Lab2 : Build and visualize a Gaussian Mixture Model (GMM)

G. Dileep Kumar
Assistant Professor

June 24, 2025

1 Key Concepts

A **Gaussian Mixture Model (GMM)** is a *probabilistic model* that assumes all the data points are generated from a mixture of several **Gaussian (normal) distributions** with unknown parameters.

1. **Mixture Model:** GMM models the data as coming from multiple subpopulations (clusters), where each is modeled by a Gaussian distribution.
2. **Gaussian Distribution:** Each component is a multivariate normal distribution defined by:
    - Mean vector μ
    - Covariance matrix Σ
3. **Soft Clustering:** Unlike K-Means (hard clustering), GMM assigns a *probability* (responsibility) to each data point for each cluster.

GMM Probability Density Function

The probability density function for a GMM is:

$$p(x) = \sum_{i=1}^{K} \pi_i \cdot \mathcal{N}(x \mid \mu_i, \Sigma_i)$$

Where:

- K = number of components
- π_i = mixing coefficient (prior probability), with $\sum_{i=1}^{K} \pi_i = 1$
- $\mathcal{N}(x \mid \mu_i, \Sigma_i)$ = multivariate Gaussian PDF

EM Algorithm for GMM

E-Step (Expectation)

Estimate the responsibility that each Gaussian has for each data point:

$$\gamma_i(x_n) = \frac{\pi_i \cdot \mathcal{N}(x_n|\mu_i, \Sigma_i)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

M-Step (Maximization)

Update the parameters using the responsibilities:

New mean:

$$\mu_i = \frac{\sum_n \gamma_i(x_n) x_n}{\sum_n \gamma_i(x_n)}$$

New covariance:

$$\Sigma_i = \frac{\sum_n \gamma_i(x_n)(x_n - \mu_i)(x_n - \mu_i)^T}{\sum_n \gamma_i(x_n)}$$

New mixing coefficients:

$$\pi_i = \frac{1}{N} \sum_n \gamma_i(x_n)$$

Repeat the E-step and M-step until convergence (log-likelihood change becomes small).

2 Implementation

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Step 1: Generate synthetic 2D data
np.random.seed(42)
n_samples = 500
mean1 = [0, 0]
cov1 = [[1, 0.5], [0.5, 1]]
data1 = np.random.multivariate_normal(mean1, cov1, n_samples
    )

mean2 = [5, 5]
cov2 = [[1, -0.3], [-0.3, 1]]
data2 = np.random.multivariate_normal(mean2, cov2, n_samples
    )
```

```
X = np.vstack((data1, data2))  # (1000, 2)

# Step 2: Initialize GMM Parameters
k = 2  # number of components
n, d = X.shape
means = X[np.random.choice(n, k, replace=False)]
covariances = [np.eye(d) for _ in range(k)]
weights = np.ones(k) / k

# Step 3-4: EM Algorithm
def e_step(X, means, covariances, weights):
    responsibilities = np.zeros((n, k))
    for i in range(k):
        rv = multivariate_normal(mean=means[i], cov=covariances[i])
        responsibilities[:, i] = weights[i] * rv.pdf(X)
    responsibilities /= responsibilities.sum(axis=1, keepdims=True)
    return responsibilities

def m_step(X, responsibilities):
    Nk = responsibilities.sum(axis=0)
    weights = Nk / n
    means = np.dot(responsibilities.T, X) / Nk[:, np.newaxis]
    covariances = []
    for i in range(k):
        diff = X - means[i]
        cov = np.dot(responsibilities[:, i] * diff.T, diff) / Nk[i]
        covariances.append(cov)
    return weights, means, covariances

# Run EM
max_iters = 100
tol = 1e-4
log_likelihoods = []

for iteration in range(max_iters):
    responsibilities = e_step(X, means, covariances, weights)
    weights, means, covariances = m_step(X, responsibilities)

    # Compute log-likelihood for convergence
    ll = 0
    for i in range(k):
        rv = multivariate_normal(mean=means[i], cov=covariances[i])
```

```
        ll += np.sum(np.log(weights[i] * rv.pdf(X) + 1e-8))
    log_likelihoods.append(ll)
    if iteration > 0 and abs(ll - log_likelihoods[-2]) < tol
        :
        break

print(f"Converged in {iteration+1} iterations.")

# Visualization
colors = responsibilities.argmax(axis=1)
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=colors, cmap='viridis',
    alpha=0.6, s=30)
plt.scatter(means[:, 0], means[:, 1], c='red', marker='x', s
    =100, label='Means')
plt.title("Gaussian Mixture Model Clustering")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 1: Sampling from a Gaussian Distribution

Output