Assignment-3

高斯混合模型

复旦大学计算机系

概要

项目主要完成了聚类数据集创建、高斯混合模型构建与 K-Means 模型构建，同时比较了高斯混合模型与 K-Means 模型的异同与特点，并通过可视化的手段对于模型进行分析。

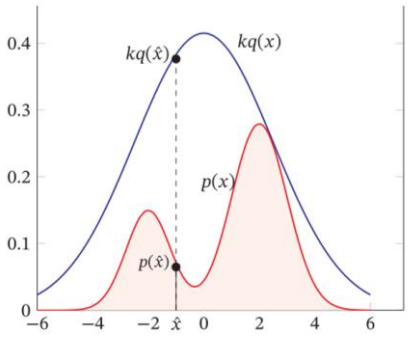
1. 数据集采样

1.1 拒绝采样

1.1.1 拒绝采样概要

拒绝采样（Rejection Sampling）是一种间接采样方法。对于原始分布 p(x)，可引入一个容易采样的分布 q(x)，一般称为提议分布（Proposal Distribution），然后以某个标准来拒绝一部分的样本使得最终采集的样本服从分布 p(x)。

在拒绝采样中，已知未归一化的分布 p'(x)，需要构建一个提议分布 q(x)和一个常数 k，使得 kq(x)可以覆盖函数 p'(x)，即 $kq(x) \geqslant p'(x), \forall x$，如课程 PPT 图所示：



1.1.2 代码实现

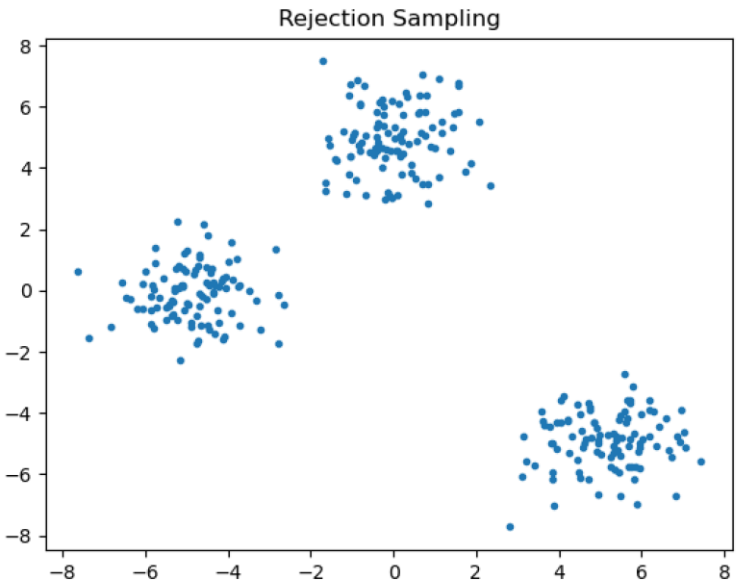
拒绝采样代码实现在 handout/package.py 的 multivariate_normal 函数中，整体构建的主要思路是以均匀采样作为提议分布：先利用均匀分布生成一定范围之内的随机坐标(a,b)，计算参数给定的正态分布下该坐标的概率 P，然后将 P 除以正态分布概率最大值 Pmax 获得接受概率 P'。此时 P'属于[0,1]，即对于该正态分布下的 1 个落在坐标（a,b）的点，采样时有 P'的概率接受，也有 1-P'的概率拒绝。再利用均匀分布生成一个[0,1]之间的随机数 U，如果 $U \leqslant P'$，则被接受并加入列表，反之 U 不被接受，继续上述过程直到样本数量足够。 这里由于最大值 Pmax 即为公式 exp 项之前的常系数项，因此

可以在计算时不考虑常系数项，直接将 exp 项与 1 的比值作为 P 与 Pmax 的比值。

具体体现上述部分的核心代码如下：

```
while len(sample) != sample_num:
    u = (np.random.rand(mean.shape[0]) *2* expand_size - expand_size) + mean
    exp_part = -0.5 * np.mat(u - mean) * np.mat(cov) * np.mat(u - mean).T
    generator = np.random.rand(1)
    if np.exp(exp_part) >= generator:
        sample.append(u)
```

1.1.2 结果可视化



不带标签的数据集

可以自定每个高斯分布的维度/平均值/协方差/数据范围/采样点数，在之后的实验中会主要使用由拒绝采样生成的样本。

1.2 Box-Muller 采样

1.2.1 Box-Muller 采样概要（简要证明）

Box-Muller 采样也同样是一种间接采样，利用概率密度函数的数学性质进行映射，以通过两个独立的遵从均匀分布的随机数得到两个独立的正态分布的随机数。从均匀分布取

得随机数 X_1，X_2 出发，令函数 g、h 各自符合以下两式：

$$g(x_1,x_2)=\sqrt{-2lnx_1}cos(2\pi x_2)\quad h(x_1,x_2)=\sqrt{-2lnx_2}sin(2\pi x_1)$$

并令：$y_1=g(x_1,x_2)\quad y_2=h(x_1,x_2)$，同时联立上式可有：$x_1=e^{-(y_1^2+y_2^2)/2}$

可以通过概率密度函数的变换公式得到：

$$f_{Y_1Y_2}(y_1,y_2)=1/\left|\partial(y_1,y_2)/\partial(x_1,x_2)\right|f_{X_1,X_2}(x_1,x_2)$$

其中：$\frac{\partial(y_1,y_2)}{\partial(x_1,x_2)}$ 为 Jacobi 矩阵，且由于 X_1，X_2 符合[0,1)上的均匀分布，有$f_{X_1X_2}(x_1,x_2)=1$

计算 Jacobi 矩阵有：$\frac{\partial(y_1,y_2)}{\partial(x_1,x_2)}=-\frac{2\pi}{x_1}$

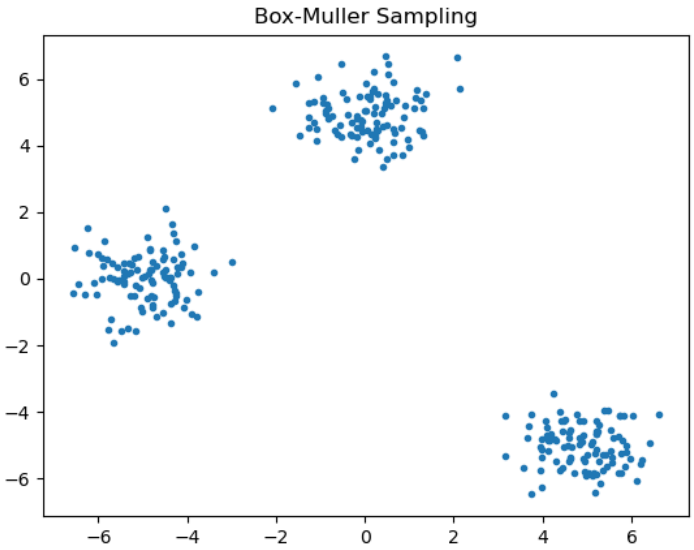
则有：$f_{Y_1Y_2}(y_1,y_2)=1/(2\pi e^{-\frac{y_1^2+y_2^2}{2}})$，正是二元标准正态分布的概率密度函数。

1.2.2 代码实现

Box-Muller 采样实现较为简单，位于 handout/package.py 的 multivariate_normal_boxmuller 函数之中。具体实现只需要通过 numpy 模块将均匀选取的随机数按照上述公式进行转化即可，具体核心代码如下：

```
sample = []
while len(sample) != sample_num:
    u, v = np.random.rand(mean.shape[0]), np.random.rand(mean.shape[0])
    u_log, v_the = -np.log(u), 2 * np.pi * v
    u_sqr = np.sqrt(u_log)
    x = u_sqr * np.cos(v_the)
    data = x * cov + mean
    sample.append(data)
return np.stack(sample, axis=0)
```

1.2.3 结果可视化



1.3 采样方法比较与分析

拒绝采样：在实际实验中，拒绝采样要对每个潜在样本点进行概率计算，因此对于低维度情况效果更友好。同时对于样本点较大的情况下，拒绝采样的采样效率需要着重考虑（即总体的接受率），比 Box-Muller 稍复杂。如果提议函数远大于原始分布函数，拒绝率往往会较高；因此需要寻找与原始分布较为接近的提议分布，可能会增加难度。但是对于项目中较低维（项目中选用 2 维，也可以更高）且数据点不是非常多情况下，该方法十分理想。

Box-Muller 采样：Box-Muller 采样的计算复杂度较低，只需要通过简单变换就可以生成低维正态分布随机变量，且不需要考虑拒绝率的问题。但其较为弱势的一点是，对于高维数据而言，需要进一步处理将各个单独产生的概率联合，以保证变量间独立性等因素（协方差之类的信息）而拒绝采样则不用经过这类处理。

之后的实验会采用拒绝采样作为数据生成手段。

2. 高斯混合模型与 K-Means 聚类

2.1 高斯混合模型（GMM）

2.1.1 高斯混合模型概要

高斯混合模型（Gaussian Mixture Model, GMM）是由多个高斯分布组成的模型，其总体密度函数为多个高斯密度函数的加权组合，其最特殊的性质是在参数估计中无法观测到样本具体由哪个分布生成。模型主体是以期望最大化算法（Expectation-Maximum，EM 算法）为核心的参数估计方法，主要常用于聚类过程。

高斯混合模型利用边际似然函数来建模包含可观测变量与隐变量（不可观测的变量）的联合概率，并以此作为最大化目标函数使其收敛。随后在迭代过程中以 E 步（以固定模型参数计算高斯分布的后验概率）、M 步（以固定后验概率计算模型参数，使得边际似然函数最大）相继迭代，直至边际似然函数最终收敛。

其核心计算式为课程 PPT 中：

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

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

其中$N_k = \sum_{n=1}^{N}\gamma_{nk}$，$\gamma_{nk}$为样本 $x^{(n)}$属于第 k 个高斯分布的后验概率。

2.1.2 高斯混合模型代码实现

项目中高斯混合模型代码位于 handout/package.py 的 GaussianMixtureModel 类内，其中__init__函数主要负责数据读取与参数初始化，**train 函数**主要负责模型迭代与最终结果可视化，**y_expect_cal 函数**用于以固定的模型参数计算分布的后验概率（E 步），**para_cal 函数**用于固定后验概率计算模型参数（M 步）。以下为各个函数实现核心部分：

Train 函数部分：

在计数迭代中依次执行 E 步（第 2 行）/M 步（第 3 行）/计算期望（第 4 行）

```
1.  for _ in range(self.count):
2.      self.gamma, _ = self.y_expect_cal(self.x, self.pi, self.mean, self.cov)
3.      self.pi, self.mean, self.cov = self.para_cal(self.x, self.gamma)
4.      _, expect = self.y_expect_cal(self.x, self.pi, self.mean, self.cov)
```

Y_expect_cal 函数核心部分：

固定模型参数在两层循环中根据 3.1.1 所示公式进行后验概率计算，这里隐去了部分变量设定行，展示代码主要是根据高维高斯分布公式进行概率计算，并乘上先验概率，在后续进一步得到后验概率。

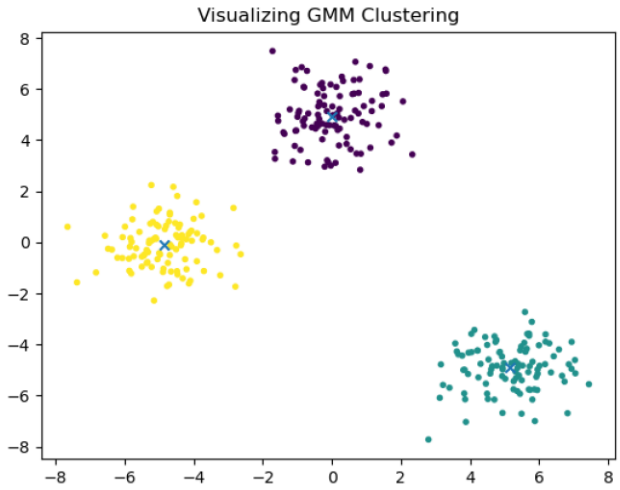
```
for n in range(x.shape[0]):
    for k in range(mean.shape[0]):
        norm_part = 1 / ((2 * np.pi) ** ((x.shape[1]) / 2) * ((np.linalg.det(cov[k, :, :])) ** 0.5))
        vec = (x[n, :] - mean[k, :])
        exp_part = -0.5 * np.dot(np.dot(vec, np.linalg.pinv(cov[k, :, :])), vec.T)
        k_list.append(norm_part * np.exp(exp_part) * pi[k])
```

para_cal 函数核心部分：

固定后验概率在单层循环中根据 3.1.1 所示公式进行参数计算。这里隐去了部分变量设定行，展示代码主要是根据前述公式计算高斯分布的平均值和协方差矩阵。

```
for k in range(pi.shape[0]):
    for n in range(x.shape[0]):
        uk += gamma[n, k] * x[n, :]
    uk = uk / pi[k]
    for n in range(x.shape[0]):
        sigmak += gamma[n, k] * np.dot((x[n, :] - uk)[:, None], (x[n, :].squeeze() - uk)[None, :])
    sigmak = sigmak / pi[k]
```

2.1.3 结果可视化



可见 GMM 模型在该数据集上效果不错，聚类表现好且平均值（蓝色 x 标记）都位于各类中心。其余数据集上的模型表现和与 K-Means 模型的优劣分析将会在 2.3 节中进行。

2.2 K-Means

2.2.1 K-Means 模型概要

K-Means 方法是一种迭代求解的聚类算法，先随机选取 K 个中心点作为初始化状态，迭代中主要步骤由两步组成：1.分配步：计算每个样本与 K 个中心点距离并将样本归入与其距离最近的中心点所在类，即以最近距离中心点为基准对样本进行划分；2.更新步：计算每个类内样本点的均值坐标，并将其作为新的中心点。反复循环直至最后收敛。

2.2.2 K-Means 代码实现

项目中 KMeans 模型代码位于 handout/package.py 的 KMeansModel 类内。其类内主要函数为：**__init__函数**，主要用于模型参数初始化和数据读取；**train 函数**，主要负责模型迭代与最终结果可视化；**label_cal 函数**，主要用于通过当前中心点计算各个样本的标签（所对应的类）；**center_cal 函数**，主要用于通过各个样本的标签计算新的各类中心点。

Label_cal 函数：

直接在双循环内计算样本与中心点的距离，并通过 np.argmin()方法获得最小距离中心点的索引号，作为新的 label 标记。

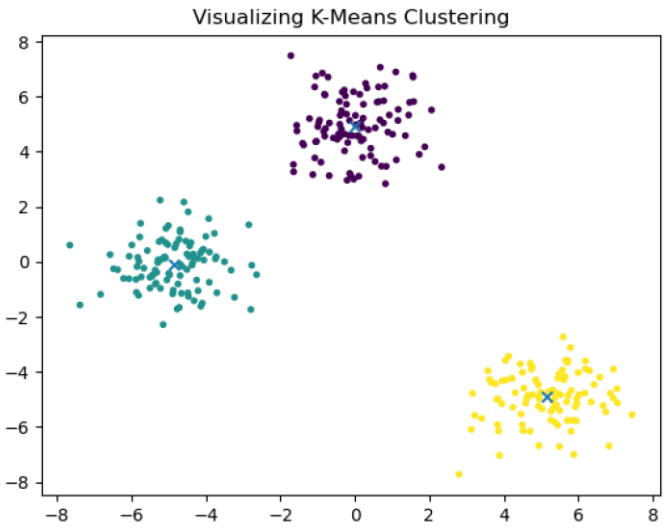
```
1.  for n in range(x.shape[0]):
2.      for k in range(center.shape[0]):
3.          k_list.append(((x[n, :] - center[k, :]) ** 2).sum())
4.      k_np = np.stack(k_list, axis=0)
5.      label.append(k_np.argmin())
```

Center_cal 函数:

在单循环内获得每类的样本点，并在类内求均值坐标。

```
1.  for k in range(k_num):
2.      dots = data[data[:, -1] == k]
3.      center.append(dots[:, :-1].mean(axis=0
```

2.2.3 结果可视化



可见在该数据集上，k-Means 算法效果也很好，聚类有效且中心点位置都在类中心。

2.3 高斯混合模型与 K-Means 算法比较

2.3.1 思想比较

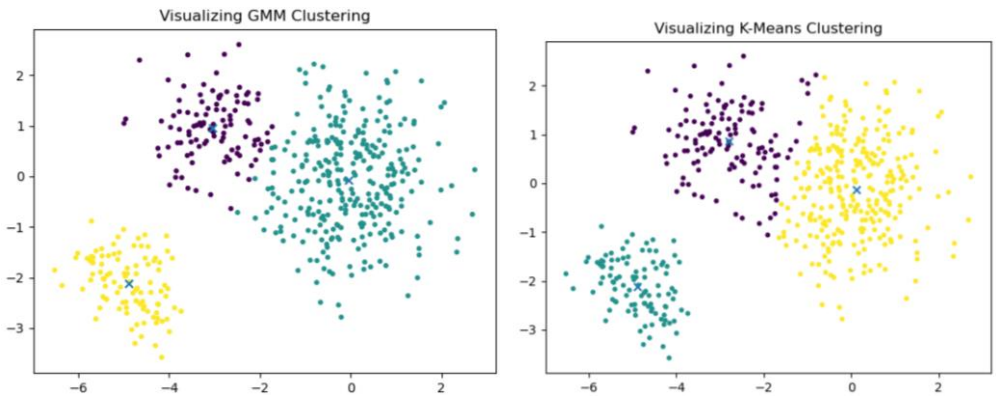
（相同）高斯混合模型和 K-Means 算法在整体架构上十分类似，都是在迭代内进行一步双循环操作和一步单循环操作。第一步双循环操作都是将样本与分布/类进行关联，第二步单循环操作都是对分布/类进行调整，更新相关参数。

（不同）但是，高斯混合模型相较于 K-Means 算法优势在于，它从概率意义上对于分布的参数进行了估计，它的每个样本点并不单独属于某个类，而是以一定概率属于某一类，是对于隐变量的考虑，这使得它的结果更加平滑（软聚类）。

因此高斯混合模型能对每个类的参数，尤其是分布方差等细节参数进行更好的估计，而 K-Means 只局限于中心点与样本距离（硬聚类），导致后者确定的类大小都差不多，而且对于重叠的数据集，K-Means 效果会相对不够好，同时这也意味这 K-Means 无法良好抓取高斯分布信息时，非常依赖于中心点初始化的选择。

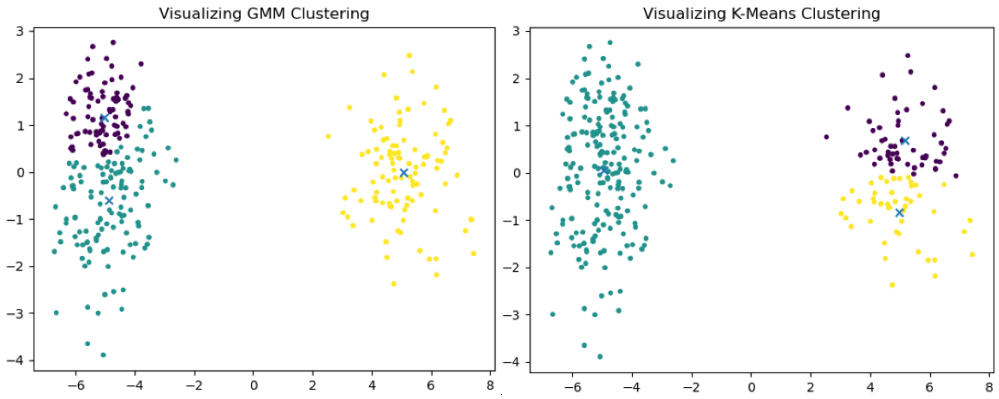
2.3.2 实验说明

不均衡数据集测试：



以上两图分别是高斯混合模型（左）和 K-Means（右）对于不均衡数据集（文件 data1.data）进行的聚类。原始数据在图像坐标(0,0)位置处有一规模更大的高斯分布，从图中可见高斯混合模型能够很好的将数据聚类成不同规模大小的高斯分布，而 Kmeans 做不到。这符合 2.3.1 节的讨论，因为高斯混合模型有更好的参数估计，而不受限于单独的距离因素。

距离较远数据集测试：

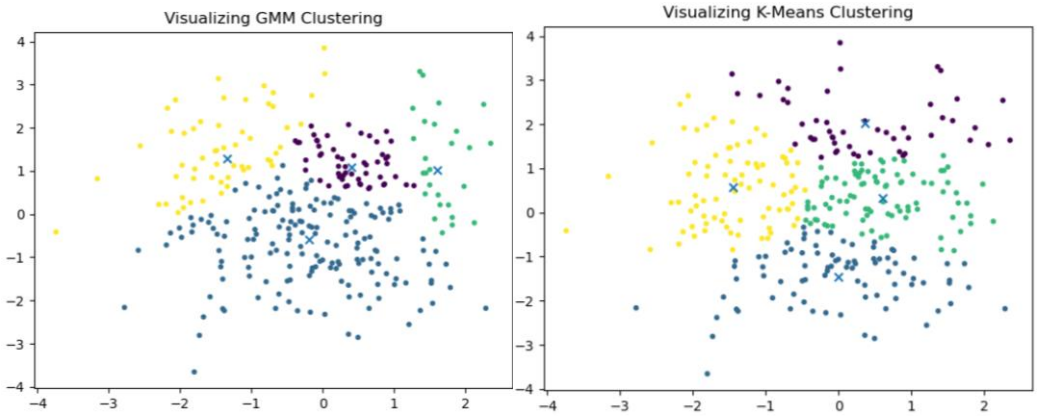


以上两图分别是高斯混合（左）和 K-Means（右）对距离较远的同规模多个高斯的数据集（文件 data2.data）聚类结果。原始数据在(-5,-1), (-5, 1), (5, 0)有三个不同协方差的高斯分布，可见由于 K-Means 算法受距离影响巨大，使得其在多个高斯模型中，如果各别高斯距离相差较大，且参数初始化不够理想，初始中心点没有合理分配时，容易导致一个中心点占据多个高斯模型的情况。但对于 GMM 而言，由于其能够学习到高斯分布的参数特征，并且是软聚类，可以避免上述情况的发生。

这同样也意味着 K-Means 作为 GMM 的初始化效果不一定会更好，因为 K-Means 在考虑距离因素时，反而可能会受距离影响，使聚类效果变差。

2.3.3 对 K 值选择的猜测

选取距离较近的规模相同的数据集（包含三个高斯分布，存储在文件 data3.data），但是在 k=4 情况下应用高斯混合模型和 Kmeans 方法进行聚类：



原始数据是分布在(0, -1), (-1, 0.5), (0.5, 1)的等规模等参数高斯分布，可见 GMM 模型的中心点比起 Kmeans 的中心点更靠近类分界边，这是其软聚类性质造成的。

对于 Kmeans 算法而言，其掌握不到分布细节信息，也就无法对聚类效果进行评定。

但是对于 GMM 而言，或许可以通过不同 k 下的期望最大化情况来决定哪个聚类更加优秀，虽然 k 更大时期望也会更大或者也考虑分布的对称性作为分布的评估以及分布融合之类。

3.程序执行与文件说明

python source.py

data1.data 是不均衡数据集/data2.data 是距离较远数据集/data3.data 是等规模且紧凑数据集.

可以在 source.py 内直接生成数据集，也可以自主选定数据集。

在指定数据集时，首先要将数据集 data 文件放在 source.py 同文件夹下，然后在 source.py 的模型内设置 file_name = 'data1.data'

```
model = GaussianMixtureModel(maxiter=50, k=3, file_name='data1.data')
model.train()
```