palette='viridis', legend='full')

plt.title("K-Means Clustering")

plt.xlabel("Feature 1 (Sepal length)")

plt.ylabel("Feature 2 (Sepal width)")

plt.show()

# Step 7: Visualize GMM Clustering Results

plt.figure(figsize=(8,6))

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=gmm\_labels, palette='viridis', legend='full')

plt.title("GMM Clustering")

plt.xlabel("Feature 1 (Sepal length)")

plt.ylabel("Feature 2 (Sepal width)")

plt.show()

# Step 8: Compare Clustering using Silhouette Score

kmeans\_score = silhouette\_score(X, kmeans\_labels)

gmm\_score = silhouette\_score(X, gmm\_labels)

print(f"K-Means Silhouette Score: {kmeans\_score:.3f}")

print(f"GMM Silhouette Score: {gmm\_score:.3f}")

# Step 9: Visualize Clusters in Reduced 2D using PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# K-Means PCA Projection

plt.figure(figsize=(8,6))

sns.scatterplot(x=X\_pca[:, 0], y=X\_pca[:, 1], hue=kmeans\_labels, palette='viridis', legend='full')

plt.title("K-Means Clustering in 2D (PCA Projection)")

plt.show()

# GMM PCA Projection

plt.figure(figsize=(8,6))

sns.scatterplot(x=X\_pca[:, 0], y=X\_pca[:, 1], hue=gmm\_labels, palette='viridis', legend='full')

plt.title("GMM Clustering in 2D (PCA Projection)")

plt.show()