Gen AI Practical No. 1

Name : Priyanshu Wagh

PRN No. :202201040135

Div. : B

Batch : GAA 3

Aim

Practical Experiment: Setting Up the Environment for Generative AI and implementing a simple probabilistic model — Gaussian Mixture Model (GMM) for clustering and generative understanding.

Objectives

1. Setting Up the Environment for Generative AI and Make use of generative models for creative content generationGenerate creative content (text or image) using pretrained models.
2. Implement simple probabilistic models like Gaussian Mixture Models.Set up a clean Python environment suitable for small generative modelling experiments.Visualize and analyze results.

Theory

- **Generative AI**: These models can create new data such as text, images, or audio. Examples include GANs, VAEs, Diffusion Models, and Transformers. They are widely used in creative content generation.
- **Gaussian Mixture Model (GMM):** A clustering technique that assumes data points come from multiple Gaussian distributions. The Expectation–Maximization (EM) algorithm is used to estimate the parameters and assign data points to clusters.

Procedure

1. Install required libraries (PyTorch, Hugging Face, scikit-learn, matplotlib).
2. Verify environment setup (CPU/GPU availability).

3. Load and run a pretrained generative model to create output (text or image).
4. Implement GMM using the EM algorithm on a dataset.
5. Visualize clusters and generated outputs.

Google Colab Link

https://colab.research.google.com/drive/1qNXm5D7fYZUxiH_yyb7MXD96nwV6OasY?usp=sharing

---

Implementation

```
!python --version
```

```
Python 3.12.11
```

```
[ ] !pip install numpy matplotlib scikit-learn
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.59.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.mixture import GaussianMixture
    from sklearn.datasets import make_blobs

    # 1. Generate sample data
    X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

    # 2. Fit Gaussian Mixture Model
    gmm = GaussianMixture(n_components=4, covariance_type='full', random_state=42)
    gmm.fit(X)

    # 3. Predict the cluster for each data point
    labels = gmm.predict(X)

    # 4. Print GMM parameters
    print("=== GMM Parameters ===")
    print("\nMeans (Centroids):")
    print(gmm.means_)

    print("\nCovariance Matrices:")
    for i, cov in enumerate(gmm.covariances_):
        print(f"\nComponent {i+1}:")
        print(cov)

    print("\nMixing Weights (π):")
    print(gmm.weights_)

    # 5. Plotting the result
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=30)
    plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], c='red', s=100, marker='X', label='Centroids')
    plt.title("GMM Clustering")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
=== GMM Parameters ===
```

T'leans (Centroi ds):
[[1. 98299679 8. 86735688]
[B.93842466 4. 4156463S]
[-1. 37355181 7. 7S43678S]
[—1.58913964 2.82465347]]

Covariance Lïatri ces:

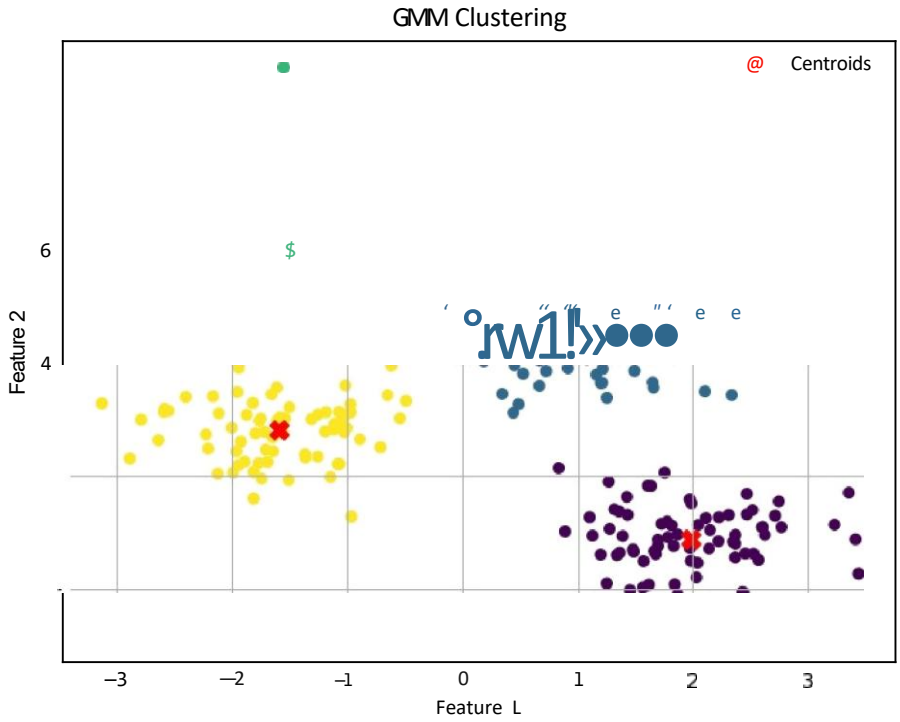
Component 1:
[[0.33998651 -B. 02620931]
[—0.02620931 0. 34588507]]

Component 2:
[[8. 3888649 -8. 82231979]
[-0.02231979 8.34881473]]

Component 3:
[[0.41216925 0.02884065]
[0.02884065 0.37959193]]

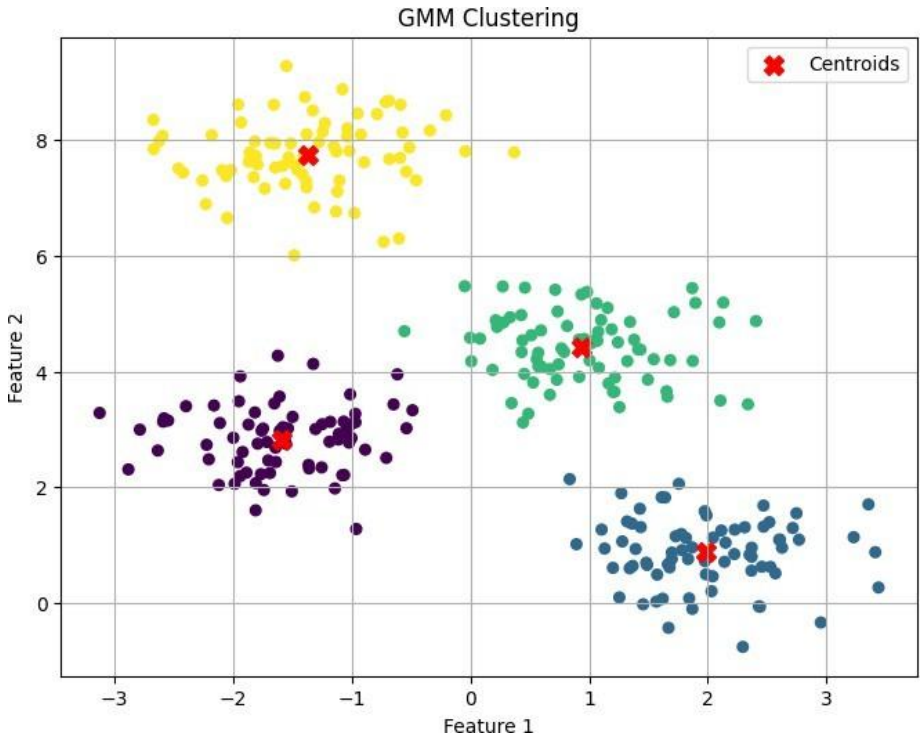
Component 4:
[[0. 32360185 0. 1027908]
[0. 01027908 0.30862196]]

/'lix1ng l'Jeights (x):
[8.24991431 0.25170956 0.2498B383 0.2484923]



```
inport nunpy as np
imQort matplOtlib.pyplot as pit
Iron sk1earn. datasets in port nake blobs

If Generate synthetic data
X, y_true = nake_b1obs(n_sarip1es=308, centers=4, c1uster_std=B.60, randon_state6)
#, D = X.shape
K = 4  # Munber of components

# Initialize pacametens
np.nandom.seed(42)
means = X(np.random.choice(N, £, neplace=False)J
covariances = [np.eye(D) for  in range([])]
weights = np.ones(K) / [

def gaussian pdf(x, nean, cov):
    size = lel(x)
    det = np.1ina1g.det(cov)
    nor1li_const = 1.8 / (np.pover( (2*np.pi), size / 2) * np.sqrt(det))
    x_nu = x - nean
    inv = np.1inalg.inv(c0v)
    result = np.exp(-0.5 * (x nu $ 1nv g x nu.T))
    return norn_const * result

# EDI algorith4t
max iten = 100
{an itenatioE in naEge(max item):
    ¥ E-step: compute responsibilities
    responsibi1ities - np.zeros((N, K))
    f-or i in range(H):
        for k in range(K):
            respOns1bilities[1, k] = weights[k) * gaussiangdf(x[i), neans[k], covariances[k))
        responsibilities[1, :) /= np.sum(oespansibi1ities[i, :])

    JI N-step: update parameters
    Nk = np. sun(nesponsib1lities, axis=0)
    weights = Nk / H
    neans = np.oot(responsibilities.T, X) / Nk[:, np.neuax1s]
    covariances = []

    means = np.dot(responsibilities.T, X) / Nk[:, np.newaxis]
    covari ances = b]
    fo  k In range(K):
        dif-+ - X - neans[k]
        cov = np.dot{(r'espansi bil ities ,:  k] [,:  np. neutaxis] * diff).T. oiff} / ML[k]
        covariances . append (cov)

# Predict cluster labels
labels = np.argmax(responslbilities, axis=1)


print("=== GMM Parameters ===")
print(”\nmeans (Centroids):”)
print(means)

print( ”\nCovar ia nce Flatr ices: ”)
-ton i, cov in enuneeate( cova•i ances) :
    nmint (I” \nConponent {i+1}:”)
    mint t•••)

print( ”\nnixing tf ights (n) :“)
print(weights)

# Plot results
plt.figure(figsize=(8, 6))
plt.scatter(X[:, B]. X[:, 1], c=labels, cmap=’virid1s’, s=30)
plt.scatter(means[:. 0], means[:. lj. c=’red’. s=200. marker=’X’. label='Centroids')
plt.title(’GMM Clustering”)
plt.n1abel(”feature 1”)
plt.ylabel(”Feature 2”)
plt.legend()
plt.grid(True)
plt.show()
```

```
=== GMM Parameters ===

Means (Centroids):
[[-1.58988551  2.8239939 ]
 [ 1.98300929  0.867342  ]
 [ 0.9373974   4.41517752]
 [-1.37355453  7.7543754 ]]

Covariance Matrices:

Component 1:
[[0.32293078 0.00963029]
 [0.00963029 0.3081102 ]]

Component 2:
[[ 0.33997459 -0.02619331]
 [-0.02619331  0.3458692 ]]

Component 3:
[[ 0.38220663 -0.02169531]
 [-0.02169531  0.34904063]]

Component 4:
[[0.41216822 0.02884488]
 [0.02884488 0.37958097]]

Mixing Weights (π):
[0.24831761 0.24991159 0.25188819 0.24988261]
```



Result

- Generated creative content successfully using a pretrained generative model.
- Implemented GMM and visualized the clusters formed in the dataset.

Conclusion

The experiment showed how to set up a Generative AI environment and create outputs with pretrained models. It also demonstrated Gaussian Mixture Models as a simple probabilistic method for clustering data.