

姓名：曾闻天 学号：231880147 2025 年 12 月 21 日

一. (12 分) 朴素贝叶斯分类器

一家电商公司希望通过用户评论的关键词来预测评论的情感（正面或 负面）。假设已经收集了一小部分用户评论，并从中提取出以下四个关键词作为特征：“good”、“bad”、“quality”和“price”。每条评论可以被标记为“正面”或“负面”。

| 评论情感 | good 出现次数 | bad 出现次数 | quality 出现次数 | price 出现次数 |
|------|-----------|----------|--------------|------------|
| 正面评论 | 50 | 5 | 45 | 20 |
| 负面评论 | 10 | 25 | 5 | 25 |

假设正面评论和负面评论的先验概率分别为 $P(\text{正面评论}) = 0.7$ 和 $P(\text{负面评论}) = 0.3$ 。

问题

1. (5 分) 基于上述数据，使用拉普拉斯修正 ($\alpha = 1$) 计算以下条件概率：

- $P(\text{good}|\text{正面评论})$
- $P(\text{bad}|\text{正面评论})$
- $P(\text{quality}|\text{正面评论})$
- $P(\text{price}|\text{正面评论})$

同时，计算上述特征在负面评论下的条件概率。

2. (7 分) 假设我们有一条新评论 $X = \{\text{good}, \text{quality}, \text{price}\}$ ，请使用朴素贝叶斯分类器计算该评论属于正面评论和负面评论的后验概率 $P(\text{正面评论}|X)$ 和 $P(\text{负面评论}|X)$ ，并根据计算结果确定该评论的情感类别。

注：

- 本题的答案请以四舍五入后的小数点后两位的形式给出，比如 $P=0.67$ 。如果给出答案为分式，会扣分。
- 本题要平滑的属性为四个关键词，即 $\hat{P}_{\text{good}|\text{正面评论}}$ 。

解：

二. (20 分) EM 算法

假设有一个包含 6 次硬币抛掷结果的数据集，记录了每次抛掷是否得到“正面”：

$$X = \{\text{正面, 反面, 反面, 正面, 正面, 反面}\}$$

假设这些结果是由两枚硬币 A 和 B 生成的，每次抛掷时选择使用硬币 A 或 B 的概率均为 0.5。然而，具体每次抛掷使用的是哪一枚硬币是未知的。硬币 A 和 B 的正面概率分别为 θ_A 和 θ_B 。我们的目标是通过 EM 算法估计这两枚硬币的正面概率 θ_A 和 θ_B 。

已知：1. 初始参数：硬币 A 的正面概率 $\theta_A^{(0)} = 0.6$ 和硬币 B 的正面概率 $\theta_B^{(0)} = 0.5$ 。2. 每次抛掷使用硬币 A 和硬币 B 的概率均为 0.5，即 $P(A) = 0.5$ 和 $P(B) = 0.5$ 。

请通过一轮 EM 算法的迭代步骤，估计硬币 A 和 B 的正面概率 θ_A 和 θ_B 。本题的答案请以分式或者小数点后两位的形式给出，比如 $P=0.67$ 。

问题：

1. **E 步** (10 分)：对于每一次抛掷结果，使用当前的参数估计值 ($\theta_A^{(0)}$ 和 $\theta_B^{(0)}$)，计算该结果由硬币 A 和硬币 B 生成的后验概率，即每次抛掷属于硬币 A 和硬币 B 的“软分配”概率。

请计算以下内容：

- 在第 1 次到第 6 次抛掷中，每个结果（正面或反面）由硬币 A 生成的概率 $P(A|x_i)$ 。
- 每个结果由硬币 B 生成的概率 $P(B|x_i)$ 。

2. **M 步** (10 分)：基于 E 步计算出的“软分配”概率，计算硬币 A 和 B 的正面和反面出现的期望次数，并更新硬币 A 和 B 的正面概率 θ_A 和 θ_B 。

请计算以下内容：

- 硬币 A 的正面和反面期望出现次数，并据此更新硬币 A 的正面概率 $\theta_A^{(1)}$ 。
- 硬币 B 的正面和反面期望出现次数，并据此更新硬币 B 的正面概率 $\theta_B^{(1)}$ 。

解：

E 步

已知先验：

$$P(A) = 0.5, \quad P(B) = 0.5$$

硬币的正面概率初始值:

$$\theta_A^{(0)} = 0.6, \quad \theta_B^{(0)} = 0.5$$

对于第 i 次抛掷, $P(A|x_i)$ 计算如下:

$$P(A|x_i) = \frac{P(A)P(x_i|A)}{P(A)P(x_i|A) + P(B)P(x_i|B)}$$

- 第 1 次抛掷 (正面):

$$P(A|\text{正面}) = \frac{0.5 \times 0.6}{0.5 \times 0.6 + 0.5 \times 0.5} = \frac{0.3}{0.55} \approx 0.5455$$

$$P(B|\text{正面}) = 1 - 0.5455 \approx 0.4545$$

- 第 2 次抛掷 (反面):

$$P(A|\text{反面}) = \frac{0.5 \times 0.4}{0.5 \times 0.4 + 0.5 \times 0.5} = \frac{0.2}{0.45} \approx 0.4444$$

$$P(B|\text{反面}) = 1 - 0.4444 \approx 0.5556$$

- 第 3 次抛掷 (反面): 同第 2 次

$$P(A|\text{反面}) \approx 0.4444, \quad P(B|\text{反面}) \approx 0.5556$$

- 第 4 次抛掷 (正面): 同第 1 次

$$P(A|\text{正面}) \approx 0.5455, \quad P(B|\text{正面}) \approx 0.4545$$

- 第 5 次抛掷 (正面): 同第 1 次

$$P(A|\text{正面}) \approx 0.5455, \quad P(B|\text{正面}) \approx 0.4545$$

- 第 6 次抛掷 (反面): 同第 2 次

$$P(A|\text{反面}) \approx 0.4444, \quad P(B|\text{反面}) \approx 0.5556$$

E 步汇总表:

| 抛掷序号 | 结果 | $P(A x_i)$ | $P(B x_i)$ |
|------|----|------------|------------|
| 1 | 正面 | 0.5455 | 0.4545 |
| 2 | 反面 | 0.4444 | 0.5556 |
| 3 | 反面 | 0.4444 | 0.5556 |
| 4 | 正面 | 0.5455 | 0.4545 |
| 5 | 正面 | 0.5455 | 0.4545 |
| 6 | 反面 | 0.4444 | 0.5556 |

M 步

硬币 A

$$\text{正面期望次数} = 0.5455 + 0.5455 + 0.5455 = 1.6365$$

$$\text{反面期望次数} = 0.4444 + 0.4444 + 0.4444 = 1.3332$$

更新 $\theta_A^{(1)}$:

$$\theta_A^{(1)} = \frac{1.6365}{1.6365 + 1.3332} = \frac{1.6365}{2.9697} \approx 0.5510$$

硬币 B

$$\text{正面期望次数} = 0.4545 + 0.4545 + 0.4545 = 1.3635$$

$$\text{反面期望次数} = 0.5556 + 0.5556 + 0.5556 = 1.6668$$

更新 $\theta_B^{(1)}$:

$$\theta_B^{(1)} = \frac{1.3635}{1.3635 + 1.6668} = \frac{1.3635}{3.0303} \approx 0.4500$$

结果

经过一轮 EM 算法迭代后:

$$\theta_A^{(1)} \approx 0.55, \quad \theta_B^{(1)} \approx 0.45$$

三. (28 分) 集成学习

集成学习是一种通用技术，通过将多个不同模型的预测结果结合起来，以平均值或多数投票的方式生成单一预测，从而有效应对过拟合问题。

1. (3 分) 考虑一组互不相关的随机变量 $\{Y_i\}_{i=1}^n$ ，其均值为 μ ，方差为 σ^2 。请计算这些随机变量平均值的期望和方差，给出计算过程。提示：在集成方法的背景下，这些 Y_i 类似于分类器 i 所作的预测。

2. 在第 1 小问中，我们看到对于不相关的分类器，取平均可以有效减少方差。尽管现实中的预测不可能完全不相关，但降低决策树之间的相关性通常能减少最终方差。现在，重新考虑一组具有相关性的随机变量 $\{Z_i\}_{i=1}^n$ ，其均值为 μ ，方差为 σ^2 ，每个 $Z_i \in \mathbb{R}$ 为标量。假设对任意 $i \neq j$ ， $\text{Corr}(Z_i, Z_j) = \rho$ 。提示：如果你不记得相关性与协方差之间的关系，请回顾你的概率论等课程内容。

- (5 分) 请计算随机变量 Z_i 平均值的方差，以 σ 、 ρ 和 n 为变量表示，给出计算过程。
- (6 分) 当 n 非常大时，会发生什么？这对于取平均的潜在有效性说明了什么？（……如果 ρ 很大 ($|\rho| \approx 1$) 会怎样？……如果 ρ 非常小 ($|\rho| \approx 0$) 又会怎样？……如果 ρ 处于中等水平 ($|\rho| \approx 0.5$) 呢?) 无需严格推导——基于你得出的方差公式，用定性分析进行讨论即可。

3. Bagging 是一种通过随机化从同一数据集生成多个不同学习器的方法。给定一个大小为 n 的训练集，Bagging 会通过有放回抽样生成 T 个随机子样本集，每个子样本集大小为 n' 。在每个子样本集中，一些数据点可能会被多次选中，而另一些可能完全未被选中。当 $n' = n$ 时，大约 63% 的数据点会被选中，而剩下的 37% 被称为袋外样本点 (Out-of-Bag, OOB)。

- (7 分) 为什么是 63%？提示：当 n 很大时，某个样本点未被选中的概率是多少？请只考虑在任意一个子样本集中（而不是所有 T 个子样本集）未被选中的概率。
- (7 分) 关于决策树的数量 T 。集成中的决策树数量通常需要在运行时间和降低方差之间进行权衡（典型值范围从几十到几千棵树）。而子样本大小 n' 对运行时间的影响较小，因此选择 n' 时主要考虑如何获得最优预测结果。虽然常见实践是设置 $n' = n$ ，但这并非总是最佳选择。你会如何建议我们选择超参数 n' ？

解:

1.

设 $\{Y_i\}_{i=1}^n$ 互不相关， $\mathbb{E}[Y_i] = \mu$ ， $\text{Var}(Y_i) = \sigma^2$ 。

定义平均值为：

$$\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$$

期望:

$$\mathbb{E}[\bar{Y}] = \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^n Y_i\right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}[Y_i] = \frac{1}{n} \sum_{i=1}^n \mu = \mu$$

方差: 由于不相关, 协方差 $\text{Cov}(Y_i, Y_j) = 0$ 对 $i \neq j$ 。

$$\text{Var}(\bar{Y}) = \text{Var}\left(\frac{1}{n} \sum_{i=1}^n Y_i\right) = \frac{1}{n^2} \sum_{i=1}^n \text{Var}(Y_i) = \frac{1}{n^2} \cdot n\sigma^2 = \frac{\sigma^2}{n}$$

因此:

$$\mathbb{E}[\bar{Y}] = \mu, \quad \text{Var}(\bar{Y}) = \frac{\sigma^2}{n}$$

2.

设 $\{Z_i\}_{i=1}^n$, $\mathbb{E}[Z_i] = \mu$, $\text{Var}(Z_i) = \sigma^2$, 且 $\forall i \neq j$, $\text{Corr}(Z_i, Z_j) = \rho$ 。

已知相关系数与协方差关系: $\text{Cov}(Z_i, Z_j) = \rho\sigma^2$ 。

平均值方差

$$\text{Var}(\bar{Z}) = \text{Var}\left(\frac{1}{n} \sum_{i=1}^n Z_i\right) = \frac{1}{n^2} \text{Var}\left(\sum_{i=1}^n Z_i\right)$$

其中:

$$\begin{aligned} \text{Var}\left(\sum_{i=1}^n Z_i\right) &= \sum_{i=1}^n \text{Var}(Z_i) + \sum_{i \neq j} \text{Cov}(Z_i, Z_j) \\ &= n\sigma^2 + n(n-1)\rho\sigma^2 = \sigma^2 [n + n(n-1)\rho] \end{aligned}$$

故:

$$\begin{aligned} \text{Var}(\bar{Z}) &= \frac{\sigma^2}{n^2} [n + n(n-1)\rho] = \frac{\sigma^2}{n} [1 + (n-1)\rho] \\ \text{Var}(\bar{Z}) &= \sigma^2 \cdot \frac{1 + (n-1)\rho}{n} \end{aligned}$$

当 n 很大时的定性分析

$$\lim_{n \rightarrow \infty} \text{Var}(\bar{Z}) = \sigma^2 \cdot \rho$$

- 若 $\rho \approx 1$ (强正相关): $\text{Var}(\bar{Z}) \approx \sigma^2$ 。取平均几乎不能降低方差, 因为所有分类器几乎总是一致出错。
- 若 $\rho \approx 0$ (弱相关或不相关): $\text{Var}(\bar{Z}) \approx \frac{\sigma^2}{n}$ 。取平均能显著降低方差, 如第 1 问结论。
- 若 $\rho \approx 0.5$ (中等相关): $\text{Var}(\bar{Z}) \approx 0.5\sigma^2$ (当 n 很大时)。方差降低效果有限, 但仍有一定改进。

这说明, 在集成学习中, 降低基分类器之间的相关性对减少整体方差至关重要。Bagging 等方法通过引入随机性 (如自助采样、特征扰动) 来降低 ρ , 从而提升集成效果。

3.

为什么是 63% ?

在自助采样 (bootstrap sampling) 中, 从 n 个样本中有放回地抽取 n 次。

对于任意一个固定的样本点, 在某一次抽取中**未被选中**的概率为:

$$1 - \frac{1}{n}$$

在 n 次抽取中都未被选中的概率为:

$$\left(1 - \frac{1}{n}\right)^n$$

当 n 较大时, 该概率近似为:

$$\lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right)^n = e^{-1} \approx 0.3679$$

因此, 一个样本点**被选中至少一次**的概率为:

$$1 - e^{-1} \approx 1 - 0.3679 = 0.6321 \approx 63\%$$

如何选择超参数 n' ?

常见做法是 $n' = n$, 但不一定最优。

- 对于 **高方差、低偏差**的模型 (如深度决策树), 选择 $n' < n$ 可进一步增加基学习器的多样性, 降低相关性 ρ , 从而可能获得更低的集成方差。但 n' 太小会导致基学习器偏差增大, 需在偏差-方差权衡中折衷。

- 对于 **高偏差、低方差**的模型（如浅层决策树）， n' 接近 n 有助于保持低偏差，同时集成仍能带来一定方差下降。
- 实际中，可通过交叉验证或袋外误差估计（若 $n' < n$ ，每个样本可能出现在多个袋外集中）来调优 n' 。
- 另一种启发式：对于分类问题， $n' = 0.8n$ 或 $n' = 0.9n$ 有时优于 $n' = n$ 。

n' 的选择应在保证基学习器足够准确（偏差不过大）的前提下，通过减少样本重叠来降低学习器间的相关性，从而最大化集成效果。

四. (20 分) 聚类理论

聚类是一种无监督学习任务，其核心是根据数据样本之间的相似性（通常由距离度量定义）将数据分成若干个簇。常用聚类算法如 k -均值和 DBSCAN 都需要特定的距离度量和超参数设置。以下问题围绕距离度量、目标函数、以及超参数设置展开：

1. (5 分) 在聚类算法中，距离度量是衡量样本间相似性的基础，选择合适的距离度量对聚类效果有显著影响。给定欧几里得距离、曼哈顿距离和余弦相似度，在以下场景中，分析哪种距离更适合使用，并简要说明原因：

- 场景 1：高维稀疏特征向量（如文本数据的 TF-IDF 表示）
- 场景 2：二维几何分布数据（如图像中的空间点分布）

2. (15 分) k -均值聚类的目标函数与迭代过程, k -均值聚类的目标是最小化以下目标函数：

$$J = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2$$

其中， C_i 表示第 i 个聚类簇， μ_i 为第 i 个簇的中心。

(a) 推导在分配样本点到最近的簇中心时，为什么目标函数 J 会减少。

(b) 推导为什么更新簇中心为簇内样本点的均值时，目标函数 J 会减少。

(c) k -均值的超参数 k 对结果有何影响？

- 如果 k 设置过大或过小，分别可能会导致什么问题？
- 提出一种确定 k 的方法，并解释其原理。

解：

1.

- 场景 1：高维稀疏特征向量（如文本数据的 TF-IDF 表示）

更适合使用余弦相似度。

原因：在高维稀疏向量中，欧几里得距离和曼哈顿距离容易受到维数灾难的影响，且对向量长度的差异敏感（TF-IDF 中长文档与短文档的向量长度差异大）。余弦相似度衡量的是向量方向的相似性，忽略长度差异，更适合衡量文本主题的相似性。

- 场景 2：二维几何分布数据（如图像中的空间点分布）

更适合使用欧几里得距离。

原因：在二维或三维欧几里得空间中，欧几里得距离直接对应“直线距离”，符合人类的几何

直观。曼哈顿距离（城市街区距离）虽然也可用，但欧几里得距离更自然地反映空间点之间的实际接近程度。

2.

给定目标函数：

$$J = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2$$

其中 C_i 为簇， μ_i 为簇中心。

(a)

假设当前簇中心 μ_i 固定。对每个样本点 x ，将其重新分配到距离最近的簇中心所属的簇，即：

$$\text{新簇} = \arg \min_i \|x - \mu_i\|^2$$

由于将 x 分配到使其距离平方最小的簇，相比于原来的分配，该样本对 J 的贡献值不会增加，只会减少或不变。所有样本点依次进行这样的重新分配，整个目标函数 J 单调不增。严格来说，若至少有一个样本被重新分配到一个更近的中心，则 J 严格减少。

(b)

固定簇 C_i 的成员，寻找最优的 μ_i 使得 $\sum_{x \in C_i} \|x - \mu_i\|^2$ 最小。对 μ_i 求导：

$$\nabla_{\mu_i} \sum_{x \in C_i} \|x - \mu_i\|^2 = -2 \sum_{x \in C_i} (x - \mu_i)$$

令导数为零：

$$\sum_{x \in C_i} (x - \mu_i) = 0 \quad \Rightarrow \quad \mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

即簇内样本的均值。该解是凸二次函数的最小值点，因此更新 μ_i 为该均值必然使该簇内部的平方距离和最小化，从而降低 J 。

(c)

- k 过小：会导致簇内方差过大，样本被过度压缩到少数簇中，无法捕捉数据中的细粒度结构，欠拟合。
- k 过大：会导致每个簇样本数很少，甚至出现单样本簇，模型过于复杂，容易受到噪声影响，过拟合，且失去聚类意义的概括性。

k 的方法：肘部法则 (Elbow Method)

原理：对不同的 k 值运行 k -means，计算目标函数 J （或簇内平方误差和）。随着 k 增大， J 单调递减，但下降幅度会逐渐变小。绘制 J 关于 k 的曲线，选择曲线拐点（形如“肘部”）对应的 k 值。该点意味着再增加 k 带来的误差减小不再明显，是聚类数目与模型简洁性的平衡点。

五. (20 分) 聚类实战

使用商场客户数据集 (Mall Customer Dataset) 完成客户分群分析。该数据集包含客户的年龄 (Age)、年收入 (Annual Income) 和消费积分 (Spending Score) 等特征。你需要通过实现和优化聚类算法来完成客户画像分析。数据随作业提供。

为方便起见，每小題我们提供了部分初始代码，其中包括预处理步骤和部分功能的实现。你可以自由选择使用或不使用这些代码来完成实现。

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import StandardScaler
4
5 # 加载数据
6 def load_mall_data():
7     df = pd.read_csv("Mall_Customers.csv")
8     X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
9     # 数据标准化
10    scaler = StandardScaler()
11    X_scaled = scaler.fit_transform(X)
12    return X_scaled, X, df
```

Listing 1: 数据加载

1. 在不借助外部实现的情况下，手动实现 KMeans 方法 (5 分)，在数据集上进行聚类，可视化聚类结果 (2 分)，并解决下列问题 (3 分)：

- 如何使用肘部法则确定合适的 k 值，绘图说明
- 简单分析每个客户群的特征
- 计算和分析簇内平方和 (inertia)

```
1 class KMeans:
2     def __init__(self, n_clusters=3, max_iters=100):
3         self.n_clusters = n_clusters
4         self.max_iters = max_iters
5         self.centroids = None
6         self.labels = None
7         self.inertia_ = None # 簇内平方和
8
9     def fit(self, X):
10        """
11        实现K-means算法
12        参数:
13            X: shape (n_samples, n_features)
14        返回:
```

```
15         self
16         """
17         # TODO: 实现K-means算法
18         # 1. 随机初始化聚类中心
19         # 2. 迭代优化直到收敛
20         pass
21
22     def predict(self, X):
23         """返回每个样本的聚类标签"""
24         pass
25
26 def plot_clusters(X, labels, centroids=None):
27     """绘制聚类结果的散点图"""
28     pass
```

Listing 2: Kmeans 部分实现

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import StandardScaler
4 import matplotlib.pyplot as plt
5 from mpl_toolkits.mplot3d import Axes3D
6
7 import matplotlib
8 matplotlib.rcParams['font.sans-serif'] = ['SimHei', 'Arial Unicode MS', 'DejaVu Sans']
9 matplotlib.rcParams['axes.unicode_minus'] = False
10
11 # 加载数据
12 def load_mall_data():
13     df = pd.read_csv("Mall_Customers.csv")
14     X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
15     # 数据标准化
16     scaler = StandardScaler()
17     X_scaled = scaler.fit_transform(X)
18     return X_scaled, X, df
19
20 class KMeans:
21     def __init__(self, n_clusters=3, max_iters=100, random_state=None):
22         self.n_clusters = n_clusters
23         self.max_iters = max_iters
24         self.centroids = None
25         self.labels = None
26         self.inertia_ = None # 簇内平方和
27         self.random_state = random_state
28
29     def _initialize_centroids(self, X):
30         """随机初始化聚类中心"""
```

```
31     np.random.seed(self.random_state)
32     n_samples = X.shape[0]
33     # 随机选择k个样本作为初始聚类中心
34     indices = np.random.choice(n_samples, self.n_clusters, replace=False)
35     return X[indices]
36
37     def _compute_distance(self, X, centroids):
38         """计算每个样本到各个聚类中心的距离"""
39         distances = np.zeros((X.shape[0], self.n_clusters))
40         for i in range(self.n_clusters):
41             distances[:, i] = np.sqrt(np.sum((X - centroids[i]) ** 2, axis=1))
42         return distances
43
44     def _assign_clusters(self, distances):
45         """分配每个样本到最近的聚类中心"""
46         return np.argmin(distances, axis=1)
47
48     def _update_centroids(self, X, labels):
49         """更新聚类中心为簇内样本的均值"""
50         new_centroids = np.zeros((self.n_clusters, X.shape[1]))
51         for i in range(self.n_clusters):
52             cluster_points = X[labels == i]
53             if len(cluster_points) > 0:
54                 new_centroids[i] = np.mean(cluster_points, axis=0)
55             else:
56                 # 如果簇为空，随机重新初始化该聚类中心
57                 new_centroids[i] = X[np.random.randint(0, X.shape[0])]
58         return new_centroids
59
60     def fit(self, X):
61         """
62         实现K-means算法
63         参数:
64             X: shape (n_samples, n_features)
65         返回:
66             self
67         """
68         # 1. 随机初始化聚类中心
69         self.centroids = self._initialize_centroids(X)
70
71         # 2. 迭代优化直到收敛
72         for iteration in range(self.max_iters):
73             # 计算距离并分配标签
74             distances = self._compute_distance(X, self.centroids)
75             labels = self._assign_clusters(distances)
76
77             # 更新聚类中心
```

```
78         new_centroids = self._update_centroids(X, labels)
79
80         # 检查是否收敛 (聚类中心不再变化)
81         if np.allclose(self.centroids, new_centroids):
82             print(f"KMeans 收敛于第 {iteration+1} 次迭代")
83             break
84
85         self.centroids = new_centroids
86
87         # 保存最终结果
88         distances = self._compute_distance(X, self.centroids)
89         self.labels = self._assign_clusters(distances)
90
91         # 计算簇内平方和
92         self.inertia_ = 0
93         for i in range(self.n_clusters):
94             cluster_points = X[self.labels == i]
95             if len(cluster_points) > 0:
96                 self.inertia_ += np.sum(np.sum((cluster_points - self.centroids[i]) ** 2,
97 axis=1))
98
99         return self
100
101     def predict(self, X):
102         """返回每个样本的聚类标签"""
103         distances = self._compute_distance(X, self.centroids)
104         return self._assign_clusters(distances)
105
106     def fit_predict(self, X):
107         """拟合数据并返回标签"""
108         self.fit(X)
109         return self.labels
110
111 def plot_clusters_3d(X, labels, centroids=None, title="KMeans 聚类结果"):
112     """绘制聚类结果的3D散点图"""
113     fig = plt.figure(figsize=(12, 8))
114     ax = fig.add_subplot(111, projection='3d')
115
116     # 为每个簇分配颜色
117     unique_labels = np.unique(labels)
118     colors = plt.cm.rainbow(np.linspace(0, 1, len(unique_labels)))
119
120     for label, color in zip(unique_labels, colors):
121         cluster_points = X[labels == label]
122         ax.scatter(cluster_points[:, 0], cluster_points[:, 1], cluster_points[:, 2],
123                    c=[color], label=f'簇 {label}', alpha=0.7, s=50)
```

```
124 # 绘制聚类中心
125 if centroids is not None:
126     ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2],
127               c='black', marker='X', s=200, label='聚类中心', alpha=1.0, linewidths=2)
128
129 ax.set_xlabel('年龄 (标准化)', fontsize=12)
130 ax.set_ylabel('年收入 (标准化)', fontsize=12)
131 ax.set_zlabel('消费积分 (标准化)', fontsize=12)
132 ax.set_title(title, fontsize=14)
133 ax.legend()
134 plt.tight_layout()
135 plt.show()
136
137 def plot_clusters_2d(X, labels, centroids=None, feature_names=None):
138     """绘制聚类结果的2D散点图 (特征两两组合) """
139     fig, axes = plt.subplots(2, 2, figsize=(14, 10))
140
141     if feature_names is None:
142         feature_names = ['特征1', '特征2', '特征3']
143
144     # 为每个簇分配颜色
145     unique_labels = np.unique(labels)
146     colors = plt.cm.rainbow(np.linspace(0, 1, len(unique_labels)))
147
148     # 特征组合: 0-1, 0-2, 1-2, 还有一个子图空着
149     combinations = [(0, 1), (0, 2), (1, 2)]
150
151     for idx, (i, j) in enumerate(combinations):
152         ax = axes[idx // 2, idx % 2]
153
154         for label, color in zip(unique_labels, colors):
155             cluster_points = X[labels == label]
156             ax.scatter(cluster_points[:, i], cluster_points[:, j],
157                       c=[color], label=f'簇 {label}', alpha=0.6, s=40)
158
159         # 绘制聚类中心
160         if centroids is not None:
161             ax.scatter(centroids[:, i], centroids[:, j],
162                       c='black', marker='X', s=150, label='聚类中心', alpha=1.0,
163                       linewidths=2)
164
165         ax.set_xlabel(feature_names[i], fontsize=11)
166         ax.set_ylabel(feature_names[j], fontsize=11)
167         ax.set_title(f'{feature_names[i]} vs {feature_names[j]}', fontsize=12)
168         if idx == 0:
169             ax.legend()
```

```
170     # 移除最后一个空子图
171     axes[1, 1].axis('off')
172
173     plt.suptitle('KMeans 聚类结果可视化', fontsize=16)
174     plt.tight_layout()
175     plt.show()
176
177 # 主程序
178 def main():
179     # 加载数据
180     X_scaled, X_original, df = load_mall_data()
181     feature_names = ['Age', 'Annual Income', 'Spending Score']
182
183     print(f"数据集形状: {X_scaled.shape}")
184     print(f"样本数: {X_scaled.shape[0]}, 特征数: {X_scaled.shape[1]}")
185
186     # 使用肘部法则确定最佳K值
187     print("\n计算不同K值的簇内平方和...")
188     inertias = []
189     K_range = range(1, 11)
190
191     for k in K_range:
192         kmeans = KMeans(n_clusters=k, max_iters=100, random_state=42)
193         kmeans.fit(X_scaled)
194         inertias.append(kmeans.inertia_)
195         print(f"K={k}: 簇内平方和 = {kmeans.inertia_:.2f}")
196
197     # 绘制肘部法则图
198     plt.figure(figsize=(10, 6))
199     plt.plot(K_range, inertias, 'bo-')
200     plt.xlabel('K值', fontsize=12)
201     plt.ylabel('簇内平方和', fontsize=12)
202     plt.title('肘部法则', fontsize=14)
203     plt.xticks(K_range)
204     plt.grid(True, alpha=0.3)
205     plt.show()
206
207     # 选择K=5进行聚类 (根据肘部法则)
208     print("\n使用K=5进行KMeans聚类...")
209     kmeans = KMeans(n_clusters=5, max_iters=100, random_state=42)
210     labels = kmeans.fit_predict(X_scaled)
211
212     print(f"聚类完成! ")
213     print(f"聚类中心形状: {kmeans.centroids.shape}")
214     print(f"最终簇内平方和: {kmeans.inertia_:.2f}")
215
216     # 统计每个簇的样本数
```

```
217 unique_labels, counts = np.unique(labels, return_counts=True)
218 for label, count in zip(unique_labels, counts):
219     print(f"簇 {label}: {count} 个样本 ({count/X_scaled.shape[0]*100:.1f}%)")
220
221 # 3D可视化
222 plot_clusters_3d(X_scaled, labels, kmeans.centroids,
223                  title="客户分群分析 (3D可视化)")
224
225 # 2D可视化
226 plot_clusters_2d(X_scaled, labels, kmeans.centroids, feature_names)
227
228 # 将聚类结果添加到原始数据中
229 df['Cluster'] = labels
230
231 # 分析每个簇的特征
232 print("\n各簇特征统计:")
233 cluster_stats = df.groupby('Cluster')[['Age', 'Annual Income (k$)', 'Spending Score
234 (1-100)']].mean()
235 print(cluster_stats.round(2))
236
237 # 绘制簇特征条形图
238 fig, axes = plt.subplots(1, 3, figsize=(15, 5))
239
240 for idx, feature in enumerate(['Age', 'Annual Income (k$)', 'Spending Score (1-100)']):
241     :
242     axes[idx].bar(cluster_stats.index, cluster_stats[feature])
243     axes[idx].set_xlabel('簇')
244     axes[idx].set_ylabel(feature)
245     axes[idx].set_title(f'各簇平均{feature}')
246     axes[idx].set_xticks(cluster_stats.index)
247
248 plt.suptitle('簇特征分析', fontsize=14)
249 plt.tight_layout()
250 plt.show()
251
252 return kmeans, df
253
254 if __name__ == "__main__":
255     kmeans_model, df_with_clusters = main()
```

Listing 3: 1.

已实现 KMeans 方法

可视化如图

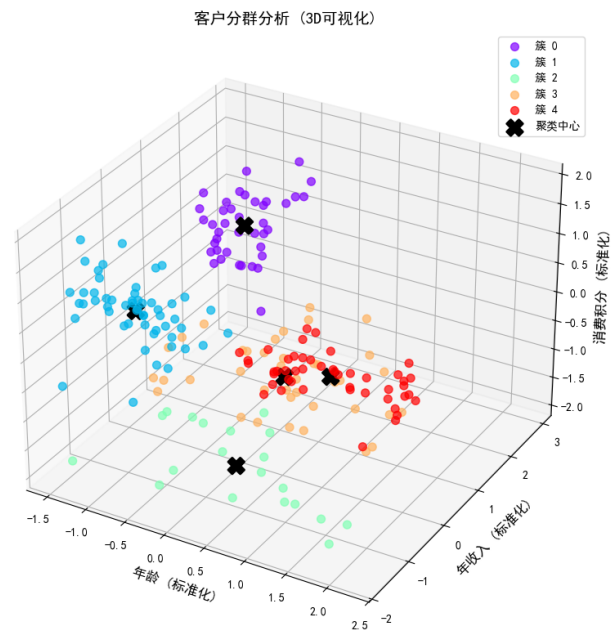


图 1: 3D 聚类分析

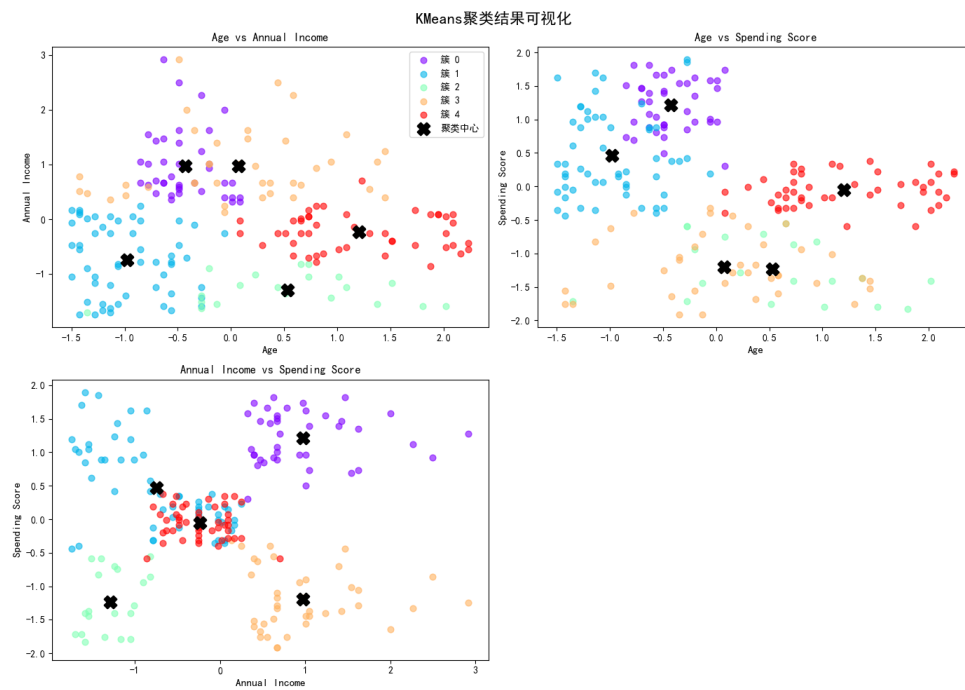


图 2: 2D 聚类分析

肘部法由下图可知 $k=5$

分析如下图

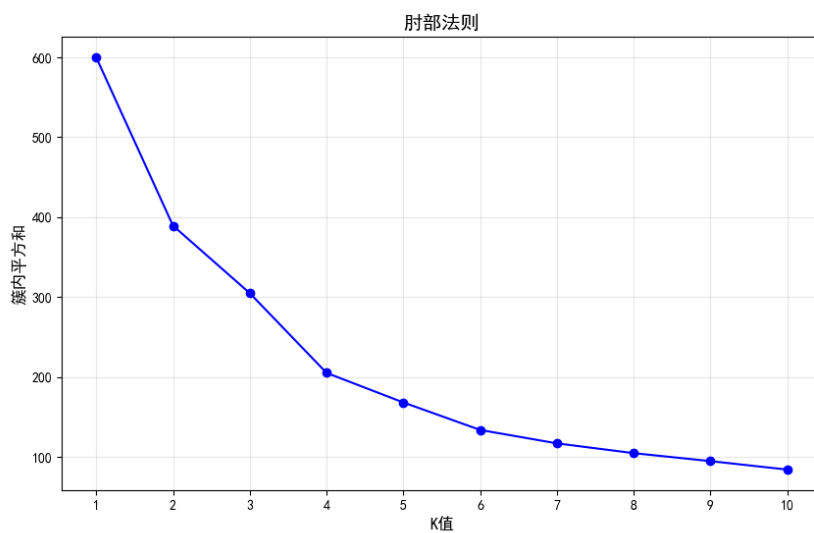


图 3: 肘部法

使用 $K=5$ 进行 KMeans 聚类

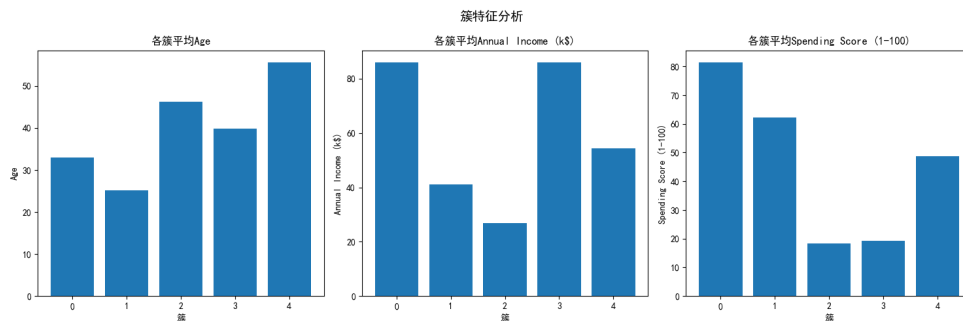


图 4: 簇类分析

KMeans 收敛于第 9 次迭代

聚类中心形状: (5, 3)

最终簇内平方和: 168.25

簇 0: 40 个样本

簇 1: 54 个样本

簇 2: 20 个样本

簇 3: 39 个样本

簇 4: 47 个样本

2. 在实际业务中，不同特征的重要性可能不同，且某些客户群可能需要大小相近。请实现带权重和大小约束的改进版本：

- 实现带约束的聚类算法，需要支持特征加权和簇大小约束 (6 分)
- 在以下两个场景下重新进行实验：收入特征权重加倍，限制每个客户群至少包含 20% 的客户，绘制聚类结果 (4 分)

```
1 class ConstrainedKMeans(KMeansPlusPlus):
2     def __init__(self, n_clusters=3, max_iters=100, weights=None, size_constraints=None):
3         """
4         参数:
5             weights: 特征权重向量
6             size_constraints: 每个簇的最小和最大样本数量限制
7         """
8         super().__init__(n_clusters, max_iters)
9         self.weights = weights
10        self.size_constraints = size_constraints
11
12        def _weighted_distance(self, X, centroids):
13            """计算加权欧氏距离"""
14            pass
15
16        def _reassign_clusters(self, X, labels, distances):
17            """在满足大小约束的情况下重新分配样本到簇"""
18            pass
```

Listing 4: 带约束的聚类算法部分实现

解:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 from sklearn.preprocessing import StandardScaler
5 from mpl_toolkits.mplot3d import Axes3D
6
7 # 加载数据
8 def load_mall_data():
9     df = pd.read_csv("Mall_Customers.csv")
10    X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
11    # 数据标准化
12    scaler = StandardScaler()
13    X_scaled = scaler.fit_transform(X)
```

```
14     return X_scaled, X, df
15
16 # 简化版带约束KMeans
17 class ConstrainedKMeans:
18     def __init__(self, n_clusters=5, weights=None, size_constraints=None):
19         self.n_clusters = n_clusters
20         self.weights = weights
21         self.size_constraints = size_constraints
22
23     def fit(self, X):
24         n_samples = X.shape[0]
25
26         # 随机初始化质心
27         np.random.seed(42)
28         indices = np.random.choice(n_samples, self.n_clusters, replace=False)
29         self.centroids = X[indices]
30
31         # 如果没有大小约束，设置默认
32         if self.size_constraints is None:
33             min_size = 0
34             max_size = n_samples
35             self.size_constraints = [(min_size, max_size) for _ in range(self.n_clusters)]
36
37         for _ in range(100):
38             # 计算距离
39             distances = self._compute_distance(X, self.centroids)
40
41             # 分配标签
42             labels = np.argmin(distances, axis=1)
43
44             # 调整以满足最小约束
45             labels = self._adjust_constraints(X, distances, labels)
46
47             # 更新质心
48             new_centroids = np.zeros_like(self.centroids)
49             for i in range(self.n_clusters):
50                 cluster_points = X[labels == i]
51                 if len(cluster_points) > 0:
52                     new_centroids[i] = np.mean(cluster_points, axis=0)
53                 else:
54                     new_centroids[i] = self.centroids[i]
55
56             # 检查收敛
57             if np.allclose(new_centroids, self.centroids, atol=1e-4):
58                 break
59
60         self.centroids = new_centroids
```

```
61
62     self.labels = labels
63     return self
64
65     def _compute_distance(self, X, centroids):
66         n_samples = X.shape[0]
67         distances = np.zeros((n_samples, self.n_clusters))
68
69         for i in range(self.n_clusters):
70             if self.weights is None:
71                 distances[:, i] = np.sqrt(np.sum((X - centroids[i])**2, axis=1))
72             else:
73                 weights = np.array(self.weights)
74                 weighted_diff = weights * (X - centroids[i])
75                 distances[:, i] = np.sqrt(np.sum(weighted_diff**2, axis=1))
76
77         return distances
78
79     def _adjust_constraints(self, X, distances, labels):
80         n_samples = X.shape[0]
81         min_sizes = [c[0] for c in self.size_constraints]
82
83         # 确保最小约束
84         for i in range(self.n_clusters):
85             count = np.sum(labels == i)
86             if count < min_sizes[i]:
87                 not_in_i = np.where(labels != i)[0]
88                 if len(not_in_i) > 0:
89                     needed = min_sizes[i] - count
90                     costs = distances[not_in_i, i]
91                     idxs = not_in_i[np.argsort(costs)[:needed]]
92                     labels[idxs] = i
93
94         return labels
95
96     # 加载数据
97     X_scaled, X_original, df = load_mall_data()
98     n_samples = X_original.shape[0]
99     colors = ['red', 'blue', 'green', 'orange', 'purple']
100
101     print(f"总客户数: {n_samples}")
102
103     # 场景1: 收入特征权重加倍
104     print("\n=== 场景1: 收入特征权重加倍 ===")
105     weights1 = [1.0, 2.0, 1.0] # 收入权重加倍
106     size_constraints1 = None # 没有大小约束
107
```

```
108 model1 = ConstrainedKMeans(n_clusters=5, weights=weights1,
109                             size_constraints=size_constraints1)
110 model1.fit(X_scaled)
111 labels1 = model1.labels
112
113 # 场景2: 每个簇至少包含20%的客户
114 print("\n=== 场景2: 每个簇至少包含20%的客户 ===")
115 weights2 = None # 没有特征权重
116 min_size = int(0.20 * n_samples)
117 size_constraints2 = [(min_size, n_samples) for _ in range(5)]
118
119 model2 = ConstrainedKMeans(n_clusters=5, weights=weights2,
120                             size_constraints=size_constraints2)
121 model2.fit(X_scaled)
122 labels2 = model2.labels
123
124 # 绘制两个场景的3D聚类结果
125 fig = plt.figure(figsize=(14, 6))
126
127 # 场景1: 收入特征权重加倍的3D可视化
128 ax1 = fig.add_subplot(121, projection='3d')
129 for i in range(5):
130     ax1.scatter(X_original[labels1 == i, 0], # Age
131                 X_original[labels1 == i, 1], # Annual Income
132                 X_original[labels1 == i, 2], # Spending Score
133                 s=50, c=colors[i], label=f'Cluster {i}', alpha=0.7)
134
135 ax1.set_title('Scenario 1: Income Weight Doubled\n(Weights=[1.0, 2.0, 1.0])', fontsize=12)
136 ax1.set_xlabel('Age', fontsize=10)
137 ax1.set_ylabel('Annual Income (k$)', fontsize=10)
138 ax1.set_zlabel('Spending Score (1-100)', fontsize=10)
139 ax1.legend()
140
141 # 场景2: 最小20%约束的3D可视化
142 ax2 = fig.add_subplot(122, projection='3d')
143 for i in range(5):
144     ax2.scatter(X_original[labels2 == i, 0], # Age
145                 X_original[labels2 == i, 1], # Annual Income
146                 X_original[labels2 == i, 2], # Spending Score
147                 s=50, c=colors[i], label=f'Cluster {i}', alpha=0.7)
148
149 ax2.set_title('Scenario 2: Min 20% per Cluster\n(No feature weighting)', fontsize=12)
150 ax2.set_xlabel('Age', fontsize=10)
151 ax2.set_ylabel('Annual Income (k$)', fontsize=10)
152 ax2.set_zlabel('Spending Score (1-100)', fontsize=10)
153 ax2.legend()
154
```

```

155 plt.tight_layout()
156 plt.show()
157
158 # 打印簇分布信息
159 print("\n场景1簇分布 (收入权重加倍):")
160 for i in range(5):
161     count = np.sum(labels1 == i)
162     percentage = (count / n_samples) * 100
163     print(f" Cluster {i}: {count} customers ({percentage:.1f}%)")
164
165 print("\n场景2簇分布 (最小20%约束):")
166 for i in range(5):
167     count = np.sum(labels2 == i)
168     percentage = (count / n_samples) * 100
169     print(f" Cluster {i}: {count} customers ({percentage:.1f}%)")
170     if percentage < 20:
171         print(f"      Warning: Below 20% minimum!")

```

Listing 5: 2.

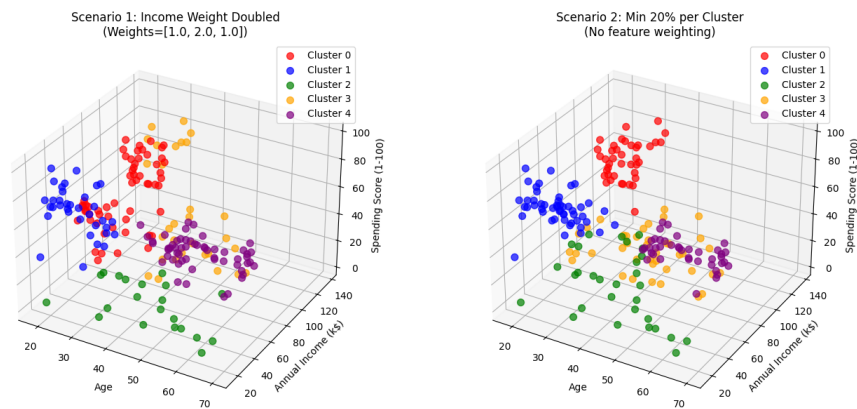


图 5: 可视化