《模式识别与机器学习 A》实验报告

实验题目：实现 k-means 聚类方法和混合高斯模型

班级：2203601

学号：2022113416

姓名：刘子康

实验报告内容

一、实验目的

- 实现一个 k-means 聚类算法和混合高斯模型，并且用 EM 算法估计模型中的参数；
- 手动生成 k 组高斯分布的数据，或利用 UCI 上的数据集，对聚类模型加以验证。

二、实验内容

手动生成 k 组不同均值和方差高斯分布的数据，参数自行设定：

（1）用 k-means 聚类，测试算法效果；

（2）用混合高斯模型和实现的 EM 算法估计参数，查看每次迭代后似然值变化情况，考察 EM 算法是否可以获得正确的结果（与设定的结果比较）。

应用：在 UCI 上找一个简单问题数据，用实现的 GMM 进行聚类。

三、实验环境

- 操作系统：Windows 11
- 编程语言：Python 3.10
- 第三方库：Numpy 1.23.4，Scipy 1.10.0，Pandas 2.1.4，Matplotlib 3.8.2
- IDE：Pycharm 2022 社区版

四、实验过程、结果及分析

4.1 实验原理

K-means 是一种基于距离的硬聚类方法，目标是将数据分成 K 个簇，每个簇内的点尽可能接近聚类中心，而簇间尽可能分开。其基本原理为维护 K 个聚类中心，将每个样本点分配到距离最近的聚类中心所在簇，并计算每个簇中所有样本点均值来更新聚类中心，重复此过程，直至聚类中心不再移动。

高斯混合模型（Gaussian Mixture Model，GMM） 是一种基于概率模型的软聚类方法，可以看作是对 K-means 的一种扩展。GMM 认为数据是由多个高斯分布混合而成的，每个簇对应一个高斯分布，每个样本点对于各个簇有一个概率值，表示它属于该簇的可能性。GMM 使用期望最大化（Expectation-Maximization，EM）算法进行参数估计，EM 算法是一种迭代算法，用于含有隐藏变量的概率模型参数的极大似然估计，主要包括两个步骤：E 步求期望，M 步极大化。

聚类评价指标选择兰德指数（RI），它是一种常用的聚类评估指标，通过计算同类且同聚类或异类且异聚类的比例，比较聚类结果与真实标签之间的相似性。

4.2 实验过程

4.2.1 生成 K 组高斯分布数据

使用 Numpy 库的 random 模块随机生成 K 组不同的均值和方差，并使用 normal()函数每组随机生成 n_samples 个符合高斯分布的样本点（为便于绘图，特征数设置为 2），拼接后返回生成的数据集。

```
# 生成k组高斯分布数据
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def generate_data(K, n_samples, mean=None, var=None):
    if mean is None:
        mean = np.random.randint(-5, 5, K * 2).reshape(K, 2)
    if var is None:
        var = np.random.uniform(1, 2, K * 2).reshape(K, 2)
    data = [[], []]
    for i in range(K):
        data[0].extend(np.random.normal(mean[i, 0], var[i, 0], n_samples))
        data[1].extend(np.random.normal(mean[i, 1], var[i, 1], n_samples))

    return np.array(data).T
```

4.2.2 K-means 聚类算法实现

（1）初始化聚类中心。随机或线性选取 K 个样本点作为初始聚类中心；

```
    # 初始化聚类中心
    sample_size = n_samples // K   # 将数据等距分成K份
    center_ini = sample_size // 2
    for i in range(K):
        centers[i, :] = data[center_ini, :]
        center_ini += sample_size
```

（2）分配样本点。对于数据集中的每个样本点，计算它到每个聚类中心的距离（通常使用欧氏距离），将其分配给距离最近的簇；

```
        # 计算样本所属簇
        for i in range(n_samples):
            dist = []
            for j in range(K):
                dist.append(np.linalg.norm(data[i] - centers[j]))
            labels[i] = np.argmin(dist)
            result[labels[i]].append(i)
```

（3）更新每个簇的中心。对于每个簇，计算其所有成员点的均值，即为新的聚类中心；

```
        # 更新聚类中心
        centers_last = centers.copy()
        for i in range(K):
            centers[i, :] = np.mean(data[result[i], :], axis=0)
```

（4）重复（2）和（3），直至聚类中心不再变化。记录前一次的聚类中心，每次循环更新完聚类中心后与之比较，若相等则结束迭代。

（5）输出聚类结果，计算 RI，绘制图像，不同颜色代表不同簇的聚类结果。

4.2.3 GMM 模型构建

（1）初始化每个簇的高斯分布参数：均值（μ）、协方差矩阵（σ）和权重（α）。随机选择 K 个样本点作为初始均值，计算数据集转置后的协方差矩阵作为初始协方差矩阵，初始权重设置为平均权重。

```
        # 初始化模型参数
        n_samples, n_features = X.shape
        self.alphas = np.ones(self.K) / self.K
        self.mus = np.array([X[i] for i in np.random.choice(n_samples, self.K, replace=False)])
        self.sigmas = np.array([np.cov(X.T) for _ in range(self.K)])
        self.likelihood_history = []
```

（2）E-Step，计算分模型 k 对观测数据 y_j 的响应度。

根据公式$\hat{\gamma}_{jk}=E\left(\gamma_{jk}|y,\theta\right)=P(\gamma_{jk}=1|y,\theta)=\frac{\alpha_k\phi(y_j|\theta_k)}{\sum_{k=1}^{k}\alpha_k\phi(y_j|\theta_k)}$，计算当前模型参数下第 j 个观测数据来自第 k 个分模型（簇）的概率。使用 Scipy 库 stats 模块的 multivariate_normal

函数，计算观测值$\hat{\gamma}_{jk}$在参数θ_k下的概率密度函数值，并乘上相应权重，最后除以各分模型响应度求和。

```
        # E-Step: 计算每个数据点属于每个高斯分布的后验概率
        单元测试 | 注释生成 | 代码解释 | 缺陷检测
        def _e_step(self, X):
            responsiveness = np.zeros((X.shape[0], self.K))
            for i in range(self.K):
                rv = multivariate_normal(mean=self.mus[i], cov=self.sigmas[i])
                responsiveness[:, i] = self.alphas[i] * rv.pdf(X)
            responsiveness /= responsiveness.sum(axis=1, keepdims=True)

            return responsiveness
```

（3）M-Step，更新新一轮迭代的模型参数。

用$\hat{\mu}_k$、$\hat{\sigma}_k^2$和$\hat{\alpha}_k$表示$\theta^{(i+1)}$的各参数，更新模型参数θ_k。求$\hat{\mu}_k$、$\hat{\sigma}_k^2$只需将$Q(\theta,\theta^{(i+1)})$分别对$\hat{\mu}_k$、$\hat{\sigma}_k^2$求偏导并令其为 0，即可得到，公式为$\hat{\mu}_k=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}y_j}{\sum_{j=1}^{N}\hat{\gamma}_{jk}}$，$\hat{\sigma}_k^2=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}(y_j-\hat{\mu}_k)^2}{\sum_{j=1}^{N}\hat{\gamma}_{jk}y_j}$；求$\hat{\alpha}_k$是在$\sum_{k=1}^{K}\hat{\alpha}_k=1$条件下求偏导并令其为 0 得到的，公式为$\hat{\alpha}_k=\frac{n_k}{N}=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}}{N}$。

```
        # M-Step: 更新模型参数
        单元测试 | 注释生成 | 代码解释 | 缺陷检测
        def _m_step(self, X, responsiveness):
            n_samples = X.shape[0]
            for i in range(self.K):
                n_k = responsiveness[:, i].sum()
                # 计算加权均值
                self.mus[i] = np.sum(responsiveness[:, i][:, np.newaxis] * X, axis=0) / n_k

                # 计算加权协方差
                diff = X - self.mus[i]
                self.sigmas[i] = np.dot((responsiveness[:, i][:, np.newaxis] * diff).T, diff) / n_k

                # 计算加权权重
                self.alphas[i] = responsiveness[:, i].sum() / n_samples
```

（4）重复（2）和（3），直至收敛。

明确隐变量，构造完全数据的对数似然函数$P(y,\gamma|\theta)=\prod_{j=1}^{N}P(y_j,\gamma_{j1},\gamma_{j2},\dots,\gamma_{jk}|\theta)$。

```
        # 对数似然函数
        单元测试 | 注释生成 | 代码解释 | 缺陷检测
        def log_likelihood(self, X):
            likelihood = 0
            for i in range(self.K):
                rv = multivariate_normal(mean=self.mus[i], cov=self.sigmas[i])
                likelihood += self.alphas[i] * rv.pdf(X)

            return np.sum(np.log(likelihood))
```

维护一个历史对数似然函数值列表，每次迭代后与上一次的似然函数值比较，若几乎没有变化则结束迭代（tol 默认值 1e-6）。

```
            # 判断是否收敛
            if iter > 0 and abs(self.likelihood_history[-1] - self.likelihood_history[-2]) < self.tol:
                break
```

（5）输出结果并绘图。每个样本点通过对属于各个簇的概率值取最大值对应簇，作为它的类别，输出聚类结果，计算 RI，绘制图像，不同颜色代表不同簇的聚类结果。

4.2.4 UCI 鸢尾花数据集测试

选择 UCI 数据库中的鸢尾花数据集进行测试验证，该数据集有 4 个特征、3 个类别，可用于聚类算法。使用 Pandas 库加载本地的鸢尾花数据集，并转换为 Numpy 数组格式，使用 GMM 模型进行测试，输出聚类结果和 RI，并选择前两个特征进行绘图。

```python
    # UCI鸢尾花数据集
    iris = pd.read_csv('./iris.csv').drop(['Id', 'Species'], axis=1).to_numpy()
    true_labels = [0] * 50 + [1] * 50 + [2] * 50

    gmm_em = GMM_EM(K=3)
    gmm_em.fit(iris)
    gmm_labels = np.argmax(gmm_em._e_step(iris), axis=1)    # 获取每个点的聚类结果
    RI = adjusted_rand_score(true_labels, gmm_labels)
    print("=" * 27 + " Iris数据集 GMM聚类结果 " + "=" * 27)
    print(gmm_labels)
    print(f"兰德指数RI：{RI:.4f}")

    # 绘图
    fig2, axs = plt.subplots(1, 2, figsize=(10, 5))
    scatter4 = axs[0].scatter(iris[:, 0], iris[:, 1], c=true_labels, s=12)
    axs[0].set_title("True Classification")
    axs[0].set_xlabel("Feature 1")
    axs[0].set_ylabel("Feature 2")
    scatter5 = axs[1].scatter(iris[:, 0], iris[:, 1], c=gmm_labels, s=12)
    axs[1].set_title("GMM Clustering")
    axs[1].set_xlabel("Feature 1")
    axs[1].set_ylabel("Feature 2")
    fig2.colorbar(scatter4, ax=axs, fraction=0.02, pad=0.03, ticks=[0, 1, 2, 3])
```

4.3 实验结果及分析

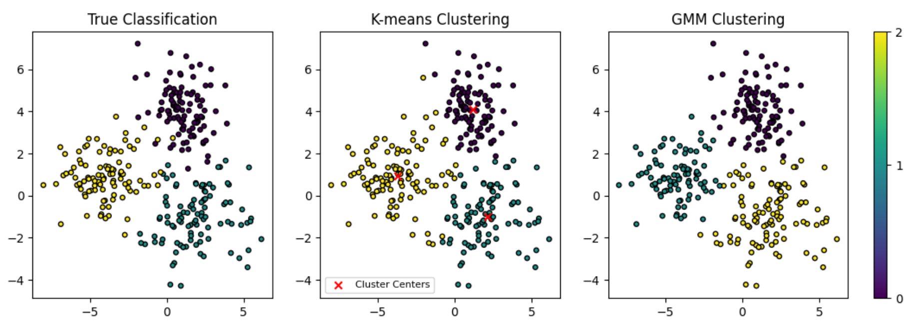
4.3.1 手动生成的 K 组高斯分布数据

聚类结果及 RI 指标如下所示，其中 K-means 聚类结果 RI 为 0.8633，GMM 聚类结果 RI 为 0.9019。

```
============================== K-means聚类结果 ==============================
[0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 1 1 1
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2]
兰德指数ARI：0.8633

=============================== GMM聚类结果 ===============================
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 2 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1
 1 1 1 1]
兰德指数ARI：0.9019
```

绘制样本点图像如下所示，注：聚类结果中的不同颜色仅用于区分不同簇，并不表示类别标签，与真实类别的颜色无一一对应关系。

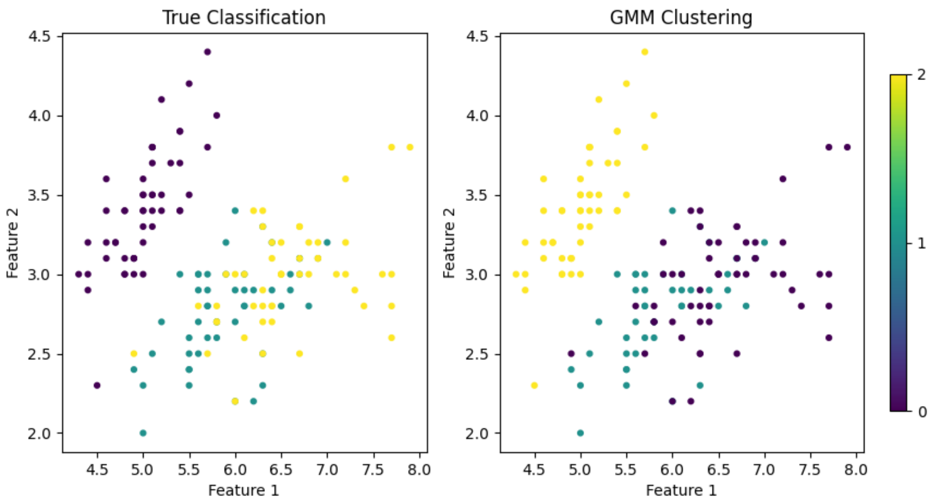


4.3.2 UCI 鸢尾花数据集

聚类结果及 RI 指标如下所示，聚类结果 RI 为 0.7184。

```
========================== Iris数据集 GMM聚类结果 ==========================
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 2 0 2 0 2 2 0 0
 0 0 0 0 0 2 2 2 2 0 0 2 0 0 2 0 0 0 2 2 2 0 2 2 0 0 2 0 0 0 0 0 0 0 0 0 0
 0 0]
兰德指数RI：0.7184
```

绘制样本点图像（前两个特征）如下所示，注：聚类结果中的不同颜色仅用于区分不同簇，并不表示类别标签，与真实类别的颜色无一一对应关系。



五、实验总体结论

K-means 算法和高斯混合模型均可用于数据集聚类，K-means 通过计算样本点和聚类中心之间的距离将样本点归类，而高斯混合模型输出每个样本点属于各个簇的概率值，取最大值对应簇进行归类。

此次实验使用 K-means 算法和基于 E-M 算法估计参数的高斯混合模型，对手动生成的 K 组高斯分布数据和 UCI 数据库中的鸢尾花数据集进行测试验证，使用兰德指数作为评价指标，达到了较好的聚类效果。

六、完整实验代码

```python
import numpy as np
import pandas as pd
from scipy.stats import multivariate_normal
from sklearn.metrics import adjusted_rand_score
import matplotlib.pyplot as plt

np.random.seed(69)

```

```python
# 生成 k 组高斯分布数据
def generate_data(K, n_samples, mean=None, var=None):
    if mean is None:
        mean = np.random.randint(-5, 5, K * 2).reshape(K, 2)
    if var is None:
        var = np.random.uniform(1, 2, K * 2).reshape(K, 2)
    data = [[], []]
    for i in range(K):
        data[0].extend(np.random.normal(mean[i, 0], var[i, 0], n_samples))
        data[1].extend(np.random.normal(mean[i, 1], var[i, 1], n_samples))

    return np.array(data).T

def Kmeans(K, data):
    n_samples, n_features = data.shape
    result = [[] for _ in range(K)] # 聚类结果
    labels = np.zeros(n_samples, dtype=int)     # 各样本聚类标签
    centers = np.zeros((K, n_features))         # 当前聚类中心
    centers_last = np.zeros((K, n_features))    # 上一次的聚类中心

    # 初始化聚类中心
    sample_size = n_samples // K    # 将数据等距分成 K 份
    center_ini = sample_size // 2
    for i in range(K):
        centers[i, :] = data[center_ini, :]
        center_ini += sample_size

    while centers.all() != centers_last.all():
        # 计算样本所属簇
        for i in range(n_samples):
            dist = []
            for j in range(K):
                dist.append(np.linalg.norm(data[i] - centers[j]))
            labels[i] = np.argmin(dist)
            result[labels[i]].append(i)

        # 更新聚类中心
        centers_last = centers.copy()
        for i in range(K):
            centers[i, :] = np.mean(data[result[i], :], axis=0)

    return labels, centers

class GMM_EM:
```

```python
    def __init__(self, K, max_iter=100, tol=1e-6):
        self.K = K
        self.max_iter = max_iter
        self.tol = tol

    # 模型迭代
    def fit(self, X):
        # 初始化模型参数
        n_samples, n_features = X.shape
        self.alphas = np.ones(self.K) / self.K
        self.mus = np.array([X[i] for i in np.random.choice(n_samples, self.K, replace=False)])
        self.sigmas = np.array([np.cov(X.T) for _ in range(self.K)])
        self.likelihood_history = []

        # 迭代过程
        for iter in range(self.max_iter):
            # E-step: 计算每个数据点属于每个高斯分布的后验概率
            responsiveness = self._e_step(X)

            # M-step: 更新模型参数
            self._m_step(X, responsiveness)

            # 计算对数似然函数
            likelihood = self.log_likelihood(X)
            self.likelihood_history.append(likelihood)
            # print(f"Iter {iter + 1}, Likelihood: {likelihood}")

            # 判断是否收敛
            if iter > 0 and abs(self.likelihood_history[-1] - self.likelihood_history[-2]) < self.tol:
                break

        return self

    # E-Step：计算每个数据点属于每个高斯分布的后验概率
    def _e_step(self, X):
        responsiveness = np.zeros((X.shape[0], self.K))
        for i in range(self.K):
            rv = multivariate_normal(mean=self.mus[i], cov=self.sigmas[i])
            responsiveness[:, i] = self.alphas[i] * rv.pdf(X)
        responsiveness /= responsiveness.sum(axis=1, keepdims=True)

        return responsiveness
```

```

    # M-Step: 更新模型参数
    def _m_step(self, X, responsiveness):
        n_samples = X.shape[0]
        for i in range(self.K):
            n_k = responsiveness[:, i].sum()
            # 计算加权均值
            self.mus[i] = np.sum(responsiveness[:, i][:, np.newaxis] * X, axis=0) / n_k

            # 计算加权协方差
            diff = X - self.mus[i]
            self.sigmas[i] = np.dot((responsiveness[:, i][:, np.newaxis] * diff).T, diff) / n_k

            # 计算加权权重
            self.alphas[i] = responsiveness[:, i].sum() / n_samples

    # 对数似然函数
    def log_likelihood(self, X):
        likelihood = 0
        for i in range(self.K):
            rv = multivariate_normal(mean=self.mus[i], cov=self.sigmas[i])
            likelihood += self.alphas[i] * rv.pdf(X)

        return np.sum(np.log(likelihood))




if __name__ == '__main__':
    K = 3
    n_samples = 100

    # 生成 k 组高斯分布数据
    data = generate_data(K, n_samples)
    true_labels = [0] * n_samples + [1] * n_samples + [2] * n_samples
    # print(data.shape)

    # K-means 聚类算法
    kmeans_labels, cluster_centers = Kmeans(K, data)
    RI = adjusted_rand_score(true_labels, kmeans_labels)
    print("=" * 30 + " K-means 聚类结果 " + "=" * 30)
    print(kmeans_labels)
    print(f"兰德指数 ARI: {RI:.4f}\n")
```

```

    # 高斯混合模型+EM 算法
    gmm_em = GMM_EM(K)
    gmm_em.fit(data)
    gmm_labels = np.argmax(gmm_em._e_step(data), axis=1)    # 获取每个点的聚类标签
    RI = adjusted_rand_score(true_labels, gmm_labels)
    print("=" * 32 + " GMM 聚类结果 " + "=" * 32)
    print(gmm_labels)
    print(f"兰德指数 ARI: {RI:.4f}\n")

    # 绘图
    cmap = plt.get_cmap("viridis")
    fig1, axs = plt.subplots(1, 3, figsize=(20, 4))
    scatter1 = axs[0].scatter(data[:, 0], data[:, 1], c=true_labels, s=15, edgecolor='black')
    axs[0].set_title("True Classification")
    scatter2 = axs[1].scatter(data[:, 0], data[:, 1], c=kmeans_labels, s=15, edgecolor='black')
    axs[1].scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='x', label='Cluster Centers')
    axs[1].legend(fontsize=8, loc='lower left')
    axs[1].set_title("K-means Clustering")
    scatter3 = axs[2].scatter(data[:, 0], data[:, 1], c=gmm_labels, s=15, edgecolor='black')
    axs[2].set_title("GMM Clustering")
    fig1.colorbar(scatter1, ax=axs, fraction=0.02, pad=0.03, ticks=[0, 1, 2, 3])


    # UCI 鸢尾花数据集
    iris = pd.read_csv('./iris.csv').drop(['Id', 'Species'], axis=1).to_numpy()
    true_labels = [0] * 50 + [1] * 50 + [2] * 50

    gmm_em = GMM_EM(K=3)
    gmm_em.fit(iris)
    gmm_labels = np.argmax(gmm_em._e_step(iris), axis=1)    # 获取每个点的聚类标签
    RI = adjusted_rand_score(true_labels, gmm_labels)
    print("=" * 27 + " Iris 数据集 GMM 聚类结果 " + "=" * 27)
    print(gmm_labels)
    print(f"兰德指数 RI: {RI:.4f}")

    # 绘图
    fig2, axs = plt.subplots(1, 2, figsize=(10, 5))
    scatter4 = axs[0].scatter(iris[:, 0], iris[:, 1], c=true_labels, s=12)
    axs[0].set_title("True Classification")
```

```
    axs[0].set_xlabel("Feature 1")
    axs[0].set_ylabel("Feature 2")
    scatter5 = axs[1].scatter(iris[:, 0], iris[:, 1], c=gmm_labels, s=12)
    axs[1].set_title("GMM Clustering")
    axs[1].set_xlabel("Feature 1")
    axs[1].set_ylabel("Feature 2")
    fig2.colorbar(scatter4, ax=axs, fraction=0.02, pad=0.03, ticks=[0, 1, 2, 3])

    plt.show()
```

七、参考文献

[1] 刘远超. 深度学习基础: 高等教育出版社, 2023.