哈爾濱工業大學

Harbin Institute of Technology

机器学习实验报告

GMM 模型

课程名称：	机器学习
学院：	计算学部
专业：	计算机科学与技术
学号：	**1180300811**
姓名：	孙骁
指导老师：	刘扬
学期：	**2020** 秋季学期

2020 年 **11** 月 **3** 日

目录

一、 实验目的和要求

1.1 实验题目

实现 k-means 聚类方法和混合高斯模型.

1.2 实验目标

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.

1.3 实验测试

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）.

1.4 实验应用

可以 UCI 上找一个简单问题数据，用你实现的 GMM 进行聚类.

二、 实验环境

1. Anaconda 4.8.4
2. Python 3.7.4
3. PyCharm 2019.1 (Professional Edition)
4. Windows 10 2004

三、 实验原理

3.1 聚类问题

聚类问题属于无监督学习问题. 假定样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m\}$ 包含 m 个无标记样本，每个样本 $\boldsymbol{x}_i = (x_{i1}; x_{i2}; \cdots; x_{in})$ 是一个 n 维特征向量，则聚类算法将样本集 D 划分为 k 个不相交的簇 $\{C_l | l = 1, 2, \cdots, k\}$，其中 $C_{l'} \cap_{l' \neq l} C_l = \varnothing$. 相应的，我们用 $\lambda_j \in \{1, 2, \cdots, k\}$ 表示样本 $\boldsymbol{x}_j$ 的“簇标记”，即 $\boldsymbol{x}_j \in C_{\lambda_j}$. 于是，聚类的结果可以用包含 m 个元素的簇标记向量 $\boldsymbol{\lambda} = (\lambda_1; \lambda_2; \cdots; \lambda_m)$ 表示.

3.2 K-Means 算法原理

对于给定的样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_m\}$，“$k$ 均值”算法针对聚类所得到的簇划分 $\mathcal{C}=\{C_1,C_2,\cdots,C_k\}$ 最小化平方误差

$$E=\sum_{i=1}^{k}\sum_{\boldsymbol{x}\in C_i}\|\boldsymbol{x}-\boldsymbol{\mu_i}\|_2^2, \tag{1}$$

其中 $\boldsymbol{\mu_i}=\frac{1}{|C_i|}\sum_{\boldsymbol{x}\in C_i}\boldsymbol{x}$ 是簇 C_i 的均值向量. 式 (1) 刻画了簇内样本围绕均值向量的紧密程度，E 值越小，则簇内样本相似度越高.

最优化式 (1) 并不容易，找到其最优解需要考查样本集 D 所有可能的簇划分，显然这是一个 NP 难问题. 因此，k 均值算法采用了贪心策略，通过迭代优化来近似最小化式 (1).

3.3 混合高斯模型

对于 n 维样本空间 $\mathcal{X}$ 中的随机变量 $\boldsymbol{x}$，若 $\boldsymbol{x}$ 服从高斯分布，其概率密度函数为

$$p(\boldsymbol{x})=\frac{1}{(2\pi)^{\frac{n}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \tag{2}$$

其中 $\boldsymbol{\mu}$ 是 n 维均值向量，$\boldsymbol{\Sigma}$ 是 $n\times n$ 的协方差矩阵，由式 (2) 可知，高斯分布完全由均值向量 $\boldsymbol{\mu}$ 和协方差矩阵 $\boldsymbol{\Sigma}$ 这两个参数决定，因此我们也将概率密度函数记为 $p(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})$.

定义 1 (混合高斯分布) 定义高斯混合分布如下：

$$p_{\mathcal{M}}(\boldsymbol{x})=\sum_{i=1}^{k}\alpha_i\cdot p(\boldsymbol{x}|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i), \tag{3}$$

该分部共由 k 个混合成分组成，每个混合成分对应一个高斯分布. 其中 $\boldsymbol{\mu}_i$ 与 $\boldsymbol{\Sigma}_i$ 是第 i 个高斯混合成分的参数，而 $\alpha_i>0$ 为相应的“混合系数”，且 $\sum_{i=1}^{k}\alpha_i=1$.

假设样本的生成过程由高斯混合分布给出：首先，根据 $\alpha_1,\alpha_2,\cdots,\alpha_k$ 定义的先验分布选择高斯混合成分，即 $p(z_j=i)=\alpha_i$，其中 α_i 是选择第 i 个混合成分的概率；然后，根据被选择的混合成分的概率密度进行采样，从而生成相应的样本. 根据贝叶斯定理，z_j 的后验分布如式 (4) 所示，

$$\begin{aligned}p_{\mathcal{M}}(z_j=i|\boldsymbol{x}_j)&=\frac{P(z_j=i)\cdot p_{\mathcal{M}}(\boldsymbol{x}_j|z_j=i)}{p_{\mathcal{M}}(\boldsymbol{x}_j)}\\&=\frac{\alpha_i\cdot p(\boldsymbol{x}_j|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)}{\sum\limits_{l=1}^{k}\alpha_l\cdot p(\boldsymbol{x}_j|\boldsymbol{\mu}_l,\boldsymbol{\Sigma}_l)},\end{aligned} \tag{4}$$

即 $p_{\mathcal{M}}(z_j=i|\boldsymbol{x}_j)$ 给出了样本 $\boldsymbol{x}_j$ 由第 i 个高斯混合成分生成的后验概率.

当混合高斯分布 (3) 已知时，混合高斯聚类算法将把样本集 D 划分为 k 个簇 $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$，每个样本 $\boldsymbol{x}_j$ 的簇标记 λ_j 如式 (5) 确定：

$$\lambda_j = \underset{i \in \{1,2,\cdots,k\}}{\arg\max}\ \gamma_{ji}. \tag{5}$$

对于给定样本集 D 式 (3) 中的模型参数 $\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \,|1 \leq i \leq k\}$ 的求解，可以采用极大似然估计，即最大化对数似然

$$\begin{aligned} LL(D) &= \ln\left(\prod_{j=1}^{m} p_{\mathcal{M}}(\boldsymbol{x}_j)\right) \\ &= \sum_{j=1}^{m} \ln\left(\sum_{i=1}^{k} \alpha_i \cdot p(\boldsymbol{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\right), \end{aligned} \tag{6}$$

对于以上问题的最优化求解，常采用 EM 算法进行迭代优化.

3.4 EM 算法

若参数 $\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \,|1 \leq i \leq k\}$ 能使得式 (6) 最大化，则由 $\frac{\partial LL(D)}{\partial \boldsymbol{\mu}_i} = 0$ 有

$$\sum_{j=1}^{m} \frac{\alpha_i \cdot p(\boldsymbol{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum\limits_{l=1}^{k} \alpha_l \cdot p(\boldsymbol{x}_j|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} (\boldsymbol{x}_j - \boldsymbol{\mu}_i) = 0, \tag{7}$$

由式 (4) 及 $\gamma_{ji} = p_{\mathcal{M}}(z_j = i|\boldsymbol{x}_j)$，有

$$\boldsymbol{\mu}_i = \frac{\sum\limits_{j=1}^{m} \gamma_{ji} \boldsymbol{x}_j}{\sum\limits_{j=1}^{m} \gamma_{ji}}, \tag{8}$$

即各混合成分的均值可以通过样本加权平均来估计，样本权重是每个样本属于该成分的后验概率. 类似的，由 $\frac{\partial LL(D)}{\partial \boldsymbol{\Sigma}_i} = 0$ 可得

$$\boldsymbol{\Sigma}_i = \frac{\sum\limits_{j=1}^{m} \gamma_{ji} (\boldsymbol{x}_j - \boldsymbol{\mu}_i)(\boldsymbol{x}_j - \boldsymbol{\mu}_i)^{\mathrm{T}}}{\sum\limits_{j=1}^{m} \gamma_{ji}}, \tag{9}$$

对于混合系数 α_i，除了要最大化 $LL(D)$，还需要满足 $\alpha_i \geq 0$，$\sum_{i=1}^{k} \alpha_i = 1$，考虑 $LL(D)$ 的拉格朗日形式

$$LL(D) + \lambda\left(\sum_{i=1}^{k} \alpha_i - 1\right), \tag{10}$$

其中 λ 为拉格朗日乘子. 由式 (10) 对 α_i 的导数为 0，有

$$\sum_{j=1}^{m} \frac{\alpha_i \cdot p\left(\boldsymbol{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)}{\sum_{l=1}^{k} \alpha_l \cdot p\left(\boldsymbol{x}_j | \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l\right)} + \lambda = 0. \tag{11}$$

两边同时乘以 α_i，对所有样本求和可知 $\lambda = -m$，有

$$\alpha_i = \frac{1}{m} \sum_{j=1}^{m} \gamma_{ji}, \tag{12}$$

即每个高斯成分的混合系数由样本属于该成分的平均后验概率确定.

四、 算法实现

4.1 K-Means 算法

K-Means 算法的迭代优化算法如算法 (1) 所示.

Algorithm 1 K Means

Input: $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m\}$, 聚类簇数 k

Output: 簇划分 $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$

从 D 中随机选择 k 个样本作为初始均值向量 $\{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \cdots, \boldsymbol{\mu_k}\}$
repeat
    令 $C_i = \varnothing \, (1 \leq i \leq k)$
    for $j = 1, 2 \cdots, m$ **do**
        计算样本 $\boldsymbol{x}_j$ 与各均值向量 $\boldsymbol{\mu_i} \, (1 \leq i \leq k)$ 的距离：$d_{ji} = \|\boldsymbol{x}_j - \boldsymbol{x}_i\|_2$;
        根据距离最近的均值向量确定 $\boldsymbol{x}_j$ 的簇标记：$\lambda_j = \arg\min_{i \in \{1,2,\cdots,k\}} d_{ji}$;
        将样本 $\boldsymbol{x}_j$ 划入相应的簇:$C_{\lambda_j} = C_{\lambda_j} \cup \{\boldsymbol{x}_j\}$;
    end for
    for $i = 1, 2, \cdots, k$ **do**
        计算新均值向量：$\boldsymbol{\mu_i'} = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$;
        if $\boldsymbol{\mu_i'} \neq \boldsymbol{\mu_i}$ **then**
            将当前均值向量 $\boldsymbol{\mu_i}$ 更新为 $\boldsymbol{\mu_i'}$
        else
            保持当前均值向量不变
        end if
    end for
until 当前均值向量均未更新

4.2 高斯混合聚类算法

高斯混合聚类算法描述如算法 (2) 所示，算法首先对高斯混合分布的模型参数进行了初始化. 然后在第 2-12 行基于 EM 算法对模型参数进行迭代更新. 若 EM 算法的停止条件满足 (最大迭代轮数，或似然函数 $LL(D)$ 增长很少甚至不再增长)，则在第 14-17 行根据高斯混合分布确定簇划分，并最后返回最终结果.

Algorithm 2 Gaussian mixture clustering algorithm

Input: $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_m\}$, 高斯混合成分个数 k

Output: 簇划分 $\mathcal{C}=\{C_1,C_2,\cdots,C_k\}$

1: 初始化高斯混合分布的模型参数 $\{(\alpha_i,\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\,|1\leq i\leq k\}$
2: **repeat**
3: **for** $j=1,2,\cdots,m$ **do**
4: 根据式 (4) 计算计算 $\gamma_{ji}=p_{\mathcal{M}}\left(z_j=i|\boldsymbol{x}_j\right)(1\leq i\leq k)$
5: **end for**
6: **for** $i=1,2,\cdots,k$ **do**
7: 计算新均值向量：$\boldsymbol{\mu}_i'=\frac{\sum_{j=1}^m\gamma_{ji}\boldsymbol{x}_j}{\sum_{j=1}^m\gamma_{ji}}$;
8: 计算新协方差矩阵：$\boldsymbol{\Sigma}_i'=\frac{\sum_{j=1}^m\gamma_{ji}(\boldsymbol{x}_j-\boldsymbol{\mu}_i')(\boldsymbol{x}_j-\boldsymbol{\mu}_i')^{\mathrm{T}}}{\sum_{j=1}^m\gamma_{ji}}$;
9: 计算新混合系数：$\alpha_i'=\frac{\sum_{j=1}^m\gamma_{ji}}{m}$;
10: **end for**
11: 将模型参数 $\{(\alpha_i,\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\,|1\leq i\leq k\}$ 更新为 $\{(\alpha_i',\boldsymbol{\mu}_i',\boldsymbol{\Sigma}_i')\,|1\leq i\leq k\}$
12: **until** 满足停止条件
13: $C_i=\varnothing\,(1\leq i\leq k)$
14: **for** $j=1,2,\cdots,m$ **do**
15: 根据式 (5) 确定 $\boldsymbol{x}_j$ 的簇标记 λ_j;
16: 将 $\boldsymbol{x}_j$ 划入相应的簇：$C_{\lambda_j}=C_{\lambda_j}\cup\{\boldsymbol{x}_j\}$
17: **end for**

五、 实验步骤

5.1 生成满足二维高斯分布数据

依照给定的均值和方差，生成指定数量的满足二维高斯分布的数据，生成三类数据，二维均值分别为 1、1；-1、0；1、-2，二维协方差矩阵为 $\begin{pmatrix}0.1 & 0\\ 0 & 0.1\end{pmatrix}$，生成三类数据各生成 100 个.

5.2 实现章节4.1中的 K_Means 算法

实现实现章节4.1中的 K-Means 算法，代码见附录 (A).

5.3 实现章节4.2中的混合高斯模型

实现章节4.2中的混合高斯模型，并使用 EM 算法进行求解，代码见附录 (B).

5.4 使用 UCI 数据集测试 K-Means 与混合高斯模型

选用 UCI 鸢尾花分类数据集，根据鸢尾花的 4 个属性对鸢尾花的分类进行预测. 数据集中一共有三种类别，每个类别各 50 个样本，一共 150 个样本，每条数据包括四个特征，分别是：

1. 萼片长度 (单位：厘米)
2. 萼片宽度 (单位：厘米)
3. 花瓣长度 (单位：厘米)
4. 花瓣宽度 (单位：厘米)

实现章节4.1中的 K-Means 算法与章节4.2中的混合高斯模型，并显示分类的准确率. 代码见附录 (C).

六、 实验结果

6.1 K-Means 聚类结果

首先采用随机选取初始簇中心的方法，实验结果如图 (1) 所示. 很明显在分类上出现了错误，分析原因为没有选好的选取初始簇中心，导致了聚类时无法区分与簇中心较远而距离大致相近的点.

第二次选择了选择相距尽可能远的点作为初始的簇中心，聚类结果如图 (2) 所示，此次初始簇中心相距较远，较好的划分了样本集.

6.2 混合高斯模型聚类结果

混合高斯模型的聚类结果如图 (3)，可以看出和 K-Means 聚类效果基本一致.

6.3 使用 UCI 数据集测试 K-Means 与混合高斯模型

数据集没有数据特征缺失的情况，对数据集分别使用 K-Means 算法聚类与基于 EM 算法的高斯混合模型聚类，得到的结果如图 (4) 所示.

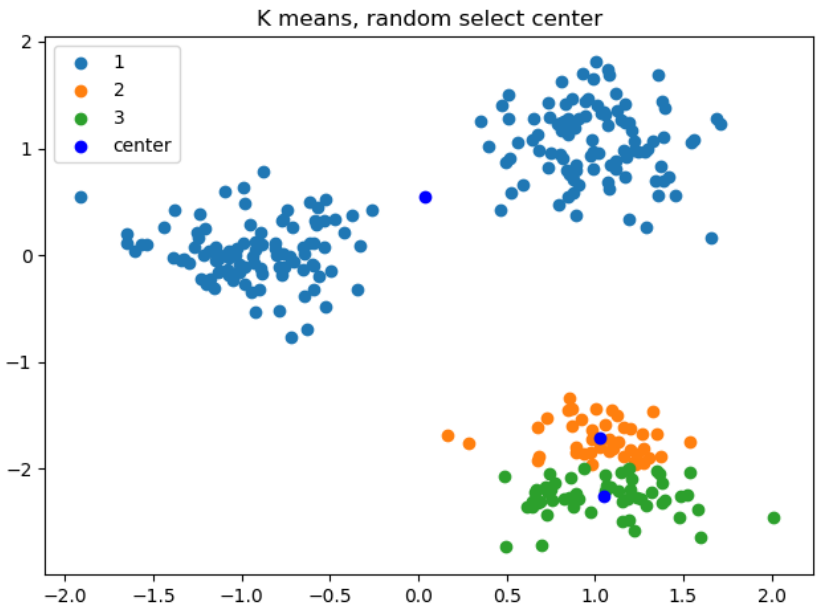

图 **1**　随机选取初始簇中心的 **K-Means** 算法结果

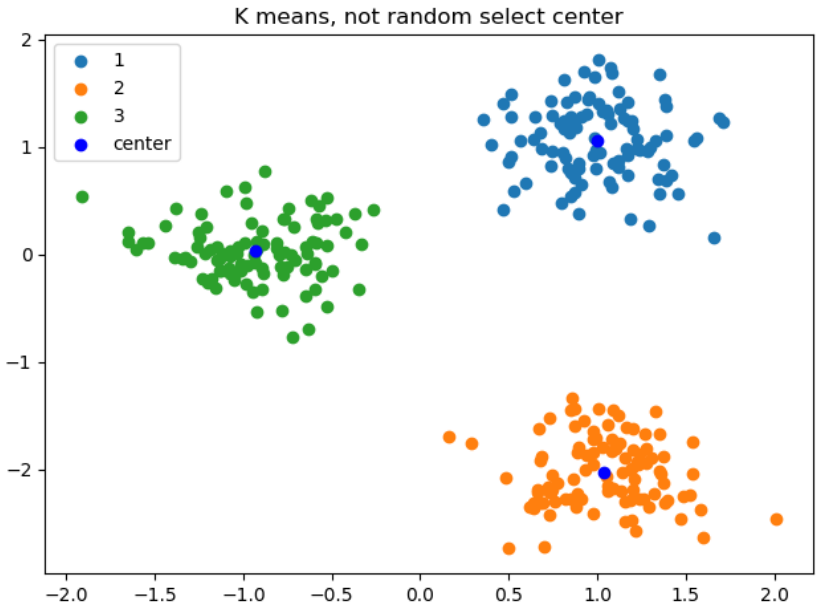

图 **2**　非随机选取初始簇中心的 **K-Means** 算法结果

七、 实验结论

1. K-Means 聚类假设数据分布在以簇中心为中心的一定范围内，混合高斯模型则假设数据符合混合高斯分布；
2. 由于 K-Means 聚类算法采用贪心的思想，未必能得到全局最优解，簇中心初始化对于最终的结果有很大的影响，如果选择不好初始的簇中心值容易使之陷入局部最优解；

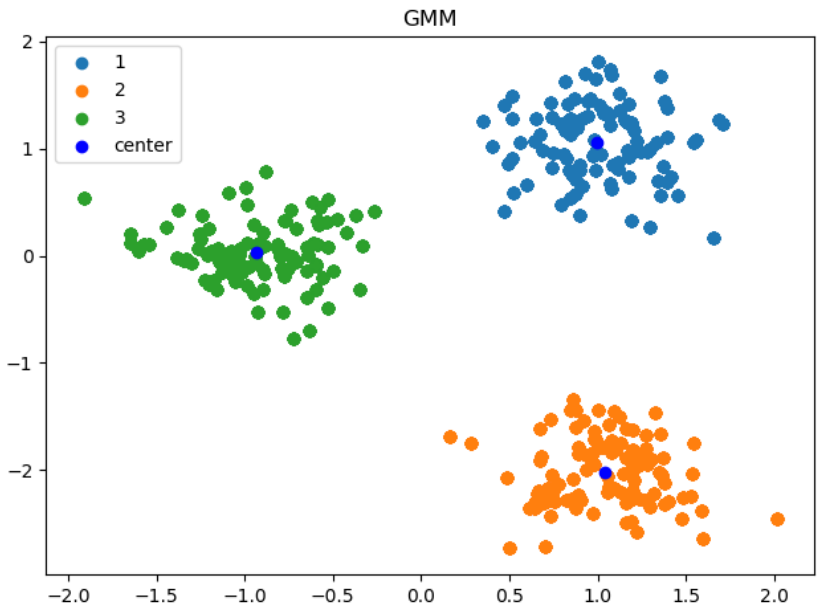

图 **3**　混合高斯模型的聚类结果

```
k means accuracy: 0.8866666666666667
GMM accuracy: 0.6866666666666666
```

图 **4**　在 **UCI** 鸢尾花数据集上测试结果

3. 使用 EM 算法解决混合高斯模型聚类问题时，如果初始高斯模型的均值和方差选取不当，可能会出现极大似然值为 0 的情况，也会出现协方差矩阵不可逆的情况.

参考文献

[1] 李航, 统计学习方法 (2019.3).

[2] 周志华, 机器学习 (2016.1).

[3] Iris Data Set. (1988.7) [Data set].

附录 A　K-Means 算法–k_means.py

```
import numpy as np
import random
import collections


def calculate_distance(x_1, x_2):
    return np.linalg.norm(x_1 - x_2)


class KMeans(object):
    def __init__(self, data, k, deviation=1e-6):
        self.data = data
        self.k = k
        self.deviation = deviation
        self.data_rows = data.shape[0]
        self.data_columns = data.shape[1]
        self.data_attribution = [-1] * self.data_rows
        self.mu = self.__initial_k_dots()

    def __initial_k_dots(self):
        mu_temp = np.random.randint(0, self.k) + 1
        mu = [self.data[mu_temp]]
        for i in range(self.k - 1):
            ans = []
            for j in range(self.data_rows):
                temp_ans = np.sum([calculate_distance(self.data[j], mu[k]) for k in
                    range(len(mu))])
                ans.append(temp_ans)
            mu.append(self.data[np.argmax(ans)])
        return np.array(mu)

    def k_means(self):
        number = 0
        while True:
            result = collections.defaultdict(list)
            for i in range(self.data_rows):
                distance = [calculate_distance(self.data[i], self.mu[j]) for j in
                    range(self.k)]
                lam_j = np.argmin(distance)
                result[lam_j].append(self.data[i].tolist())
                self.data_attribution[i] = lam_j
            new_mu = np.array([np.mean(result[i], axis=0).tolist() for i in
                range(self.k)])
            new_loss = np.sum(calculate_distance(self.mu[i], new_mu[i]) for i in
```

```
                range(self.k))
            if new_loss > self.deviation:
                self.mu = new_mu
            else:
                break
            # print(number)
            number += 1
        # print(self.mu)
        return self.mu, result

    def random_select_center(self):
        self.mu = self.data[random.sample(range(self.data_rows), self.k)]
        return self.k_means()

    def not_random_select_center(self):
        self.mu = self.__initial_k_dots()
        return self.k_means()
```

附录 B　混合高斯模型与 EM 算法求解–gaussian_mixture_model.py

```
import numpy as np
import numpy.random as random
import collections
from scipy.stats import multivariate_normal


def calculate_distance(x_1, x_2):
    return np.linalg.norm(x_1 - x_2)


class GaussianMixtureModel(object):
    def __init__(self, data, k, deviation, iteration_number):
        self.data = data
        self.k = k
        self.deviation = deviation
        self.iteration_number = iteration_number
        self.alpha = np.ones(self.k) * (1.0 / self.k)
        self.data_rows = data.shape[0]
        self.data_columns = data.shape[1]
        self.mu, self.sigma = self.__init_params()
        self.data_attribution = [-1] * self.data_rows
        self.result = collections.defaultdict(list)
        self.gamma = None

```

```
    def __init_params(self):
        mu_0 = random.randint(0, self.k)
        # print(mu_0)
        mu_temp = [self.data[mu_0]]
        for index in range(self.k - 1):
            temp_ans = []
            for i in range(self.data_rows):
                temp_ans.append(np.sum([calculate_distance(self.data[i], mu_temp[j]) for
                    j in range(len(mu_temp))]))
            mu_temp.append(self.data[np.argmax(temp_ans)])
        mu = np.array(mu_temp)
        # print("mu", mu)
        sigma = collections.defaultdict(list)
        for i in range(self.k):
            sigma[i] = np.eye(self.data_columns, dtype=float) * 0.1

        # print("sigma", sigma)
        # for i in range(self.k):
        #     print("sigma", i, sigma[i])
        return mu, sigma

    def calculate_likelihood(self):
        likelihood = np.zeros((self.data_rows, self.k))
        for i in range(self.k):
            # print("------------")
            # print("mu[i]", self.mu[i])
            # print("sigma[i]", self.sigma[i])
            likelihood[:, i] = multivariate_normal.pdf(self.data, self.mu[i],
                self.sigma[i])
            # print(likelihood[:, i])
        return likelihood

    def calculate_expectation(self):
        likelihoods = self.calculate_likelihood() * self.alpha
        sum_likelihood = np.expand_dims(np.sum(likelihoods, axis=1), axis=1)

        self.gamma = likelihoods / sum_likelihood
        # print(self.gamma)
        self.data_attribution = self.gamma.argmax(axis=1)
        for i in range(self.data_rows):
            self.result[self.data_attribution[i]].append(self.data[i].tolist())

    def max_function(self):
        for i in range(self.k):
            gamma_ji = np.expand_dims(self.gamma[:, i], axis=1)
            mu_i = (gamma_ji * self.data).sum(axis=0) / gamma_ji.sum()
            cov = (self.data - mu_i).T.dot((self.data - mu_i) * gamma_ji) / gamma_ji.sum()
```

```
            self.mu[i], self.sigma[i] = mu_i, cov
        self.alpha = self.gamma.sum(axis=0) / self.data_rows

    def gmm(self):
        pre_alpha = self.alpha
        pre_mu = self.mu
        pre_sigma = self.sigma
        for i in range(self.iteration_number):
            self.calculate_expectation()
            self.max_function()
            diff = np.linalg.norm(pre_alpha - self.alpha) + np.linalg.norm(pre_mu -
                self.mu) + np.sum(
                [np.linalg.norm(pre_sigma[i] - self.sigma[i]) for i in range(self.k)])
            if diff > self.deviation:
                pre_alpha = self.alpha
                pre_sigma = self.sigma
                pre_mu = self.mu
            else:
                break
        self.calculate_expectation()
        return self.mu, self.result
```

附录 C　读取鸢尾花数据集–read_iris_data.py

```
import numpy as np
import pandas as pd


def read_iris_data():
    data_set = pd.read_csv("../data/iris.csv")
    X = data_set.drop("class", axis=1)
    Y = data_set['class']
    # print(Y)
    return np.array(X, dtype=float), np.array(Y, dtype=str)
```

附录 D　主程序–k_means_GMM.py

```
import itertools as it

from matplotlib import pyplot as plt
from src.generate_data import *
from src.k_means import *
```

```
from src.read_iris_data import *
from src.gaussian_mixture_model import *


def generate_picture(k, random_mu, random_result, not_random_mu, not_random_result,
    gmm_mu, gmm_result):
    plt.title("K means, random select center")
    for i in range(k):
        plt.scatter(np.array(random_result[i])[:, 0], np.array(random_result[i])[:, 1],
            label=str(i + 1))
    plt.scatter(random_mu[:, 0], random_mu[:, 1], c="b", label="center")
    plt.legend()
    plt.show()

    plt.title("K means, not random select center")
    for i in range(k):
        plt.scatter(np.array(not_random_result[i])[:, 0],
            np.array(not_random_result[i])[:, 1], label=str(i + 1))
    plt.scatter(not_random_mu[:, 0], not_random_mu[:, 1], c="b", label="center")
    plt.legend()
    plt.show()

    plt.title("GMM")
    for i in range(k):
        plt.scatter(np.array(gmm_result[i])[:, 0], np.array(gmm_result[i])[:, 1],
            label=str(i + 1))
    plt.scatter(gmm_mu[:, 0], gmm_mu[:, 1], c="b", label="center")
    plt.legend()
    plt.show()


def test_iris_data():
    data_X, data_Y = read_iris_data()
    k_means = KMeans(data_X, 3)
    k_means__mu, k_means_result = k_means.not_random_select_center()
    k_means_attribution = k_means.data_attribution

    number = len(data_Y)
    counts_kmeans = []
    result = list(it.permutations(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],
        3))
    for i in range(len(result)):
        count = 0
        for index in range(number):
            # print(data_Y[index])
            # print(result[i][k_means_attribution[index]])
            if data_Y[index] == result[i][k_means_attribution[index]]:
```

```
                count += 1
        counts_kmeans.append(count)
    kmeans_accuracy = 1.0 * np.max(counts_kmeans) / number
    print("k means accuracy:", kmeans_accuracy)

    deviation = 1e-12
    iteration_number = 10000
    # print(data_X.shape)
    gmm = GaussianMixtureModel(data_X, 3, deviation, iteration_number)
    gmm_mu, gmm_result = gmm.gmm()

    counts_gmm = []
    gmm_attribution = gmm.data_attribution
    # print(gmm_attribution)
    for i in range(len(result)):
        count = 0
        for index in range(number):
            if data_Y[index] == result[i][gmm_attribution[index]]:
                count += 1
        counts_gmm.append(count)
    gmm_accuracy = 1.0 * np.max(counts_gmm) / number
    print("GMM accuracy:", gmm_accuracy)


def main():
    k = 3
    category_means = [[1, 1], [-1, 0], [1, -2]]
    category_number = [100, 100, 100]
    data = generate_2_dimension_data(category_means, category_number, k)
    k_means_result = KMeans(data, k)
    random_mu, random_result = k_means_result.random_select_center()
    not_random_mu, not_random_result = k_means_result.not_random_select_center()

    deviation = 1e-12
    iteration_number = 10000
    gmm = GaussianMixtureModel(data, k, deviation, iteration_number)
    gmm_mu, gmm_result = gmm.gmm()

    generate_picture(k, random_mu, random_result, not_random_mu, not_random_result,
        gmm_mu, gmm_result)

    test_iris_data()


if __name__ == '__main__':
    main()
```

附录 E　生成二维高斯分布数据–generate_data.py

```python
import numpy as np


def generate_2_dimension_data(category_means, category_number, k_number):
    cov = [[0.1, 0], [0, 0.1]]
    data = []
    for i in range(k_number):
        for index in range(category_number[i]):
            data.append(
                np.random.multivariate_normal([category_means[i][0],
                    category_means[i][1]], cov).tolist())
    return np.array(data)
```