Experiment No. 1 Introduction to Generative AI

Practical Experiment: 1. Setting Up the Environment for Generative AI and Make use of generative models for creative content generation.
2. Implement simple probabilistic models like Gaussian Mixture Models.

Name:- Gourav Sable
PRN/roll no. :- 202302040019

1. Setting Up the Environment for Generative AI

1.1 Environment Checks and Installation Hints

> These checks are safe to run offline. If you have internet and a GPU, you can later extend this section to install and use deep generative libraries like `torch`, `transformers`, or `diffusers`.

Tips:

- Use Python ≥ 3.9.
- Keep experiments in a virtual environment (e.g., `conda` or `venv`).
- Pin versions in `requirements.txt` for reproducibility.

```
# Basic environment info (safe offline)
import sys, platform, importlib

print("Python:", sys.version.split()[0])
print("Platform:", platform.platform())

# Check optional packages (will print 'missing' if not installed)
for pkg in ["numpy", "pandas", "matplotlib",  "torch", "transformers", "diffusers"]:
    try:
        mod = importlib.import_module(pkg.replace("-", "_"))
        ver = getattr(mod, "__version__", "unknown")
        print(f"{pkg:>12}: {ver}")
    except Exception:
        print(f"{pkg:>12}: missing")

# --- Optional installs (commented) ---
# If you have internet access and want to use deep generative models:
# !pip install --upgrade pip
# !pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121  # pick your CUDA/CPU wheel
# !pip install transformers diffusers accelerate safetensors
```

```
Python: 3.12.11
Platform: Linux-6.1.123+-x86_64-with-glibc2.35
       numpy: 2.0.2
      pandas: 2.2.2
  matplotlib: 3.10.0
       torch: 2.8.0+cu126
transformers: 4.56.0
   diffusers: 0.35.1
```

2. Implement simple probabilistic models like Gaussian Mixture Models.

Gaussian Mixture Model (GMM): A mixture model that represents data as a combination of multiple Gaussian distributions. Each Gaussian component is defined by its mean and covariance, and the overall model uses mixing coefficients to represent each component's proportion in the data.

EM Algorithm for GMM: The Expectation-Maximization algorithm iteratively estimates the parameters by:

E-step: Calculating the probability (responsibility) that each data point belongs to each Gaussian component.

M-step: Updating the parameters (means, covariances, mixing coefficients) to maximize the expected likelihood.

This sklearn implementation optimizes parameters via EM internally. Visualization with ellipses shows the shape and orientation of each learned Gaussian component, overlaid on clustered data points.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
from sklearn.mixture import GaussianMixture
```

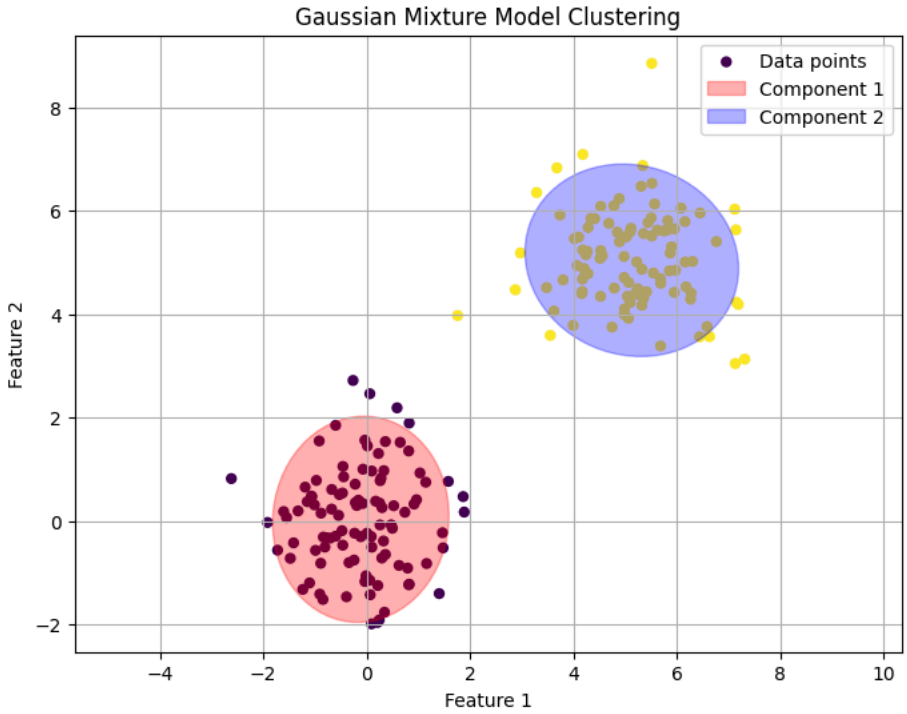
```
# Rough data
np.random.seed(42)
X = np.concatenate([
    np.random.normal(loc=[0, 0], scale=[1, 1], size=(100, 2)),
    np.random.normal(loc=[5, 5], scale=[1, 1], size=(100, 2))
])


# Initialize the GMM
n_components = 2
gmm = GaussianMixture(n_components=n_components, random_state=42)
gmm.fit(X)
```

```
GaussianMixture
GaussianMixture(n_components=2, random_state=42)
```

```
# Calculation of means of each Gaussian component
labels = gmm.predict(X)
responsibilities = gmm.predict_proba(X)

# Extract learned parameters
means = gmm.means_
covariances = gmm.covariances_
weights = gmm.weights_

print("Learned Means:")
print(means)
print("\nLearned Covariances:")
for i, cov in enumerate(covariances):
    print(f"Component {i+1} covariance:\n{cov}\n")
print("Mixing Coefficients:")
print(weights)
```

```
Learned Means:
[[-0.11500848  0.03519083]
 [ 5.12924798  5.0438043 ]]

Learned Covariances:
Component 1 covariance:
[[0.72676721 0.02955168]
 [0.02955168 0.99205403]]

Component 2 covariance:
[[ 1.0668767  -0.08151428]
 [-0.08151428  0.86501281]]

Mixing Coefficients:
[0.50014826 0.49985174]
```

```
# Visualization function
def plot_gmm_clusters(X, labels, means, covariances):
    plt.figure(figsize=(8,6))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=25, label='Data points')

    colors = ['red', 'blue']
    ax = plt.gca()

    for k, (mean, cov) in enumerate(zip(means, covariances)):
        # Compute ellipse parameters from covariance matrix
        eigenvalues, eigenvectors = np.linalg.eigh(cov)
        order = eigenvalues.argsort()[::-1]
        eigenvalues, eigenvectors = eigenvalues[order], eigenvectors[:, order]
        angle = np.degrees(np.arctan2(eigenvectors[1, 0], eigenvectors[0, 0]))
        width, height = 2 * np.sqrt(eigenvalues) * 2  # 2 std devs

        ellipse = Ellipse(xy=mean, width=width, height=height, angle=angle,
                          color=colors[k % len(colors)], alpha=0.3, label=f'Component {k+1}')
        ax.add_patch(ellipse)

    plt.title('Gaussian Mixture Model Clustering')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.grid(True)
    plt.axis('equal')
    plt.show()

    # Plot the clustering results with Gaussian ellipses
plot_gmm_clusters(X, labels, means, covariances)
```


Gaussian Mixture Model Clustering
Data points
Component 1
Component 2
Feature 1
Feature 2
8
6
4
2
0
−2
−4
−2
0
2
4
6
8
10