Python与大数据分析

--聚类

知识点概括

- 聚类概述
- 聚类算法介绍
- KMeans的原理
- KMeans的实现与应用

聚类概述

聚类定义

聚类 是一种将数据集划分为多个“簇”的技术，其中同一簇内的元素相似度较高，而不同簇之间的元素相似度较低。

无监督学习：聚类属于无监督学习方法，因为它不需要预先标注的数据标签

形式化地说，假定样本集 $D = \{x_1, x_2, \dots, x_m\}$ 包含m 个无标记样本，每个样本$x_i = \{x_{i1}, x_{i2}, \dots, x_{in}\}$是一个 n 维特征向量,则聚类算法将样本集划分为$D$个不相交的簇$\{C_l \mid l = 1,2, \dots, k\}$,其中$C_l \cap C_{l' \neq l} = \emptyset$ 且$D = \cup_{l=1}^{k} C_l$.相应地,我们用$\lambda_j \in \{1,2, \dots, k\}$ 表示样本的x_j的“簇标记”(cluster label),即$x_j \in C_{\lambda_j}$.于是,聚类的结果可用包含m个元素的标记向量 $\boldsymbol{\lambda} = \{\lambda_1; \lambda_2; \dots; \lambda_m\}$表示.

聚类概述

聚类的目标和重要性

目标：

- **发现数据中的潜在结构**：聚类帮助我们在没有标签的情况下揭示数据的内在模式。
- **分组相似数据**：将数据点根据相似性进行分组，便于后续的分析、处理或预测。
- **简化数据复杂度**：通过将大量数据压缩为较少的簇，简化数据处理和理解。

重要性：

- **帮助决策**：聚类分析可以为企业提供洞察，帮助制定更为精准的市场策略或产品推荐。
- **提高数据处理效率**：通过聚类，将不同的任务或数据进行合理的分组，降低计算复杂度。
- **数据预处理**：聚类可以作为其他机器学习算法的前处理步骤，如为分类问题提供簇标签，或者为降维提供参考

聚类算法的应用领域

- **数据分析：**
  - 数据的探索性分析，发现数据中的潜在结构。
  - 如：客户数据分析、社交网络分析。
- **图像识别：**
  - 图像分割：通过聚类算法将图像中的不同区域分割开来，应用于人脸识别、物体识别等领域。
  - 如：基于像素的聚类，颜色和形状的聚类分析。
- **市场细分：**
  - 根据消费者行为、购买习惯等特征，将市场分成若干个群体，帮助企业制定有针对性的营销策略。
  - 如：将顾客分为高价值顾客、潜力顾客等不同群体。
- **生物信息学：**
  - 基因表达数据的聚类分析，帮助发现基因的功能。
  - 如：基因簇的分析，疾病模式识别。
- **文本挖掘：**
  - 文档聚类，自动将类似的文档归为一类，用于信息检索、推荐系统等。
  - 如：新闻文章分类、邮件垃圾分类。

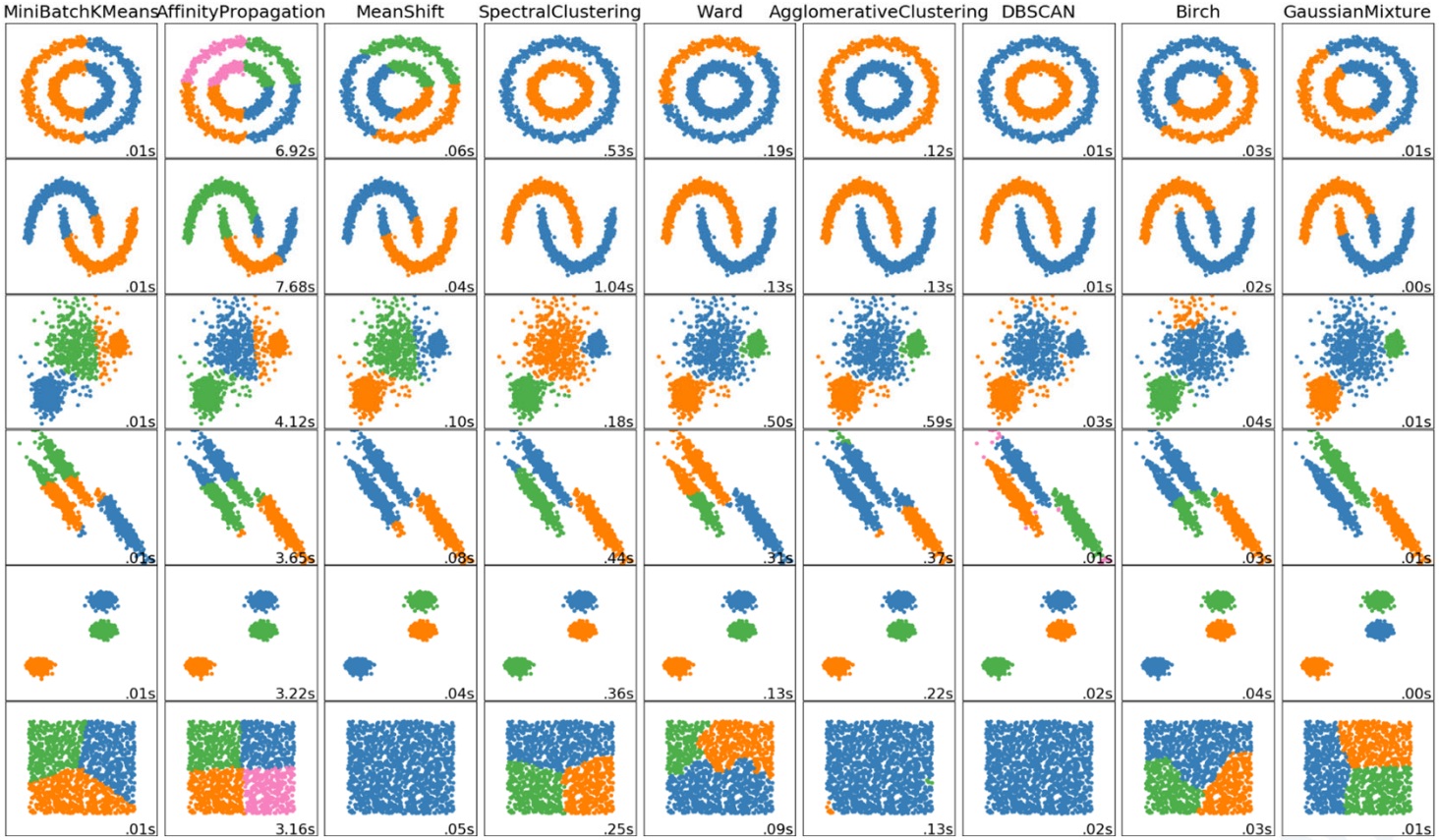
聚类算法

聚类算法有多种类型，每种方法都有不同的优缺点和适用场景。常见的聚类算法包括：

- **K-Means算法**：是一种基于划分的聚类算法，它通过最小化簇内样本的方差来划分数据点。
- **层次聚类**：是一种通过构建树形结构（树状图）来逐步合并（或分割）簇的聚类方法。主要分为**自底向上**（凝聚型）和**自顶向下**（分裂型）两种方法。
- **DBSCAN（Density-Based Spatial Clustering of Applications with Noise）**：是一种基于密度的聚类方法，能够发现任意形状的簇，并且能够自动识别噪声数据。
- **高斯混合模型（Gaussian Mixture Model）**：是一种基于概率模型的聚类方法，它假设数据来自多个高斯分布的混合。

聚类概述

在不同数据集上不同聚类算法的效果图



不同分类算法对比

特性	K-Means算法	层次聚类	DBSCAN	高斯混合模型
算法类型	划分型聚类	层次聚类	基于密度的聚类	基于概率的聚类
簇形状假设	簇为球形（均匀密度）	无假设（可处理任意形状的簇）	簇形状不固定，能够发现任意形状的簇	簇为高斯分布（圆形或椭圆形）
簇数	需要预设K值	不需要预设簇数	不需要预设簇数	需要预设簇数
对异常值的鲁棒性	不鲁棒，对噪声和离群点敏感	不鲁棒，对噪声和离群点敏感	对噪声和离群点鲁棒	对噪声不太鲁棒
算法复杂度	时间复杂度为O(nKd)（n为样本数，K为簇数，d为特征数）	时间复杂度为$O(n^2)$（n为样本数）	时间复杂度为O(n*log(n))（n为样本数）	时间复杂度为O(nKd)（n为样本数，K为簇数，d为特征数）
初始化要求	需要初始化K个簇中心	无初始化要求	无初始化要求	需要初始化（通过EM算法估算高斯分布参数）
处理高维数据的能力	对高维数据表现较差，可能会受到“维度灾难”影响	对高维数据表现较差，计算复杂度高	对高维数据效果较差，可能需要降维	在高维数据中表现较好，但对初始化较为敏感
适用场景	聚类数已知，数据分布均匀，簇为球形	用于探索数据的层次结构或多层次聚类	适用于有噪声数据及具有不规则簇形的数据	适用于数据集包含多个高斯分布的情况，特别是密度不同的簇
优点	简单，效率高，适用于大规模数据集	可以处理任意形状的簇，提供层次结构	能发现任意形状的簇，能有效识别噪声点和边界点	能够处理不同密度的簇，适应性强
缺点	对K值敏感，对噪声敏感，容易陷入局部最优	计算复杂度高，时间消耗大	对参数选择敏感（如邻域半径），难以处理密度差异较大的数据	对初始化和参数选择敏感，容易收敛到局部最优解
常见应用	图像压缩、市场细分、客户分群	生物学中的物种分类、图像分割、基因数据分析	异常检测、地理信息系统（GIS）、图像处理等	图像分割、语音识别、异常检测等

距离算法设计的两个基本问题

➢ 距离计算

对距离$dist(\cdot, \cdot)$，若它是一个“距离度量”，则它满足一些基本性质：

- 非负性：$dist(x_i, x_j) \geq 0$；
- 同一性：$dist(x_i, x_j) = 0$ 当且仅当$x_i = x_j$；
- 对称性：$dist(x_i, x_j) = dist(x_j, x_i)$
- 三角不等式：$dist(x_i, x_j) \leq dist(x_i, x_k) + dist(x_k, x_j)$

距离算法设计的两个基本问题

- 距离计算

给定样本 $x_i = (x_{i1}; x_{i2}; \ldots; x_{in})$ 与 $x_j = (x_{j1}; xj_2; \ldots; x_{jn})$，最常用的是“闵可夫斯基距离”：

$$\mathrm{dist}_{\mathrm{mk}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{u=1}^{n} |x_{iu} - x_{ju}|^p \right)^{\frac{1}{p}}$$

其中$p \geq 1$，其显然满足前面的四条距离的基本性质

距离算法设计的两个基本问题

➢ 距离计算

- 当 p = 2，闵可夫斯基距离即为欧式距离：

$$\mathrm{dist}_{\mathrm{ed}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2 = \sqrt{\sum_{u=1}^{n} |x_{iu} - x_{ju}|^2}$$

- 当 p = 1，闵可夫斯基距离即为曼哈顿距离：

$$\mathrm{dist}_{\mathrm{man}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = ||\boldsymbol{x}_i - \boldsymbol{x}_j||_1 = \sum_{u=1}^{n} |x_{iu} - x_{ju}|$$

区别：曼哈顿距离计算简单，适合高维数据；欧式距离计算更为精确，但可能受到维度灾难影响。

距离算法设计的两个基本问题

➢ 性能度量

性能度量：亦称“有效性指标”。与监督学习中的性能度量作用相似,对聚类结果,我们需通过某种性能度量来评估其好坏;另一 方面,若明确了最终将要使用的性能度量,则可直接将其作为聚类过程的优化目标, 从而更好地得到符合要求的聚类结果.

聚类是将样本集 D 划分为若干互不相交的子集,即样本簇.那么,什么样的聚类结果比较好?.

- “簇内相似度”(intra-cluster similarity) 高
- “簇间相似度”(inter-cluster similarity) 低

距离算法设计的两个基本问题

➢ 性能度量

聚类性能度量大致有两类：

- 内部指标 (internal index)：直接考察聚类结果而不利用任何参考模型。

  常见的内部指标有DB指数、Dunn指数等

- 外部指标 (external index)：将聚类结果与某个“参考模型”(reference model) 进行比较。

  常见的外部指标有Jaccard系数、FM指数、Rand指数等

距离算法设计的两个基本问题

内部指标

考虑聚类结果的簇划分 $C = \{C_1, C_2, \ldots, C_k\}$，定义：

$$\mathrm{avg}(C) = \frac{2}{|C|(|C|-1)} \sum_{1 \leqslant i < j \leqslant |C|} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) ,$$

$$\mathrm{diam}(C) = \max_{1 \leqslant i < j \leqslant |C|} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) ,$$

$$d_{\min}(C_i, C_j) = \min_{\boldsymbol{x}_i \in C_i, \boldsymbol{x}_j \in C_j} \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j) ,$$

$$d_{\mathrm{cen}}(C_i, C_j) = \mathrm{dist}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) ,$$

其中，$dist(\cdot, \cdot)$ 用于计算两个样本之间的距离；μ 代表簇 C 的中心点 $\mu = \frac{1}{|C|}\sum_{1 \leq i \leq |C|} x_i$。显然，$avg(\cdot)$ 对应于簇内 C 样本间的平均距离，$diam(\cdot)$ 对应于簇 C 内样本间的最远距离，$d_{min}(C_i, C_j)$ 对应于簇 C_i 与簇 C_j 最近样本间的距离，$d_{cen}(C_i, C_j)$ 对应于簇 C_i 与簇 C_j 中心点间的距离。

距离算法设计的两个基本问题

内部指标

基于avg，diam， d_{min}， d_{cen}可导出下面这些常用的聚类性能度量内部指标:

- DB指数（Davies-Bouldin Index，简称DBI）：

$$DBI = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{\mathrm{avg}(C_i)+\mathrm{avg}(C_j)}{d_{\mathrm{cen}}(\boldsymbol{\mu}_i,\boldsymbol{\mu}_j)}\right)$$

- Dunn指数（Dunn Index，简称DI）：

$$DI = \min_{1\leqslant i\leqslant k}\left\{\min_{j\neq i}\left(\frac{d_{\min}(C_i,C_j)}{\max_{1\leqslant l\leqslant k}\mathrm{diam}(C_l)}\right)\right\}$$

显然，DBI的值越小越好，而DI则相反，值越大越好。

距离算法设计的两个基本问题

外部指标

对数据集 $D = \{x_1, x_2, \ldots, x_m\}$，假定通过聚类给出的划分为 $C = \{C_1, C_2, \ldots, C_k\}$ 参考模型给出的簇划分为 $C^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$。相应地，令 $\boldsymbol{\lambda}$ 与 $\boldsymbol{\lambda}*$ 分别表示与 C 和 C*对应的标记向量。我们将样本两两配对考虑，定义：

$$
\begin{aligned}
a &= |SS|, \quad SS = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j)\}, \\
b &= |SD|, \quad SD = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j)\}, \\
c &= |DS|, \quad DS = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j)\}, \\
d &= |DD|, \quad DD = \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j)\},
\end{aligned}
$$

其中集合 SS 包含了在 C 中隶属于相同簇且在 C* 中也隶属于相同簇的样本对，集合 SD 包含了在 C 中隶属于相同簇但在 C* 中隶属于不同簇的样本对。由于每个样本$(x_i, x_j)(i < j)$仅能出现在一个集合中，因此有 $a + b + c + d = m(m-1)/2$ 成立.

距离算法设计的两个基本问题

外部指标

基于a，b，c，d可导出下面这些常用的聚类性能度量外部指标:

- Jaccard系数（Jaccard Coefficient，简称JC）：

$$\mathrm{JC} = \frac{a}{a+b+c}$$

- FM指数（Fowlkes and Mallows Index，简称FMI）：

$$\mathrm{FMI} = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}$$

- Rand指数（Rand Index，简称RI）：

$$\mathrm{RI} = \frac{2(a+d)}{m(m-1)}$$

上述性能度量的结果值都在 [0,1] 区间，值越大越好。

K-means

K-Means算法：是一种通过最小化簇内数据点到簇中心的距离的方式进行数据聚类的方法。其目标是将数据集分成K个簇，每个簇内的点与簇中心尽量相似，不同簇之间的点差异尽量大。

给定样本集集 $D = \{x_1, x_2, \ldots, x_m\}$，K-means算法针对所得簇划分 $C = \{C_1, C_2, \ldots, C_k\}$ 最小化平方误差：

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} ||\boldsymbol{x} - \boldsymbol{\mu}_i||_2^2$$

其中 $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ 是 C_i 的均值向量，直观来看，平方误差 E 在一定程度上刻画了簇内样本围绕簇均值向量的紧密程度，值越小则簇内样本相似度越高。

K-means

最小化式平方误差 E 并不容易, 找到它的最优解需考察样本集 D 所有可能的簇划分,这是一个NP难问题。因此,均值算法采用了贪心策略,通过迭代优化来近似求解式.

- 第1行对均值向量进行初始化；
- 在第4-8行对当前簇划分迭代更新；
- 第9-16行对当前均值向量迭代更新；
- 若迭代更新后聚类结果保持不变,则在第18行将当前划分结果返回.

输入: 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
聚类簇数 k.

过程:

1: 从 D 中随机选择 k 个样本作为初始均值向量 $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$
2: **repeat**
3: 令 $C_i = \varnothing \ (1 \leqslant i \leqslant k)$
4: **for** $j = 1, 2, \ldots, m$ **do**
5: 计算样本 $\boldsymbol{x}_j$ 与各均值向量 $\boldsymbol{\mu}_i \ (1 \leqslant i \leqslant k)$ 的距离: $d_{ji} = ||\boldsymbol{x}_j - \boldsymbol{\mu}_i||_2$;
6: 根据距离最近的均值向量确定 $\boldsymbol{x}_j$ 的簇标记: $\lambda_j = \arg\min_{i \in \{1,2,\ldots,k\}} d_{ji}$;
7: 将样本 $\boldsymbol{x}_j$ 划入相应的簇: $C_{\lambda_j} = C_{\lambda_j} \bigcup \{\boldsymbol{x}_j\}$;
8: **end for**
9: **for** $i = 1, 2, \ldots, k$ **do**
10: 计算新均值向量: $\boldsymbol{\mu}'_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$;
11: **if** $\boldsymbol{\mu}'_i \neq \boldsymbol{\mu}_i$ **then**
12: 将当前均值向量 $\boldsymbol{\mu}_i$ 更新为 $\boldsymbol{\mu}'_i$
13: **else**
14: 保持当前均值向量不变
15: **end if**
16: **end for**
17: **until** 当前均值向量均未更新

输出: 簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$

算法流程图

K-means

我们以西瓜数据集4.0为例来演示k均值算法的学习过程.为方便叙述,我们将编号为 i 的样本称为 x_i ,这是一个包含“密度”与“含糖率” 两个属性值的二维向量.

表 9.1 西瓜数据集 4.0

编号	密度	含糖率	编号	密度	含糖率	编号	密度	含糖率
1	0.697	0.460	11	0.245	0.057	21	0.748	0.232
2	0.774	0.376	12	0.343	0.099	22	0.714	0.346
3	0.634	0.264	13	0.639	0.161	23	0.483	0.312
4	0.608	0.318	14	0.657	0.198	24	0.478	0.437
5	0.556	0.215	15	0.360	0.370	25	0.525	0.369
6	0.403	0.237	16	0.593	0.042	26	0.751	0.489
7	0.481	0.149	17	0.719	0.103	27	0.532	0.472
8	0.437	0.211	18	0.359	0.188	28	0.473	0.376
9	0.666	0.091	19	0.339	0.241	29	0.725	0.445
10	0.243	0.267	20	0.282	0.257	30	0.446	0.459

K-means

假定聚类簇数 $k=3$，算法开始时随机选取三个样本x_6，x_{12}， x_{27}作为初始均值向量，即：

$$\mu_1=(0.403，0.237)，\ \mu_2=(0.343，0.099)，\ \mu_3=(0.532，0.472)$$

考察样本 $x_1=(0.697，0.460)$，它与当前均值向量 μ_1，μ_2，μ_3 的距离分别为 0.369，0.506，0.166，因此 x_1将被划入簇 C_3 中，类似的，对数据集中的所有样本考察一遍后,可得当前簇划分为：

$$C_1=\{\boldsymbol{x}_5,\boldsymbol{x}_6,\boldsymbol{x}_7,\boldsymbol{x}_8,\boldsymbol{x}_9,\boldsymbol{x}_{10},\boldsymbol{x}_{13},\boldsymbol{x}_{14},\boldsymbol{x}_{15},\boldsymbol{x}_{17},\boldsymbol{x}_{18},\boldsymbol{x}_{19},\boldsymbol{x}_{20},\boldsymbol{x}_{23}\};$$

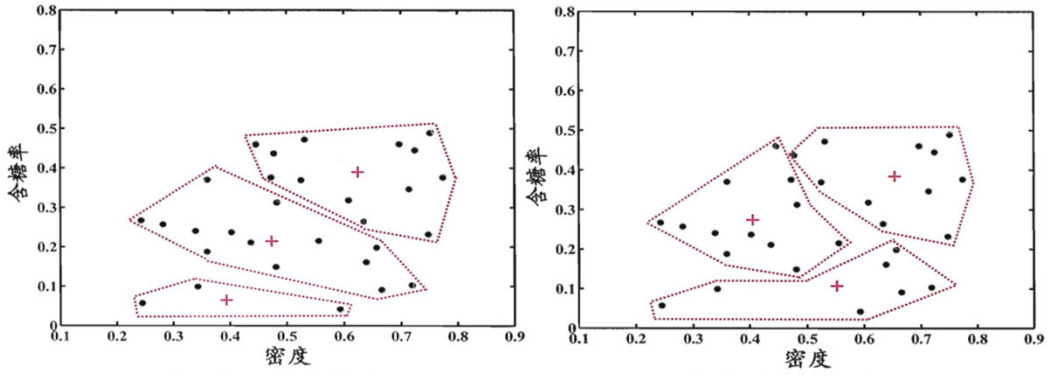
$$C_2=\{\boldsymbol{x}_{11},\boldsymbol{x}_{12},\boldsymbol{x}_{16}\};$$

$$C_3=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3,\boldsymbol{x}_4,\boldsymbol{x}_{21},\boldsymbol{x}_{22},\boldsymbol{x}_{24},\boldsymbol{x}_{25},\boldsymbol{x}_{26},\boldsymbol{x}_{27},\boldsymbol{x}_{28},\boldsymbol{x}_{29},\boldsymbol{x}_{30}\}.$$

于是可以从C_1、 C_2、 C_3分别求出新的均值向量： $\boldsymbol{\mu}_1'=(0.473;0.214)$, $\boldsymbol{\mu}_2'=(0.394;0.066)$, $\boldsymbol{\mu}_3'=(0.623;0.388)$.

确定簇数K的方法

通过更新均值向量后，不断重复上述过程，直至到迭代的结果相同为止，得到最终的簇划分。



(a) 第一轮迭代后　(b) 第二轮迭代后

(c) 第三轮迭代后　(d) 第四轮迭代后

K-means++

K-Means++ 是 K-Means 聚类算法的一种初始化方法，用于提高 K-Means 聚类算法在选择初始质心时的效果，减少聚类结果对初始化的敏感性，并且通常能够加快收敛速度，避免陷入局部最优解。

核心思想:

K-Means 算法依赖于初始化质心的位置，然而，传统的随机初始化方式可能导致初始质心分布不均匀，进而影响算法的聚类效果。K-Means++ 通过一种更智能的方式来选择初始质心，确保质心的初始化更有代表性，从而提高聚类结果的质量。

K-means++

初始化步骤:

1. **选择第一个质心：** 随机选择数据集中的一个点作为第一个质心。
2. **计算每个样本点到当前已选择的质心的距离：** 对于每个样本点，计算它与当前已经选择的质心中最近一个质心的距离（即点到质心的最短距离）。记这些距离为 D(x)，其中 x 表示样本点。
3. **选择下一个质心：** 根据每个样本点到最近质心的距离 D(x) 来选择下一个质心。具体来说，选择一个点的概率与其距离的平方成正比。也就是说，距离当前质心较远的数据点更有可能被选为新的质心。这个步骤保证了质心分布得更加均匀。
4. **重复步骤 2 和 3，直到选择了 k 个质心：** 重复以上步骤，直到选出了 k个质心。

K-means++

选择质心的概率:

在选择下一个质心时，样本点 x 被选中的概率是：

$$P(x) = \frac{D(x)^2}{\sum_{i=1}^{n} D(x_i)^2}$$

其中，D(x) 是数据点 x 到已选质心的最小距离，$\sum_{i=1}^{n} D(x_i)^2$ 是所有数据点到当前质心的最小距离的平方和。

K-means++的优点：

- **更好的初始质心选择：** 由于 K-Means++ 使得质心选择更加均匀，算法能够避免某些最差的初始化情况，降低了初始化敏感性。
- **更高的聚类质量：** 由于初始质心更加合理，K-Means++ 通常能获得更好的聚类结果。
- **较少的迭代次数：** K-Means++ 通常能使算法更快收敛，减少了迭代次数。

确定簇数K的方法

（一）肘部法

- **目标**：确定聚类算法（如K-Means）中最佳聚类数 K。
- **原理**：计算不同 K值对应的聚类总平方误差（SSE，Sum of Squared Errors）。当 K增加到真实聚类数时，SSE下降幅度会突然变缓，形成一个“肘部”拐点（Inflection point），此时对应的 K 即为最佳值。
- **步骤**：
  1. 对不同的 K*K* 值运行聚类算法（如K-Means）。
  2. 计算每个 K*K* 对应的SSE。
  3. 绘制 K*K*-SSE 曲线，找到拐点（肘部）。

确定簇数K的方法

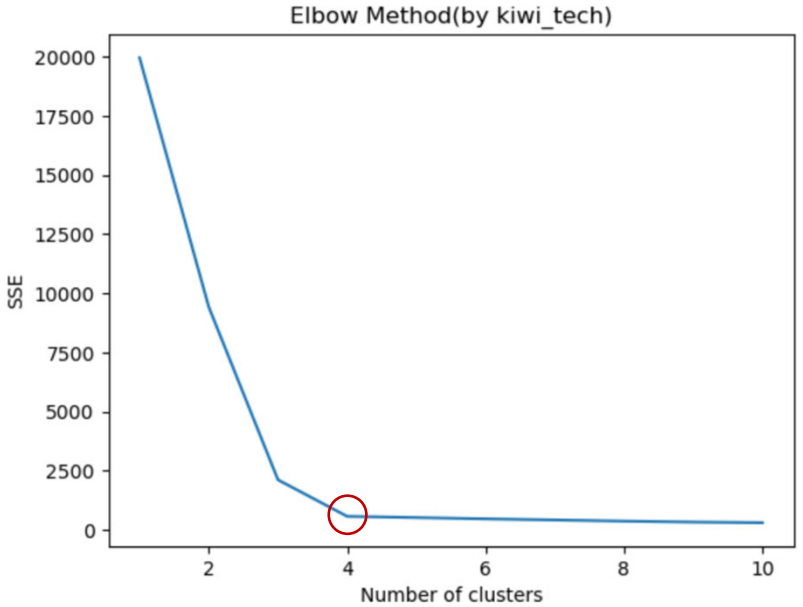
（一）肘部法

误差平方和（SSE）计算方式：

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2$$

其中，

- SSE代表簇划分的好坏，是所有样本的聚类误差
- C_i 代表其中一个簇，也就是第i个簇
- p 代表簇 C_i 中的样本点
- m_i 代表簇 C_i 的质心，也就是 C_i 中所有样本的平均值



最佳 k = 4

确定簇数K的方法

（一）轮廓系数法

- **目标**：评估聚类结果的紧密性和分离性，选择使轮廓系数最大的 K。
- **原理**：对每个样本计算以下两个值：
  - a(i)：样本 i 到同簇其他样本的平均距离（内聚度）。
  - b(i)：样本 i 到最近其他簇所有样本的平均距离（分离度）。
  - 轮廓系数公式：

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

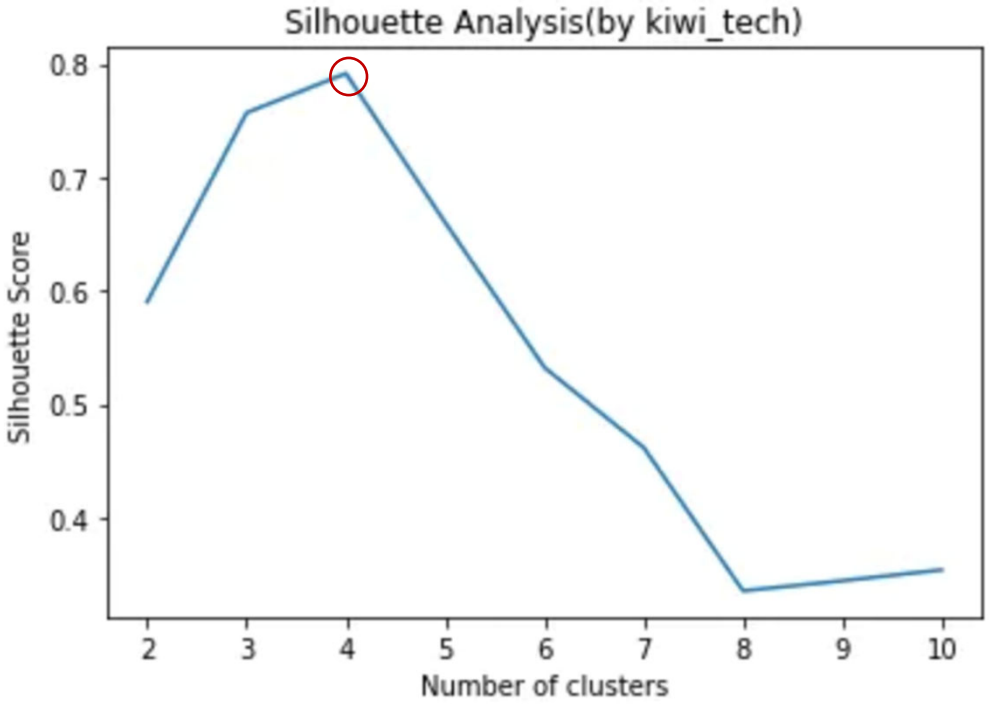
s(i) ∈ [−1,1]，值越大表示聚类效果越好。

确定簇数K的方法

（二）轮廓系数法

步骤：

1. 对不同的 K 值运行聚类算法。
2. 计算每个 K 对应的轮廓系数均值。
3. 画出轮廓系数折线图。
4. 选择轮廓系数最大的 K。



最佳 k = 4

Sklearn方法介绍

LabelEncoder() 是 scikit-learn 库中用于将类别型数据转换为数值型数据的工具。许多机器学习算法要求输入数据为数值型，因此需要对类别型特征进行编码。LabelEncoder 将每个类别映射为一个唯一的整数标签

工作原理：

LabelEncoder 会为每个类别分配一个整数标签，标签的顺序取决于类别的字典序。

- 例如，给定类别 ['paris', 'paris', 'tokyo', 'amsterdam']，LabelEncoder 会将其转换为 [1, 1, 2, 0]。

Sklearn方法介绍

LabelEncoder使用方法

导入 LabelEncoder 方法

```
from sklearn.preprocessing import LabelEncoder
```

创建数据

```
data = ['paris', 'paris', 'tokyo', 'amsterdam']
```

初始化 LabelEncoder

```
le = LabelEncoder()
```

拟合模型

```
encoded_data = le.fit_transform(data)
```

查看结果

```
print(encoded_data)
```

```
[1 1 2 0]
```

获取类别映射

```
print(le.classes_)
```

```
['amsterdam' 'paris' 'tokyo']
```

逆转换

```
original_data = le.inverse_transform(encoded_data)
print(original_data)
```

```
['paris' 'paris' 'tokyo' 'amsterdam']
```

Sklearn方法介绍

MinMaxScaler ( ) 是 scikit-learn 库中用于数据归一化的工具。它通过将每个特征缩放到指定的范围（通常是 [0, 1]）来转换数据。这种归一化方法对于许多机器学习算法非常重要，因为它可以避免特征值的量纲差异对模型训练造成影响。

工作原理：

MinMaxScaler 将每个特征的值线性地映射到指定的范围 (min, max):

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \times (\max - \min) + \min$$

- X 是原始数据。
- X_{min} 和 X_{max}分别是特征的最小值和最大值。
- min和 max是目标范围的最小值和最大值，默认为 0 和 1。

Sklearn方法介绍

MinMaxScaler ( ) 是 scikit-learn 库中用于数据归一化的工具。它通过将每个特征缩放到指定的范围（通常是 [0, 1]）来转换数据。这种归一化方法对于许多机器学习算法非常重要，因为它可以避免特征值的量纲差异对模型训练造成影响。

参数说明：

- feature_range：指定转换后的数据范围，默认为 (0, 1)。
- copy：布尔值，默认为 True。设置为 False 时，数据会在原地进行归一化，避免复制。
- clip：布尔值，默认为 False。设置为 True 时，转换后的数据会被限制在指定的范围内。

Sklearn方法介绍

MinMaxScaler使用方法

导入 MinMaxScaler 方法

```
from sklearn.preprocessing import MinMaxScaler
```

创建数据

```
import numpy as np

# 示例数据
data = np.array([[4, 2, 3],[1, 5, 6]])
```

初始化 MinMaxScaler

```
# 设置特征范围为 (0, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
```

拟合并转换数据

```
scaled_data = scaler.fit_transform(data)
```

查看结果

```
print(scaled_data)
```

```
[[1. 0. 0.]
 [0. 1. 1.]]
```

Sklearn方法介绍

Kmeans ( ) 是 scikit-learn 库中用于执行 K-means 聚类分析的类。

参数说明：

- **n_clusters**：指定簇的数量，即 K 的值。
- **init**：初始化簇中心的方法。可选值包括：
  - 'k-means++'：智能选择初始簇中心，通常能加速收敛。
  - 'random'：随机选择初始簇中心。
  - 数组：直接提供初始簇中心的坐标。
  - 可调用对象：自定义初始化方法。
- **n_init**：算法运行的次数，每次使用不同的初始簇中心。最终结果取最优的运行结果。默认值为 'auto'，表示根据 init 的值自动选择运行次数。
- **max_iter**：每次运行的最大迭代次数。
- **random_state**：控制随机数生成的种子，以确保结果的可重复性。

Sklearn方法介绍

KMeans使用方法

导入 KMeans 方法

```
from sklearn.cluster import KMeans
```

创建数据

```
import numpy as np

data = np.array([[1, 2], [1, 4], [1, 0],
                 [10, 2], [10, 4], [10, 0]])
```

初始化 KMeans

```
kmeans = KMeans(n_clusters=2, max_iter=300, n_init=10, random_state=0)
```

拟合模型

```
kmeans.fit(data)
```

查看结果

```
print(kmeans.labels_)  # 输出每个数据点所属的簇标签
print(kmeans.cluster_centers_)  # 输出簇中心坐标
```

```
[1 1 1 0 0 0]
[[10.  2.]
 [ 1.  2.]]
```

K-means实现

导入数据

下面我们基于Sklearn利用一个泰国Facebook直播卖家的数据集为例子，来实现K-means分析：

```
import pandas as pd

#读取下载好的数据集
data = 'Live.csv'
df = pd.read_csv(data)

# 查看数据集
print(df)
```

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	0	0	111
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	0	0	204
...	...	...	...	...	...	...	...
7045	1050855161656896_1061863470556065	photo	9/24/2016 2:58	89	0	0	89
7046	1050855161656896_1061334757275603	photo	9/23/2016 11:19	16	0	0	14
7047	1050855161656896_1060126464063099	photo	9/21/2016 23:03	2	0	0	1
7048	1050855161656896_1058663487542730	photo	9/20/2016 0:43	351	12	22	349
7049	1050855161656896_1050858841656528	photo	9/10/2016 10:30	17	0	0	17

7050 rows × 16 columns

我们可以看到数据集中有 7050 个实例和 16 个属性，其中有四个额外属性

数据下载地址：https://www.kaggle.com/datasets/ashishg21/facebook-live-sellers-in-thailand-uci-ml-repo

K-means实现

探索性数据分析

```
# 查看df的基本信息
print(df.info())

# 删除额外属性
df.drop(['Column1', 'Column2', 'Column3',
'Column4'], axis=1, inplace=True)

# 再次查看df的信息
print(df.info())
```

现在，我们可以看到冗余列已从数据集中删除。

删除前的df

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   status_id         7050 non-null   object
 1   status_type       7050 non-null   object
 2   status_published  7050 non-null   object
 3   num_reactions     7050 non-null   int64
 4   num_comments      7050 non-null   int64
 5   num_shares        7050 non-null   int64
 6   num_likes         7050 non-null   int64
 7   num_loves         7050 non-null   int64
 8   num_wows          7050 non-null   int64
 9   num_hahas         7050 non-null   int64
 10  num_sads          7050 non-null   int64
 11  num_angrys        7050 non-null   int64
 12  Column1           0 non-null      float64
 13  Column2           0 non-null      float64
 14  Column3           0 non-null      float64
 15  Column4           0 non-null      float64
dtypes: float64(4), int64(9), object(3)
memory usage: 881.4+ KB
None
```

删除后的df

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   status_id         7050 non-null   object
 1   status_type       7050 non-null   object
 2   status_published  7050 non-null   object
 3   num_reactions     7050 non-null   int64
 4   num_comments      7050 non-null   int64
 5   num_shares        7050 non-null   int64
 6   num_likes         7050 non-null   int64
 7   num_loves         7050 non-null   int64
 8   num_wows          7050 non-null   int64
 9   num_hahas         7050 non-null   int64
 10  num_sads          7050 non-null   int64
 11  num_angrys        7050 non-null   int64
dtypes: int64(9), object(3)
memory usage: 661.1+ KB
None
```

我们可以看到，有 3 个字符变量（数据类型 为object）和剩余的 9 个数值变量（数据类型为int64）。

K-means实现

探索性数据分析

数据集中有 3 个分类变量。我们将逐一探讨它们。

```
# 查看变量中的标签
print(df['status_id'].unique())

# 查看有多少种不同类型的变量
print(len(df['status_id'].unique()))
```

```
['246675545449582_1649696485147474' '246675545449582_1649426988507757'
 '246675545449582_1648730588577397' ...
 '1050855161656896_1060126464063099' '1050855161656896_1058663487542730'
 '1050855161656896_1050858841656528']
6997
```

```
# 查看变量中的标签
print(df['status_published'].unique())

# 查看有多少种不同类型的变量
print(len(df['status_published'].unique()))
```

```
['4/22/2018 6:00' '4/21/2018 22:45' '4/21/2018 6:17' ... '9/21/2016 23:03'
 '9/20/2016 0:43' '9/10/2016 10:30']
6913
```

```
# 查看变量中的标签
print(df['status_type'].unique())

# 查看有多少种不同类型的变量
print(len(df['status_type'].unique()))
```

```
['video' 'photo' 'link' 'status']
4
```

K-means实现

探索性数据分析

我们可以看到 status_id 变量中有 6997 个唯一标签而 status_published 变量中有 6913 个唯一标签。数据集中的实例总数为 7050。因此，它们大约是每个实例的唯一标识符。因此，这不是我们可以使用的变量。因此，我将删除它们。

```
# 删除status_id，status_published列
df.drop(['status_id', 'status_published'], axis=1, inplace=True)

# 输出df
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   status_type     7050 non-null   object
 1   num_reactions   7050 non-null   int64
 2   num_comments    7050 non-null   int64
 3   num_shares      7050 non-null   int64
 4   num_likes       7050 non-null   int64
 5   num_loves       7050 non-null   int64
 6   num_wows        7050 non-null   int64
 7   num_hahas       7050 non-null   int64
 8   num_sads        7050 non-null   int64
 9   num_angrys      7050 non-null   int64
dtypes: int64(9), object(1)
memory usage: 550.9+ KB
None
```

K-means实现

将分类变量转化为整数

```
from sklearn.preprocessing import LabelEncoder

# 创建 LabelEncoder 对象
le = LabelEncoder()

# 将status_type进行编码，转变为数值型，用于后续聚类
df['status_type'] = le.fit_transform(df['status_type'])

# 输出X
print(df.head())
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	3	529	512	262	432	92	3	1	1	0
1	1	150	0	0	150	0	0	0	0	0
2	3	227	236	57	204	21	1	1	0	0
3	1	111	0	0	111	0	0	0	0	0
4	1	213	0	0	204	9	0	0	0	0

K-means实现

特征缩放

```
from sklearn.preprocessing import MinMaxScaler

# 保留标题
cols = X.columns

# 创建 MinMaxScaler 对象
ms = MinMaxScaler()

# 归一化 X
X = ms.fit_transform(X)

# 将 X 转为数据框
X = pd.DataFrame(X, columns=[cols])

# 查看 X
print(X.head())
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	1.000000	0.112314	0.024393	0.076519	0.091720	0.140030	0.010791	0.006369	0.019608	0.0
1	0.333333	0.031847	0.000000	0.000000	0.031847	0.000000	0.000000	0.000000	0.000000	0.0
2	1.000000	0.048195	0.011243	0.016647	0.043312	0.031963	0.003597	0.006369	0.000000	0.0
3	0.333333	0.023567	0.000000	0.000000	0.023567	0.000000	0.000000	0.000000	0.000000	0.0
4	0.333333	0.045223	0.000000	0.000000	0.043312	0.013699	0.000000	0.000000	0.000000	0.0

K-means实现

具有两个簇的kmeans模型

```
from sklearn.cluster import KMeans

# 创建聚类数为2的 KMeans 对象
kmeans = KMeans(n_clusters=2, random_state=0)

# 对 X 聚类
kmeans.fit(X)

# 输出簇中心坐标
print(kmeans.cluster_centers_)

# 输出内部误差平方和
print(kmeans.inertia_)
```

K-means实现

具有两个簇的kmeans模型

cluster_centers_：

该属性返回一个形状为 (n_clusters, n_features) 的 NumPy 数组，其中 n_clusters 是簇的数量，n_features 是每个数据点的特征数量。每一行表示一个簇的中心坐标，即该簇内所有数据点的均值。

```
print(kmeans.cluster_centers_)
```

```
[[3.28506857e-01 3.90710874e-02 7.54854864e-04 7.53667113e-04
  3.85438884e-02 2.17448568e-03 2.43721364e-03 1.20039760e-03
  2.75348016e-03 1.45313276e-03]
 [9.54921576e-01 6.46330441e-02 2.67028654e-02 2.93171709e-02
  5.71231462e-02 4.71007076e-02 8.18581889e-03 9.65207685e-03
  8.04219428e-03 7.19501847e-03]]
```

inertia_：

该属性表示所有样本到其最近簇中心的距离的平方和，即聚类的“惯性”或“内部误差平方和”。惯性越小，表示簇内的数据点越紧密，聚类效果越好。

```
print(kmeans.inertia_)
```

```
237.7572640441955
```

K-means实现

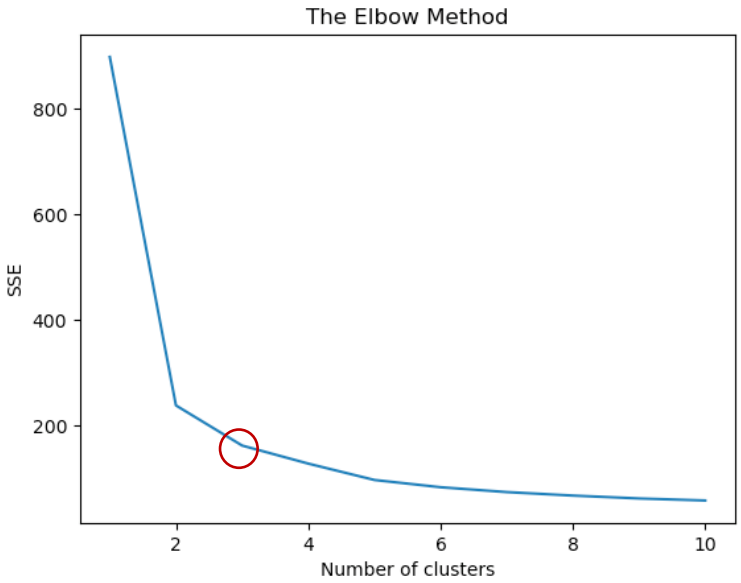
使用肘部法寻找最佳聚类数

```
from sklearn.cluster import KMeans

# 创建误差数组
sse = []

# 记录不同聚类簇数下的误差
for i in range(1, 11):
kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init =
10, random_state = 0)
kmeans.fit(df)
sse.append(kmeans.inertia_)

# 绘制误差图
import matplotlib.pyplot as plt
plt.plot(range(1, 11), sse)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```



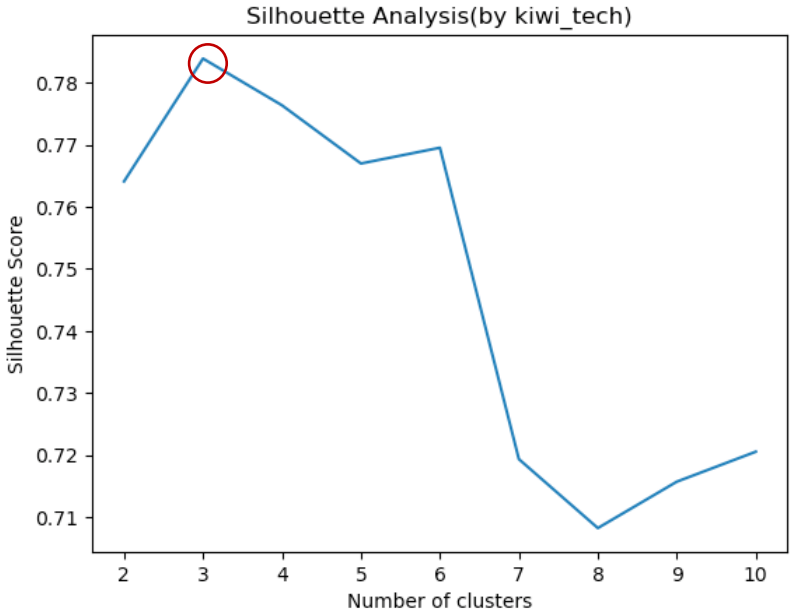
K-means实现

使用轮廓分析法寻找最佳聚类数

```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

# 使用轮廓分析法找到最佳的集群数
silhouette_scores = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300,
    n_init=10, random_state=0)
    kmeans.fit(df)
    silhouette_scores.append(silhouette_score(df, kmeans.labels_))

# 绘制轮廓分析法图
plt.plot(range(2, 11), silhouette_scores)
plt.title('Silhouette Analysis(by kiwi_tech)')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



K-means实现

具有3个簇的kmeans模型

```
from sklearn.cluster import KMeans

# 创建聚类数为3的 KMeans 对象
kmeans = KMeans(n_clusters=2, random_state=0)

# 对 X 聚类
kmeans.fit(X)

# 输出簇中心坐标
print(kmeans.cluster_centers_)

# 输出内部误差平方和
print(kmeans.inertia_)
```

```
[[3.28742853e-01 1.99588196e-02 6.50282622e-04 5.37894046e-04
  1.94880247e-02 1.93982105e-03 2.03104006e-03 1.16647149e-03
  2.84240297e-03 1.51976868e-03]
 [9.63364293e-01 5.06532935e-02 2.77360441e-02 3.04804142e-02
  4.28478700e-02 4.90844018e-02 8.17287347e-03 1.00742442e-02
  8.36216840e-03 7.50274643e-03]
 [4.79102956e-01 4.00444107e-01 3.03960092e-03 4.33797051e-03
  3.99082569e-01 5.26906195e-03 9.70232988e-03 1.26609400e-03
  9.59405169e-04 1.97297031e-04]]
161.59633400033613
```

Q & A