2020 年春季学期
计算学部《机器学习》课程

Lab 3 实验报告

姓名	包宇昊
学号	1180300605
班号	1803105
电子邮件	2568235796@qq.com
手机号码	18267116288

目录

1 .实验目标 2

2 .实验要求及环境 3

3 .数学思想 4

3.1 k-means 聚类 4

3.2 高斯混合模型 4

3.3 高斯混合模型参数估计的 EM 算法 5

4 .实验步骤 5

4.1 生成数据 5

4.2 K-means 聚类 6

4.2.1 随机选取初始中心点，设置标志位以避免选取相同的中心点 6

4.2.2 计算欧氏距离，选取距离最小的中心点进行分类 7

4.2.3 根据分类重新计算中心点 7

4.2.4 如果差距较小则跳出循环，循环结束 7

4.2.5 测试效果 8

4.3 GMM-EM 10

4.3.1 用 k-means 分类结果初始化，沿用 k-means 得到的均值，计算 k-means 分类得到的各类的协方差矩阵作为各类初始的协方差矩阵 10

4.3.2 根据数学原理进行 E 步、M 步迭代 10

4.4 UCI 数据测试 12

5 .实验结论 12

建议写出：问题的描述，解决问题的思路，实验的做法，实验结果的分析，结论，自拟标题

1 .实验目标

- 理解 k-means 聚类过程
- 理解混合高斯模型(GMM)用 EM 估计参数的实现过程

- 掌握 k-means 和混合高斯模型的联系
- 学会 EM 估计参数的方法和代码实现
- 学会用 GMM 解决实际问题

2 .实验要求及环境

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）。

实验环境：

pycharm 2020.2.2

python 3.7

win10

X86-64

3 .数学思想

3.1 k-means 聚类

k-means 聚类就是根据某种度量方式(本次采用欧氏距离)，将相关性较大的一些样本点聚集在一起，一共聚成 k 个堆，每一个堆称为一“类”。k-means 的过程为：先在样本点中选取 k 个点作为暂时的聚类中心，然后依次计算每一个样本点与这 k 个点的距离，将每一个与距离这个点最近的中心点聚在一起，这样形成 k 个类“堆”，求每一个类的期望，将求得的期望作为这个类的新的中心点。一直不停地将所有样本点分为 k 类，直至中心点不再改变停止。

3.2 高斯混合模型

高斯混合模型是指具有如下形式的概率分布模型：

$$P(y|\theta) = \sum_{k=1}^{K} \alpha_k \varphi(y|\theta_k)$$

其中α_k 是样本中类 k 中的数据所占的比率，$\sum_{k=1}^{K} \alpha_k = 1$, $\varphi(y|\theta_k)$ 是第 k 类中的高斯分布的概率分布函数。

其中$\varphi(y|\theta_k)$为

$$\varphi(y|\theta_k) = \frac{1}{2\pi^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} exp(-\frac{(y-\mu_k)^T \Sigma_k^{-1} (y-\mu_k)}{2})$$

其中 D 为样本数据的维数，$\Sigma\, k$ 为第 k 个高斯模型的协方差矩阵，如果数据为一维，则是方差；μ_k为第 k 个模型的均值。当样本为一维数据时

$$\varphi(y|\theta_k) = \frac{1}{\sqrt{2\pi}\sigma_k} exp(-\frac{(y-\mu_k)^2}{2\sigma_k^2})$$

，其中σ_k为第 k 类的标准差。

通常将$(\alpha, \sum ,\mu)$是高斯混合模型中的隐变量，由于隐变量的存在，高斯混合模型无法求出解析解，但是可以用 EM 算法迭代求解隐变量，并完成分类。

具体迭代过程如下：

1. 根据输入的超参数 K 首先初始化一些向量（可以从现有的向量中挑选），作为各簇的均值向量。
2. 根据初始化的均值向量给出训练样本的一个划分，计算各个训练样本到各个均指向量的距离，找出距离最近的均值向量，并将该样本分至该均值向量所代表的簇。

3. 根据新的簇划分，重新计算每个簇的均值向量，如果新的均值向量与旧的均值向量差小于 ε ，则认为算法收敛；否则，更新均值向量，回到第 2 步重新迭代求解。

3.3 高斯混合模型参数估计的 EM 算法

1.初始化响应度矩阵γ,协方差矩阵,均值和α

2.E 步：定义响应度矩阵γ,其中γ_{jk}表示第 j 个样本属于第 k 类的概率，计算式如下

$$\gamma_{jk}=\frac{\alpha_k\varphi(y_j|\theta_k)}{\sum_{k=1}^{K}\alpha_k\varphi(y_j|\theta_k)},j=1,2,...,N;k=1,2,...,K$$

3.M 步：将响应度矩阵视为定值，更新均值，协方差矩阵和α

$$\mu_k=\frac{\sum_{k=1}^{N}\gamma_{jk}y_j}{\sum_{j=1}N\gamma_{jk}},k=1,2,...,K$$

$$\Sigma_k=\frac{\sum_{j=1}^{N}\gamma_{jk}(y-\mu_k)(y-\mu_k)^T}{\sum_{j=1}^{N}\gamma_{jk}},k=1,2,..,K$$

4. 重复 2.3.步，直到收敛

4 .实验步骤

4.1 生成数据

生成四组高斯分布数据，每组数据数量为 50，其中每组的均值和方差如下所示

```python
config = {
    'K': 4,
    'dim': 2,
    'n': 50,

    'miu': np.array([[2, 2], [9,9], [9,1], [2, 7]]),
    'sigma': np.array([[[2, 0], [0, 2]],
                       [[1, 0], [0, 1]],
                       [[2, 0], [0, 2]],
                       [[1, 0], [0, 1]]])

}
```

```python
data[i] = np.random.multivariate_normal(miu[i], sigma[i], n)
```

4.2 K-means 聚类

4.2.1 随机选取初始中心点，设置标志位以避免选取相同的中心点

```python
def select_initial_center(data, k, n, dim):
    center = np.zeros((k, dim))
    flags = np.zeros((n, 1))
    for i in range(k):
        # 此处flags避免选择同一初始点
        while True:
            a = np.random.randint(0, n)
            if flags[a] == 0:
                center[i, :] = data[a, :]
                flags[a] = 1
                break
    return center
```

4.2.2 计算欧氏距离，选取距离最小的中心点进行分类

```
while True:
    distance = np.zeros(k)
    for i in range(n):
        for j in range(k):
            distance[j] = np.linalg.norm(data[i, :] - center[j, :])
        arg = np.argmin(distance)
        classes[i, dim] = arg
```

4.2.3 根据分类重新计算中心点

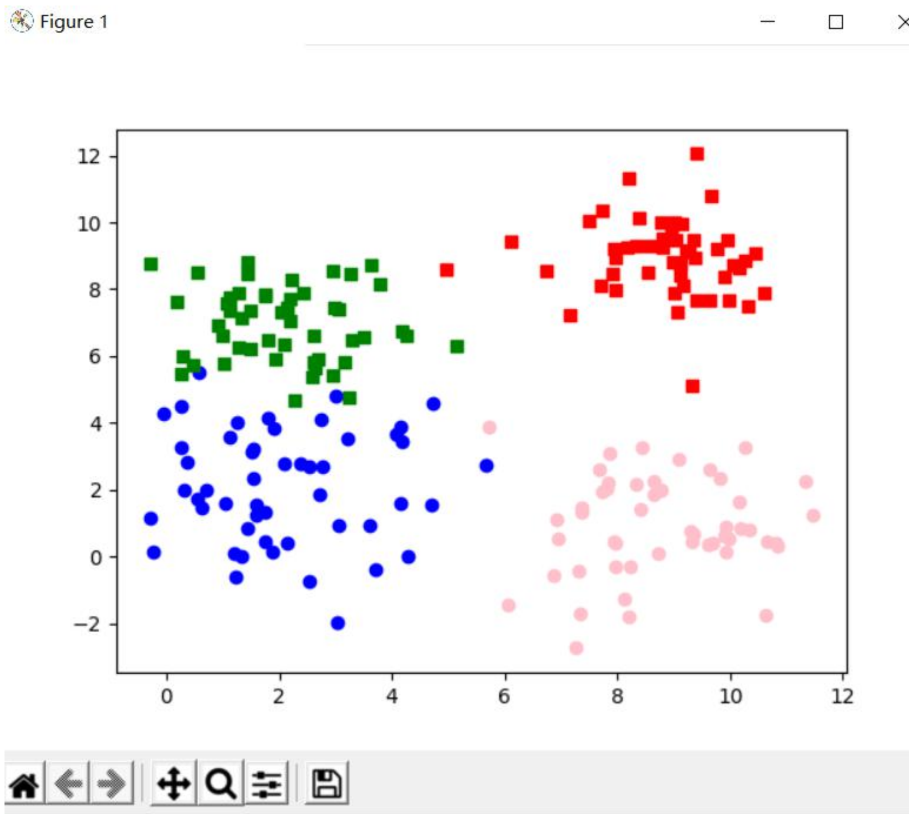
```
def new_center(data, k, n, dim, classes):
    new_center = np.zeros((k, dim))
    num = np.zeros(k)
    for i in range(n):
        if classes[i, dim] < k:
            c = int(classes[i, dim])
            new_center[c, :] = new_center[c, :] + classes[i, :dim]
            num[c] += 1
    for i in range(k):
        if num[i] != 0:
            new_center[i, :] /= num[i]
    return new_center
```

4.2.4 如果差距较小则跳出循环，循环结束

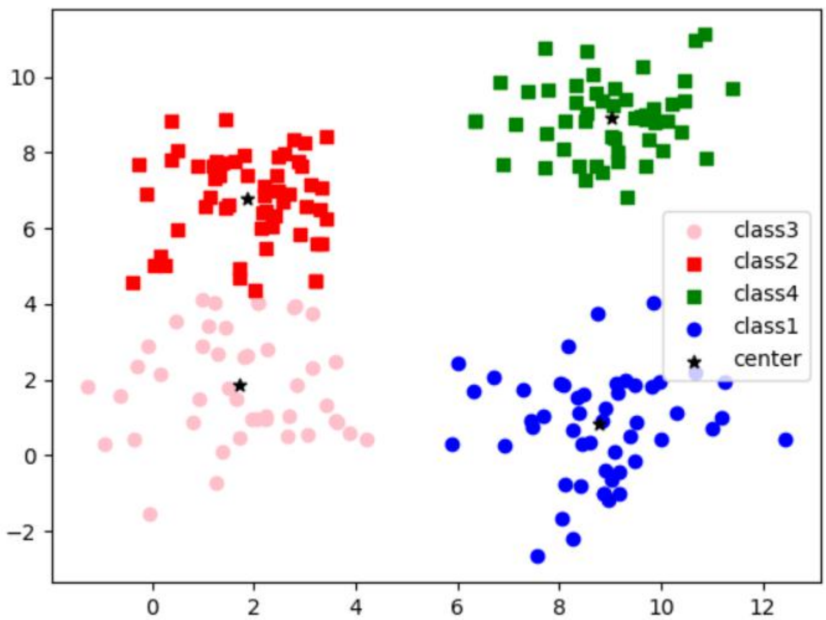
```
for i in range(k):
    distance_bias = np.linalg.norm(newcenter[i, :] - center[i, :])
    if distance_bias < 1e-15:
        p += 1
        print('class%d聚类成功' % p)
```

4.2.5 测试效果

（1）初始生成 4 类，未通过 k-means 分类

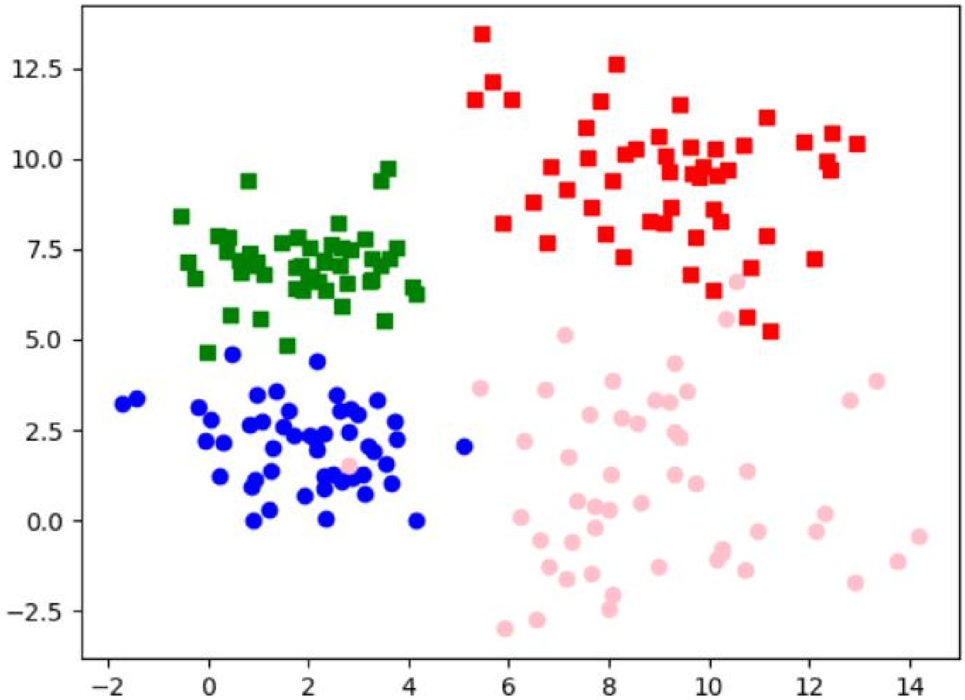


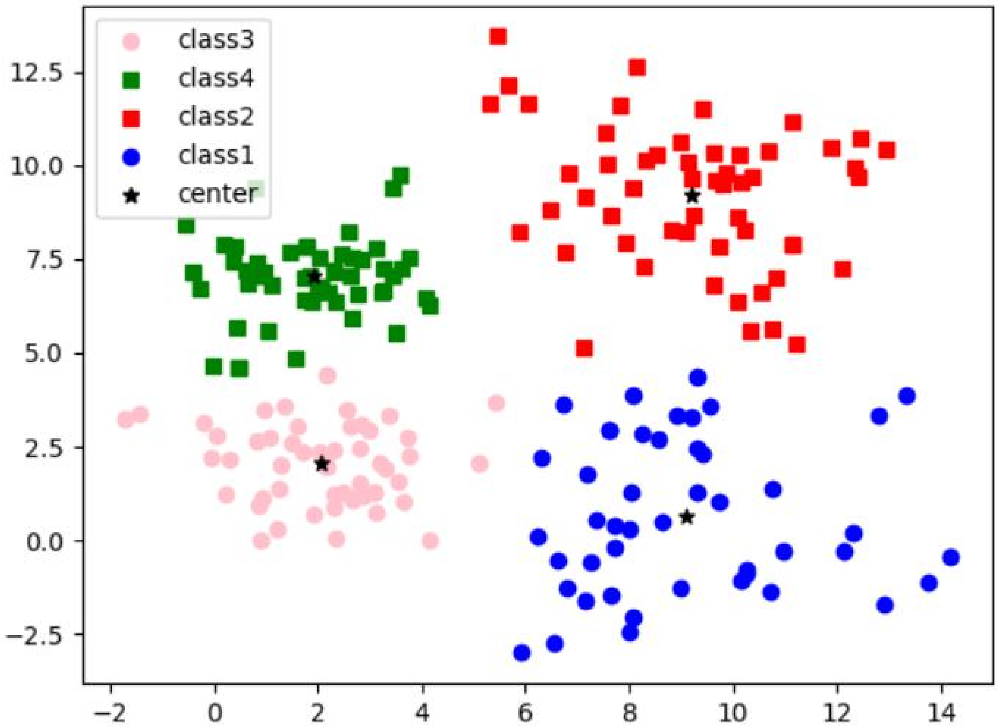
（2）k-means 分类完成后：



（3）修改方差，观察

```
'sigma': np.array([[[2, 0], [0, 2]],
                   [[3, 0], [0, 3]],
                   [[5, 0], [0, 5]],
                   [[1, 0], [0, 1]]])
```





4.3 GMM-EM

4.3.1 用 k-means 分类结果初始化，沿用 k-means 得到的均值，计算 k-means 分类得到的各类的协方差矩阵作为各类初始的协方差矩阵

4.3.2 根据数学原理进行 E 步、M 步迭代

E 步：求期望

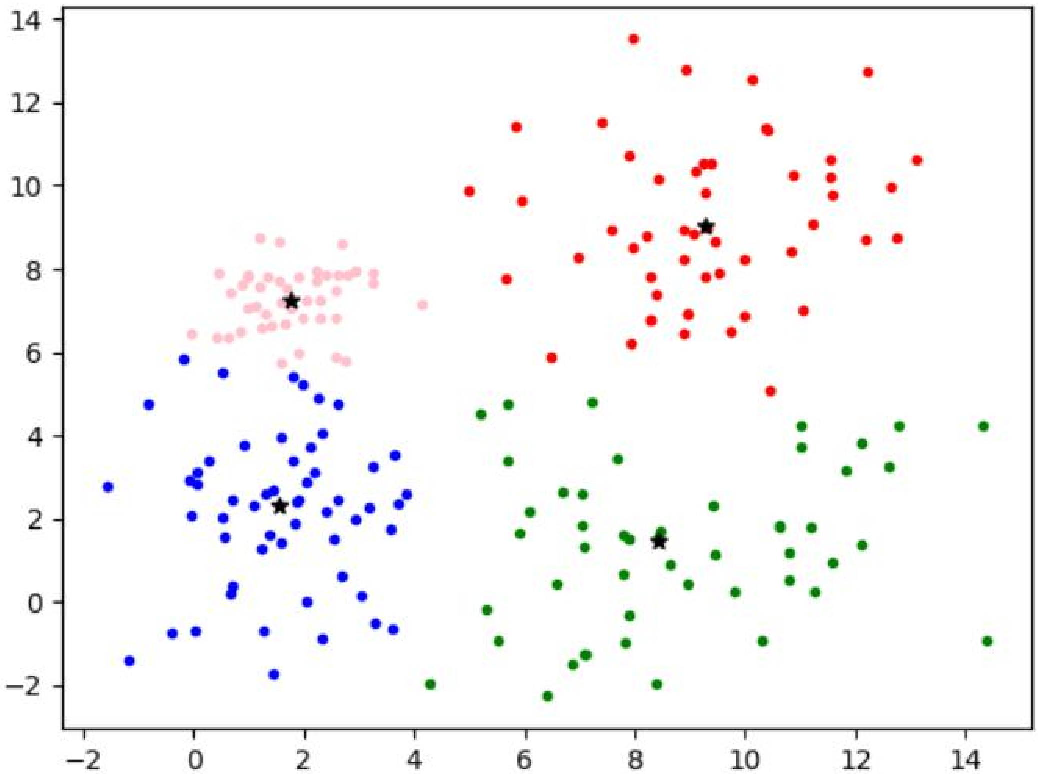
M 步：求最大似然

```
for j in range(n):
    mixture_model = 0
    for k in range(K):
        mixture_model += alpha[k] * scipy.stats.multivariate_normal.pdf(data[j],
                                                                      miu[k], sigma[k])
    for k in range(K):
        gamma[j][k] = alpha[k] * scipy.stats.multivariate_normal.pdf(data[j],
                                                                   miu[k], sigma[k]) / mixture_model
```

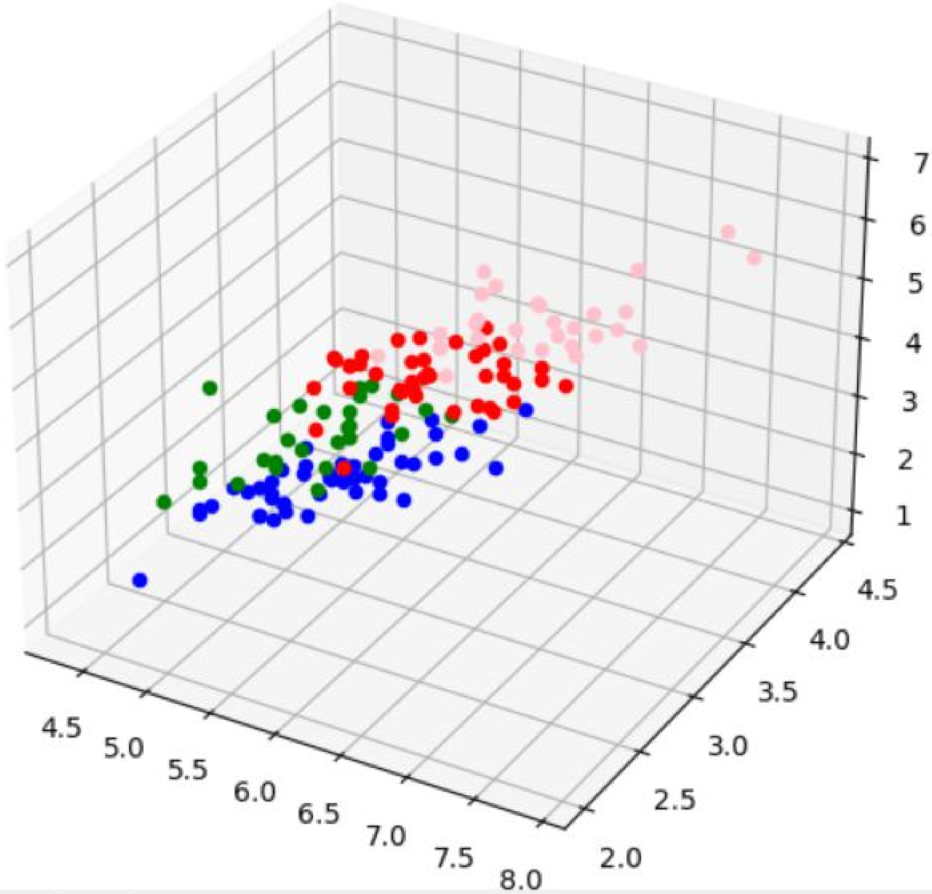
```
for k in range(K):
    Nk = 0
    for j in range(n):
        Nk += gamma[j][k]
    miu2 = np.zeros(dim)
    for j in range(n):
        miu2 += gamma[j][k] * data[j]
    miu1[k] = miu2 / Nk

    sigma2 = np.zeros(dim)
    for j in range(n):
        sigma2 += (data[j] - miu[k]) ** 2 * gamma[j][k]
    sigma3 = np.eye(dim)
    sigma3[0, 0] = sigma2[0]
    sigma3[1, 1] = sigma2[1]
    sigma1[k] = sigma3 / Nk

    alpha1[k] = Nk / n
return miu1, sigma1, alpha1, gamma
```

显示结果

```
0 135.21369101912148
1 2.5121833901430364
2 0.5785421522202796
3 0.19425252840494522
4 0.08935747156078833
5 0.048700222098204904
6 0.02831411454269528
7 0.01671093691174974
8 0.009835296305254815
```



4.4 UCI 数据测试



5 .实验结论

K-Means 其实就是一种特殊的高斯混合模型，假设每种类在样本中出现的概率相等均为 1/k ，而且假设高斯模型中的每个变量之间是独立的，即变量间的协方差矩阵是对角阵，可以直接用欧氏距离作为 K-Means 的协方差去衡量相似性；K-Means 对响应度也做了简化，每个样本只属于一个类，即每个样本属于某个类响应度为 1，对于不属于的类响应度设为 0。而在高斯混合模型中，用协方差矩阵替代 K-Means 中的欧式距离去度量点和点之间的相似度，响应度也由离散的 0，1 变成了需要通过全概率公式计算的值。由于 GMM 不像 K-means 做了很多假设，所以分类最终效果比 K-Means 好，但是 GMM-EM 算法容易被噪声影响，所以适合对 K-Means 的分类结果进行进一步优化。