LAB3: K-Means & GMM

21304219
刘文婧

Requirements

1. 实现 K-Means 算法及用 EM 算法训练 GMM 模型的代码。可调用 numpy, scipy 等软件包中的基本运算，但不能直接调用机器学习包（如 sklearn）中上述算法的实现函数；
2. 在 K-Means 实验中，探索两种不同初始化方法对聚类性能的影响；
3. 在 GMM 实验中，探索使用不同结构的协方差矩阵（如：对角且元素值都相等、对角但对元素值不要求相等、普通矩阵等）对聚类性能的影响。同时，也观察不同初始化对最后结果的影响；
4. 在给定的训练集上训练模型，并在测试集上验证其性能。使用聚类精度(Clustering Accuracy, ACC)作为聚类性能的评价指标。由于 MNIST 数据集有 10 类，故在实验中固定簇类数为 10。

Overview of K-Means & GMM model

> 从 latent-variable model 到 EM algorithm 训练 GMM，和 K-Means 和 Soft K-Means，相关公式、算法训练流程

- K-Means 和 GMM 都是无监督学习，也就是说，现在我们有很多无标签样本 $\{\mathbf{X}^{(n)}\}$。

Latent-Variable Model

1. 我们希望找到一个这些样本服从的分布 $p(x)$，然后甚至利用这个分布去生成新的 x, 从统计学的角度，就是极大化对数似然函数 $\log p(x^{(1)}, x^{(2)}..., x^{(n)})$
2. 然而，直接计算 $p(x)$ 并不容易，我们想到，这些样本背后应该是有一定结构的，也就是说，我们可以引入隐变量 z，有：

$$p(\boldsymbol{x}) = \int_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}, \quad p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$$

3. 这样，用 $p(\boldsymbol{x}|\boldsymbol{z}), p(\boldsymbol{z})$ 两个概率去算就好了。

Gaussian Latent-Variable Model

1. 最直接地，都认为服从高斯分布（同时，高斯分布具有线性性，后面可以看到，这会方便对 $p(x)$ 的计算）：

$$\boldsymbol{z} \sim N(\boldsymbol{0}, \boldsymbol{I}), \quad \boldsymbol{x}|\boldsymbol{z} \sim N(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$$

2. 根据 $p(\boldsymbol{x}|\boldsymbol{z})=N(\boldsymbol{W}\boldsymbol{z}+\boldsymbol{\mu},\sigma^2\boldsymbol{I})$，我们就能生成 样本点或数据点 $\boldsymbol{x}_n$，**<u>生成方式：均值+噪声，噪声服从方差对应的均值为0的高斯分布</u>**，即：

$$\boldsymbol{x}_n=\boldsymbol{W}\boldsymbol{z}_n+\boldsymbol{\mu}+\boldsymbol{\epsilon}_n,\quad \boldsymbol{z}_n\sim N(\boldsymbol{0},\boldsymbol{I}),\quad \boldsymbol{\epsilon}_n\sim N(\boldsymbol{0},\sigma^2\boldsymbol{I})$$

3. 由于高斯分布具有线性性，现在， $\boldsymbol{x}_n$ 服从的也是高斯分布，而且均值和方差容易计算如下：（噪声和隐变量是独立的）

$$\begin{aligned}
&\mathrm{E}[\boldsymbol{x}_n]=\mathrm{E}[\boldsymbol{W}\boldsymbol{z}_n+\boldsymbol{\mu}+\boldsymbol{\epsilon}_n]=\boldsymbol{\mu},\\
\mathrm{Var}[\boldsymbol{x}_n]=&\mathrm{E}\left[(\boldsymbol{W}\boldsymbol{z}_n+\boldsymbol{\epsilon}_n)(\boldsymbol{W}\boldsymbol{z}_n+\boldsymbol{\epsilon}_n)^\top\right]\\
=&\mathrm{E}\left[\boldsymbol{W}\boldsymbol{z}_n\boldsymbol{z}_n^\top\boldsymbol{W}^\top+\boldsymbol{W}\boldsymbol{z}_n\boldsymbol{\epsilon}_n^\top+\boldsymbol{\epsilon}_n\boldsymbol{z}_n^\top\boldsymbol{W}^\top+\boldsymbol{\epsilon}_n\boldsymbol{\epsilon}_n^\top\right]\\
=&\boldsymbol{W}\mathrm{E}[\boldsymbol{z}_n\boldsymbol{z}_n^\top]\boldsymbol{W}^\top+\mathrm{E}[\boldsymbol{\epsilon}_n\boldsymbol{\epsilon}_n^\top]\\
=&\boldsymbol{W}\boldsymbol{W}^\top+\sigma^2\boldsymbol{I}\\
&\Longrightarrow\ \boldsymbol{x}\sim N(\boldsymbol{\mu},\boldsymbol{W}\boldsymbol{W}^\top+\sigma^2\boldsymbol{I})
\end{aligned}$$

6. 可见，此时我们能生成较复杂的 $p(x)$ 了，同时，用我们的样本 $\boldsymbol{x}_n$ 来训练这个分布中的 $\boldsymbol{W},\sigma$ 参数，对似然函数求导得：（记 $\boldsymbol{\Sigma}=\boldsymbol{W}\boldsymbol{W}^\top+\sigma^2\boldsymbol{I}$）

$$\log p(\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_N)=\sum_{n=1}^{N}\log N(\boldsymbol{x}_n;\boldsymbol{\mu},\boldsymbol{W}\boldsymbol{W}^\top+\sigma^2\boldsymbol{I})$$

$$\begin{cases}\frac{\partial}{\partial\boldsymbol{\mu}}\log p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)=0\\ \frac{\partial}{\partial\boldsymbol{\Sigma}}\log p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N)=0\end{cases}$$

$$\begin{cases}\boldsymbol{\mu}=\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n=\bar{\boldsymbol{x}}\\ \boldsymbol{\Sigma}=\boldsymbol{W}\boldsymbol{W}^\top+\sigma^2\boldsymbol{I}=\boldsymbol{S},\boldsymbol{S}\text{ denotes the covariance matrix}\end{cases}$$

7. 此时发现， $\boldsymbol{W}\boldsymbol{W}^\top+\sigma^2\boldsymbol{I}=\boldsymbol{S}=\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$，即 $\boldsymbol{W}=\boldsymbol{U}(\boldsymbol{\Lambda}-\sigma^2\boldsymbol{I})^{\frac{1}{2}}$，可以看出来和主成分分析PCA的一些关系：此时的权重矩阵 $\boldsymbol{W}$ 相当于是特征向量（主成分） $\boldsymbol{u}_i$ 组成的矩阵 $\boldsymbol{U}$ 进行一定的放缩，即：

$$\boldsymbol{u}_i\xrightarrow{\text{scale}}\boldsymbol{u}_i\sqrt{\lambda_i-\sigma^2},\ \lambda_i\text{ denotes the i-th eigenvalue}$$

Gaussian Mixture Model

1. 高斯混合模型，每簇权重 π_k，每簇内，样本都服从一个高斯分布， $p(\boldsymbol{x})$ 即：

$$p(\boldsymbol{x})=\sum_{k=1}^{K}\pi_kN(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$$

从隐变量模型的角度看高斯混合模型：

2. 隐变量服从的是 Categorical Distribution，即：

$$p(\boldsymbol{z})=\prod_{k=1}^{K}\pi_k^{z_k},\quad \text{or}\quad p(\boldsymbol{z}=\mathbf{1}_k)=\pi_k$$

其中， $\mathbf{1}_k$ 是 One-Hot 编码， $\boldsymbol{z}_k$ 是 $\boldsymbol{z}$ 的第 k 个分量， $\in\{0,1\}$。

3. 并且，对每个隐变量 $\boldsymbol{z}=\mathbf{1}_k$， 设变量 $\boldsymbol{x}$ 分别属于某第 k 个高斯分布，即给出他们的 joint distribution：

$$p(\boldsymbol{x},\boldsymbol{z})=\prod_{k=1}^{K}\left[\pi_k N(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right]^{z_k},\quad \text{or}\quad p(\boldsymbol{x},z=1_k)=\pi_k N(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$$

4. 当然，此时也容易给出隐变量 $\boldsymbol{z}$ 对变量 $\boldsymbol{x}$ 的后验分布，对 $\boldsymbol{z}=\mathbf{1}_k$，这个后验分布指出的正是样本属于第 $\boldsymbol{k}$ 簇的概率：

$$p(\boldsymbol{z}|\boldsymbol{x})=\frac{\prod_{k=1}^{K}\left[\pi_k N(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right]^{z_k}}{\sum_{k=1}^{K}\pi_k N(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}$$

$$p(\boldsymbol{z}=\mathbf{1}_k|\boldsymbol{x})=\frac{\pi_k N(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}{\sum_{k=1}^{K}\pi_k N(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}=p(\boldsymbol{x}\text{ in k-th cluster})$$

5. 此时，我们的目标也是去极大化对数似然函数：

$$\log p(\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_N)=\sum_{n=1}^{N}\log\sum_{k=1}^{K}\pi_k N(\boldsymbol{x}_n;\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$$

6. 但此时求偏导、然后梯度下降的方式计算较复杂：

$$\begin{cases}\frac{\partial}{\partial\boldsymbol{\mu}_k}\log p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N),k=1,2...K\\ \frac{\partial}{\partial\boldsymbol{\Sigma}_k}\log p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N),k=1,2...K\\ \frac{\partial}{\partial\boldsymbol{\pi}_k}\log p(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N),k=1,2,...K\end{cases}$$

7. 那么我们换成使用EM算法来训练GMM。

Expectation-Maximization Algorithm

1. 回到最开始的想法，我们还是希望尽量好地拟合

$$p(\boldsymbol{x};\boldsymbol{\theta})\quad \text{s.t.}\quad \log p(\boldsymbol{x};\boldsymbol{\theta})\text{ larger}$$

2. 引入辅助分布 $q(\boldsymbol{z})$，且 $q(\boldsymbol{z})$ 是对后验分布 $p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})$ 的一种近似，把单个样本的对数概率函数写成：

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

其中，后一项 $\sum_{\boldsymbol{z}} q(\boldsymbol{z}) \log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})}$ 给出的是 $\mathcal{KL}$ 散度 $\mathcal{KL}(q(\boldsymbol{z}) \,||\, p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}))$，表示两个函数之间的距离；

前一项则给出的是函数 $\log p(\boldsymbol{x};\boldsymbol{\theta})$ 减去 $\mathcal{KL}$ 距离之后，相应的一个 $LowerBound$。

3. 要让 $q(\boldsymbol{z})$ 对后验分布 $p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})$ 的近似效果最好，比如在某个 t 时刻的迭代中，得到$p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)})$，就取如下的 $q(\boldsymbol{z})$，使得 $\mathcal{KL}$ 散度 为0：

$$
q(\boldsymbol{z}) = p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)})
$$

4. Now $LowerBound$：

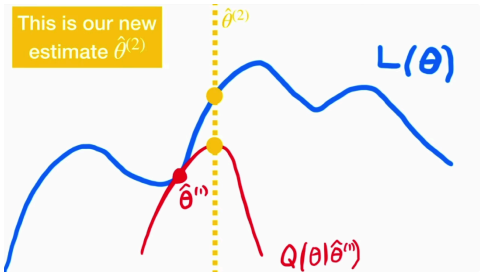
$$
\sum_{\boldsymbol{z}} q(\boldsymbol{z}) \log \frac{p(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta})}{q(\boldsymbol{z})} = \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)}) \log \frac{p(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta})}{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)})}
$$

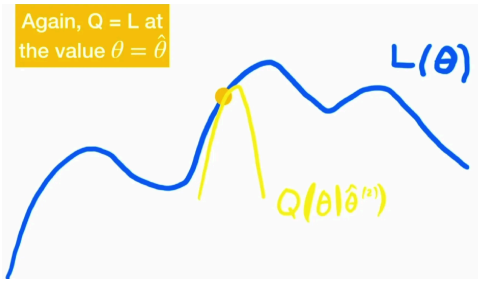
5. 接下来提升这个 $LowerBound$，即，$t+1$ 时刻的迭代，选一个 $\boldsymbol{\theta}^{(t+1)}$ 让下界函数值尽可能地更大：

$$
\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)}) \log \frac{p(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta})}{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)})}
$$

也即，

$$
\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)}) \log p(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta})
$$





6. 这就是EM算法，也即：

$$
\text{E-step}: \quad Q(\boldsymbol{\theta},\boldsymbol{\theta}^{(t)}) = \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}^{(t)})} \left[\log p(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta})\right]
$$

这是只针对 1 个样本 x 的，针对 N 个样本，即

$$\text{E-step}: \quad Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{p(\boldsymbol{z}^{(n)}|\boldsymbol{x}^{(n)};\boldsymbol{\theta}^{(t)})} \left[\log p(\boldsymbol{x}^{(n)}, \boldsymbol{z}^{(n)}; \boldsymbol{\theta}) \right]$$

$$\text{M-step}: \quad \boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$$

EM算法保证了每一轮迭代后，对每个样本 x 估计的概率对数似然函数总是不降的：

$$\log p(\boldsymbol{x}; \boldsymbol{\theta}^{(t+1)}) \geq \sum_{\boldsymbol{z}} p(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}^{(t)}) \log \frac{p(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}^{(t+1)})}{p(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}^{(t)})} \geq \log p(\boldsymbol{x}; \boldsymbol{\theta}^{(t)})$$

Train GMM with EM Algorithm

$$\text{E-step}: \quad Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{p(\boldsymbol{z}^{(n)}|\boldsymbol{x}^{(n)};\boldsymbol{\theta}^{(t)})} \left[\log p(\boldsymbol{x}^{(n)}, \boldsymbol{z}^{(n)}; \boldsymbol{\theta}) \right]$$

$$\text{M-step}: \quad \boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$$

1. 代入GMM的概率分布式：

$$p(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) = \prod_{k=1}^{K} \left[\pi_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right]^{z_k}, \; p(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{\prod_{k=1}^{K} \left[\pi_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right]^{z_k}}{\sum_{k=1}^{K} \pi_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

2. 得到EM算法训练GMM的公式：

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{\pi_k^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{i=1}^{K} \pi_i^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_i^{(t)}, \boldsymbol{\Sigma}_i^{(t)})} \log \pi_k N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$$

其中，$\boldsymbol{\theta}^{(t)} = \{\pi_k^{(t)}, \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})\}_{k=1}^{K}$

记

$$\gamma_{nk}^{(t)} = \frac{\pi_k^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{i=1}^{K} \pi_i^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_i^{(t)}, \boldsymbol{\Sigma}_i^{(t)})}$$

并在 $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$ 表达式中代入高斯分布表达式 $N = \frac{\exp\left\{-\frac{1}{2}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k)\right\}}{(2\pi)^{\frac{d}{2}} |\boldsymbol{\Sigma}_k|^{\frac{1}{2}}}$，得：

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk}^{(t)} \left[-\frac{1}{2} (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k) - \frac{1}{2} \log |\boldsymbol{\Sigma}_k| + \log \pi_k \right] + C$$

3. 求偏导，并同时考虑概率和为1就得到

$$\frac{\partial Q}{\partial \boldsymbol{\mu}_k}=0,\quad \frac{\partial Q}{\partial \boldsymbol{\Sigma}_k}=0, k=1,2...K,$$

$$\sum_{k=1}^{K}\pi_k^{(t)}=1,$$

$$\gamma_{nk}^{(t)}=\frac{\pi_k^{(t)}N(\boldsymbol{x}^{(n)};\boldsymbol{\mu}_k^{(t)},\boldsymbol{\Sigma}_k^{(t)})}{\sum_{i=1}^{K}\pi_i^{(t)}N(\boldsymbol{x}^{(n)};\boldsymbol{\mu}_i^{(t)},\boldsymbol{\Sigma}_i^{(t)})}\quad (\text{Probability of } \boldsymbol{x}^{(n)} \text{ in k-th cluster })$$

$$\text{denote:}\quad N_k^{(t)}=\sum_{n=1}^{N}\gamma_{nk}^{(t)},\quad (\text{all samples in k-th cluster})$$

$$\Rightarrow\begin{cases}\boldsymbol{\mu}_k^{(t+1)}=\frac{1}{N_k^{(t)}}\sum_{n=1}^{N}\gamma_{nk}^{(t)}\boldsymbol{x}^{(n)}, & (\text{Center of cluster k})\\ \boldsymbol{\Sigma}_k^{(t+1)}=\frac{1}{N_k^{(t)}}\sum_{n=1}^{N}\gamma_{nk}^{(t)}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k^{(t+1)})^{\top}, & (\text{Covariance of cluster k})\\ \pi_k^{(t+1)}=\frac{N_k^{(t)}}{N}, & (\text{Weight of cluster k})\end{cases}$$

4. 在实际训练中，**E-STEP**是计算：

$$\gamma_{nk}^{(t)}=\frac{\pi_k^{(t)}N(\boldsymbol{x}^{(n)};\boldsymbol{\mu}_k^{(t)},\boldsymbol{\Sigma}_k^{(t)})}{\sum_{i=1}^{K}\pi_i^{(t)}N(\boldsymbol{x}^{(n)};\boldsymbol{\mu}_i^{(t)},\boldsymbol{\Sigma}_i^{(t)})}\quad (\text{Probability of } \boldsymbol{x}^{(n)} \text{ in k-th cluster })$$

往往会把这个后验概率 γ_{nk} 又称为 **责任值（responsibility）**；

然后**M-STEP**是利用计算好的责任值，进行如下这些模型参数的更新

$$\begin{cases}\boldsymbol{\mu}_k^{(t+1)}=\frac{1}{N_k^{(t)}}\sum_{n=1}^{N}\gamma_{nk}^{(t)}\boldsymbol{x}^{(n)}, & (\text{Center of cluster k})\\ \boldsymbol{\Sigma}_k^{(t+1)}=\frac{1}{N_k^{(t)}}\sum_{n=1}^{N}\gamma_{nk}^{(t)}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k^{(t+1)})^{\top}, & (\text{Covariance of cluster k})\\ \pi_k^{(t+1)}=\frac{N_k^{(t)}}{N}, & (\text{Weight of cluster k})\end{cases}$$

5. 如果设协方差矩阵较简单： $\boldsymbol{\Sigma}_k=\sigma_k^2\boldsymbol{I}$，有

$$\gamma_{nk}=\frac{\pi_k e^{-\frac{1}{2\sigma_k^2}\|\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k\|^2}}{\sum_{k=1}^{K}\pi_k e^{-\frac{1}{2\sigma_k^2}\|\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k\|^2}}$$

然后再简化一点，并假设每簇的权重 π_k 相等：

$$\gamma_{nk}=\frac{e^{-\beta\|\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k\|^2}}{\sum_{k=1}^{K}e^{-\beta\|\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k\|^2}}$$

这就回到了 Soft K-Means。

K-Means

1. 优化目标：

$$\min_{\gamma,\boldsymbol{\mu}} \quad J = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{nk}\|\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k\|^2$$

2. 分步地这样更新 $(\gamma_{nk} \to \boldsymbol{\mu}_k \to \gamma_{nk} \to \boldsymbol{\mu}_k \to \dots)$:

$$\gamma_{nk} = \begin{cases} 1, & \text{if } k = \arg\min_k \|\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k\|^2, \\ 0, & \text{else.} \end{cases}$$

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N}\gamma_{nk}\boldsymbol{x}^{(n)}}{\sum_{n=1}^{N}\gamma_{nk}}$$

Soft K-Means

在K-Means的基础上，仍然:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N}\gamma_{nk}\boldsymbol{x}^{(n)}}{\sum_{n=1}^{N}\gamma_{nk}}$$

但此时样本不是一定地被分给某一最近的cluster，而是根据样本到不同 cluster 中心的距离，并加上系数 β 来调节决定样本被分到每个 cluster 的概率:

$$\gamma_{nk} = \frac{e^{-\beta\|\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k\|^2}}{\sum_{k=1}^{K}e^{-\beta\|\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k\|^2}}$$

K-Means Clustering

> 探索两种不同初始化方法对聚类性能的影响。

K-Means 算法流程

相应的公式在上面已经给出了，下面给出对应的具体代码。

初始化聚类中心：2 种不同的初始化方法

1. random: 随机选10个样本点作为初始聚类中心。

```
if self.init_method == "random":
    random_indices = np.random.choice(self.samples_num, size=self.cluster_num, replace=False)
    self.center = self.samples[random_indices, :]
```

2. dist-based（dist-based+random）：先随机选1个样本点作聚类中心，再找 k 个离这个样本点最远的点，从这 k 个中选 1 个作为下1个聚类中心。

```
if self.init_method == "random":
    random_indices = np.random.choice(self.samples_num, size=self.cluster_num, replace=False)
    self.center = self.samples[random_indices, :]

elif self.init_method == "dist-based":
    random_index = np.random.choice(self.samples_num, size=1, replace=False)
    self.center[0, :] = np.squeeze(self.samples[random_index, :])

    # choose next clustering center based on distance
    for i in range(1, 10):
        last_center = self.center[i-1, :]
        k = 100  # we choose top-k far points
        far_k_points = far(last_center, self.samples, k)
        choose = np.random.choice(far_k_points)
        self.center[i, :] = np.squeeze(self.samples[choose, :])

else:
    raise ValueError("Invalid center-init-method")
```

更新样本点n被分配到第k个簇的情况（gamma）

```
def update_cluster(self):
    """
    update cluster-matrix {gamma_nk}
    :return:
    """
    for i in range(self.samples_num):
        sample_i = self.samples[i]
        distances = np.linalg.norm(self.center - sample_i, axis=1)
        k = np.argmin(distances)
        # One-Hot Code
        self.gamma[i, :] = 0  # Remember to set ZERO !
        self.gamma[i, k] = 1
```

更新聚类中心

```
def update_center(self):
    """
    update clustering center mu_k
    :return:
    """
    # for k in range(self.cluster_num):
    #     gamma_k = self.gamma[:, k]
    #     samples_in_k = np.dot(gamma_k, self.samples)
    #     mu_k = samples_in_k / np.sum(gamma_k)
    #     self.center[k, :] = mu_k
    gamma_sample_sum = np.dot(self.gamma.T, self.samples)
    gamma_sum = np.sum(self.gamma, axis=0)
    # RuntimeWarning: invalid value encountered in divide
    epsilon = 1e-10
    self.center = np.where(gamma_sum[:, np.newaxis] > 0, gamma_sample_sum / (gamma_sum[:, np.newaxis]+ epsilon), self.center)
```

整个训练过程

即，先更新聚类分配、再更新聚类中心，这样循环进行：

```python
def train(self):
    self.init_center()
    with tqdm(range(self.iters), desc="Training Progress") as pbar:
        for i in pbar:
            # early stop if CONVERGE
            prev_center = self.center.copy()
            self.update_cluster()
            self.update_center()
            # print(f'--finish iter-{i} traning')
            # if ord == Inf:
            # ValueError: The truth value of an array with more than one element is ambiguo
            # or, Solution: use ord = 'fro', caculate Matrix's Frobenius Norm
            convergence = np.linalg.norm(prev_center - self.center, ord='fro')
            pbar.set_postfix({"Iter": i, "Convergence": convergence})
            if convergence < 1e-6:
                print("CENTERS ALREADY CONVERGED")
                break
```

比较 2 种不同的初始化方法对聚类性能的影响

- 在train方法中，我设置了 convergence < 1e-5 就停止，从而能大概找到多少轮迭代后，K-Means的中心就变化不大了。
- 使用 random 初始化聚类中心，训练2次，得到：

```
K-Means Training Progress:  21%|██        | 104/500 [01:02<03:59,  1.65it/s, Iter=104, Convergence=0]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model KMeans with Init-Method random: 0.5963
```

```
K-Means Training Progress:  19%|█▉        | 93/500 [00:54<03:57,  1.71it/s, Iter=93, Convergence=0]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model KMeans with Init-Method random: 0.5378
```

大概100次迭代更新后，聚类中心点的变化就不大了，此时得到的ACC平均一下大概有 56% 左右。

- 使用 dist-based （其实是dist-based+random）初始化聚类中心，训练2次，得到：

```
K-Means Training Progress:  32%|███▏      | 158/500 [01:33<03:22,  1.69it/s, Iter=158, Convergence=0]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model KMeans with Init-Method dist-based: 0.5452
```

```
K-Means Training Progress:  32%|███▏      | 161/500 [01:31<03:13,  1.75it/s, Iter=161, Convergence=0]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model KMeans with Init-Method dist-based: 0.5499
```

发现虽然ACC差不多（甚至降低了一点），但所需要的迭代次数增加了，要160次左右。

- 如果把dist-based的初始选择从100个点改为10个点，即变成在10个最远的点里random选，接近于单纯的dist-based（也防止了Outliers），得到：

```
K-Means Training Progress:  10%|▉         | 48/500 [00:29<04:33,  1.65it/s, Iter=48, Convergence=0]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model KMeans with Init-Method dist-based: 0.5423
```

```
K-Means Training Progress:   8%|▉         | 40/500 [00:23<04:35,  1.67it/s, Iter=40, Convergence=0]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model KMeans with Init-Method dist-based: 0.5169

K-Means Training Progress:  17%|██        | 87/500 [00:52<04:07,  1.67it/s, Iter=87, Convergence=0]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model KMeans with Init-Method dist-based: 0.5647
```

此时的平均ACC其实也差不多，但所需要的迭代次数减少了几十次。

- 结论：
- 其实对MNIST数据集，不同的 CENTER 初始化方法影响不是很大，都能在测试集上得到 55% 左右的聚类精度；
- 越偏向于纯dist-based的方法所需要的收敛迭代次数会更少一些，但ACC也会降低一点（ $2\% - 3\%$ ）

GMM Clustering

> 探索不同结构的协方差矩阵（对角且元素值都相等、对角但元素值不要求相等、普通矩阵等）对聚类性能的影响；也观察不同初始化对最后结果的影响。

GMM 算法流程

模型参数大致如下：

```python
class GMM:
    def __init__(self, init_method, covar_type, cluster_num, samples, iters):
        """
        Gaussian Mixture Model
        :param init_method: random or KMeans-PreTrain
        :param covar_type: type of covariance matrix, full or tied or diag or spherical
        :param cluster_num: 10 in this lab
        :param samples: samples (N=60000, D)
        """
        self.init_method = init_method
        self.covariance_type = covar_type
        self.cluster_num = cluster_num
        self.samples = samples
        self.samples_num = samples.shape[0]
        self.feature_dim = samples.shape[1]
        self.iters = iters

        # parameters of GMM model
        self.means = np.zeros((self.cluster_num, self.feature_dim))  # μ_k, center of k-th cluster
        # "tied" has different covariances-shape from other three, so we just set it None here
        self.covariances = None  # Σ_k: covariance matrices
        self.weights = np.ones(self.cluster_num) / self.cluster_num  # π_k: cluster weights
        self.resp = np.zeros((self.samples_num, self.cluster_num))  # γ_nk
```

不同的初始化方法

- 高斯分布均值初始化：

1. 随机选10个样本点作为初始聚类中心，即初始各个簇的均值 $\boldsymbol{\mu}_k$ 。
2. 先用 KMeans 训练几轮，得到 10 个簇的中心，即均值 $\boldsymbol{\mu}_k, k = 0, 1...9$ 。

```
if self.init_method == "random":
    random_indices = np.random.choice(self.samples_num, size=self.cluster_num, replace=False)
    self.means = self.samples[random_indices, :]

elif self.init_method == "KMeans-PreTrain":
    kmeans = KMeans( init_method: "dist-based",  cluster_num: 10, self.samples,  iters: 30)
    kmeans.train()
    self.means = kmeans.get_centers()

else:
    raise ValueError("Invalid means-init-method")
```

- 每个分布的权重初始化如下：放在了__init__方法里

```
self.weights = np.ones(self.cluster_num) / self.cluster_num  # π_k: cluster weights
```

- 协方差矩阵初始化：

见下面分析

不同结构的协方差矩阵

回顾上面训练GMM模型的流程：

E-STEP：计算 t 时刻的 responsibility

$$\gamma_{nk}^{(t)} = \frac{\pi_k^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{i=1}^{K} \pi_i^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_i^{(t)}, \boldsymbol{\Sigma}_i^{(t)})} \quad (\text{Probability of } \boldsymbol{x}^{(n)} \text{ in k-th cluster })$$

M-STEP：更新下一时刻 t+1 的模型参数

$$\begin{cases} \boldsymbol{\mu}_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{n=1}^{N} \gamma_{nk}^{(t)} \boldsymbol{x}^{(n)}, & (\text{Center of cluster k}) \\ \boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{n=1}^{N} \gamma_{nk}^{(t)} (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})^\top, & (\text{Covariance of cluster k}) \\ \pi_k^{(t+1)} = \frac{N_k^{(t)}}{N}, & (\text{Weight of cluster k}) \end{cases}$$

1. 使用不同的协方差矩阵，会影响后验概率即责任值计算所需的复杂度：

$$\gamma_{nk}^{(t)} = \frac{\pi_k^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{i=1}^{K} \pi_i^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_i^{(t)}, \boldsymbol{\Sigma}_i^{(t)})} \quad (\text{Probability of } \boldsymbol{x}^{(n)} \text{ in k-th cluster })$$

2. 可以想到，简单一点的协方差矩阵会使得计算没那么复杂，但这时的高斯分布可能就对实际分布的拟合效果不够精确。
3. 一般情况下的普通协方差矩阵：（记为"full"）

$$\boldsymbol{\Sigma}_k = \begin{bmatrix} \sigma_{k,11} & \sigma_{k,12} & \cdots & \sigma_{k,1D} \\ \sigma_{k,21} & \sigma_{k,22} & \cdots & \sigma_{k,2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{k,D1} & \sigma_{k,D2} & \cdots & \sigma_{k,DD} \end{bmatrix}$$

并代入具体的高斯分布表达式，即得到：

$$\gamma_{nk} = \frac{\pi_k \cdot \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k)\right)}{\sum_{j=1}^{K} \pi_j \cdot \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_j|^{\frac{1}{2}}} \cdot \exp\left(-\frac{1}{2}(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1}(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_j)\right)}$$

4. 如果是对角且元素值都相等的协方差矩阵： $\boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}$，就可以把表达式简化为：（记为"spherical"）

$$\gamma_{nk} = \frac{\pi_k e^{-\frac{1}{2\sigma_k^2}\|\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k\|^2}}{\sum_{k=1}^{K} \pi_k e^{-\frac{1}{2\sigma_k^2}\|\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k\|^2}}$$

5. 如果是对角但元素值不要求相等的协方差矩阵： $\boldsymbol{\Sigma}_k = \mathrm{diag}(\sigma_{k,1}^2, \sigma_{k,2}^2, \ldots, \sigma_{k,D}^2)$，此时比上面复杂一点，但仍然是按维度可分的，能拆成简单的一维计算再求和：（记为"diag"）

$$\gamma_{nk} = \frac{\pi_k \exp\left(-\sum_{d=1}^{D} \frac{\left(x_d^{(n)} - \mu_{k,d}\right)^2}{2\sigma_{k,d}^2}\right)}{\sum_{j=1}^{K} \pi_j \exp\left(-\sum_{d=1}^{D} \frac{\left(x_d^{(n)} - \mu_{j,d}\right)^2}{2\sigma_{j,d}^2}\right)}$$

6. 而事实上，在sklearn对GMM的实现中，还有一种协方差矩阵设置方式，即，各个簇共享同1个协方差矩阵：（记为"tied"）

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_{DD} \end{bmatrix}$$

不同结构的协方差矩阵的初始化

为了防止出现奇异矩阵，无法求逆/做除法，在可能出现奇异矩阵的地方，给对角线元素加上 $epsilon = 1e-n$。

1. "full"：

$$\boldsymbol{\Sigma}_k = \begin{bmatrix} \sigma_{k,11} & \sigma_{k,12} & \cdots & \sigma_{k,1D} \\ \sigma_{k,21} & \sigma_{k,22} & \cdots & \sigma_{k,2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{k,D1} & \sigma_{k,D2} & \cdots & \sigma_{k,DD} \end{bmatrix}$$

由于刚开始没有具体分好的簇，直接计算全体样本的协方差矩阵，分配给每个簇的 $\boldsymbol{\Sigma}_k$：

（这里计算的时候注意一下，`np.cov` 计算时，默认输入矩阵的每一行是一个特征，而每一列是一个样本，这和我们的 samples矩阵不一样，要转秩一下）

```
elif self.covariance_type == "full":
    cov_k = np.cov(self.samples.T) + np.eye(self.feature_dim) * epsilon
    self.covariances = np.array([cov_k for _ in range(self.cluster_num)])
```

2. “spherical”： $\boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}$

由于刚开始没有具体分好的簇，直接把全体样本的方差（把samples矩阵直接展平）分配给每个簇的 σ_k^2 ：

```
if self.covariance_type == "spherical":
    unit_matrix = np.eye(self.feature_dim)  # generate unit matrix I
    # haven't any cluster yet, so we set all sigma_k the same as var(all samples)
    squared_sigma_k = np.var(self.samples)
    cov_k = squared_sigma_k * unit_matrix
    # generate cov_matrix of cluster_num
    self.covariances = np.array([cov_k for _ in range(self.cluster_num)])
```

3. “diag”： $\boldsymbol{\Sigma}_k = \mathrm{diag}(\sigma_{k,1}^2, \sigma_{k,2}^2, \ldots, \sigma_{k,D}^2)$

由于刚开始没有具体分好的簇，直接把全体样本，按每个维度计算方差分配给每个 $\sigma_{k,d}^2$ ，然后生成对角矩阵：

```
elif self.covariance_type == "diag":
    squared_sigma_k_for_each_dim = np.var(self.samples, axis=0) + epsilon
    cov_k = np.diag(squared_sigma_k_for_each_dim)
    self.covariances = np.array([cov_k for _ in range(self.cluster_num)])
```

4. “tied”：

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1D} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{D1} & \sigma_{D2} & \cdots & \sigma_{DD} \end{bmatrix}$$

计算全体样本的协方差矩阵，分配给 $\boldsymbol{\Sigma}$ ，然后所有的簇共享使用这个协方差矩阵：

```
elif self.covariance_type == "tied":
    self.covariances = np.cov(self.samples.T) + np.eye(self.feature_dim) * epsilon
```

E-STEP：计算 t 时刻后验概率

计算 t 时刻的 responsibility

$$\gamma_{nk}^{(t)} = \frac{\pi_k^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{i=1}^{K} \pi_i^{(t)} N(\boldsymbol{x}^{(n)}; \boldsymbol{\mu}_i^{(t)}, \boldsymbol{\Sigma}_i^{(t)})} \quad (\text{Probability of } \boldsymbol{x}^{(n)} \text{ in k-th cluster})$$

调用`scipy.stats`的`multivariate_normal`来计算

```
def e_step(self):
    """
    compute the responsibility γ_nk: the probability of sample_n belongs to k-th cluster
    """
    # clear previous result
    self.resp = np.zeros((self.samples_num, self.cluster_num))

    for k in range(self.cluster_num):
        if self.covariance_type in ["spherical", "diag", "full"]:
            cov_matrix_k = self.covariances[k]
        elif self.covariance_type == "tied":
            cov_matrix_k = self.covariances
        else:
            raise ValueError("Invalid covariance_type")

        gaussian = multivariate_normal(mean=self.means[k], cov=cov_matrix_k)
        self.resp[:, k] = self.weights[k] * gaussian.pdf(self.samples)

    self.resp = self.resp / np.sum(self.resp, axis=1, keepdims=True)
```

NOTE：这里计算的时候要注意尽量利用矩阵的特性，减少循环，而且这里的多维高斯分布可以同时对矩阵中的多个samples进行计算。

一开始这里写了2重循环，导致训练一个迭代都要几分钟，改成1层循环，速度就大大提升了。

M-STEP：更新 t+1 时刻模型参数

更新下一时刻 t+1 的模型参数

$$\begin{cases} \boldsymbol{\mu}_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{n=1}^{N} \gamma_{nk}^{(t)} \boldsymbol{x}^{(n)}, & (\text{Center of cluster k}) \\ \boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{n=1}^{N} \gamma_{nk}^{(t)} (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})^\top, & (\text{Covariance of cluster k}) \\ \pi_k^{(t+1)} = \frac{N_k^{(t)}}{N}, & (\text{Weight of cluster k}) \end{cases}$$

其中

$$N_k^{(t)} = \sum_{n=1}^{N} \gamma_{nk}^{(t)}, \quad (\text{all samples in k-th cluster})$$

均值和权重的更新都是一样的：

```
        # update π_k
        self.weights = N_k / self.samples_num

        # update μ_k
        for k in range(self.cluster_num):
            self.means[k, :] = np.dot(self.resp[:, k], self.samples) / N_k[k]
```

主要是协方差矩阵有不同更新计算方式。

1. 对普通矩阵，即“full”，每个簇的协方差矩阵这样更新：

$$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{n=1}^{N} \gamma_{nk}^{(t)} (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})^\top$$

代码实现：

```
if self.covariance_type == "full":
    for k in range(self.cluster_num):
        diff = self.samples - self.means[k]
        # resp[:, k]: (N,)一维向量
        # 这里要进行逐元素加权操作，用[:, np.newaxis]变成 N*1 的列向量
        # 这样，如果每行是1，那么对应的 diff 行（第n个样本与第k个中心的差）才会被加权1，否则加权0
        gamma_weighted_diff = self.resp[:, k][:, np.newaxis] * diff
        # 这里注意一下，样本是按行存储的，和手动数学计算（按列存储）不同，所以转秩不一样
        self.covariances[k, :] = np.dot(gamma_weighted_diff.T, diff) / N_k[k]
```

2. 对“diag”类型的每个簇的协方差矩阵 $\boldsymbol{\Sigma}_k = \mathrm{diag}(\sigma_{k,1}^2, \sigma_{k,2}^2, \ldots, \sigma_{k,D}^2)$，可以分开按维度更新：

$${\sigma_{k,d}^2}^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{n=1}^{N} \gamma_{nk}^{(t)} \left(x_d^{(n)} - {\mu_{k,d}}^{(t+1)} \right)^2, d = 1, 2 \ldots D$$

在实际代码实现中，由于存储的直接是 $\boldsymbol{\Sigma}_k = \mathrm{diag}(\sigma_{k,1}^2, \sigma_{k,2}^2, \ldots, \sigma_{k,D}^2)$，直接矩阵化地计算：

$$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k^{(t)}} \sum_{n=1}^{N} \gamma_{nk}^{(t)} \left[(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)}) \odot (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)}) \right]$$

和“full”的计算公式比起来就是，少了不同特征维度的交叉项 $(x_d^{(n)} - {\mu_{k,d}}^{(t+1)})(x_q^{(n)} - {\mu_{k,q}}^{(t+1)}), d \neq q$，只计算平方项。

代码实现：

按上面的公式实现时，要注意其实等号右边算出来是一个D维向量，要调整回对角矩阵的形式：

```
elif self.covariance_type == "diag":
    for k in range(self.cluster_num):
        diff = self.samples - self.means[k]
        gamma_weighted_diff = self.resp[:, k][:, np.newaxis] * (diff ** 2)
        # 记得把向量还原成对角矩阵（np.diag()）
        self.covariances[k, :] = np.diag(np.sum(gamma_weighted_diff, axis=0) / N_k[k])
```

3. 对"spherical"类型，每个簇的协方差矩阵 $\boldsymbol{\Sigma}_k = \sigma_k^2 \boldsymbol{I}$，更新 σ_k^2 即可，即，把所有维度的 $(x_d^{(n)} - {\mu_{k,d}}^{(t+1)})^2$ 加起来，再取均值，来更新所有维度上一样的 σ_k^2，即向量二范数（再除以维度数）计算：

$$\sigma_k^{2(t+1)} = \frac{1}{N_k^{(t)} D} \sum_{n=1}^{N} \gamma_{nk}^{(t)} \|\boldsymbol{x}^{(n)} - {\boldsymbol{\mu}_k}^{(t+1)}\|^2$$

和"full","diag"的实现不同，这里先去计算二范数的平方，得到一标量：

```
elif self.covariance_type == "spherical":
    for k in range(self.cluster_num):
        diff = self.samples - self.means[k]
        squared_l2_norm = np.sum(diff ** 2, axis=1)
        squared_sigma_k = np.sum(self.resp[:, k] * squared_l2_norm) / (N_k[k] * self.feature_dim)
        self.covariances[k, :] = np.eye(self.feature_dim) * squared_sigma_k
```

4. 对"tied"类型，就是在"full"的基础上，直接对所有样本的协方差取平均，去更新共享的 $\boldsymbol{\Sigma}$：

$$\boldsymbol{\Sigma}^{(t+1)} = \frac{1}{N} \sum_{k=1}^{K} \sum_{n=1}^{N} \gamma_{nk}^{(t)} (\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})^\top$$

代码实现：

```
elif self.covariance_type == "tied":
    cov_sum = np.zeros((self.feature_dim, self.feature_dim))
    for k in range(self.cluster_num):
        diff = self.samples - self.means[k]
        gamma_weighted_diff = self.resp[:, k][:, np.newaxis] * diff
        cov_sum += np.dot(gamma_weighted_diff.T, diff)

    self.covariances = cov_sum / self.samples_num
```

整个训练过程

即EM算法流程，先E步，后M步，循环进行：

```
def train(self):
    self.init_parameters()
    with tqdm(range(self.iters), desc="EM Training Progress") as pbar:
        for i in pbar:
            prev_means = self.means.copy()
            # print("e-step")
            self.e_step()
            # print("m-step")
            self.m_step()
            # print(f'--finish iter-{i} EM traning')
            pbar.set_postfix({"Iter": i, "Convergence": np.linalg.norm(prev_means - self.means, ord='fro')})
            if np.linalg.norm(prev_means - self.means, ord='fro') < 1e-6:
                print("CENTERS ALREADY CONVERGED")
                break
```

不同的初始化对聚类性能的影响

考虑2种：1. 高斯分布均值点随机从样本点中选；2.先用KMeans训练50个以内iter，训练得到的聚类中心作为高斯分布的均值点

- 比如，都取球形协方差矩阵 spherical，
- random 初始化：

```
EM Training Progress:  90%|████████▉ | 179/200 [01:48<00:12,  1.64it/s, Iter=179, Convergence=9.25e-6]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model GMM with Init-Method random, Cov spherical: 0.537
```

- KMeans-Pretrain 初始化：

```
K-Means Training Progress: 100%|██████████| 30/30 [00:18<00:00,  1.67it/s, Iter=29, Convergence=15.7]
EM Training Progress:  49%|████▉     | 146/300 [01:33<01:39,  1.55it/s, Iter=146, Convergence=9.94e-6]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model GMM with Init-Method KMeans-PreTrain, Cov spherical: 0.4633
```

- 或者，都取 full 协方差矩阵：
- random 初始化：

```
EM Training Progress:  79%|███████▉  | 238/300 [03:25<00:53,  1.16it/s, Iter=238, Convergence=9.45e-6]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model GMM with Init-Method random, Cov full: 0.6111
```

- KMeans-Pretrain 初始化：

```
K-Means Training Progress: 100%|██████████| 30/30 [00:17<00:00,  1.69it/s, Iter=29, Convergence=19.4]
EM Training Progress:  97%|█████████▋| 291/300 [04:16<00:07,  1.14it/s, Iter=291, Convergence=9.56e-6]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model GMM with Init-Method KMeans-PreTrain, Cov full: 0.6624
```

- 实验结果显示，一个不太精确的 spherical 协方差矩阵的模型，random初始化得到了最终更好的ACC；精确一点的 full 协方差矩阵的模型，KMeans-Pretrain 初始化得到了最终更好的ACC。
- 当然，K-Means预训练得到的中心也有一定的偶然性，所以**其实不同的初始化方法对ACC的影响不会太大，ACC更多地还是受到协方差矩阵的影响**。

不同的协方差矩阵对聚类性能的影响

- 都按KMeans-Pretrain 初始化高斯分布均值
- 普通矩阵 full：

```
K-Means Training Progress: 100%|██████████| 30/30 [00:17<00:00,  1.69it/s, Iter=29, Convergence=19.4]
EM Training Progress:  97%|█████████▋| 291/300 [04:16<00:07,  1.14it/s, Iter=291, Convergence=9.56e-6]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model GMM with Init-Method KMeans-PreTrain, Cov full: 0.6624
```

- spherical：

```
K-Means Training Progress: 100%|██████████| 30/30 [00:18<00:00,  1.67it/s, Iter=29, Convergence=15.7]
EM Training Progress:  49%|████▉     | 146/300 [01:33<01:39,  1.55it/s, Iter=146, Convergence=9.94e-6]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model GMM with Init-Method KMeans-PreTrain, Cov spherical: 0.4633
```

- diag：

```
K-Means Training Progress: 100%|██████████| 30/30 [00:17<00:00,  1.73it/s, Iter=29, Convergence=16.8]
EM Training Progress:  86%|████████▌ | 258/300 [02:58<00:29,  1.44it/s, Iter=258, Convergence=9.77e-6]
CENTERS ALREADY CONVERGED
Clustering Accuracy for Model GMM with Init-Method KMeans-PreTrain, Cov diag: 0.3185
```

- tied：

```
K-Means Training Progress: 100%|██████████| 30/30 [00:18<00:00,  1.64it/s, Iter=29, Convergence=60]
EM Training Progress: 100%|██████████| 300/300 [04:17<00:00,  1.17it/s, Iter=299, Convergence=0.000707]
Clustering Accuracy for Model GMM with Init-Method KMeans-PreTrain, Cov tied: 0.4711
```

- 普通矩阵full拥有最好的聚类性能，最终的ACC最高可以有 66% 左右。从公式理论的角度而言，普通矩阵 full 确实能经过多个维度的精细调整，得到用高斯分布对数据分布进行更好的拟合效果。对角元素可以不同的对角矩阵有最低的ACC，只有 $30\%+$。
- 可见，full以外的其他矩阵为了节省计算资源/存储资源，使用不那么精确的协方差矩阵来拟合高斯分布，得到的最终ACC就会低一点。这确实是符合公式理论的。

Training Methods

> 采用的训练方法，包括参数初始化方法、优化方法、其他的训练技巧等

- K-Means和GMM模型相关的训练流程+参数初始化方法在上面已经写了，下面写一些其他的数据处理、训练技巧方法：

数据预处理

- 数据预处理：同样都是MNIST数据集，沿用第一次实验的数据预处理方式就好。

Early Stop

- 加入 early stop 机制，如果中心点已经收敛了，就不再训练了：

```
if np.linalg.norm(prev_means - self.means, ord='fro') < 1e-5:
    print("CENTERS ALREADY CONVERGED")
    break
```

PCA降维

- 一开始不管怎么训练GMM模型，都会出现一大堆0，后验概率在每个簇上全是0。

- 搜索发现，高维空间的数据点是较为稀疏的，原本的MNIST数据在各个维度上的信息贡献量偏差较大，有的维度全是0，这对分类显然没有什么贡献，要把真正对分类有较大贡献的主成分提取出来，然后降维在这些主成分上计算。
- 用PCA降维：

`from sklearn.decomposition import PCA`

实际训练中降到了50维：

```
arguments = args()
train_labels, train_samples_before = extract_data('mnist_train.csv')
# train_samples = standardization(train_samples)
test_labels, test_samples_before = extract_data('mnist_test.csv')
# test_samples = standardization(test_samples)
pca = PCA(50)
pca.fit(train_samples_before)
train_samples = pca.transform(train_samples_before)
test_samples = pca.transform(test_samples_before)
```

- 这样就能正常进行GMM模型的训练了

聚类精度ACC

- 使用聚类精度ACC作为聚类性能的评价指标。
- clustering 只是把样本分成 10 簇，但并不意味着这分类后的 第0、1、2... 9簇 就直接对应 原本样本标签中的 0-9，比如，原本的一类item标签记为1，但实际聚类后大部分该类item被聚在了第9簇中，所以要使用二部图匹配的方法来衡量。
- 使用匈牙利算法解决二部图匹配问题，在`calculate_acc`函数中。

tqdm

为了方便查看训练进度，使用tqdm：把原本的循环for语句改成类似

`with tqdm(range(self.iters), desc="EM Training Progress") as pbar:  \\  for i in pbar:`

以及加入

`pbar.set_postfix({"Iter": i, "Convergence": np.linalg.norm(prev_means - self.means, ord='fro')})`

就能如下方便地看到训练进度了：

```
K-Means Training Progress: 100%|██████████| 30/30 [00:17<00:00,  1.69it/s, Iter=29, Convergence=19.4]
EM Training Progress:  69%|███████   | 208/300 [03:00<01:18,  1.18it/s, Iter=207, Convergence=0.00121]
```

Observation & Comparison & Analysis between GMM and K-Means

> 观察实验结果，结合理论知识，比较 K-means 聚类方法和 EM 训练的 GMM 聚类方法之间的优劣，以及实验结果的相关讨论。

- 比较聚类中心收敛所需的时间：

根据上面的到的实验结果，K-Means总是能在1分钟以内得到收敛的聚类中心；而GMM往往要好几分钟，使用full普通矩阵的GMM要4分多钟。

- 比较最后得到的聚类精度ACC：

根据上面的到的实验结果，K-Means得到的ACC能平均有 53% 左右，而GMM会非常取决于所使用的协方差矩阵，最好的full矩阵可以有最高能达到 66% 的ACC。

- 结合理论知识分析，根据上面的公式推导流程就可以看出来，EM算法训练的GMM其实是比Soft-KMeans都更精确的，能得到更好的拟合效果，所以使用full矩阵的GMM能得到最好的聚类精度。
- 同时，如果只是简单地限制高斯分布的协方差矩阵是 diag 或者 spherical类型的，就未必能比较精确地拟合，得到的ACC甚至可能低于K-Means。
- K-Means的优点主要在于聚类中心收敛所需的时间短，如果对最终ACC的要求只是 50% 以上，可以用K-Means来更快地得到收敛的聚类中心。
- 而使用 full 协方差矩阵的GMM，会花 K-Means 5 倍左右的训练时间，但最终可以达到较高的聚类精度，适合对ACC要求更高的任务。
- 而使用其他矩阵的GMM，从目前的模型而言，训练时长劣于K-Means，ACC也并不好，可能是针对这个MNIST数据集，确实是需要更完全的普通协方差矩阵来拟合数据分布的原因。
- 当然，GMM耗时长，是因为GMM的EM算法要计算协方差矩阵、计算多维高斯分布概率等，要进行的计算比K-Means更多。