[redacted]学生实验报告

开课学院及实验室：计算机科学与网络工程学院 412 实验室　　2023 年 12 月 5 日

学院	计算机科学与网络工程学院	年级/专业/班		姓名	[redacted]	学号	[redacted]
实验课程名称	数据挖掘与机器学习实验					成绩	
实验项目名称	聚类分析					指导老师	[redacted]

实验三　聚类分析

一、 实验目的

本实验课程是计算机、人工智能、软件工程等专业学生的一门专业课程，通过实验，帮助学生更好地掌握数据挖掘与机器学习相关概念、技术、原理、应用等；通过实验提高学生编写实验报告、总结实验结果的能力；使学生对机器学习模型、算法等有比较深入的认识。

要掌握的知识点如下：

1. 掌握机器学习中涉及的相关概念、模型、算法；
2. 熟悉机器学习模型训练、验证、测试的流程；
3. 熟悉常用的数据预处理方法；
4. 掌握聚类分析问题的表示、求解及编程。

二、基本要求

1. 实验前，复习《数据挖掘与机器学习》课程中的有关内容。
2. 准备好实验数据，编程完成实验内容，收集实验结果。
3. 独立完成实验报告。

三、实验软件

推荐使用 Python 编程语言（允许使用 numpy 库，需实现详细实验步骤，不允许直接调用 scikit-learn 中回归、分类、聚类等高层 API）。

四、实验内容：

基于 IRIS 鸢尾花数据集，完成关于鸢尾花的聚类分析。

1 准备数据集并认识数据

下载 IRIS 数据集

https://archive.ics.uci.edu/ml/datasets/iris

了解数据集各个维度特征的含义

列名	说明	类型
SepalLength	花萼长度	float
SepalWidth	花萼宽度	float
PetalLength	花瓣长度	float
PetalWidth	花瓣宽度	float
Class	类别变量，0 表示山鸢尾，1 表示变色鸢尾，2 表示维吉尼亚鸢尾。	int

2 探索数据并预处理数据

观察数据集各个维度特征的数值类型与分布

挑选 sepal length、petal length 两维特征作为聚类依据

3 求解聚类中心

编程实现 k-means 聚类、混合高斯聚类

4 测试和评估模型

在数据集上计算聚类的性能指标

五、实验报告

（1）简要介绍 k-means、混合高斯聚类的原理

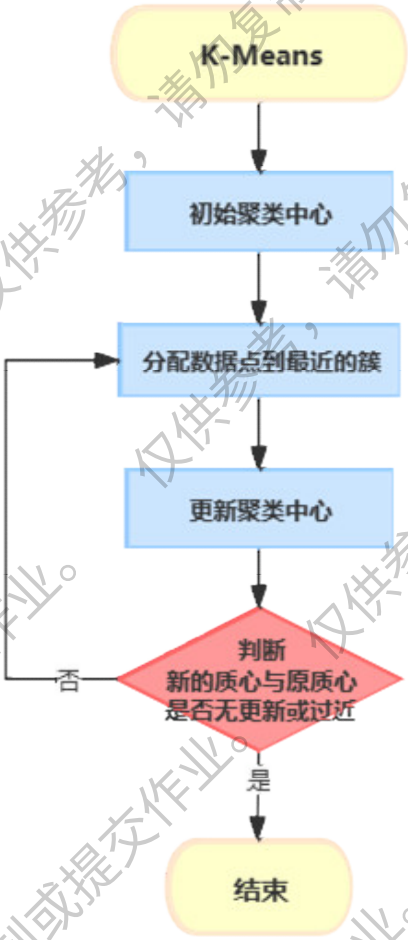
1.k-means 算法原理：

K 均值（K-Means）算法是一种经典的基于距离的聚类算法，其核心思想是利用对象之间的距离作为相似性的评价指标，认为距离越近的两个对象相似度越大。其作用是将数据集划分为 K 个不重叠的簇。

以下是 K 均值算法的主要步骤：

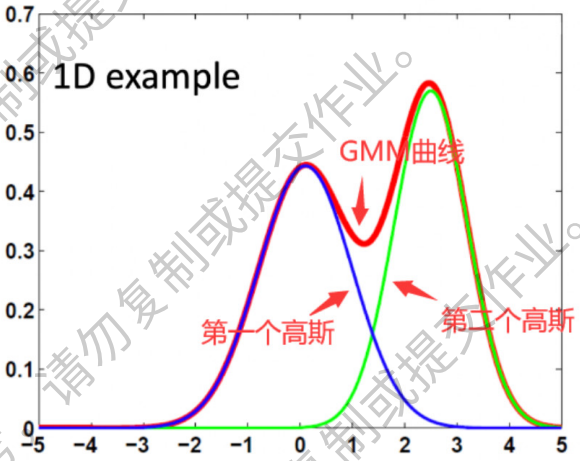
STEP 1. 随机选取 k 个样本点作为初始聚类中心
STEP 2. 分配数据点到最近的簇
STEP 3. 更新聚类中心
STEP 4. 重复 2 和 3

算法流程图如下：



2. 混合高斯聚类原理：

混合高斯聚类（GMM）是一种基于概率模型的**软聚类**方法，它假设数据是由多个高斯分布组成的线性混合体。与 K 均值算法不同，混合高斯聚类允许一个数据点同时属于多个簇，每个簇对应一个高斯分布（但是最终选取概率最大的簇做为聚类结果）。



以下是混合高斯聚类的基本原理：

GMM 表达式：

$$p_M(x)=\sum_{k=1}^{K}\pi_k\cdot \mathrm{N}\left(x\mid\mu_k,\Sigma_k\right)$$

$$\sum_{k=1}^{K}\pi_k=1,\quad \pi_k>0$$

其中：π_k 为第 k 高斯分布的权重

μ_k 为第 k 高斯分布的均值

Σ_k 为第 k 高斯分布的（协）方差

GMM 中一共有 3 个（类）参数需要求解，采用期望最大化 EM 算法求解，EM 算法流程如下：

STEP 1. 初始化 K 个高斯分布的参数（均值、方差、权重）

STEP 2. E-STEP （计算所有样本由第 K 个高斯生成的后验概率）

STEP 3. M-STEP （根据上一步计算的概率更新参数π_k、 μ_k、 Σ_k）

STEP 4. 重复 2 和 3，直到收敛

输入: 样本集 $D=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$;
高斯混合成分个数 K

过程:

1: 初始化高斯混合分布的模型参数 $\{(\pi_k,\boldsymbol{\mu}_k,\Sigma_k)\mid 1\leqslant k\leqslant K\}$
2: **repeat**
EM 算法的 E 步. 3: **for** $i=1,2,\ldots,N$ **do**
4: 计算 $\boldsymbol{x}_i$ 由各混合成分生成的后验概率, 即

$$\gamma_{ik}=\frac{\pi_k\cdot\mathcal{N}(\boldsymbol{x}_i\mid\boldsymbol{\mu}_k,\Sigma_k)}{\sum_{j=1}^{K}\pi_j\cdot\mathcal{N}(\boldsymbol{x}_i\mid\boldsymbol{\mu}_j,\Sigma_j)}$$

5: **end for**
EM 算法的 M 步. 6: **for** $k=1,2,\ldots,K$ **do**
7: 计算新均值向量: $\boldsymbol{\mu}'_k=\frac{\sum_{i=1}^{N}\gamma_{ik}\boldsymbol{x}_i}{\sum_{i=1}^{N}\gamma_{ik}}$
8: 计算新协方差矩阵: $\boldsymbol{\Sigma}'_k=\frac{\sum_{i=1}^{N}\gamma_{ik}(\boldsymbol{x}_i-\boldsymbol{\mu}'_k)(\boldsymbol{x}_i-\boldsymbol{\mu}'_k)^{\mathrm{T}}}{\sum_{i=1}^{N}\gamma_{ik}}$
9: 计算新混合系数: $\pi'_k=\frac{1}{N}\sum_{i=1}^{N}\gamma_{ik}$
10: **end for**
11: 将模型参数 $\{(\pi_k,\boldsymbol{\mu}_k,\Sigma_k)\mid 1\leqslant k\leqslant K\}$ 更新为 $\{(\pi'_k,\boldsymbol{\mu}'_k,\boldsymbol{\Sigma}'_k)\mid 1\leqslant k\leqslant K\}$
12: **until** 满足停止条件 (例如已达到最大迭代轮数, 或似然函数 $LL(D)$ 增长很少)
13: $C_k=\varnothing$ $(1\leqslant k\leqslant K)$
14: **for** $i=1,2,\ldots,N$ **do**
15: 根据下式确定 $\boldsymbol{x}_i$ 的簇标记 λ_i;

$$\lambda_i=\underset{k\in\{1,2,\cdots,K\}}{\arg\max}\ \gamma_{ik}$$

16: 将 $\boldsymbol{x}_i$ 划入相应的簇: $C_{\lambda_i}=C_{\lambda_i}\bigcup\{\boldsymbol{x}_i\}$
17: **end for**
输出: 簇划分 $C=\{C_1,C_2,\ldots,C_K\}$

简单推导如下：

E-STEP[1] 计算给定样本点 x^i，由第 k 个高斯分布生成的概率 γ_{ik}

$$\begin{aligned}\gamma_{ik}&\triangleq p_M(z=k\mid x^i)\\&=\frac{p_M(z=k,x^i)}{p_M(x^i)}\\&=\frac{P(z=k)\cdot p_M(x^i\mid z=k)}{p_M(x^i)}\\&=\frac{P(z=k)\cdot p_M(x^i\mid z=k)}{\sum_{j=1}^{K}P(z=j)\cdot \mathrm{N}(x^i\mid z=j)}\\&=\frac{\pi_k\cdot p_M(x^i\mid z=k)}{\sum_{j=1}^{K}\pi_j\cdot \mathrm{N}(x^i\mid z=j)}\\&=\frac{\pi_k\cdot \mathrm{N}(x^i\mid \mu_k,\Sigma_k)}{\sum_{j=1}^{K}\pi_j\cdot \mathrm{N}(x^i\mid \mu_j,\Sigma_j)}\end{aligned}$$

[1] 核心是引入隐变量 z，这样直接知道求的是哪个高斯，简化求解

M-STEP 由上一步计算的γ_{ik}来更新参数π_k、 μ_k、 Σ_k

似然函数[2]:

$$\begin{aligned} L(\theta) &= \ln(p_M(x)) \\ &= \ln(\prod_{i=1}^{N} p_M(x^i)) \\ &= \sum_{i=1}^{N} \ln(p_M(x^i)) \\ &= \sum_{i=1}^{N} \ln(\sum_{k=1}^{K} \pi_k \cdot \mathrm{N}(x^i \mid \mu_k, \Sigma_k)) \end{aligned}$$

分别对参数π_k, μ_k, Σ_k进行求偏导，

然后分别令$\frac{\partial}{\partial \mu_k} L(\theta) = 0$ $\frac{\partial}{\partial \Sigma_k} L(\theta) = 0$，解得：

$$\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x^i}{\sum_{i=1}^{N} \gamma_{ik}} \qquad \Sigma_k = \frac{\sum_{i=1}^{N} \gamma_{ik} (x^i - \mu_k)(x^i - \mu_k)^{\mathsf{T}}}{\sum_{i=1}^{N} \gamma_{ik}}$$

对π_k，因为存在约束条件$\sum_{k=1}^{K} \pi_k = 1, \ \pi_k > 0$，用拉格朗日乘数法来求解条件极值问题。

拉格朗日函数：

$$L(\theta) + \lambda \left(\sum_{k=1}^{N} \pi_k - 1 \right)$$

令上式对π_k的导数为 0，求解下述方程组：

$$\begin{cases} \sum_{i=1}^{N} \frac{\mathrm{N}(x^i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathrm{N}(x^i \mid \mu_j, \Sigma_j)} + \lambda = 0 \\ \sum_{k=1}^{N} \pi_k = 1 \end{cases}$$

解得：$\pi_k = \frac{1}{N} \sum_{i=1}^{N} \gamma_{ik}$

[2] 似然函数是衡量样本成为观察到数据的概率（回顾一下概念，脑子老是糊..）

（2）程序清单（包含详细求解步骤）

0.导包

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.spatial.distance import cdist
from scipy.stats import multivariate_normal
from sklearn.preprocessing import LabelEncoder

%matplotlib inline
%config InlineBackend.figure_format = 'svg'
plt.rcParams['font.sans-serif']=['SimHei']
```

1. 数据处理

1）导入数据

```
df = pd.read_csv("./iris.data", names=['sepalLength', 'sepalWidth',
'petalLength','petalWidth', 'class'])
df.info()
```

2）查看数据

```
df.head()
```

3）提取 sepalLength 和 petalLength 列数据

```
feature = df[['sepalLength', 'petalLength']]
label = df[['class']]
encode = LabelEncoder()
label = encode.fit_transform(label)

data = feature.values
feature
```

2. K-Means 聚类

```
class KMeans:
    def __init__(self, n_clusters=3, max_iters=100, random_seed=None):
        self.n_clusters = n_clusters  # 簇的数量
        self.max_iters = max_iters  # 最大迭代次数
        self.random_seed = random_seed  # 随机种子
```

```
        self.centroids = None  # 簇中心

    # 初始化簇中心，随机选择数据中的点作为初始中心
    def _initialize_centroids(self, data):
        # 设置随机种子
        np.random.seed(self.random_seed)
        # 从数据中随机选择 self.n_clusters 个索引
        indices = np.random.choice(len(data), self.n_clusters,
replace=False)
        # 提取选择的索引对应的数据作为初始化的聚类中心
        centroids = data[indices]

        return centroids

    # 将数据点分配到最近的簇
    def _assign_clusters(self, data, centroids):
        # 计算每个数据点到各个聚类中心的距离
        distances = cdist(data, centroids)
        # 返回每个样本对应的聚类标签（最近聚类中心的索引）
        return np.argmin(distances, axis=1)

    # 更新簇中心为每个簇内所有数据点的平均值
    def _update_centroids(self, data, labels):
        new_centroids = np.array([data[labels == k].mean(axis=0) for k in
range(self.n_clusters)])
        return new_centroids

    # 训练函数
    def fit(self, data):
        # 初始化簇中心
        self.centroids = self._initialize_centroids(data)
        # 算法迭代主循环
        for _ in range(self.max_iters):
            # 分配样本到最近的簇
            labels = self._assign_clusters(data, self.centroids)
            # 更新簇中心
            new_centroids = self._update_centroids(data, labels)

            # 如果簇中心不再变化，则停止迭代
            if np.all(self.centroids == new_centroids):
                break

            self.centroids = new_centroids
```

```
        return labels

    # 计算 sse
    def calculate_sse(self, data, labels):
        # 检查是否已经拟合模型并有簇中心
        if self.centroids is None:
            raise ValueError("请先聚类！")

        sse = 0.0
        for i in range(self.n_clusters):
            # 获取属于当前簇的数据点
            cluster_points = data[labels == i]
            # 获取当前簇的中心
            centroid = self.centroids[i]
            # 计算数据点到簇中心的平方距离
            squared_distances = np.sum((cluster_points - centroid) ** 2,
axis=1)
            # 将平方距离求和，添加到总的 SSE 中
            sse += np.sum(squared_distances)

        return sse

    # 查看聚类结果
    def report(self):
        return pd.DataFrame(self.centroids)
```

1）求解 k-means 算法

```
# 实例化
kmeans = KMeans(n_clusters=3, max_iters=100, random_seed=42)
# 开始训练
label_predict = kmeans.fit(data)
# 聚类中心
kmeans.report()
```

2）可视化 k-means 算法聚类结果

```
# 绘制聚类结果
plt.scatter(data[:, 0], data[:, 1], c=label_predict, cmap='viridis',
edgecolors='k')
plt.scatter(kmeans.centroids[:, 0], kmeans.centroids[:, 1], marker='X',
s=50, color='red', label='Centroids')
plt.title('KMeans Clustering')
plt.xlabel('sepalLength')
plt.ylabel('petalLength')
```

```
plt.legend()
plt.savefig("1.png",dpi=800)
plt.show()
```

3）k-means 聚类数量与聚类指标的关系

```
# 尝试不同的 n_clusters，以肘部图确定
sse = []

for i in np.arange(1, 10):
    # 实例化
    kmeans = KMeans(n_clusters=i, max_iters=100, random_seed=1314)
    # 开始训练
    labels = kmeans.fit(data)
    # 计算 sse
    sse.append(kmeans.calculate_sse(data, labels))

# 绘制 sse 曲线
plt.plot(range(1, 10), sse, marker='o')
plt.scatter(3, sse[2], color='y', label='拐点', s=100)
plt.xlabel("n_clusters")
plt.xticks(np.arange(1, 10))
plt.ylabel("SSE")
plt.legend()
plt.title("肘部图")
```

3. 混合高斯分布 GMM

```
class GMM:
    def __init__(self, n, max_iters=100, tol=1e-4, random_seed=None):
        self.n = n  # 高斯分布的个数
        self.max_iters = max_iters  # 最大迭代次数
        self.tol = tol  # 收敛阈值
        self.random_seed = random_seed  # 随机种子
        self.weights = None  # 每个高斯分布的权重
        self.means = None  # 每个高斯分布的均值
        self.covariances = None  # 每个高斯分布的协方差矩阵

    # 初始化参数
    def init (self, data):
        np.random.seed(self.random_seed)
        # 随机选择数据中的点作为初始均值
        random_indices = np.random.choice(len(data), self.n,
replace=False)
        self.means = data[random_indices]
```

```
        # 初始权重相等
        self.weights = np.ones(self.n) / self.n
        # 初始协方差矩阵为数据的协方差
        self.covariances = [np.cov(data.T) for _ in range(self.n)]

    # E-STEP
    def expectation_step(self, data):
        respon = np.zeros((len(data), self.n))

        # 对每个高斯
        for k in range(self.n):
            # 创建一个多元正态分布对象，由参数（均值，(协)方差）
            multivariate_normal_dist =
multivariate_normal(mean=self.means[k], cov=self.covariances[k])
            # 计算每个样本属于每个高斯分布的概率，权重乘以每个样本属于当前高斯分布
的概率
            respon[:, k] = self.weights[k] *
multivariate_normal_dist.pdf(data)

        # 归一化概率
        respon /= np.sum(respon, axis=1, keepdims=True)
        return respon

    # M-STEP
    def maximization_step(self, data, respon):
        # 计算每个高斯分布在整体数据集上的权重之和
        total_weight = np.sum(respon, axis=0)
        # 归一，更新权重
        self.weights = total_weight / len(data)
        # 计算均值
        self.means = (respon.T @ data) / total_weight[:, np.newaxis]
        # 更新协方差矩阵
        self.covariances = []
        for k in range(self.n):
            diff = data - self.means[k]
            covariance = np.dot(respon[:, k] * diff.T, diff) /
total_weight[k]
            self.covariances.append(covariance)

    # 训练
    def fit(self, data):
        self.init (data)

        for _ in range(self.max_iters):
```

```
            # E 步：计算每个样本属于每个高斯分布的概率
            respon = self.expectation_step(data)

            old_means = np.copy(self.means)

            # M 步：更新均值、权重和协方差矩阵
            self.maximization_step(data, respon)

            # 如果均值不再变化，则停止迭代
            if np.linalg.norm(self.means - old_means) < self.tol:
                break

        return respon.argmax(axis=1)
```

1）求解 GMM

```
# 实例化
gmm = GMM(n=3, max_iters=100, random_seed=42)
# 训练
predict_labels = gmm.fit(data)
```

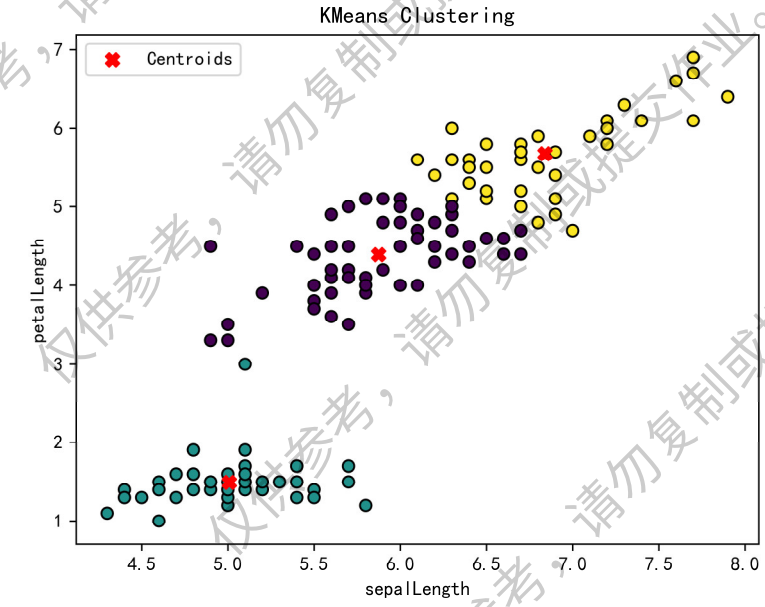
2）可视化求解 GMM 结果

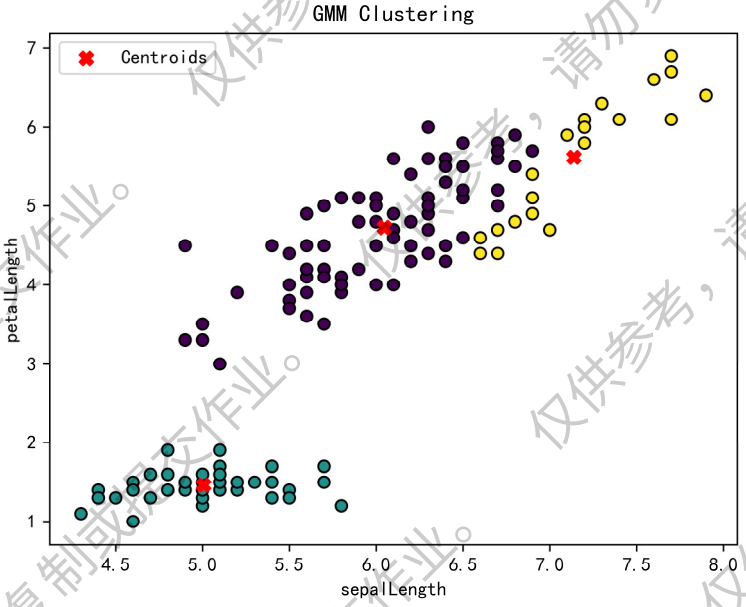
```
# 绘制混合高斯分布的聚类结果
plt.scatter(data[:, 0], data[:, 1], c=predict_labels, cmap='viridis',
edgecolors='k')
plt.scatter(gmm.means[:, 0], gmm.means[:, 1], marker='X', s=50,
color='red', label='Centroids')
plt.title('GMM Clustering')
plt.xlabel('sepalLength')
plt.ylabel('petalLength')
plt.legend()
plt.savefig("2.png",dpi=800)
plt.show()
```

（3）展示实验结果，可视化聚类结果

0	1
5.874138	4.393103
5.007843	1.494118
6.839024	5.678049

0	1
6.046630	4.730670
5.006024	1.463942
7.139048	5.619336





（4）讨论实验结果，分析 k-means 聚类数量与聚类指标的关系

1. 讨论实验结果

混合高斯聚类是一种软聚类方法，相比硬聚类方法，它更能反映数据点属于不同簇的不确定性。

K 均值（K-Means）聚类算法和混合高斯模型（GMM）聚类算法之间存在一些差异：

首先 K-Means 是一种**硬聚类**算法，每个数据点只能属于一个簇；而 GMM 算法是一种**软聚类**算法，每个数据点可以以不同的概率属于多个簇。

并且两者对每个簇的形状假设有所差异，K-Means 假设每个簇是以质心为中心的**球形簇**；而 GMM 算法使用**高斯分布**建模每个簇，可以适应不同形状的簇，每个簇的形状由其对应的高斯分布决定。

再者，K-Means 对**异常值**非常敏感，可能导致簇产生较大的偏移；而 GMM 算法是一种软聚类算法，异常值对其的影响相对较小。

最后，两者的差异本质上是软聚类与硬聚类的区别。GMM 以概率为基础，而 K-Means 以“距离”为基础。它们都是迭代算法，先对参数赋初值，然后交替执行对数据的估计和用上一步估计值重新计算参数值的步骤。对于 K-Means 是簇心位置，对于 GMM 是混合系数和各个高斯分布的中心位置和协方差矩阵。但是，GMM 还可以给出某个样本点属于某个类别的概率，提供更丰富的信息。

从实验结果来看，的确是 GMM 的效果会更加合理，但是 GMM 会比 K-Means 复杂得多，背后的原理更加充实，所以效果好其实也是情理之中。

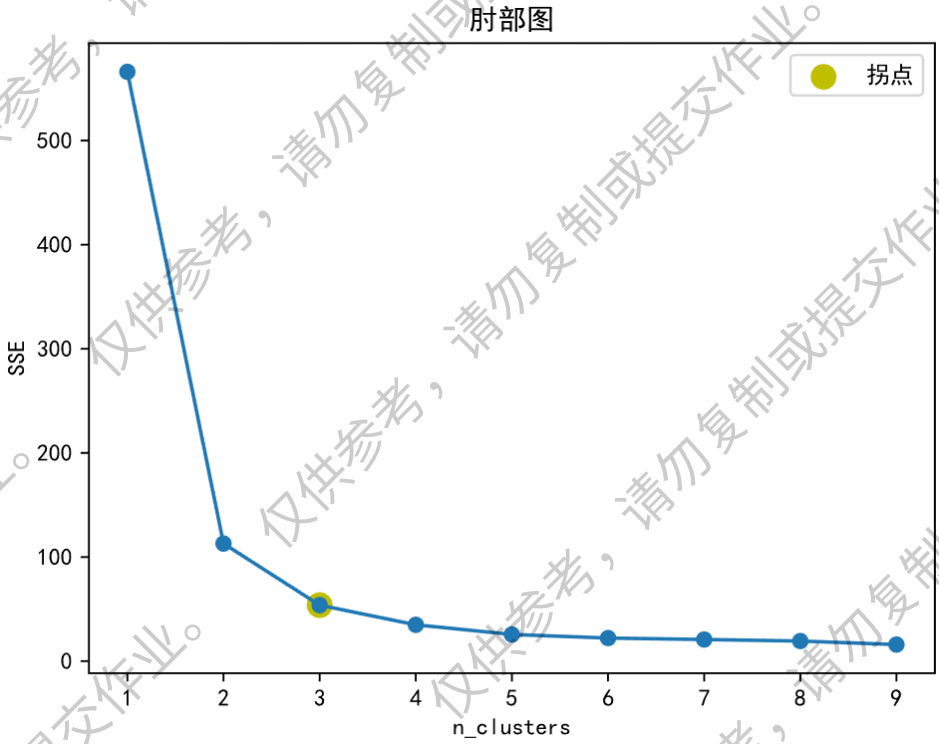
2. 分析 k-means 聚类数量与聚类指标的关系

我这里采用的聚类指标是 SSE（Sum of Squared Error），它衡量了每个数据点与其所属簇的中心之间的距离的平方和，即簇内方差的总和。SSE 越小表示数据点越接近其所属簇的中心，即簇内的紧密度越高，聚类效果越好。

$$SSE = \sum_{i=1}^{n}\sum_{j=1}^{k}\left\|x^i - \mu^j\right\|^2$$

x^i第i个数据点

μ^j第j个簇中心



从上图可以看到随着簇数的增加，SSE 减小，更多的簇导致了更小的簇内距离。但当簇数达到真实簇数 3 时，增加簇数对 SSE 的影响会减缓，形成一个肘部。