School of Computer Science Engineering and Technology
Assignment-08

Course- B.Tech	**Type-** Core
Course Code-	**Course Name-** Statistical Machine Learning
Year- 2023-2024	**Semester-** odd
Date– 11-09-2023	**Batch-** AIML-B, D

Implement Gaussian Mixture Model using Synthetic Dataset

Challenges in K-Means can be overcomed using:

- You could measure uncertainty in cluster assignment by comparing the distances of each point to all cluster centers, rather than focusing on just the closest.
- You might also imagine allowing the cluster boundaries to be ellipses rather than circles, so as to account for non-circular clusters.

A **Gaussian mixture model (GMM)** attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset. In the simplest case, GMMs can be used for finding clusters in the same manner as k-means.

However, because GMM contains a probabilistic model under the hood, it is also possible to find probabilistic cluster assignments. in Scikit-Learn this is done using the`predict_proba` method. This returns a matrix of size `[n_samples, n_clusters]` which measures the probability that any point belongs to the given cluster.

Step -1: Generate Synthetic Data using unlabeled blobs

```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4,
cluster_std=0.7, random_state=0)
X = X[:, ::-1] # flip axes for better plotting
```

Step-2: Visualize the uncertinity by making data point size proportional to probability

You have to use `predict_proba` and `probs_max` to predict the max probability for the Gaussian Mixture Model.

```
from matplotlib.patches import Ellipse

def draw_ellipse(position, covariance, ax=None, **kwargs):
"""Draw an ellipse with a given position and covariance"""
ax = ax or plt.gca()
```

```
# Convert covariance to principal axes
if covariance.shape == (2, 2):
U, s, Vt = np.linalg.svd(covariance)
angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
width, height = 2 * np.sqrt(s)
else:
angle = 0
width, height = 2 * np.sqrt(covariance)

# Draw the Ellipse
for nsig in range(1, 4):
ax.add_patch(Ellipse(position, nsig * width, nsig * height,
angle, **kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
ax = ax or plt.gca()
labels = gmm.fit(X).predict(X)
if label:
ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis',
zorder=2,edgecolor='k')
else:
ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2,cmap='viridis',edgecolor='k')
ax.axis('equal')

w_factor = 0.2 / gmm.weights_.max()
for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
draw_ellipse(pos, covar, alpha=w * w_factor)

gmm = GaussianMixture(n_components=4, covariance_type='full',
random_state=42)
plot_gmm(gmm, X_stretched)
```

Step-3: GMM as Density Estimation and Generative Model

Using `sklearn.datasets import make_moons` we can plot the data using scatter plot.