哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： 聚类算法

学号：1190201018
姓名： 李昆泽

Lab3-Report

1. 实验目的

实现一个k-means算法和混合高斯模型，并且用EM算法估计模型中的参数。

2. 实验要求及实验环境

实验要求

测试：用高斯分布产生k个高斯分布的数据（不同均值和方差）（其中参数自己设定）。

（1）用k-means聚类，测试效果；

（2）用混合高斯模型和你实现的EM算法估计参数，看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（与你设定的结果比较）。

应用：可以UCI上找一个简单问题数据，用你实现的GMM进行聚类。

实验环境

OS：Win 10

Python 3.8

3. 设计思想

3.1 手工生成数据

需要手工生成 K 类二维高斯分布的数据以供实验，每一类的高斯分布不同。做法就是用各类的均值和协方差矩阵生成多维高斯分布，再将生成的 K 类数据 concatenate 起来：

Python

```python
def generate_data(k, means, sample_num):
    """
    生成k类二维高斯分布的数据
    :param k: 类别数
    :param means: list[list], 代表每一类的均值
    :param sample_num: 每一类的样本数
    :return: 返回生成的数据
    """
    samples = np.zeros((1, 1))
    for i in range(k):
        data_temp = np.random.multivariate_normal(means[i], cov, sample_num)
        if i == 0:
            samples = data_temp
        else:
            samples = np.concatenate((samples, data_temp), axis=0)
    return samples
```

3.2 K-means基本原理

聚类问题不同于之前接触到的分类问题，它是个无监督问题，即数据的标记信息是未知的，直接根据数据的属性将数据划分为各个类别。

问题描述：给定样本集 D 和划分聚类的数量 k ，聚类需要将样本划分为 k 个不相交的簇 $N = \mathbf{N_1}, \mathbf{N_2}, \ldots, \mathbf{N_k}$ ，记 k 类每一类的中心点为 $\mu = \mu_1, \mu_2, \ldots, \mu_k$ 。相应地，用 C 来表示一个指派， C 将样本集中的每个点指派给某一中心点 μ_i 。记样本中所有点到其所属的类别中心的距离之和为 $F(\mu, C)$ ，关于距离可以有多种不同的定义，这里取欧式距离作为距离的度量，其中 $F(\mu, C)$ 公式如下：

$$F(\mu, C) = \sum_{i=1}^{k} \sum_{x:C(x)=i} ||x - \mu_i||^2$$

其中， $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x_i$ ， E 刻画了簇内样本围绕簇的均值向量的紧密程度，其值越小表明簇内样本的相似度越高。

聚类任务的优化目标就是要使此距离之和最小，即目标如下：

$$\min_{\mu} \min_{C} F(\mu, C) = \min_{\mu} \min_{C} \sum_{i=1}^{k} \sum_{x:C(x)=i} ||x - \mu_i||^2$$

K-Means采用迭代优化的策略，优化算法如下：

1. 首先初始化 k 个点作为中心点 μ ；

2. 将每一个点划分到离它最近的那个中心(固定 μ ，优化 C)；

3、根据新的 C 重新计算各类的 μ (固定 C ，优化 μ)，回到第二步继续迭代求解。

算法迭代终止的条件：当一轮迭代前后每个点所属的类别都不再变化，或者一轮迭代前后， μ 的变化很小，小于某个极小值，则停止迭代。

3.3 GMM基本原理

GMM中假设各类的分布为高斯分布，多元高斯分布生成的 d 维随机变量 x 的密度函数为：

$$p(x|\mu,\Sigma)=\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

其中 μ 为均值向量， Σ 为协方差矩阵。

给定训练样本集 $X=\{x_1,x_2,...,x_n\}$,其中 n 为样本数量。对于一个样本 x_i ，我们可以认为它是由多个对应维度的多元高斯分布所生成，可以由高斯分布的线性叠加来表征数据，假设数据由 k 个高斯分布混合生成，则

$$p(x_i)=\sum_{j=1}^{k}\pi_j p(x_i|\,u_j,\Sigma_j)$$

其中 μ_j 和 Σ_j 分别表示第 j 个高斯分布的均值和协方差矩阵， π_j 为相应的混合系数，满足 $\sum_{j=1}^{k}\pi_j=1$ 。令随机变量 $z_j\in\{1,2,...,k\}$ 表示生成样本 x_j 的高斯混合成分，其取值未知。根据贝叶斯定理， z_j 的后验分布对应于

$$\gamma(z_j)\equiv p(z_j=i|x_j)=\frac{p(z_j=i)p(x_j|z_j=i)}{p(x_j)}=\frac{\pi_i p(x_j|\mu_i,\Sigma_i)}{\sum_{l=1}^{k}\pi_l p(x_j|\mu_l,\Sigma_l)}$$

当后验概率已知时，混合高斯模型将训练样本划分成了 k 个簇 $C=C_1,C_2,...,C_k$ ，对于每一个样本 x_i ，其类别为 j ，满足 $j=\arg\max_j\gamma(z_j)$ ，即选择后验概率最大的类别作为其标签类别。其极大似然函数为

$$LL(D)=\ln p(X|\pi,\mu,\Sigma)=\ln\prod_{i=1}^{n}p(x_i)=\sum_{i=1}^{n}\ln\sum_{j=1}^{k}\pi_j p(x_i|\,u_j,\Sigma_j)$$

使上式最大化，对 μ_j 求偏导，并令导数为0，则

$$\frac{\partial\ln p(X|\pi,\mu,\Sigma)}{\partial\mu_j}=\sum_{i=1}^{n}\frac{\pi_j p(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}\Sigma_j^{-1}(x_i-\mu_j)=0$$

令

$$\gamma_{ji}=\frac{p(z_j=i|x_j)}{\sum\limits_{j=1}^{k}p(z_j=i|x_j)}=\frac{\pi_i p(x_j|\mu_i,\Sigma_i)}{\sum\limits_{l=1}^{k}\pi_l p(x_j|\mu_l,\Sigma_l)}$$

可解得

$$n_i=\sum_j\gamma_{ji}$$

$$\mu_i=\frac{1}{n_i}\sum_{i=1}^{n}\gamma_{ji}x_j$$

同理，对 Σ_j 求导令导数为0：

$$\frac{\partial\ln p(X|\pi,\mu,\Sigma)}{\partial\Sigma_j}=\sum_{i=1}^{n}\frac{\pi_j p(x_i|\mu_j,\Sigma_j)}{\sum\limits_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}(\Sigma_j^{-1}-\Sigma_j^{-1}(x_i-\mu_j)(x_i-\mu_j)^T\Sigma_j^{-1})=0$$

解得

$$\Sigma_i=\frac{\sum\limits_{j=1}^{n}\gamma_{ji}(x_j-\mu_i)(x_j-\mu_i)^T}{n_i}$$

对于混合系数 π_j ，还需要满足约束条件 $\sum\limits_{j=1}^{k}\pi_j=1$ 。构造拉格朗日多项式：

$$\ln p(X|\pi,\mu,\Sigma)+\lambda(\sum_{j=1}^{k}\pi_j-1)$$

对 π_j 求导，令导数为0：

$$\frac{\partial\ln p(X|\pi,\mu,\Sigma)+\lambda(\sum\limits_{j=1}^{k}\pi_j-1)}{\partial\pi_j}=\sum_{i=1}^{n}\frac{p(x_i|\mu_j,\Sigma_j)}{\sum\limits_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}+\lambda=0$$

同乘 π_j 并将 $j\in\{1,2,...,k\}$ 代入相加得：

$$\sum_{j=1}^{k}\pi_j\sum_{i=1}^{n}\frac{p(x_i|\mu_j,\Sigma_j)}{\sum\limits_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}+\lambda\sum_{j=1}^{k}\pi_j=0$$

将约束条件代入：

$$\sum_{i=1}^{n}(\frac{\sum_{j=1}^{k}\pi_j p(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)})+\lambda\sum_{j=1}^{k}\pi_j=n+\lambda=0$$

即 $\lambda=-n$ ，代入 $\sum_{i=1}^{n}\frac{p(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}+\lambda=0$ 中，得

$$\pi_i=\frac{n_i}{n}$$

GMM算法过程如下：

1. 随机初始化参数 $\pi_i,\mu_i,\Sigma_i,\ \ i\in 1,2,\dots,k$.
2. E步：根据式 $\gamma(z_j)=\frac{\pi_j p(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_j)}$ 计算每个样本由各个混合高斯成分生成的后验概率.
3. M步：用下式更新参数 $\pi_i,\mu_i,\Sigma_i,\ \ i\in 1,2,\dots,k$

$$\mu_i=\frac{1}{n_i}\sum_{j=1}^{n}\gamma_{ji}x_j$$

$$\Sigma_i=\frac{\sum_{j=1}^{n}\gamma_{ji}(x_j-\mu_i)(x_j-\mu_i)^T}{n_i}$$

$$\pi_i=\frac{n_i}{n}$$

4. 重复E步和M步直至收敛.

算法迭代结束条件：某一次迭代后参数的变化小于一个极小数。

E步算法实现：

Python

```python
def E_step(X, alpha, mu, sigma):
    sample_size = X.shape[0]
    cluster_size = mu.shape[0]
    gamma = np.zeros((sample_size, cluster_size))
    p = np.zeros(cluster_size)
    p_x = np.zeros(cluster_size)
    for i in range(sample_size):
        for j in range(cluster_size):
            p[j] = Gaussian(X[i], mu[j], sigma[j])
            p_x[j] = alpha[j] * p[j]
        for j in range(cluster_size):
            gamma[i, j] = p_x[j] / np.sum(p_x)
    return gamma
```

M步算法实现：

Fortran

```fortran
def M_step(X, k, gamma):
    sample_size = X.shape[0]
    feature_size = X.shape[1]

    mu = np.zeros((k, feature_size))
    sigma = np.zeros((k, feature_size, feature_size))

    for i in range(k):
        # 计算新均值向量
        mu[i] = np.sum(X * gamma[:, i].reshape((-1, 1)), axis=0) / np.sum(gamma, axis=0)[i]

        # 计算新协方差矩阵
        sigma[i] = 0
        for j in range(sample_size):
            sigma[i] += (X[j].reshape((1, -1)) - mu[i]).T.dot((X[j] - mu[i]).reshape((1, -1))) * gamma[j, i]
        sigma[i] = sigma[i] / np.sum(gamma, axis=0)[i]

    # 计算新混合系数
    alpha = np.sum(gamma, axis=0) / sample_size

    return alpha, mu, sigma
```

另外，由于需要观察每次迭代后似然值的变化，我们还需要编写一个计算（对数）似然的函数，用以计算每轮迭代的似然值，具体函数如下：

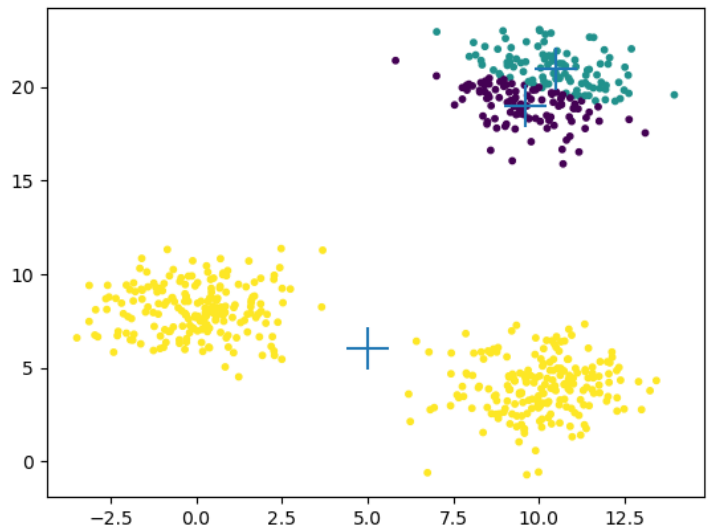
Python

```python
def likelihood_calculate(X, alpha, mu, sigma):
    sample_size = X.shape[0]
    k = len(alpha)
    likelihood = 0

    for j in range(sample_size):
        temp = 0
        for i in range(k):
            temp += alpha[i] * Gaussian(X[j], mu[i], sigma[i])
        likelihood += np.log(temp)

    return likelihood
```

4. 实验结果与分析

4.1 关于初始均值向量的选择

通过上述对K-means和GMM算法的分析，可以看出，这两种算法都非常依赖于初始均值向量的选择。在实验中，我发现如果随机选择初始均值向量，经过迭代可能得不到最优结果，而是会陷入局部最优，如下图所示。



在理想情况下，初始的均值向量应该尽量分散，这样才能避免陷入局部最优的情况。而上面这种情况的发生显然是因为有两个初始均值向量位置太过接近了。

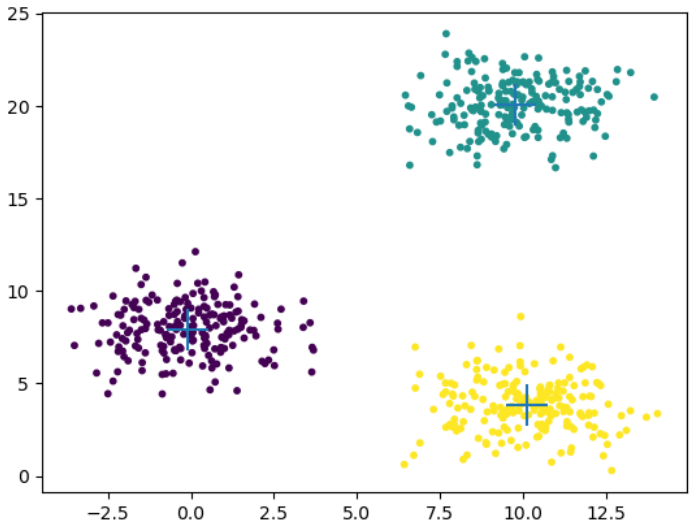
我们提出一种新的确定初始均值向量的办法，可以使均值向量尽量分散，具体算法如下。

- 首先随机选择任意一个样本作为均值向量
- 按照下面的步骤进行迭代，直到选择到 k 个均值向量

- 假设当前已经选择到 i 个均值向量，构成集合 $D=\{\mu_1,\mu_2,\ldots,\mu_i\}$ ，则在剩余样本中选择距离已选出的 i 个均值向量距离最远的样本
- 将其加入初始均值向量集合，得到 $D=\{\mu_1,\mu_2,\ldots,\mu_i,\mu_{i+1}\}$

通过这种初始化均值向量的方式，能够有效使均值向量尽量分散，从而在一定程度上避免结果陷入局部最优解。

下面是采用了优化算法之后的聚类效果。

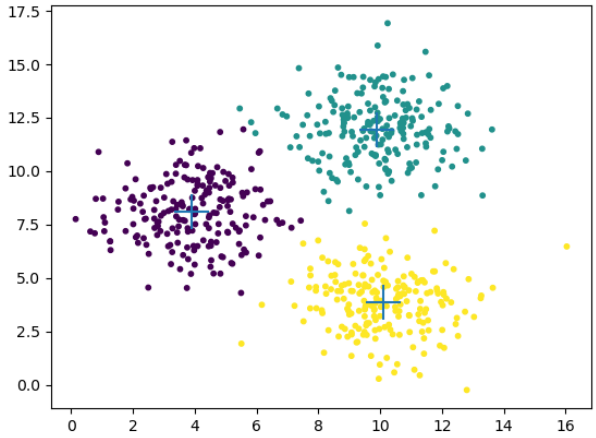


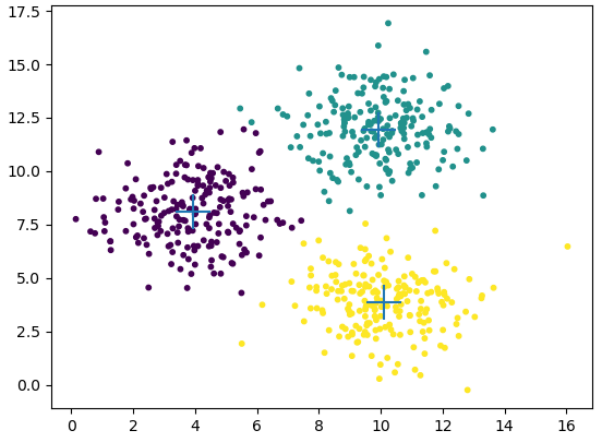
经过实验验证，采用优化后的均值向量初始化方法可以有效避免陷入局部最优。

而对于GMM算法来说，其实也存在类似的问题。而且相较于K-means算法，GMM算法常常需要更多的迭代次数。可以先使用K-means算法得到一个大致的簇中心向量，将其作为GMM算法的初始簇中心向量，以加速GMM算法的收敛。

4.2 K-means与GMM算法的对比

下面是使用同一组数据分别采用K-means和GMM算法得到的分类效果图（假设三类样本均服从多元高斯分布，高斯分布的中心向量分别为[4 8]，[10 4]，[10 12]）。





下面给出采用两种方法得到的中心向量。

算法	类别1	类别2	类别3
K-means	[3.90370845 8.09558768]	[9.89901316 11.95099421]	[10.10209492 3.87993093]
GMM	[3.93392178 8.11598331]	[9.91626945 11.94466274]	[10.09871307 3.87533911]

可以看出，采用两种方法得到不同类别的簇中心向量坐标与产生数据时高斯分布的中心向量很接近，结合图像来看，都是不错的聚类结果。

根据实验要求，我在代码里将GMM算法中每次迭代后的似然值进行了输出，如下图所示。

```
epoch  1: likelihood = -2733.9179179197
epoch  2: likelihood = -2733.7036651631
epoch  3: likelihood = -2733.6845171062
epoch  4: likelihood = -2733.6822167232
epoch  5: likelihood = -2733.6819206708
epoch  6: likelihood = -2733.6818814912
epoch  7: likelihood = -2733.6818762436
epoch  8: likelihood = -2733.6818755370
epoch  9: likelihood = -2733.6818754417
epoch 10: likelihood = -2733.6818754288
epoch 11: likelihood = -2733.6818754270
```

可以看到，似然值始终在增大，这与预期相符。

4.3 UCI数据集

使用UCI的Iris(鸢尾花)数据集，根据其4个属性：

- 花萼长度

- 花萼宽度
- 花瓣长度
- 花瓣宽度

来预测鸢尾花属于(Setosa, Versicolour, Virginica)三类中的哪一类。

最终的测试结果如下表所示。

算法	准确率
K-means	0.89
GMM	0.97

5. 结论

- K-Means算法实际上假设数据呈球状分布，而GMM算法则假设数据呈多元高斯分布，相比之下GMM算法的假设更一般
- K-Means的簇中心初始化对于最终的结果有很大的影响，如果初始的簇中心距离过近可能会使其陷入局部最优解，而采用优化后的簇中心优化算法可以较好的解决这个问题

6. 参考文献

[1] 周志华 著. 机器学习, 北京: 清华大学出版社, 2016.1

[2] 李航 著. 统计学习方法, 北京: 清华大学出版社, 2019.5

[3] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

7. 附录：源代码

kmeans.py

Python

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from itertools import permutations

# 超参数
mean = [[0, 8], [10, 4], [10, 20]]
cov = [np.diag([2, 2]), np.diag([2, 2]), np.diag([2, 2])]
train_size = 200
```

```python
train_size = 200


# 随机生成数据
def generate_data(k, means, covs, sample_num):
    """
    生成k类二维高斯分布的数据
    :param k: 类别数
    :param means: list[list], 代表每一类的均值
    :param sample_num: 每一类的样本数
    :return: 返回生成的数据
    """
    samples = np.zeros((1, 1))
    for i in range(k):
        data_temp = np.random.multivariate_normal(means[i], covs[i], sample_num)
        if i == 0:
            samples = data_temp
        else:
            samples = np.concatenate((samples, data_temp), axis=0)
    return samples


def initialization(X, k):
    dimension = X.shape[1]
    sample_size = X.shape[0]

    center = np.zeros((k, dimension))
    center[0, :] = X[0, :]
    _index = [0]
    i = 1
    while i < k:
        max_distance = 0
        max_index = 0
        for j in range(sample_size):
            if j in _index:
                continue
            temp_distance = 0
            for l in range(len(_index)):
                temp_distance += np.linalg.norm(X[j, :] - X[_index[l], :])
            if temp_distance > max_distance:
                max_distance = temp_distance
                max_index = j
        center[i, :] = X[max_index, :]
        _index.append(max_index)
        i += 1

    return center


```

```python
def kmeans(X, k, epsilon=1e-5):
    """
    K-means算法实现
    """
    dimension = X.shape[1]
    sample_size = X.shape[0]

    # 初始化中心点坐标和标签
    center = np.zeros((k, dimension))
    label = np.zeros(sample_size)

    # 均值坐标初始化优化
    center = initialization(X, k)

    while True:
        distance = np.zeros(k)

        # 根据中心重新给每个点贴分类标签
        for i in range(sample_size):
            for j in range(k):
                distance[j] = np.linalg.norm(X[i, :] - center[j, :])
            label[i] = np.argmin(distance)  # 把距某点最近的中心点作为它的分类标签

        # 根据每个点的标签计算新的中心点坐标
        new_center = np.zeros((k, dimension))
        count = np.zeros((k, 1))

        for i in range(X.shape[0]):
            new_center[int(label[i]), :] += X[i, :]  # 对每个类的所有点坐标求和
            count[int(label[i]), 0] += 1

        # 计算新的中心点坐标
        new_center /= count

        if np.linalg.norm(new_center - center) < epsilon:
            break
        else:
            center = new_center

    return X, label, center


def showResult(X, label, center):
    plt.scatter(X[:, 0], X[:, 1], c=label, s=10)
    plt.scatter(center[:, 0], center[:, 1], marker='+', s=500)
    plt.show()
```

```
106 def accuracy(real_label, class_label, k):
107     classes = list(permutations(range(k), k))
108     counts = np.zeros(len(classes))
109     for i in range(len(classes)):
110         for j in range(real_label.shape[0]):
111             if int(real_label[j]) == classes[i][int(class_label[j])]:
112                 counts[i] += 1
113     return np.max(counts) / real_label.shape[0]
114 
115 
116 def uci_iris():
117     data_set = pd.read_csv("./iris.csv")
118     classes = data_set['class']
119     X = np.zeros((data_set.shape[0], data_set.shape[1]-1))
120     X[:, :] = np.array(data_set.drop('class', axis=1), dtype=float)
121     label = np.zeros(data_set.shape[0])
122     for i in range(classes.shape[0]):
123         if classes[i] == 'Iris-setosa':
124             continue
125         elif classes[i] == 'Iris-versicolor':
126             label[i] = 1
127         elif classes[i] == 'Iris-virginica':
128             label[i] = 2
129     return X, label
130 
131 
132 train_x = generate_data(3, means=mean, covs=cov, sample_num=train_size)
133 
134 _result, _label, _center = kmeans(train_x, 3)
135 showResult(_result, _label, _center)
136 
137 # uci测试
138 test_x, real_label = uci_iris()
139 
140 _result, _label, _center = kmeans(test_x, 3)
141 print(accuracy(real_label, _label, 3))
```

GMM.py

Python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 from itertools import permutations
5 
6 # 超参数
7 mean = [[4, 8], [10, 4], [10, 12]]
```

```python
cov = [np.diag([2, 2]), np.diag([2, 2]), np.diag([2, 2])]
train_size = 200


# 随机生成数据
def generate_data(k, means, covs, sample_num):
    """
    生成k类二维高斯分布的数据
    :param k: 类别数
    :param means: list[list]，代表每一类的均值
    :param sample_num: 每一类的样本数
    :return: 返回生成的数据
    """
    samples = np.zeros((1, 1))
    for i in range(k):
        data_temp = np.random.multivariate_normal(means[i], covs[i], sample_num)
        if i == 0:
            samples = data_temp
        else:
            samples = np.concatenate((samples, data_temp), axis=0)
    return samples


def initialization(X, k):
    dimension = X.shape[1]
    sample_size = X.shape[0]

    center = np.zeros((k, dimension))
    center[0, :] = X[0, :]
    _index = [0]
    i = 1
    while i < k:
        max_distance = 0
        max_index = 0
        for j in range(sample_size):
            if j in _index:
                continue
            temp_distance = 0
            for l in range(len(_index)):
                temp_distance += np.linalg.norm(X[j, :] - X[_index[l], :])
            if temp_distance > max_distance:
                max_distance = temp_distance
                max_index = j
        center[i, :] = X[max_index, :]
        _index.append(max_index)
        i += 1

    return center
```

```python

def kmeans(X, k, epsilon=1e-5):
    dimension = X.shape[1]
    sample_size = X.shape[0]

    # 初始化中心点坐标和标签
    center = np.zeros((k, dimension))
    label = np.zeros(sample_size)

    # 均值坐标初始化优化
    center = initialization(X, k)

    while True:
        distance = np.zeros(k)

        # 根据中心重新给每个点贴分类标签
        for i in range(sample_size):
            for j in range(k):
                distance[j] = np.linalg.norm(X[i, :] - center[j, :])
            label[i] = np.argmin(distance)  # 把距某点最近的中心点作为它的分类标签

        # 根据每个点的标签计算新的中心点坐标
        new_center = np.zeros((k, dimension))
        count = np.zeros((k, 1))

        for i in range(X.shape[0]):
            new_center[int(label[i]), :] += X[i, :]  # 对每个类的所有点坐标求和
            count[int(label[i]), 0] += 1

        # 计算新的中心点坐标
        new_center /= count

        if np.linalg.norm(new_center - center) < epsilon:
            break
        else:
            center = new_center

    return X, label, center


def Gaussian(x, mu, sigma):
    return 1 / ((2 * np.pi) * pow(np.linalg.det(sigma), 0.5)) * np.exp(
        -0.5 * (x - mu).dot(np.linalg.pinv(sigma)).dot((x - mu).T))


def E_step(X, alpha, mu, sigma):
    sample_size = X.shape[0]
    cluster_size = mu.shape[0]
```

```
    gamma = np.zeros((sample_size, cluster_size))
    p = np.zeros(cluster_size)
    p_x = np.zeros(cluster_size)
    for i in range(sample_size):
        for j in range(cluster_size):
            p[j] = Gaussian(X[i], mu[j], sigma[j])
            p_x[j] = alpha[j] * p[j]
        for j in range(cluster_size):
            gamma[i, j] = p_x[j] / np.sum(p_x)
    return gamma


def M_step(X, k, gamma):
    sample_size = X.shape[0]
    feature_size = X.shape[1]

    mu = np.zeros((k, feature_size))
    sigma = np.zeros((k, feature_size, feature_size))

    for i in range(k):
        # 计算新均值向量
        mu[i] = np.sum(X * gamma[:, i].reshape((-1, 1)), axis=0) / np.sum(gamma, axis=0)[i]

        # 计算新协方差矩阵
        sigma[i] = 0
        for j in range(sample_size):
            sigma[i] += (X[j].reshape((1, -1)) - mu[i]).T.dot((X[j] - mu[i]).reshape((1, -1))) * gamma[j, i]
        sigma[i] = sigma[i] / np.sum(gamma, axis=0)[i]

    # 计算新混合系数
    alpha = np.sum(gamma, axis=0) / sample_size

    return alpha, mu, sigma


def likelihood_calculate(X, alpha, mu, sigma):
    sample_size = X.shape[0]
    k = len(alpha)
    likelihood = 0

    for j in range(sample_size):
        temp = 0
        for i in range(k):
            temp += alpha[i] * Gaussian(X[j], mu[i], sigma[i])
        likelihood += np.log(temp)

```

```
151     return likelihood
152 
153 
154 def GMM(data, k, epsilon=1e-5):
155     epoch = 0
156     sample_size = data.shape[0]
157     feature_size = data.shape[1]
158 
159     # 初始化alpha, mu, sigma
160     alpha = np.ones(k) / k
161     _, _, mu = kmeans(data, k)  # 利用kmeans算法初始化簇中心坐标
162     sigma = np.full((k, feature_size, feature_size), np.diag(np.full(feature_siz
    e, 0.1)))
163 
164     while True:
165         epoch += 1
166         last_alpha, last_mu, last_sigma = alpha, mu, sigma
167 
168         # E_step
169         gamma = E_step(data, alpha, mu, sigma)
170 
171         # M_step
172         alpha, mu, sigma = M_step(data, k, gamma)
173 
174         # 计算似然值
175         likelihood = likelihood_calculate(data, alpha, mu, sigma)
176 
177         print("epoch {:>2d}: likelihood = {:+.10f}".format(epoch, likelihood))
178 
179         if np.linalg.norm(alpha - last_alpha) < epsilon and np.linalg.norm(mu -
     last_mu) < epsilon and np.linalg.norm(sigma - last_sigma) < epsilon:
180             break
181 
182     label = np.argmax(gamma, axis=1)
183 
184     return alpha, mu, sigma, label
185 
186 
187 def showResult(X, label, center):
188     plt.scatter(X[:, 0], X[:, 1], c=label, s=10)
189     plt.scatter(center[:, 0], center[:, 1], marker='+', s=500)
190     plt.show()
191 
192 
193 def accuracy(real_label, class_label, k):
194     classes = list(permutations(range(k), k))
195     counts = np.zeros(len(classes))
196     for i in range(len(classes)):
```

```
            for j in range(real_label.shape[0]):
                if int(real_label[j]) == classes[i][int(class_label[j])]:
                    counts[i] += 1
        return np.max(counts) / real_label.shape[0]


def uci_iris():
    data_set = pd.read_csv("./iris.csv")
    classes = data_set['class']
    X = np.zeros((data_set.shape[0], data_set.shape[1]-1))
    X[:, :] = np.array(data_set.drop('class', axis=1), dtype=float)
    label = np.zeros(data_set.shape[0])
    for i in range(classes.shape[0]):
        if classes[i] == 'Iris-setosa':
            continue
        elif classes[i] == 'Iris-versicolor':
            label[i] = 1
        elif classes[i] == 'Iris-virginica':
            label[i] = 2
    return X, label


train_x = generate_data(3, means=mean, covs=cov, sample_num=train_size)

_alpha, _mu, _sigma, _label = GMM(train_x, 3)
showResult(train_x, _label, _mu)

# uci测试
test_x, real_label = uci_iris()

_alpha, _mu, _sigma, _label = GMM(test_x, 3)
print(accuracy(real_label, _label, 3))
```