哈尔滨工业大学计算学部

实验报告

课程名称：机　　器　　学　　习

课程类型：　　选　　　　修

实验题目：实现k-means聚类方法和混合高斯模型

学号：1190201019

姓名：罗家乐

一、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。

二、实验要求及实验环境

实验要求

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）。

实验环境

Programming Language: python 3.9.7 64-bit

Imported Model: numpy matplotlib sklearn.datasets sklearn.metrics

三、设计思想（本程序中的用到的主要算法及数据结构）

EM算法思想

本次实验涉及到的Kmeans模型与GMM模型，都通过EM算法进行参数学习，达成最大似然。故此处先对EM算法进行分析：

EM 算法概述

EM算法具有E、M两步：

Repeat until convergence {
(E-step) For each i, set

$$Q_i\left(z^{(i)}\right):=p\left(z^{(i)} \mid x^{(i)} ; \theta\right)$$

(M-step) Set

$$\theta:=\arg \max _\theta \sum_i \sum_{z^{(i)}} Q_i\left(z^{(i)}\right) \log \frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{Q_i\left(z^{(i)}\right)}$$

}

E步

在E步，EM算法构造z的分布Q，使其满足z的后验分布，使得KL散度为0，构造似然在现有θ下的一个下确界。

M步

在M步，通过计算θ，得到一个使得这个下界最大化的θ,此时,由于θ变化，原先的下确界不一定再是下确界，故在此进行E步。直到收敛。

EM的证明

设定我们的似然为$\ell(\theta)$，一个关于θ的函数，考虑到隐变量z，我们得到以下式子。

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^m \log p(x ; \theta) \\
&= \sum_{i=1}^m \log \sum_z p(x, z ; \theta)
\end{aligned}$$

在M步，我们给定一个z的分布Q，有：

$$\begin{aligned}
\sum_i \log p\left(x^{(i)} ; \theta\right) &= \sum_i \log \sum_{z^{(i)}} p\left(x^{(i)}, z^{(i)} ; \theta\right) \\
&= \sum_i \log \sum_{z^{(i)}} Q_i\left(z^{(i)}\right) \frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{Q_i\left(z^{(i)}\right)} \\
&\geq \sum_i \sum_{z^{(i)}} Q_i\left(z^{(i)}\right) \log \frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{Q_i\left(z^{(i)}\right)}
\end{aligned}$$

通过构造以下函数，我们可以按照Jensen不等式进行第二行到第三行的推导，

$$f\left(\mathrm{E}_{z^{(i)} \sim Q_i}\left[\frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{Q_i\left(z^{(i)}\right)}\right]\right) \geq \mathrm{E}_{z^{(i)} \sim Q_i}\left[f\left(\frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{Q_i\left(z^{(i)}\right)}\right)\right]$$

当X为常量时，等号成立（Jensen不等式性质，也可以从信息论角度推导）。
于是，在E步，我们计算出下确界，及分布Q：

$$\frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{Q_i\left(z^{(i)}\right)}=c$$

也即

$$Q_i\left(z^{(i)}\right)=c\, p\left(x^{(i)}, z^{(i)} ; \theta\right)$$

且

$$\sum_z Q_i\left(z^{(i)}\right)=1$$

故

$$\begin{aligned} Q_i\left(z^{(i)}\right) &=\frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{\sum_z p\left(x^{(i)}, z ; \theta\right)} \\ &=\frac{p\left(x^{(i)}, z^{(i)} ; \theta\right)}{p\left(x^{(i)} ; \theta\right)} \\ &=p\left(z^{(i)} \mid x^{(i)} ; \theta\right) \end{aligned}$$

证毕。

这个过程十分有趣，在E我们设置Q，构造$\ell(\theta)$的下确界，而后在M，我们计算θ，使得这个下确界取得最大值。然后重复这个过程，直至收敛。我们设下界为$h_t(\theta)$,有：

$$\ell(\theta^{t+1}) \geq h_t(\theta^{t+1}) \geq h_t(\theta^t)=\ell(\theta^t)$$

在迭代过程中，$\ell(\theta^t)$将随迭代次数增长，最终收敛到一个局部或全局最优解。

Kmeans

Kmeans概述

Kmeans是一种简单的聚类算法，其目标在于训练出几个聚类的中心centeroids，然后按照样本点离中心点的远近划分类别，达成对原数据的聚类与新数据的分类。

算法概述如下：

重复下列过程直到收敛（convergence）: {
(E-步骤) 对每个样本点贴标签:

$$c^{(i)} := \arg\min_j \left\| x^{(i)} - \mu_j \right\|^2$$

(M -步骤) 更新参数(聚类中心):

$$\mu_j := \frac{\sum_{i=1}^m 1\left\{c^{(i)} = j\right\} x^{(i)}}{\sum_{i=1}^m 1\left\{c^{(i)} = j\right\}}$$

}

对应EM算法:

随机初始化几个中心的位置（一般从样本中随机抽取）。

E步

按照距离远近划分样本点，将其划分给不同的聚类。

```
def Exception(self):
    for k in range(self.data_shape[0]):
        sample = self.samples[k]
        nearest_centroid = -1
        nearest_distance = 0
        for i in range(self.n_cluster):
            center = self.centroids[i]
            dist = self.distance(sample,center)
            if dist < nearest_distance or nearest_centroid==-1:
                nearest_centroid = i
                nearest_distance = dist
        self.labels[k] = nearest_centroid
```

M步

根据聚类内的样本的属性，更新聚类中心的属性。

```
def Maximum(self):
        for i in range(self.n_cluster):
            self.centroids[i] = np.average(self.samples[self.labels == i],axis=0)
```

循环直至收敛:

```
for i in range(self.max_iter):
    self.Exception()
    old_score = self.score()
    self.Maximum()
    if old_score == self.score():
        break
```

评价标准

在Kmeans中我们使用mean distortion distance作为评价标准：

```
def score(self):
    return sum(np.min(cdist(self.samples, self.centroids, 'euclidean'), axis=1))/self.data_shap
```

评价与分析

优缺点

Kmeans在面对较为分散，交叉不明显、分布较为贴近原形的分布有较好的表现。然而，由于对样本进行简单的硬划分，在交叉较明显的多个分布中难以对其进行分离；而近圆形的判断范围又使其面对分布的形式较特殊，如协方差某一维较大的高斯分布时，难以进行较好的聚类。

Kmeans++

此外，Kmeans还对初始化有较高的敏感度，于是查阅资料，使用Kmeans++初始化算法：

```
def initial_centroids(self):
    id = []
    first_id = random.sample(range(self.data_shape[0]), 1)
    id.append(first_id[0])
    center = self.samples[first_id]
    for j in range(self.n_cluster-1):
        all_distance = np.zeros(self.data_shape[0])
        for i in range(self.data_shape[0]):
            if not i in id:
                all_distance[i] = self.distance(self.samples[i],center)
        dist_p = all_distance/all_distance.sum()
        next_id = np.random.choice(range(self.data_shape[0]), 1, p=dist_p)
        id.append(next_id[0])
        center = np.average(self.samples[id,:],axis=0)
    centroids = self.samples[id,:]
    return centroids
```

通过在选取初始点时，尽可能使初始点分散（通过在随机抽取中给距离较远的点更大的权重），提升聚类效果。

GMM

GMM概述

高斯混合模型，使用多个高斯分布来拟合一组样本，通过EM学习几个高斯分布的参数以及样本点属于每个高斯的先验，来达成聚类原始数据及判别新样本所属的目的。

GMM算法概述:

重复下列过程直到收敛（convergence）: {
$(\mathrm{E}$-步骤) 对每个 i, j, 设

$$\mathrm{w}_{\mathrm{j}}^{(\mathrm{i})}:=\mathrm{p}\left(\mathrm{z}^{(\mathrm{i})}=\mathrm{j} \mid \mathrm{x}^{(\mathrm{i})} ; \phi, \mu, \Sigma\right)$$

(M -步骤) 更新参数:

$$\begin{aligned}
\phi_{\mathrm{j}} &=\frac{1}{\mathrm{m}} \sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{w}_{\mathrm{j}}^{(\mathrm{i})} \\
\mu_{\mathrm{j}} &=\frac{\sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{w}_{\mathrm{j}}^{(\mathrm{i})} \mathrm{x}^{(\mathrm{i})}}{\sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{w}_{\mathrm{j}}^{(\mathrm{i})}} \\
\Sigma_{\mathrm{j}} &=\frac{\sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{w}_{\mathrm{j}}^{(\mathrm{i})}\left(\mathrm{x}^{(\mathrm{i})}-\mu_{\mathrm{j}}\right)\left(\mathrm{x}^{(\mathrm{i})}-\mu_{\mathrm{j}}\right)^{\mathrm{T}}}{\sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{w}_{\mathrm{j}}^{(\mathrm{i})}}
\end{aligned}$$

}

E步

按照样本点属于每个分布的概率密度，归一化处理，得出每个样本点属于每一类的概率（GMM）。即对应分布Q的给出，得到似然的下确界。

```
def Exception(self):
    self.update_weight()

def update_weight(self):
    for i in range(self.data_shape[0]):
        sample = self.samples[i]
        for j in range(self.n_cluster):
            gaussion = self.gaussions[j]
            self.weight[i,j]=gaussion.pdf(sample)*self.phi[j]
        self.weight[i,:] /= sum(self.weight[i,:])
```

M步

按照样本点的所属，按权重加和样本点的属性得到每个高斯分布的均值，计算对应方差；同时算出样本属于每个高斯概率的先验。

```
def Maximum(self):
    self.update_phi()
    self.update_Gaussions()
def update_phi(self):
    self.phi = self.weight.sum(axis=0)/self.data_shape[0]

def update_Gaussions(self):
    for j in range(self.n_cluster):
        gaussion = self.gaussions[j]
        means = self.samples.T.dot(self.weight[:,j])/(self.phi[j]*self.data_shape[0])
        Z = self.samples-means
        cov = np.dot(Z.T*self.weight[:,j], Z) / (self.phi[j]*self.data_shape[0])+0.001*np.eye(s
        gaussion.update(means,cov)
```

评价标准

在GMM中我们使用直接使用平均似然作为评价标准：

```
def score(self):
    score = 0
    for i in range(self.data_shape[0]):
        temp = 0
        for j in range(self.n_cluster):
            temp = temp + self.gaussions[j].pdf(self.samples[i])*self.phi[j]
        score = score + np.log(temp)
    return score/self.data_shape[0]
```

评价与分析

优缺点

相较于Kmeans，GMM加入软分类，不再简单贴标签，而是按权重分割样本，更能适应交错较大的样本集合，同时，由于高斯可学习的协方差，又能够对不同形状的数据样本分布进行聚类。但由于权重、高斯模型的加入，其计算较为耗时。

概率崩塌

混合高斯相对单个高斯模型，由于权重的存在，当某个样本与其余样本差距较大，且初始化时样本与某个高斯模型均值高度重合时，便会发生概率崩塌。在学习过程中，E步：由于该点过于靠近该高斯，其概率密度将非常大，权重较高，而其余点的权重就会相对低；M步：更新高斯分布的参数时，其主要数值来源将为这个点，故该高斯分布将进一步偏向这个点，且方差缩小。如此反复，就会得到一个收缩到一个点上的高斯分布，影响学习效果。

对此，需要限制协方差的缩小，避免概率崩塌。

初始化

GMM的初始化同样选择了Kmeans中使用的Kmeans++算法。

四、实验、结果与分析

实验一 kmeans聚类测试效果

多次使用numpy.random提供的multivariate_normal函数，给定均值和方差，生成4个高斯分布下产生的各30共120个样本点，分别使用klearn.cluster._kmeans提供的Kmeans模型与自己实现的Kmeans模型进行聚类，查看结果。

4个高斯分布的均值与方差分别为：

$1.[0,1]\begin{pmatrix}0.1 & 0\\ 0 & 0.1\end{pmatrix}$

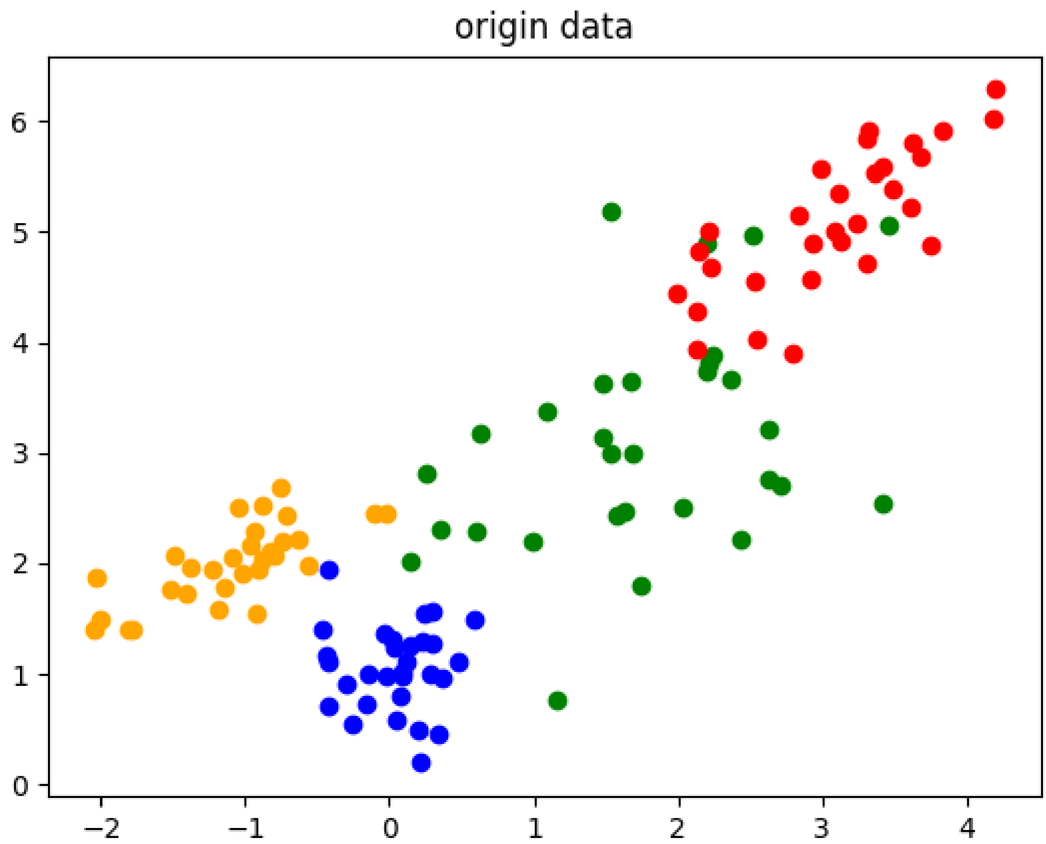
$2.[2,3]\begin{pmatrix}0.8 & 0.2\\ 0.2 & 0.8\end{pmatrix}$

$3.[3,5]\begin{pmatrix}0.3 & 0.4\\ 0.4 & 0.3\end{pmatrix}$

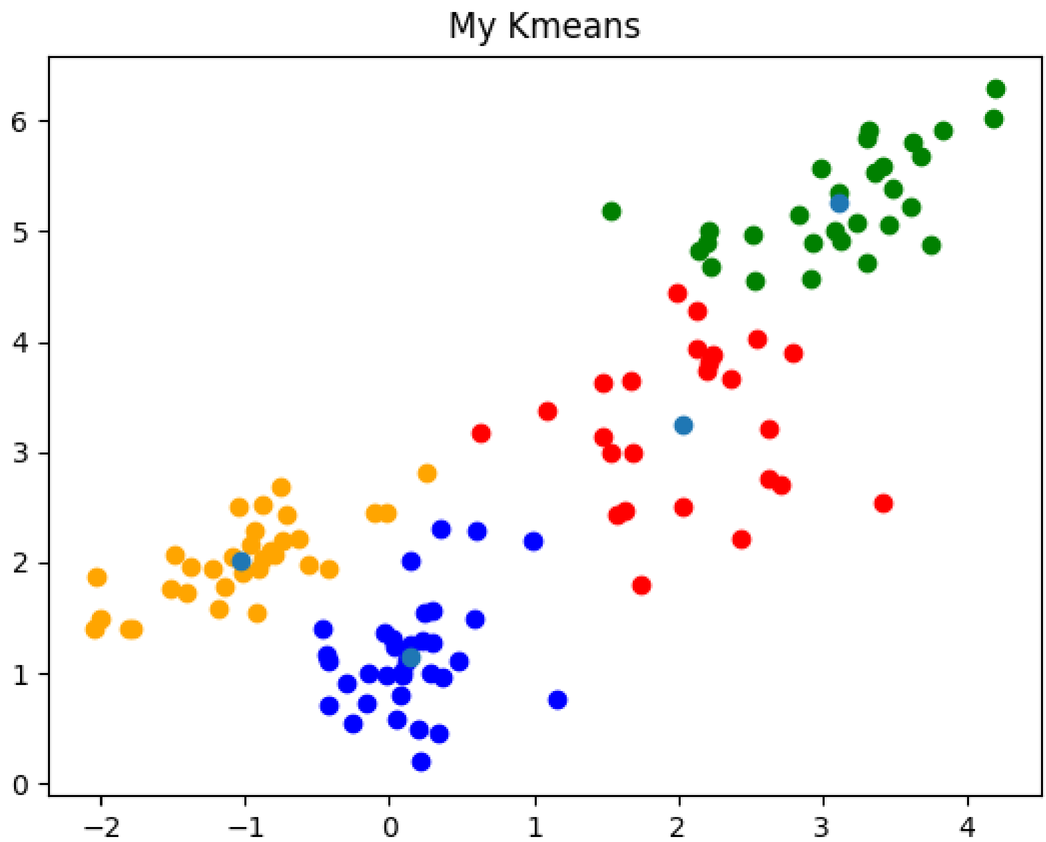
$4.[-1,2]\begin{pmatrix}0.5 & 0.3\\ 0.3 & 0.1\end{pmatrix}$

设置聚类数量为4，分别使用自己实现的My_Kmeans，数据分析库提供的Kmeans进行聚类，并与原始数据进行比对。

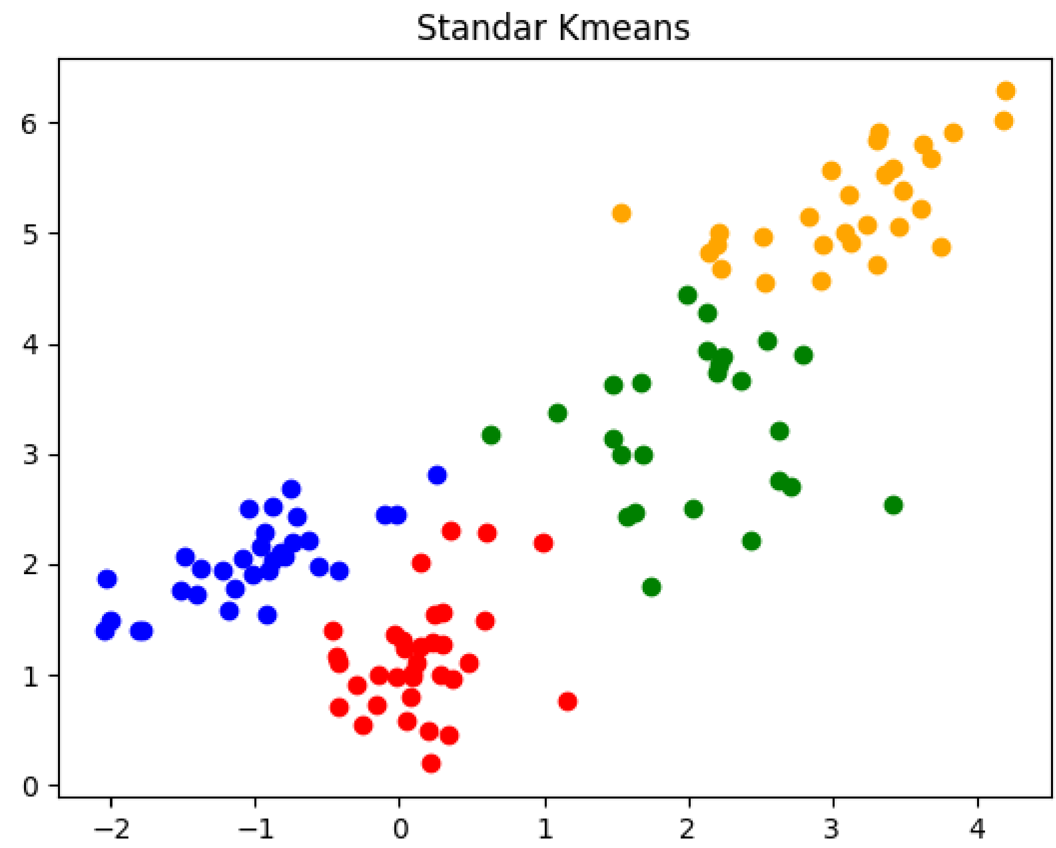
生成数据：



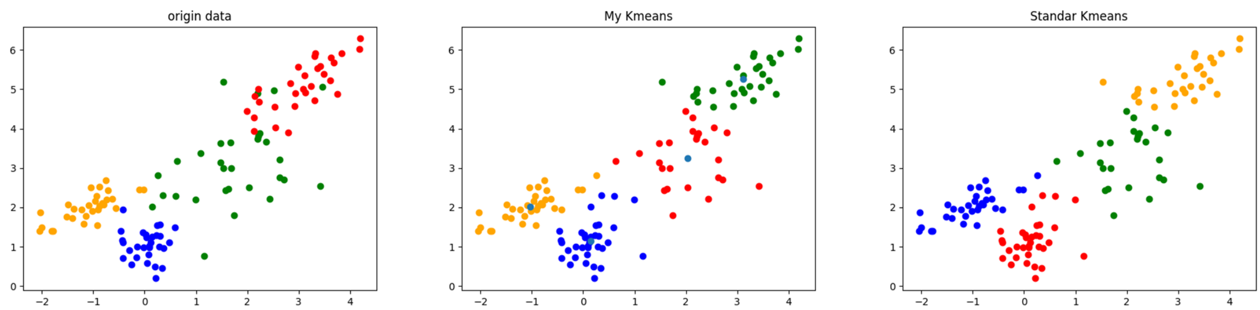
使用自己实现的Kmeans进行聚类，设置最大迭代轮数为300（提前收敛），使用Kmeans++进行初始化，执行10轮，选择平局扭曲程度最小的一轮作为结果。



使用标准的Kmeans库进行聚类：



进行对比：

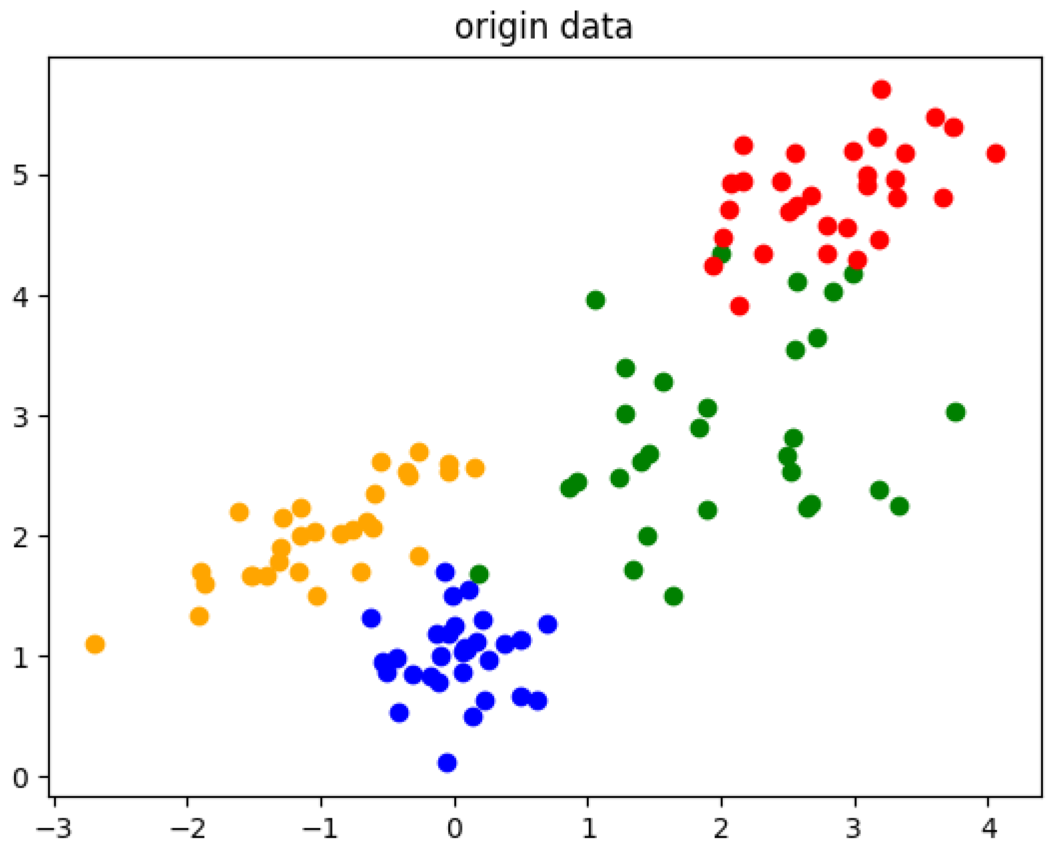


如上图所示，自己实现的Kmeans（My_Kmeans）与标准库Kmeans具有相当的聚类效果，都能得到与原数据分类相近的结果。

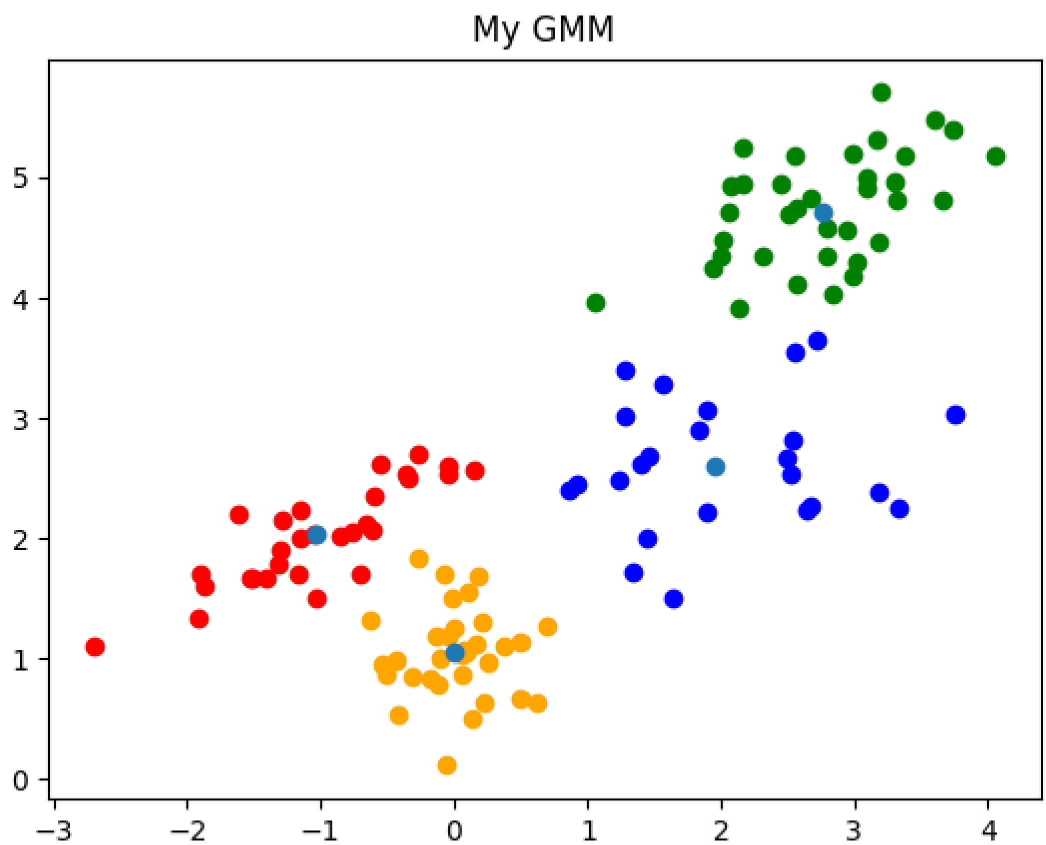
实验二 GMM聚类测试效果

使用Kmeans实验中的参数，生成数据，进行测试，并于sklearn中的标准GMM模型的聚类结果进行比对。

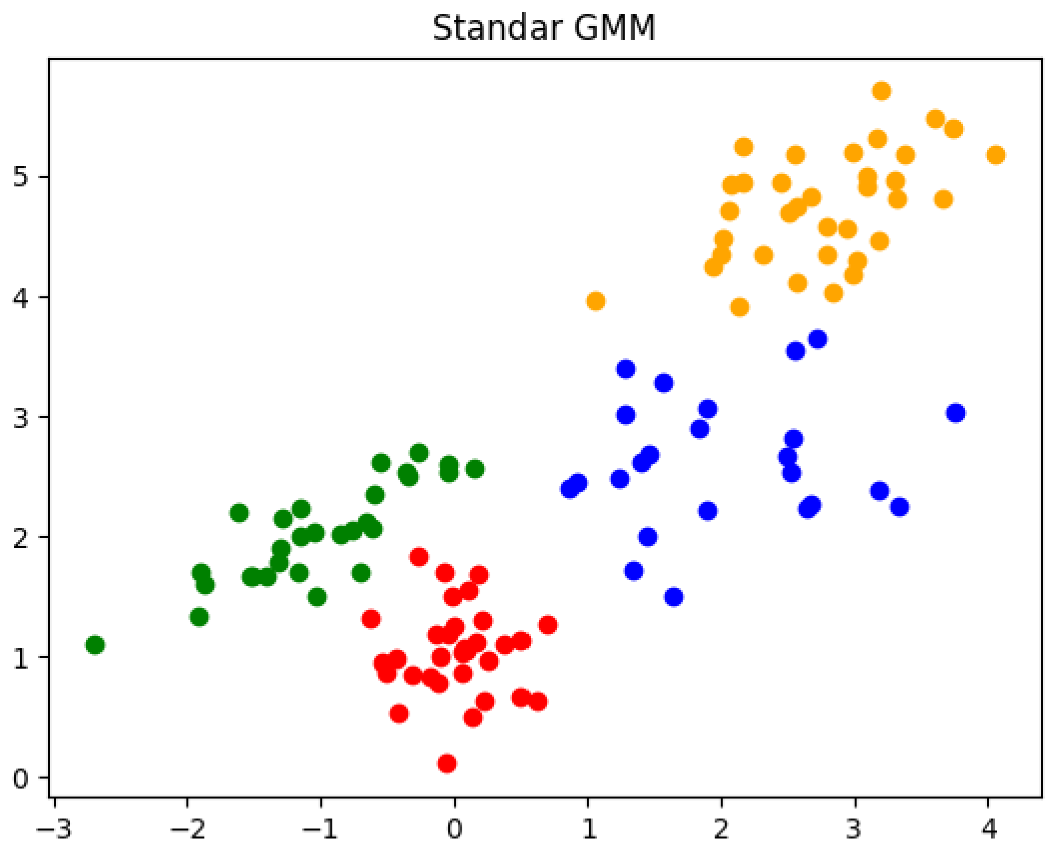
生成数据：



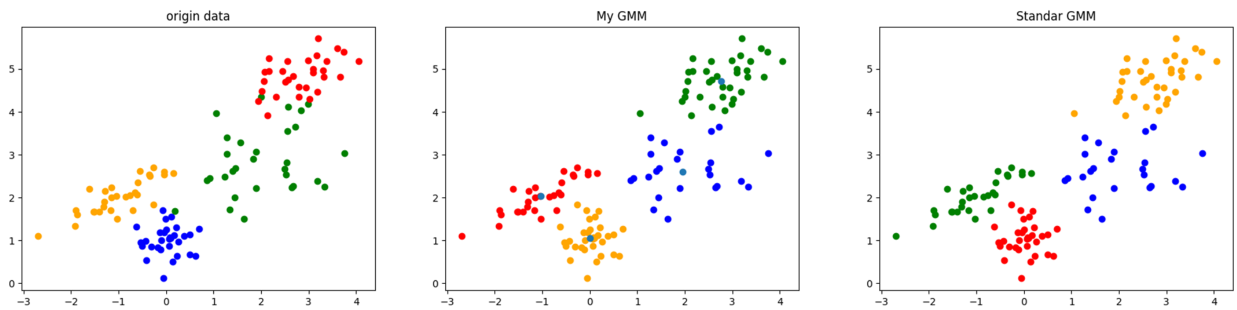
使用自己实现的GMM模型进行拟合，最大迭代次数300，初始协方差为标准阵，使用Kmeans++进行初始化，执行3轮，选择似然最大的一轮结果。



调用sklearn提供的GMM进行聚类：



进行对比：



可以观测到，由于原始数据在右上角的两个高斯类（红色与绿色）之间存在一定程度的交叉，自己训练成的GMM与标注库的GMM均不能对其进行有效的分类，除此之外，两个GMM均能够将数据样本呢聚类成类似原始数据多个高斯分布的情况，均值（由天蓝色点标注）亦符合分布的直观质心，因此聚类效果相当不错。

实验三 UCI数据集

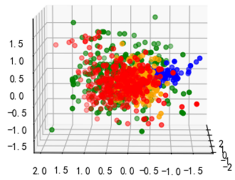
为了实践GMM模型在实际问题上的表现，在UCI上寻得数据集：

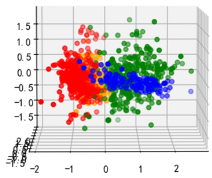
"tripadvisor_review.csv"

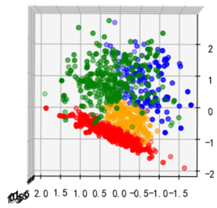
该数据集提供了980个游客对其在东亚进行的10个类别的景点的平均评价（0-4）。我将用GMM算法对游客进行聚类，便于后续的旅游广告推送。

由于数据维数过多，不便于直观展示聚类结果，且10个类别的景点之间亦可能存在某种关联性，故采用PCA（标准库）降维至3维，再进行聚类。得到PCA降维结果的3个维度的方差贡献率为：[0.4252009 0.17723144 0.12453292]，仅表达了70%左右的数据特征，但处于简单实验与可视化角度考虑，仍旧使用3维降维。

聚类后，使用My GMM进行聚类，由于样本无标签，故尝试聚类为4类（未进行进一步探索，使用轮廓系数等……）。最大迭代次数为400（预计提前收敛），得到的聚类效果如下：

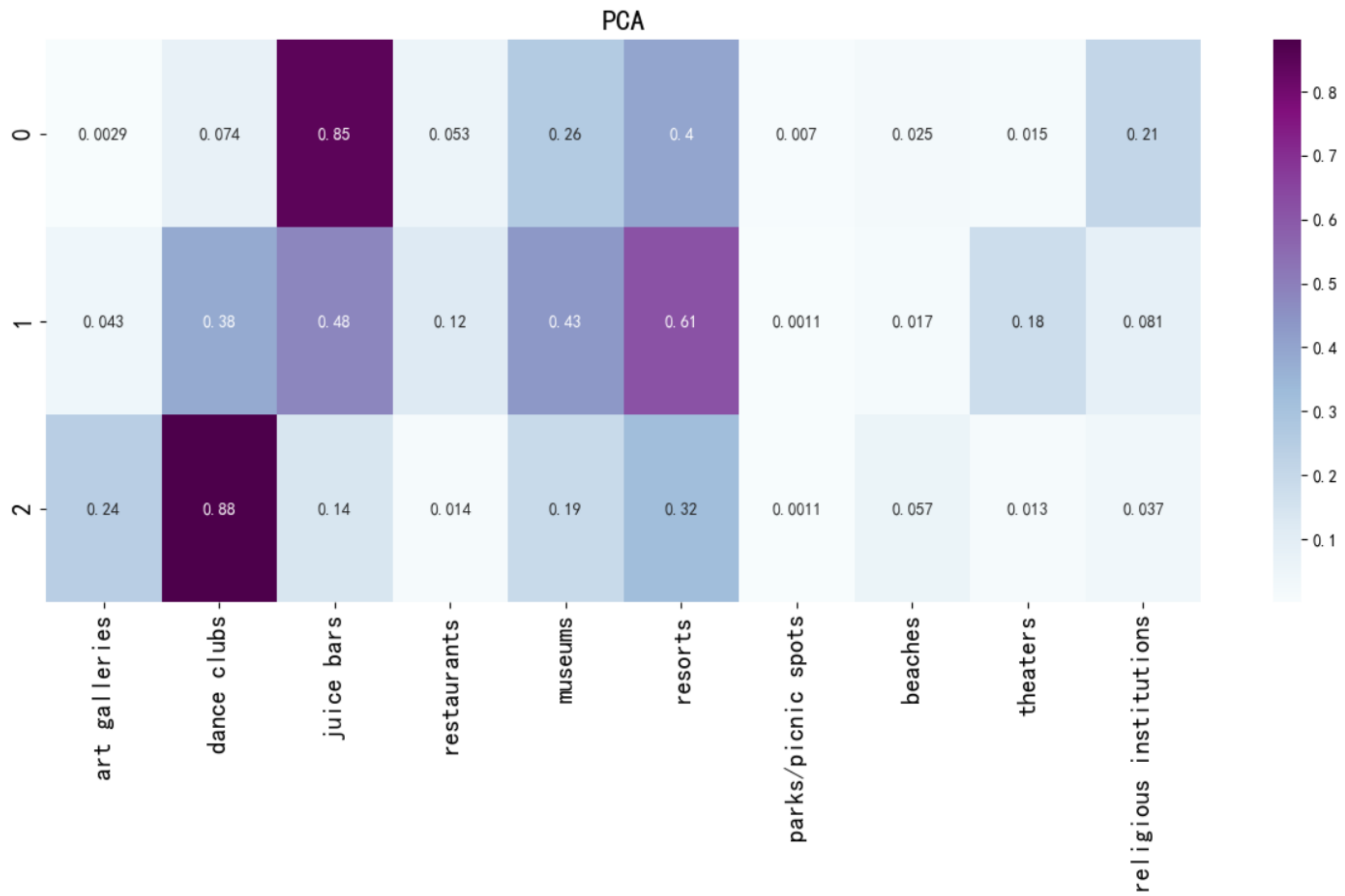




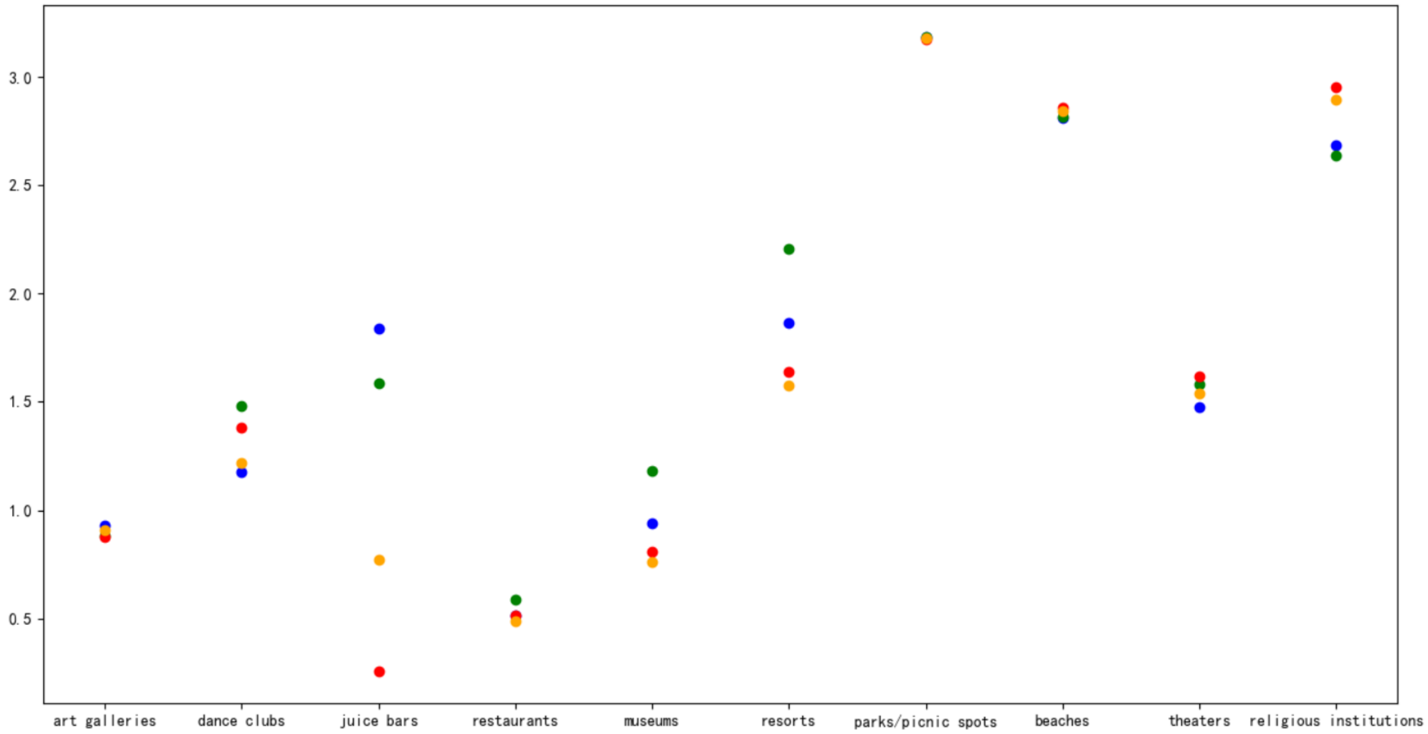


对聚类结果进行分析：

查看原特征与主成分的相关系数：



计算几个聚类均值对应的原特征数据：



从原始特征图中可以看出：

- 橙色聚类对酒吧、饭店、画廊等场所的评分很低，而对户外、舞厅、沙滩具有偏爱，应该试图给这类人群推送户外、运动的旅行计划。
- 蓝色聚类对酒吧具有较为突出的喜好，可以为其推送酒吧类旅行地点。
- 红色聚类与橙色聚类类似，但其偏好较红色聚类不那么复有特性，故对红色聚类喜好以外的旅行计划也有一定接受度。
- 绿色聚类偏好自然景观，应当推送风景奇观类旅游计划。

五、结论

在本次对Kmeans与GMM的实验中：

1. 通过手动实现Kmeans与GMM代码，对Kmeans与GMM的算法细节有了较深的理解，且对算法中的一些计算问题有了理解与解决方案：奇异矩阵——加噪声，概率崩塌——重随，初始值不佳——Kmeans++与多个模型取最优。
2. 通过与标准库的Kmeans与GMM算法的对比，逐步改进模型，获得较好的成果，对于使用Kmeans与GMM更加娴熟。
3. 通过在UCI数据集上使用自己实现的GMM模型，在整个数据分析过程中，经历了PCA降维、选择维数、返回原始数据、数据解析几个步骤，实现了一次完整的机器学习——数据分析流程，提高了个人的机器学习项目执行能力。

六、参考文献

七、附录：源代码（带注释）

Kmeans代码

```python
import numpy as np
from scipy.spatial.distance import cdist
import random


class My_Kmeans:
    '''自己实现的Kmeans聚类模型'''

    def __init__(self,n_cluster,max_iter=300,distance = None) -> None:
        '''初始化'''
        self.n_cluster = n_cluster
        self.max_iter = max_iter
        if distance == None:
            self.distance = self.euclidean
        else:
            self.distance = distance

    '''欧氏距离'''
    @staticmethod
    def euclidean(A,B):
        vec_dist = A-B
        scale_distance = vec_dist.dot(vec_dist.T)
        return scale_distance

    def initial_centroids(self):
        '''初始化中心点'''
        id = []
        first_id = random.sample(range(self.data_shape[0]), 1)
        id.append(first_id[0])
        center = self.samples[first_id]
        for j in range(self.n_cluster-1):
            all_distance = np.zeros(self.data_shape[0])
            for i in range(self.data_shape[0]):
                if not i in id:
                    all_distance[i] = self.distance(self.samples[i],center)
            dist_p = all_distance/all_distance.sum()
            next_id = np.random.choice(range(self.data_shape[0]), 1, p=dist_p)
            id.append(next_id[0])
            center = np.average(self.samples[id,:],axis=0)
        centroids = self.samples[id,:]
        return centroids

    def Exception(self):
        '''E步给样本贴标签'''
        for k in range(self.data_shape[0]):
            sample = self.samples[k]
            nearest_centroid = -1
            nearest_distance = 0
            for i in range(self.n_cluster):
                center = self.centroids[i]
                dist = self.distance(sample,center)
                if dist < nearest_distance or nearest_centroid==-1:
                    nearest_centroid = i
                    nearest_distance = dist
```

```
            self.labels[k] = nearest_centroid

    def Maximum(self):
        '''最大化似然'''
        for i in range(self.n_cluster):
            self.centroids[i] = np.average(self.samples[self.labels == i],axis=0)

    def fit(self,X):
        '''拟合数据'''
        self.data_shape = X.shape
        self.samples = X.copy()
        self.centroids = self.initial_centroids()
        self.labels = np.zeros((len(self.samples),))

        for i in range(self.max_iter):
            self.Exception()
            old_score = self.score()
            self.Maximum()
            if old_score == self.score():
                break

    def score(self):
        '''评价标准 mean distortion'''
        return sum(np.min(cdist(self.samples, self.centroids, 'euclidean'), axis=1))/self.data_
```

GMM源码

```python
import numpy as np
import random
from scipy.spatial.distance import euclidean

class Gaussion:
    '''高斯模型'''

    def __init__(self,N,cov) -> None:
        self.N = N
        self.means = np.zeros((N,))
        self.cov = np.eye(N)*cov

    def update(self,means,cov):
        '''更新均值、协方差'''
        self.means = means
        self.cov = cov
        return self

    def pdf(self,A):
        '''概率密度'''
        Z = A-self.means
        half = (np.power((2*np.pi),-self.N/2)*
                np.power(np.linalg.det(self.cov),-1/2))
        half2 = np.exp(-1/2*Z.T.dot(np.linalg.inv(self.cov)).dot(Z))
        return half*half2

    def means(self):
        '''返回均值'''
        return self.means

class My_GMM:

    def __init__(self,n_cluster,max_iter=300,cov=0.5) -> None:
        '''初始化'''
        self.n_cluster = n_cluster
        self.max_iter = max_iter
        self.cov = cov

    @staticmethod
    def euclidean(A,B):
        '''欧氏距离'''
        vec_dist = A-B
        scale_distance = vec_dist.dot(vec_dist.T)
        return scale_distance

    def initial_gaussions(self):
        '''初始化高斯'''
        id = []
        first_id = random.sample(range(self.data_shape[0]), 1)
        id.append(first_id[0])
        center = self.samples[first_id]
        for j in range(self.n_cluster-1):
            all_distance = np.zeros(self.data_shape[0])
            for i in range(self.data_shape[0]):
```

```
                if not i in id:
                    all_distance[i] = euclidean(self.samples[i],center)
            dist_p = all_distance/all_distance.sum()
            next_id = np.random.choice(range(self.data_shape[0]), 1, p=dist_p)
            id.append(next_id[0])
            center = np.average(self.samples[id,:],axis=0)
        means = self.samples[id,:]
        gaussions = [Gaussion(self.data_shape[1],self.cov).update(means[i],np.eye(self.data_sha
        return gaussions

    def update_phi(self):
        '''更新phi'''
        self.phi = self.weight.sum(axis=0)/self.data_shape[0]

    def update_Gaussions(self):
        '''更新高斯模型'''
        for j in range(self.n_cluster):
            gaussion = self.gaussions[j]
            means = self.samples.T.dot(self.weight[:,j])/(self.phi[j]*self.data_shape[0])
            Z = self.samples-means
            cov = np.dot(Z.T*self.weight[:,j], Z) / (self.phi[j]*self.data_shape[0])+0.001*np.e
            gaussion.update(means,cov)

    def update_weight(self):
        '''更新权重'''
        for i in range(self.data_shape[0]):
            sample = self.samples[i]
            for j in range(self.n_cluster):
                gaussion = self.gaussions[j]
                self.weight[i,j]=gaussion.pdf(sample)*self.phi[j]
            self.weight[i,:] /= sum(self.weight[i,:])

    def Exception(self):
        '''期望'''
        self.update_weight()

    def Maximum(self):
        '''最大化'''
        self.update_phi()
        self.update_Gaussions()

    def fit(self,X):
        '''拟合数据'''
        self.data_shape = X.shape
        self.samples = X.copy()
        self.gaussions = self.initial_gaussions()
        self.weight = np.ones((self.data_shape[0],self.n_cluster))/self.n_cluster
        self.phi = np.zeros((self.n_cluster,))
        self.update_phi()

        for i in range(self.max_iter):
            self.Exception()
            old_score = self.score()
            self.Maximum()
            if old_score == self.score():
```

```
                break

        self.labels = np.argmax(self.weight,axis=1)

    def score(self):
        '''评分'''
        score = 0
        for i in range(self.data_shape[0]):
            temp = 0
            for j in range(self.n_cluster):
                temp = temp + self.gaussions[j].pdf(self.samples[i])*self.phi[j]
            score = score + np.log(temp)
        return score/self.data_shape[0]

    def means(self):
        '''均值'''
        return  np.array([self.gaussions[i].means for i in range(self.n_cluster)])
```

实验代码

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import multivariate_normal
from My_Kmeans import My_Kmeans
from My_GMM import My_GMM
from sklearn.mixture import GaussianMixture as GMM
from sklearn.cluster._kmeans import KMeans
from sklearn.decomposition import PCA
import pandas as pd
import seaborn as sns
import os

def make_blobs(n,means,covs,sample_numbers):
    '''生成高斯数据'''
    X = []
    y_true = []
    for i in range(n):
        x = multivariate_normal(means[i],covs[i],sample_numbers)
        X.append(x)
        y_true.append(np.ones(sample_numbers)*i)
    X = np.concatenate(X,axis=0)
    y_true = np.concatenate(y_true)
    return X,y_true

def show_correlation(components,name):
    '''展示原数据与主成分的相关性'''
    df_cm = pd.DataFrame(np.abs(components), columns=name)
    plt.figure(figsize = (8,4))
    ax = sns.heatmap(df_cm, annot=True, cmap="BuPu")
    # 设置y轴的字体的大小
    ax.yaxis.set_tick_params(labelsize=15)
    ax.xaxis.set_tick_params(labelsize=15)
    plt.title('PCA', fontsize='xx-large')
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus']=False
    plt.show()

def k_means(k):
    '''Kmeans模型测试'''
    X, y_true = make_blobs(4,[[0,1],[2,3],[3,5],[-1,2]],\
        [[[0.1,0],[0,0.1]],[[0.8,0.2],[0.2,0.8]],[[0.3,0.4],[0.4,0.3]],[[0.5,0.3],[0.3,0.1]]],3
    distortion = np.Infinity
    for i in range(10):
        k_means = My_Kmeans(k,300)
        k_means.fit(X)
        score = k_means.score()
        print(score)
        if distortion > score:
            distortion = score
            labels = k_means.labels
            centers = k_means.centroids

    color = ['b','g','r','orange','gray']
    plt.figure(1)
```

```
    for index in range(k):
        A = X[(labels==index),0]
        B = X[(labels==index),1]
        plt.scatter(A,B,c=color[index])
    plt.scatter(centers[:,0],centers[:,1])
    plt.title("My Kmeans")

    k_means = KMeans(k,max_iter=500)
    k_means.fit(X)
    labels = k_means.predict(X)

    color = ['b','g','r','orange','gray']
    plt.figure(2)
    for index in range(k):
        A = X[(labels==index),0]
        B = X[(labels==index),1]
        plt.scatter(A,B,c=color[index])
    plt.title("Standar Kmeans")

    plt.figure(3)
    for index in range(k):
        A = X[(y_true==index),0]
        B = X[(y_true==index),1]
        plt.scatter(A,B,c=color[index])
    plt.title("origin data")
    plt.show()

def Gmm(k):
    '''高斯混合模型测试'''
    X, y_true = make_blobs(4,[[0,1],[2,3],[3,5],[-1,2]],\
        [[[0.1,0],[0,0.1]],[[0.8,0.2],[0.2,0.8]],[[0.3,0.4],[0.4,0.3]],[[0.5,0.3],[0.3,0.1]]],3

    likelihood = -np.Infinity
    for i in range(3):
        gmm = My_GMM(k,500,cov=1)
        gmm.fit(X)
        score = gmm.score()
        print(score)
        if likelihood < score:
            likelihood = score
            labels = gmm.labels
            means = gmm.means()

    color = ['b','g','r','orange','gray']
    plt.figure(1)
    for index in range(k):
        A = X[(labels==index),0]
        B = X[(labels==index),1]
        plt.scatter(A,B,c=color[index])
    plt.scatter(means[:,0],means[:,1])
    plt.title("My GMM")

    gmm = GMM(n_components=4).fit(X)
    labels = gmm.predict(X)
    color = ['b','g','r','orange','gray']
```

```python
    plt.figure(2)
    for index in range(k):
        A = X[(labels==index),0]
        B = X[(labels==index),1]
        plt.scatter(A,B,c=color[index])
    plt.title("Standar GMM")

    plt.figure(3)
    for index in range(k):
        A = X[(y_true==index),0]
        B = X[(y_true==index),1]
        plt.scatter(A,B,c=color[index])
    plt.title("origin data")
    plt.show()

def UCI():
    '''UCI数据测试'''
    data = pd.read_csv(os.path.join(os.getcwd(),"tripadvisor_review.csv"),header=0)
    feature_name = data.columns[1:]
    X = np.array(data.iloc[:,1:])

    pca = PCA(3)
    pca.fit(X)
    print(pca.explained_variance_ratio_)
    component = pca.components_
    new_X = pca.transform(X)


    fig = plt.figure(1)
    ax = fig.add_subplot(projection='3d')

    color = ['b','g','r','orange','gray']
    gmm_ = My_GMM(4,400)
    gmm_.fit(new_X)
    labels_ = gmm_.labels
    means = gmm_.means()

    for i in range(4):
        ax.scatter(new_X[(labels_==i),0],new_X[(labels_==i),1],new_X[(labels_==i),2],c=color[i]

    plt.show()
    show_correlation(components=component,name=feature_name)

    attrs = pca.inverse_transform(means)
    plt.xticks(ticks=range(10),labels=["art galleries","dance clubs","juice bars","restaurants"
        "resorts","parks/picnic spots","beaches","theaters","religious institutions"])
    for i in range(len(means)):
        attr = attrs[i]
        plt.scatter(range(10),attr,c = color[i])
    plt.show()

k_means(4)
Gmm(4)
UCI()
```