Assignment-3 GMM 模型

一、概念介绍

1.GMM 模型

混合高斯模型，GMM，由多个不同的高斯分布共同产生样本数据。简单地，设有 K 个不同的高斯分布，每个高斯分布的维度均为 D。记第 k 个高斯分布为 $N_k(u_k, \sigma_k^2)$，有选取概率 π_k。

样本数据的产生分两步骤：1）根据概率分布 $\pi = (\pi_1, \ldots, \pi_K)$ 随机选取第 k 个高斯分布；2）根据高斯分布的密度函数 $N_k(u_k, \sigma_k^2)$，随机选取样本点 x_n.

于是 $p(x_n) = \sum_{k=1}^{K} \pi_k * N_k(x_n \mid u_k, \sigma_k^2)$. 可见，待学习的参数是 π_k, u_k, σ_k^2

其中 u_k 是均值，σ_k^2 是方差或协方差矩阵。而 K 是超参数

2.EM 算法

1）思路

$$\begin{aligned}\log p(\boldsymbol{x};\theta) &= \log \sum_{\boldsymbol{z}} q(\boldsymbol{z}) \frac{p(\boldsymbol{x},\boldsymbol{z};\theta)}{q(\boldsymbol{z})} \\ &\geq \sum_{\boldsymbol{z}} q(\boldsymbol{z}) \log \frac{p(\boldsymbol{x},\boldsymbol{z};\theta)}{q(\boldsymbol{z})} \\ &\triangleq ELBO(q,\boldsymbol{x};\theta),\end{aligned}$$

E 步：固定参数 θ_t，找到一个分布 $q_{t+1}(z)$ 使得证据下界 $ELBO(q, x; \theta_t) = \log p(x; \theta_t)$；根据 Jensen 不等式，最理想的分布 q(z) 即后验分布 $p(z|x; \theta_t)$

M 步：固定 $q_{t+1}(z)$，找到一组参数使得证据下界最大，即

$$\theta_{t+1} = \arg\max_{\theta} ELBO(q_{t+1}, \boldsymbol{x}; \theta).$$

2）GMM 中的 EM 应用

E步　先固定参数μ,σ,计算后验分布$p(z^{(n)}|x^{(n)})$,即

$$\begin{aligned}\gamma_{nk} &\triangleq p(z^{(n)}=k|x^{(n)})\\ &= \frac{p(z^{(n)})p(x^{(n)}|z^{(n)})}{p(x^{(n)})}\\ &= \frac{\pi_k\mathcal{N}(x^{(n)};\mu_k,\sigma_k)}{\sum_{k=1}^{K}\pi_k\mathcal{N}(x^{(n)};\mu_k,\sigma_k)},\end{aligned}$$

其中γ_{nk}定义了样本$x^{(n)}$属于第k个高斯分布的后验概率.

M步　令$q(z=k)=\gamma_{nk}$,训练集$\mathcal{D}$的证据下界为

$$ELBO(\gamma,\mathcal{D};\pi,\mu,\sigma)=\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{nk}\log\frac{p(x^{(n)},z^{(n)}=k)}{\gamma_{nk}}$$

$$=\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{nk}\left(\log\mathcal{N}(x^{(n)};\mu_k,\sigma_k)+\log\frac{\pi_k}{\gamma_{nk}}\right)$$

$$=\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{nk}\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}-\log\sigma_k+\log\pi_k\right)+C,$$

其中C为和参数无关的常数.

将参数估计问题转为优化问题：

$$\begin{aligned}&\max_{\pi,\mu,\sigma} ELBO(\gamma,\mathcal{D};\pi,\mu,\sigma),\\ &\text{s.t.}\quad \sum_{k=1}^{K}\pi_k=1.\end{aligned}$$

应用拉格朗日乘数法解得：

$$\pi_k=\frac{N_k}{N},$$

$$\mu_k=\frac{1}{N_k}\sum_{n=1}^{N}\gamma_{nk}x^{(n)},$$

$$\sigma_k^2=\frac{1}{N_k}\sum_{n=1}^{N}\gamma_{nk}(x^{(n)}-\mu_k)^2,$$

3.Kmeans 算法

1）初始化 K 个中心点

2）迭代执行如下两步，直至不存在样本点的分类情况发生改变：

分配步：遍历数据集中样本点，将其归类至距离它最近的中心点，即聚类。

更新步：在每个类中，计算类的平均位置，作为该类的新中心点。

二、模型描述

1.数据生成

（详见 data.py 文件）

简单起见，设计有两维数据和三维数据，均包含 3 个高斯分布。其中，两维数据用于可视化观察；三维数据充当高维数据，高维散点图观察不方便，转为观察学习的π_k, u_k, σ_k^2在一定误差范围内的准确度。

在产生数据时，并不真实实现步骤 1）中的根据π_k随机选取高斯分布。而是，第 k 个高斯分布选取$\pi_k * N$个样本点，其中 N 是总的样本数。当 N 较大，时，该方式近似于根据π_k随机选取高斯分布。然后，将得到的 K 个数据集合并作为整个数据集。

2.GMM 模型

详见前文，本例中高斯分布个数 K=3，维度 D=2 或 3。

3.GMM 参数学习

参数学习使用 EM 算法，具体如下：

初始化：有如下两种初始化方式，下文先选择第一种初始化方式。

1）调用 kmeans 算法生成π_k, u_k, σ_k^2，作为初始值。注意，在生成协方差矩阵σ_k^2时，不需精确计算，可假设其是对角阵。

2）π_k统一初始化为 1/K，任取数据集中 k 个样本作为u_k，σ_k^2则初始化为单位阵。

迭代：

E 步：根据前文，学习γ_{nk}

M 步：固定 γ_{nk}，根据前文，计算 π_k, u_k, σ_k^2

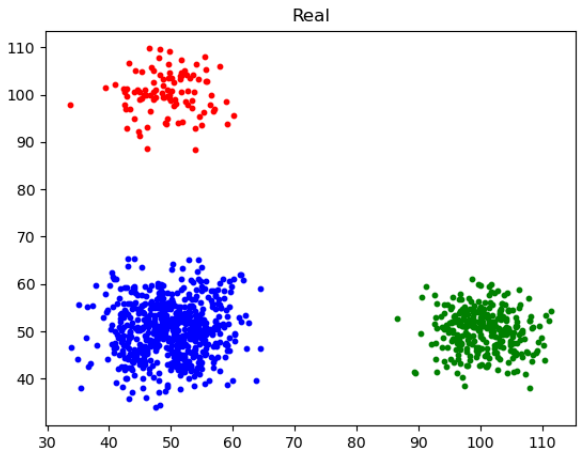
until 对数边际分布 $\sum_{n=1}^{N} \log p(x^{(n)})$ 收敛;

三、模型性能

1. 二维相离数据

真实参数值，其中 N=1000

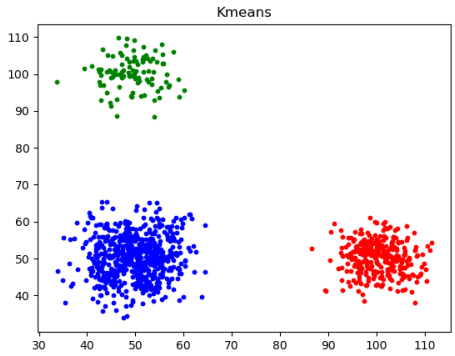
```
freq = [0.6, 0.3, 0.1]

mean0 = [50, 50]
cov0 = [[30, 0], [0, 30]]
mean1 = [100, 50]
cov1 = [[20, 0], [0, 20]]
mean2 = [50, 100]
cov2 = [[20, 0], [0, 20]]
```

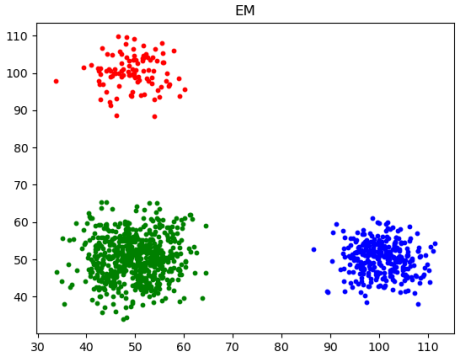


学习得到的参数值

```
2D------ Kmeans ------
km_pi is
[0.6 0.1 0.3]
km_mean is
[[ 49.81386401  50.36747575]
 [ 49.40149925 100.22624137]
 [100.46725577  50.24025596]]
km_cov is
[[[30.78469672  0.        ]
  [ 0.         33.98510002]]

 [[22.57272912  0.        ]
  [ 0.         20.04196946]]

 [[20.09533791  0.        ]
  [ 0.         19.16333351]]]
```

```
2D------ EM ------
em_pi is
[0.3 0.6 0.1]
em_mean is
[[100.46725577  50.24025596]
 [ 49.81386401  50.36747576]
 [ 49.40149925 100.22624139]]
em_cov is
[[[20.09533788 -1.06220876]
  [-1.06220876 19.16333351]]

 [[30.78469682  1.69400754]
  [ 1.69400754 33.98510036]]

 [[22.57272913  0.04396487]
  [ 0.04396487 20.04196928]]]
```

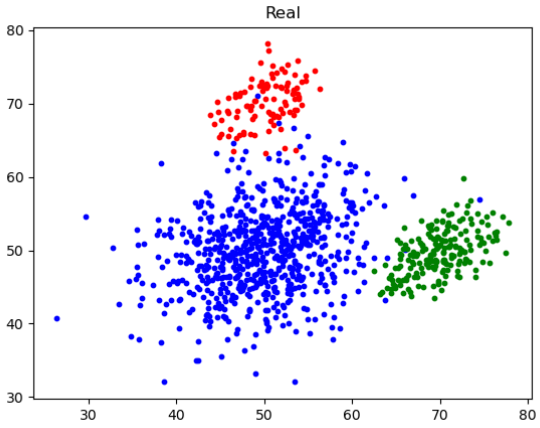




2. 二维相近数据

真实参数值，其中 N=1000

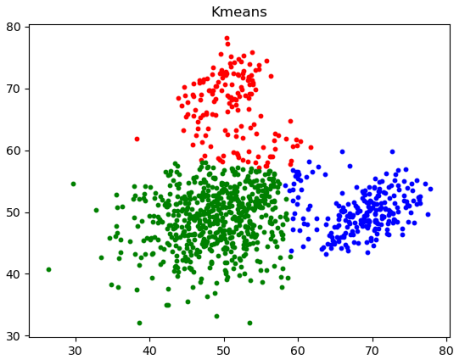
```
freq = [0.7, 0.2, 0.1]

mean0 = [50, 50]
cov0 = [[35, 10], [10, 35]]
mean1 = [70, 50]
cov1 = [[10, 5], [5, 10]]
mean2 = [50, 70]
cov2 = [[10, 5], [5, 10]]
```

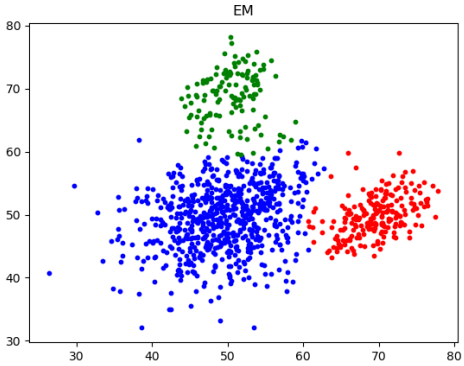


学习得到的参数值

```
2D------ Kmeans ------
km_pi is
[0.239 0.606 0.155]
km_mean is
[[68.48884762 50.1522953 ]
 [48.6959999  48.81748385]
 [51.17741244 66.94799347]]
km_cov is
[[[20.13977456  0.        ]
  [ 0.         11.59866325]]

 [[28.80272794  0.        ]
  [ 0.         23.79220294]]

 [[15.16174652  0.        ]
  [ 0.         28.49450057]]]
```

```
2D------ EM ------
em_pi is
[0.64808788 0.13728696 0.21462516]
em_mean is
[[49.29588657 49.32682183]
 [50.84181061 67.63777554]
 [69.34473712 49.82096606]]
em_cov is
[[[33.90579847  7.39501307]
  [ 7.39501307 27.48950891]]

 [[12.05465449 -2.44394891]
  [-2.44394891 29.10607745]]

 [[14.90114348  4.88954954]
  [ 4.88954954 10.06525907]]]
```





3. 三维数据

真实参数值，其中 N=1000

```
freq = [0.5, 0.3, 0.2]
mean0 = [50, 50, 50]
cov0 = [[50, 5, 10],
        [5, 50, 5],
        [10, 5, 50]]
mean1 = [100, 50, 40]
cov1 = [[30, 5, 5],
        [5, 30, 5],
        [5, 5, 30]]
mean2 = [50, 100, 100]
cov2 = [[40, 5, 15],
        [5, 40, 5],
        [15, 5, 40]]
```

学习得到的参数值

```
3D------ Kmeans ------
km_pi is
[0.5 0.2 0.3]
km_mean is
[[ 50.10394937  49.5540353   49.57937324]
 [ 50.41066577  99.96667286 100.31552009]
 [ 99.80886224  49.99112842  39.88042208]]
km_cov is
[[[52.17185385  0.          0.        ]
  [ 0.         50.33610502  0.        ]
  [ 0.          0.         50.79145462]]

 [[39.08688057  0.          0.        ]
  [ 0.         46.30521902  0.        ]
  [ 0.          0.         47.03886317]]

 [[28.80956249  0.          0.        ]
  [ 0.         27.42110648  0.        ]
  [ 0.          0.         26.98326977]]]
```

```
3D------ EM ------
em_pi is
[0.50019679 0.29980321 0.2       ]
em_mean is
[[ 50.11492846  49.55157981  49.57588391]
 [ 99.82317094  49.99551213  39.87987733]
 [ 50.41066577  99.96667287 100.3155201 ]]
em_cov is
[[[52.45764642  2.13463831 12.22149994]
  [ 2.13463831 50.33163514  3.53656255]
  [12.22149994  3.53656255 50.80241823]]

 [[28.51626745  3.18967699  3.16029472]
  [ 3.18967699 27.40978686  2.91962863]
  [ 3.16029472  2.91962863 27.00051052]]

 [[39.08688058  5.04103924 19.36734571]
  [ 5.04103924 46.3052189   5.58738597]
  [19.36734571  5.58738597 47.0388631 ]]]
```

四、分析与讨论

1. Kmeans VS EM

以二维相近数据为例进行分析，从散点图可以直观发现，基于距离的 Kmeans 方法，会使得在真实情况中较小的区域被放大，相应地真实较大的区域会被缩小。就高斯分布而言，当 N 变大时，在协方差的值适当时，在该高斯分布的边缘区域，会有一定数量的点，而这些点距离该高斯分布的中心点较远，却有可能距离其他高斯分布的中心点较近，从而错误聚类。而基于概率的 EM 算法，在尽管也存在这种现象，但程序较轻。观察学习得到的π_k，相比于 Kmeans，EM 算法得到的结果更贴近真实值，与直观发现相印证。

2. EM 的初始化

在实践中发现，EM 算法的初始化对结果有着直接影响。实际上，EM 算法得到的是局部最优解，而非全局最优解。因此，不同的初始化结果，极有可能导致结果落入不同的局部最优解。

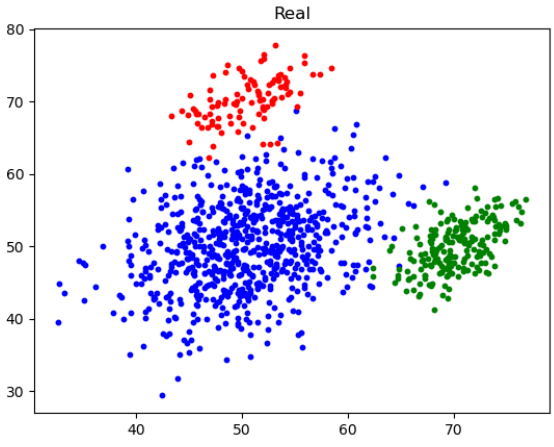
不同于前文中的以 Kmeans 的结果作为 EM 的参数初始值，我们作如下初始化：π_k统一初始化为 1/K，任取数据集中 k 个样本作为u_k，σ_k^2则初始化为单位阵。因为中心点的初始化是随机的，这会导致落入不同的局部最优解，从而产生不同的结果，准确率没有有力的保证。以下列举了两次尝试过程，其中尝试 1 的结果显然较好，尝试 2 的结果则较差。

利用 Kmeans 的结果作为初始值，结果的稳定性较高。但同样地，结果是局部最优解而非全局最优解。使用随机的方式，我们有可能得到极佳的学习结果；而使用 Kmeans 方法，由于结果较稳定，常常距全局最优解有一定差距。

尝试 1：

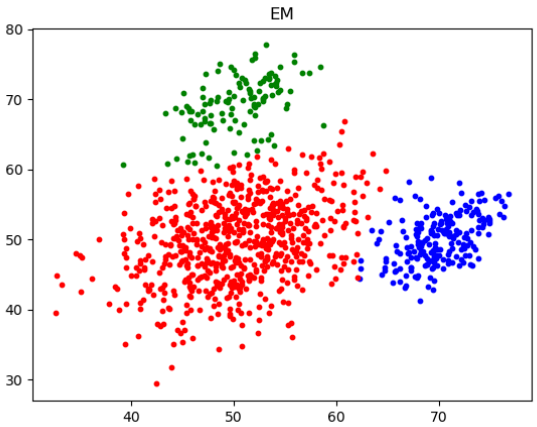
真实参数值，N=1000

```
freq = [0.7, 0.2, 0.1]

mean0 = [50, 50]
cov0 = [[35, 10], [10, 35]]
mean1 = [70, 50]
cov1 = [[10, 5], [5, 10]]
mean2 = [50, 70]
cov2 = [[10, 5], [5, 10]]
```



学习得到的参数值

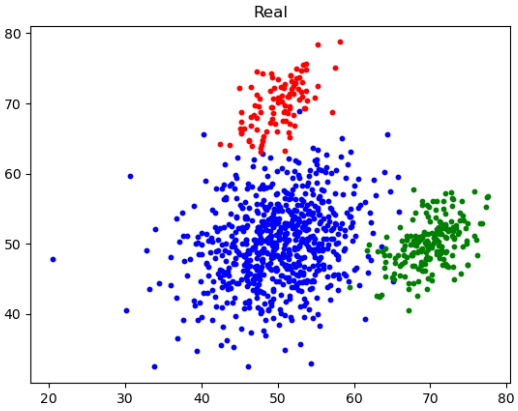
```
2D------ EM ------
em_pi is
[0.20778396 0.12024352 0.67197253]
em_mean is
[[69.92092045 50.07225219]
 [50.27986262 68.82384324]
 [50.26085879 49.74883329]]
em_cov is
[[[ 9.4297739   4.6300557 ]
  [ 4.6300557  11.91394528]]

 [[12.95641303  7.88256762]
  [ 7.88256762 20.99739306]]

 [[32.61632613 11.38166143]
  [11.38166143 34.13688171]]]
```



尝试 2：

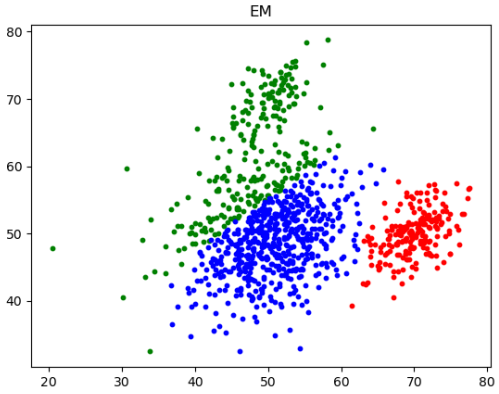
真实参数值，其中 N＝1000

```
freq = [0.7, 0.2, 0.1]

mean0 = [50, 50]
cov0 = [[35, 10], [10, 35]]
mean1 = [70, 50]
cov1 = [[10, 5], [5, 10]]
mean2 = [50, 70]
cov2 = [[10, 5], [5, 10]]
```



学习得到的参数值

```
2D------ EM ------
em_pi is
[0.50447906 0.29670351 0.19881743]
em_mean is
[[51.15678872 48.73513799]
 [48.00724505 59.05300241]
 [69.88284041 49.94621466]]
em_cov is
[[[29.36770878 10.97282534]
  [10.97282534 26.68444885]]

 [[29.40036438 25.16136167]
  [25.16136167 87.28196308]]

 [[10.37041428  5.27301015]
  [ 5.27301015 11.29722079]]]
```



五、运行指令

1.运行模型

直接键入 python source.py 即可

2.修改数据的生成参数

可在 data.py 文件的 generate_2D_data 和 generate_3D_data 中修改 π_k, u_k, σ_k^2，从而生成不同分布的数据。此外，通过 N 值可控制生成的样本数。

3.变换 EM 的初始化方式

在 EM.py 文件的 myEM()函数中，有如下代码：

```
# 初始化
    # pi, mean, cov, tmp = kmeans(X, K, max_epoch=km_epoch)
    pi, mean, cov = simple_init(X, K)
```

通过注释不同行，可选不同的初始化方式，如上选择了第二种初始化方式。

附注

本文中的系列公式来源于：邱锡鹏,神经网络与深度学习,机械工业出版社,2020，ISBN 9787111649687.