



实验课程: 机器学习与数据挖掘

实验名称: 聚类K-Means 和 GMM

专业名称: 计算机科学与技术

学生姓名: 吴臻

学生学号: 21307371

- [实验目的](#)
- [实验要求](#)
- ▼ [实验内容](#)
  - [K-Means](#)
  - [GMM \(EM算法训练\)](#)
- [实验总结](#)
- [实验资料](#)

# 实验目的

以 MNIST 数据集为例，探索 K-Means 和 GMM 这两种聚类算法的性能。

## 实验要求

1. 手动实现 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(ClusteringAccuracy, 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。

## 实验内容

### 数据预处理

1. 读入数据和特征归一化

```
def load_data(file_path):  
    data = pd.read_csv(file_path)  
    labels = data.iloc[:, 0].values  
    features = data.iloc[:, 1:].values / 255.0 # 对特征值进行归一化  
    return labels, features
```

2. PCA降维(用于GMM)

```
pca = PCA(n_components=100) # 选择主成分  
X_train = pca.fit_transform(X_train)  
X_test = pca.transform(X_test)
```

分析：为了避免由于数据维度过高导致高斯函数计算值过小、协方差矩阵不可逆，我们使用 PCA 对数据进行降维

## K-Means

- 算法流程

1. 选择初始化的  $k$  个样本作为初始聚类中心
2. 针对数据集中每个样本，计算它到  $k$  个聚类中心的距离并将其分到距离最小的聚类中心所对应的类中
3. 针对新划分后的每个类别，重新计算它的聚类中心（即属于该类的所有样本的质心）；
4. 重复上面 2 3 两步操作，直到达到某个中止条件（迭代次数、最小误差变化等）。

- 代码实现

1. **初始化方法**（随机初始化和k-means++初始化）

```
# 随机初始化
```

```
centers = X[np.random.choice(range(len(X)), k, replace=False)]
```

## k-means++初始化:

*Kmeans* ++在初始化簇中心时的方法总结成一句话就是: \*\*逐个选取 $k$ 个簇中心, 且离其它簇中心越远的样本点越有可能被选为下一个簇中心。 \*\*其具体做法如下 (其中引用英文部分论文原文) :

①从数据集 $\mathcal{X}$ 中随机 (均匀分布) 选取一个样本点作为第一个初始聚类中心 $c_1$ ;

1a. Take one center  $c_1$ , chosen uniformly at random from  $\mathcal{X}$ .

②接着计算每个样本与当前已有聚类中心之间的最短距离, 用 $D(x)$ 表示; 然后计算每个样本点被选为下一个聚类中心的概率 $P(x)$ , 最后选择最大概率值 (或者概率分布) 所对应的样本点作为下一个簇中心;

1b. Take a new center  $c_i$ , choosing  $x \in \mathcal{X}$  with probability  $P(x)$ .

$$P(x) = \frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2} \quad (1)$$

③重复第②步, 直到选择出 $k$ 个聚类中心;

1c. Repeat Step 1b. until we have taken  $k$  centers altogether.

从公式(1)也可以看成, 距离现有簇中心越远的样本点, 越可能被选为下一个簇中心。

# 初始化方法: 使用k-means++算法进行初始化 (不平方)

```
def kmeans_pp_init(X, k):
    centers = [X[np.random.randint(0, len(X))]]

    for _ in range(1, k):
        distances = cdist(X, centers, 'euclidean')
        min_distances = np.min(distances, axis=1)
        weights = min_distances / np.sum(min_distances)
        next_center = X[np.random.choice(range(len(X)), p=weights)]
        centers.append(next_center)

    return np.array(centers)
```

## 2. kmeans算法

```
def kmeans(X, Y, k, max_iterations=100):
    # 随机选择k个中心点
    # centers = X[np.random.choice(range(len(X)), k, replace=False)]

    # 初始化方法2
    centers = kmeans_pp_init(X, k)

    for t in range(max_iterations):
        # 计算样本到每个中心点的距离
        distances = cdist(X, centers, 'euclidean')

        # 找到每个样本对应的最近的中心点
        labels = np.argmin(distances, axis=1)

        # 更新中心点的位置
        new_centers = np.array([X[labels == i].mean(axis=0) for i in range(k)])

        # 判断中心点是否变化不大, 如果不变化则停止迭代
        if np.all(centers == new_centers):
            break

        centers = new_centers

        if t % 10 == 0:
            # 给每个中心点一个0-9的标签
            matrix = np.zeros((k, 10))
            distances = cdist(X, centers, 'euclidean')
            y_pred = np.argmin(distances, axis=1)
            for i in range(len(X)):
                matrix[y_pred[i]][Y[i]] += 1
            dict = np.argmax(matrix, axis=1)

            distances = cdist(X_test, centers, 'euclidean')
            y_pred = np.argmin(distances, axis=1)
            accuracy = clustering_accuracy(y_test, y_pred, dict)
            accuracy_list.append(accuracy * 100)
```

```

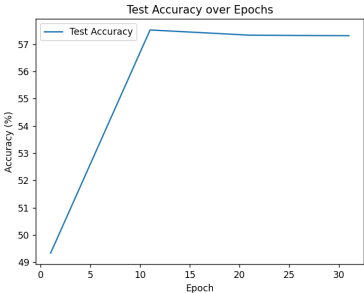
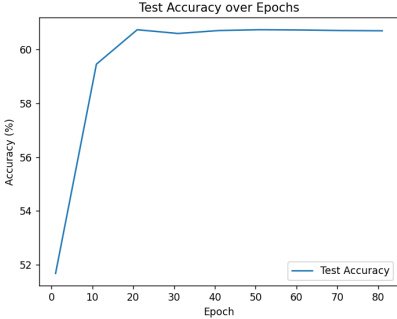
print("Epoch:", t)
print("Clustering Accuracy:", accuracy * 100, "%")

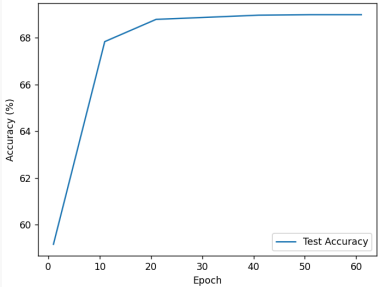
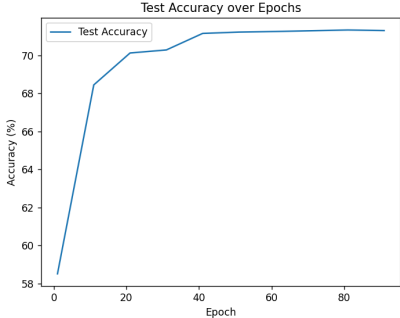
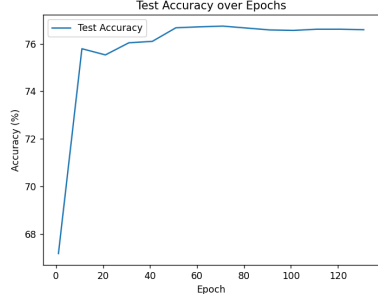
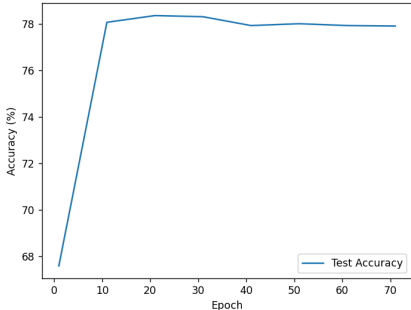
# 给中心点分配标签
matrix = np.zeros((k, 10))
distances = cdist(X, centers, 'euclidean')
y_pred = np.argmin(distances, axis=1)
for i in range(len(X)):
    matrix[y_pred[i]][V[i]] += 1
dict = np.argmax(matrix, axis=1)

# t是迭代次数
return centers, t, dict

```

## 结果展示

簇类 数	随机初始化	k-means++初始化
k=10	<div> <p>Epoch: 30 Clustering Accuracy: 57.31 % Max Clustering Accuracy: 57.31 % It takes 32.26149368286133 s</p>  </div>	<div> <p>Epoch: 80 Clustering Accuracy: 60.69 % Max Clustering Accuracy: 60.69 % It takes 69.2758994102478 s</p>  </div>

簇类数	随机初始化	k-means++初始化
k=20	<div>Epoch: 60 Clustering Accuracy: 68.979999999999 % Max Clustering Accuracy: 68.979999999999 % It takes 81.14911413192749 s</div> <div><p>Test Accuracy over Epochs</p></div>	<div>Epoch: 90 Clustering Accuracy: 71.31 % Max Clustering Accuracy: 71.34 % It takes 126.6805648803711 s</div> <div><p>Test Accuracy over Epochs</p></div>
k=30	<div>Epoch: 130 Clustering Accuracy: 76.59 % Max Clustering Accuracy: 76.74 % It takes 228.37493896484375 s</div> <div><p>Test Accuracy over Epochs</p></div>	<div>Epoch: 70 Clustering Accuracy: 77.91 % Max Clustering Accuracy: 78.36 % It takes 145.67713451385498 s</div> <div><p>Test Accuracy over Epochs</p></div>

分析: K-means算法在该数据集上表现还算不错, 准确率基本在60%以上

- 1.算法采用了两种初始化方法, K-means++的初始化相比随机初始化, 表现得稍微好一些, 结果比较稳定, K-means++ 就是选择离已选中心点最远的点, 这也比较符合常理, 聚类中心当然是互相离得越远越好。
- 2.实验还探究了不同簇类数对准确率的影响, 设置多个中心点, 给每个中心点一个标签(用来检测准确率), 实验结果表明, 提高簇类数确实可以提高准确率, 这也符合常理, 提高簇类数让分类结果更详细, 不过簇类数也不能过高, 否则会出现过拟合, 并且训练时间也会大大增加

# GMM (EM算法训练)

- 理论知识

- i. GMM定义 (高斯混合模型) :

$$p(X) = \sum_{l=1}^k \alpha_l \mathcal{N}(X; \mu_l, \Sigma_l) \quad \sum_{l=1}^k \alpha_l = 1$$

$$\Theta = \{\alpha_1, \dots, \alpha_k, \mu_1, \dots, \mu_k, \Sigma_1, \dots, \Sigma_k\}$$

GMM参数向量 $\Theta$ 的极大似然估计 (直接求解公式GMM参数向量 $\Theta$ 的极大似然估计会十分复杂。因此后面引入隐变量 $Z$ )

$$\Theta_{MLE} = \arg \max_{\Theta} \mathcal{L}(\Theta|X) = \arg \max_{\Theta} \left( \sum_{i=1}^n \log \sum_{l=1}^k \alpha_l \mathcal{N}(X; \mu_l, \Sigma_l) \right)$$

- ii. EM(Expectation-Maximization)算法

定义EM算法的参数更新过程为(引入隐变量 $Z$ ):

$$\Theta^{(g+1)} = \arg \max_{\theta} \int_Z \log p(X, Z|\theta) \cdot p(Z|X, \Theta^{(g)}) dZ$$

## 优化目标的更改

**初始目标:** 极大化 $\mathcal{L}(\theta|X)$  (似然函数)

$$\begin{aligned} \mathcal{L}(\theta|X) &= \int_Z \ln\left(\frac{p(X, Z|\theta)}{Q(Z)}\right) Q(Z) + \int_Z \ln\left(\frac{Q(Z)}{p(Z|X, \theta)}\right) Q(Z) \\ &= F(\theta, Q) + \text{KL}(Q(Z)||p(Z|X, \theta)) \end{aligned}$$

**步骤1:** 固定 $\Theta = \Theta^{(g)}$ , 最大化 $Q(Z)$ (和后面的 $Q(\theta, \theta^{(t)})$ 不是同一个) :

$\mathcal{L}(\Theta|X)$ 的大小不受 $Q(Z)$ 的影响, 而KL散度大于等于0, 所以 $\mathcal{L}(\Theta|X)$ 是 $F(\Theta, Q)$ 的上确界

如果要使 $\mathcal{L}(\Theta|X) = F(\Theta, Q)$ , 需要 $\text{KL}(\cdot) = 0$ , 因此令 $Q(Z) = p(Z|X, \Theta^{(g)})$

此时原本的似然函数 $\mathcal{L}(\Theta|X)$ 变成:

$$\mathcal{L}(\theta|X) = \int_Z \ln\left(\frac{p(X, Z|\theta)}{p(Z|X, \Theta^{(g)})}\right) p(Z|X, \Theta^{(g)})$$



**步骤2:** 固定 $Q(Z)$ , 更新参数 $\Theta$ , 将上面的 $\ln$ 拆开, 下面的是常数, 可以删去

$$\begin{aligned}\Theta^{(g+1)} &= \arg \max_{\Theta} \int_Z \log p(X, Z|\Theta) \cdot p(Z|X, \Theta^{(g)}) dZ \\ &= \arg \max_{\Theta} Q(\Theta, \Theta^{(g)})\end{aligned}$$

**最后目标:** 极大化 $Q(\Theta, \Theta^{(g)})$ , 并更新参数

### iii. GMM中应用EM算法

对于数据集 $X = \{x_1, x_2, \dots, x_n\}$ 引入隐变量 $Z = \{z_1, z_2, \dots, z_n\}$ , 每个 $z_i$ 变量表示数据 $x_i$ 属于第几个高斯分布

$$\begin{aligned}\Theta^{(g+1)} &= \arg \max_{\Theta} \int_Z \log p(X, Z|\Theta) \cdot p(Z|X, \Theta^{(g)}) dZ \\ &= \arg \max_{\Theta} Q(\Theta, \Theta^{(g)})\end{aligned}$$

◦ **E过程**, 即计算 $Q(\Theta, \Theta^{(g)})$

计算 $p(Z|X, \Theta)$ :

$$\begin{aligned}p(Z|X, \Theta) &= \prod_{i=1}^n p(z_i|x_i, \Theta) \\ &= \prod_{i=1}^n \frac{p(x_i, z_i|\Theta)}{p(x_i|\Theta)} \\ &= \prod_{i=1}^n \frac{\alpha_{z_i} \mathcal{N}(\mu_{z_i}, \Sigma_{z_i})}{\sum_{l=1}^k \alpha_l \mathcal{N}(\mu_l, \Sigma_l)}\end{aligned}$$

用 $l$ 替换 $z_i$ , 最终可得:

$$\begin{aligned}
Q(\Theta, \Theta^{(g)}) &= \sum_{l=1}^k \sum_{i=1}^n \ln p(l, x_i | \Theta) p(l | x_i, \Theta^{(g)}) \\
&= \sum_{l=1}^k \sum_{i=1}^n \ln [\alpha_l \mathcal{N}(x_i | \mu_l, \Sigma_l)] p(l | x_i, \Theta^{(g)}) \\
&= \sum_{l=1}^k \sum_{i=1}^n \ln(\alpha_l) p(l | x_i, \Theta^{(g)}) \\
&\quad + \sum_{l=1}^k \sum_{i=1}^n \ln[\mathcal{N}(x_i | \mu_l, \Sigma_l)] p(l | x_i, \Theta^{(g)})
\end{aligned}$$

◦ M过程，即最大化 $Q(\Theta, \Theta^{(g)})$

最大化 $\alpha$ ：

$$\alpha_l = \frac{1}{N} \sum_{i=1}^n p(l | x_i, \Theta^{(g)})$$

最大化 $\mu$ 和 $\Sigma$ ：

$$\mu_l = \frac{\sum_{i=1}^n x_i p(l | x_i, \Theta^{(g)})}{\sum_{i=1}^n p(l | x_i, \Theta^{(g)})}$$

$$\Sigma_l = \frac{\sum_{i=1}^n M_l}{\sum_{i=1}^n p(l | x_i, \Theta^{(g)})} = \frac{\sum_{i=1}^n (x_i - \mu_l)(x_i - \mu_l)^\top p(l | x_i, \Theta^{(g)})}{\sum_{i=1}^n p(l | x_i, \Theta^{(g)})}$$

## • 算法流程

设  $Q(\theta, \theta^{(t+1)}) = \sum_z P(Z|X, \theta(t)) \ln P(X, Z|\theta)$ ，我们就可以得到 EM 算法的整个流程：

- (1) 初始化参数  $\theta(0)$ ,  $t = 0$ ;
- (2) E (期望) 步：计算  $P(Z|X, \theta(t))$ ;
- (3) M (最大化) 步：计算期望  $Q(\theta, \theta(t+1))$ ，然后找到新的参数估计： $\theta^{(t+1)} = \arg \max_{\theta} Q(\theta, \theta^{(t)})$ ;
- (4)  $t = t + 1$ ，然后重复 (2) 和 (3)，直到对数似然  $L(\theta)$  收敛；

## • 训练技巧

1. 训练GMM模型时，总会遇到概率值太小导致出现NaN的现象，所以采用PCA降维
2. 有时会出现矩阵不可逆的现象，查阅资料，发现可以加一个微小对角矩阵，使得协方差矩阵可逆

## • 代码实现

E步

# E步, 计算给定数据点属于每个簇的概率

```
def expectation_step(X, weights, means, covariances):  
    n_samples, n_features = X.shape  
    n_clusters = len(weights)  
  
    # 初始化概率数组  
    probabilities = np.zeros((n_samples, n_clusters))  
  
    # 计算每个数据点属于每个簇的概率  
    for k in range(n_clusters):  
        probabilities[:, k] = weights[k] * multivariate_normal.pdf(X, mean=means[k], cov=covariances[k])  
  
    # 检查概率数组是否包含无效值, 并将其替换为零  
    probabilities[np.isnan(probabilities) | np.isinf(probabilities)] = 0  
  
    # 归一化概率数组  
    probabilities /= (np.sum(probabilities, axis=1, keepdims=True))  
  
    # 检查概率数组是否包含无效值, 并将其替换为零  
    probabilities[np.isnan(probabilities) | np.isinf(probabilities)] = 0.1  
  
    return probabilities
```

M步

# M步, 更新模型参数

```
def maximization_step(X, probabilities):  
    n_samples, n_features = X.shape  
    n_clusters = probabilities.shape[1]  
  
    # 更新权重  
    weights = np.sum(probabilities, axis=0) / n_samples  
  
    # 初始化均值  
    means = np.zeros((n_clusters, n_features))  
  
    for k in range(n_clusters):
```

```

# 计算每个簇的均值
means[k] = np.average(X, axis=0, weights=probabilities[:, k])

# 初始化协方差矩阵
covariances = np.zeros((n_clusters, n_features, n_features))
for k in range(n_clusters):
    diff = X - means[k]
    weighted_diff = np.dot(probabilities[:, k] * diff.T, diff)
    # 计算每个簇的协方差矩阵
    covariances[k] = weighted_diff / np.sum(probabilities[:, k])
    # 加上一个微小对角矩阵, 使得协方差矩阵可逆
    small_diag = 1e-6 * np.eye(n_features)
    covariances[k] = covariances[k] + small_diag
return weights, means, covariances

```

## 结果展示

- (1) 'full': 每个高斯子模型各自拥有一个普通的协方差矩阵;
- (2) 'tied': 所有高斯子模型共用一个普通的协方差矩阵;
- (3) 'diag': 每个高斯子模型分别使用一个对角形式的协方差矩阵, 矩阵内的对角值不一定相等;
- (4) 'spherical': 每个高斯子模型分别使用一个对角形式的协方差矩阵, 矩阵内的对角值相等;

参数初始化 ( $\mu$ ; $\Sigma$ )	full	tied	diag	spherical
随机初始化	<p>Max Clustering Accuracy: 63.36% It costs 161.55s</p>	<p>Max Clustering Accuracy: 63.61% It costs 75.54s</p>	<p>Max Clustering Accuracy: 59.90% It costs 101.65s</p>	<p>Clustering Accuracy: 57.96% Max Clustering Accuracy: 60.09% It costs 94.90s</p>
kmeans++初始化	<p>Max Clustering Accuracy: 63.64% It costs 162.12s</p>	<p>Max Clustering Accuracy: 65.77% It costs 164.65s</p>	<p>Max Clustering Accuracy: 63.26% It costs 157.56s</p>	<p>Max Clustering Accuracy: 56.57% It costs 77.16s</p>

## 分析:

### 探索使用不同结构的协方差矩阵对聚类性能的影响

a. 从上面的结果可以看到在该数据集上表现最好的协方差矩阵是full(每个高斯子模型各自拥有一个普

通的协方差矩阵)和tied(所有高斯子模型共用一个普通的协方差矩阵),每个协方差矩阵内各个变量之间有一定联系,能适用更普遍的情况,不过因为情况比较复杂,所以训练时间较久

**观察不同初始化对最后结果的影响**

b. 本次实验共采用了两种初始化方法,随机初始化是从样本随机取出部分点当作均值,结果波动比较大,不过有时结果不错,第二种初始化方法是采用了K-means++的初始化方法,相当于是有方向的选择,这使得结果比较稳定

## 实验总结

**比较 K-means 聚类方法和 EM 训练的 GMM 聚类方法之间的优劣**

**K-means聚类方法的优势:**

- 1.简单易实现: K-means算法相对简单,容易理解和实现。
- 2.计算效率高: K-means算法的计算复杂度较低,适用于大规模数据集。
- 3.结果易解释: K-means产生的聚类结果是硬聚类,每个样本只属于一个簇,结果直观易解释。

**K-means聚类方法的劣势:**

- 1.初始值敏感: K-means算法对初始聚类中心的选择敏感,不同的初始值可能导致不同的聚类结果。

**EM训练的GMM聚类方法的优势:**

- 1.处理更复杂的分布: GMM聚类方法可以建模更复杂的数据分布,每个簇可以由一个或多个高斯分布组成。
- 2.软聚类: GMM算法产生的聚类结果是软聚类,每个样本可以属于多个簇,提供了更丰富的信息。

**EM训练的GMM聚类方法的劣势:**

- 1.计算复杂度高: GMM算法的计算复杂度较高,特别是在高维数据集上。
- 2.结果解释相对复杂: 由于GMM产生的是软聚类,结果解释相对复杂,不够直观。

**综上所述**, K-means聚类方法在简单性和计算效率方面具有优势,适用于大规模数据集和简单形状的簇。而EM训练的GMM聚类方法在处理复杂分布、软聚类和噪声数据方面更具优势。

在本次实验中,当采用固定簇数时, GMM的准确率比K-means更高,不过GMM的算法理解比较困难,耗费了好长时间才大致搞懂GMM,并且在编写代码时,还遇到了很多困难,比如: nan、矩阵可逆等问题

## 实验资料

1. K-Means: <https://zhuanlan.zhihu.com/p/78798251>
2. GMM: <https://uttermost-brain-693.notion.site/Expectation-Maximization-EM-3e3d8414f21240e4876234c694789e1a?pvs=25#d53cd66cecad493ea25bc5c61efb0945>  
公式推导:
  - a. <https://zhuanlan.zhihu.com/p/85338773>
  - b. <https://uttermost-brain-693.notion.site/Expectation-Maximization-EM-3e3d8414f21240e4876234c694789e1a?pvs=25#d53cd66cecad493ea25bc5c61efb0945>