**【评价类】创新型算法汇总**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **编号** | **组合模型** | **主要优势** | **适用题型** | **赛题特点** |
| 1 | AHP + 熵权法 | 结合主观经验与客观数据，权重更合理 | 供应商选择、绩效评估 | 适合需平衡主观经验与客观数据的综合评价类赛题（如多维度指标的权重分配与方案评选） |
| 2 | TOPSIS + 灰色关联分析 | 增强对不确定信息的处理能力，提高评价稳定性 | 项目投资决策、方案优选 | 适合含不确定信息的多方案决策类赛题（如信息不完全、数据波动大的场景下的最优方案选择） |
| 3 | 模糊综合评价 + BP 神经网络 | 处理模糊信息同时具备自学习能力 | 教学质量评价、区域发展评估 | 适合含模糊指标且需动态学习的评价类赛题（如涉及模糊描述且需通过数据自主优化评价标准的场景） |
| 4 | PCA + SVM | 降维同时保持分类精度，提高运算效率 | 信用评级、故障诊断、图像分类 | 适合高维数据下的精准分类赛题（如需剔除冗余信息并保持高分类精度的场景）。 |
| 5 | DEA + TOPSIS | 兼顾效率评价与多属性决策 | 企业效率评估、医院绩效评价 | 适合兼顾效率与多属性的综合评估类赛题（如既需评价效率又需综合多维度指标排序的场景） |
| 6 | 粗糙集 + 模糊聚类 | 处理不确定信息并实现自动分类 | 客户细分、质量分类 | 适合含不确定信息的自动分类赛题（如处理不完整数据并实现模糊信息分类的场景） |
| 7 | 贝叶斯网络 + 证据理论 | 增强不确定性推理能力，融合多源信息 | 风险评估、故障诊断 | 适合多源信息融合的不确定性推理赛题（如需融合多种信息进行风险或故障等推理的场景） |
| 8 | 随机森林 + AHP | 结合机器学习与专家经验，提高分类可靠性 | 疾病诊断、客户流失预测 | 适合需结合数据与专家经验的分类赛题（如提升分类可靠性，融合机器学习结果与专家判断的场景） |
| 9 | 聚类分析 + 因子分析 | 先降维再分类，提高聚类效果 | 市场细分、用户画像 | 适合高维数据先降维再分类的赛题（如从多维度数据中提取核心因子后进行精准聚类的场景） |
| 10 | 神经网络 + 遗传算法 | 提高神经网络优化效率，避免局部最优 | 优化问题、模式识别 | 适合需高效优化避免局部最优的赛题（如神经网络参数或结构优化，以及模式识别等场景） |
| 11 | SVM + 粒子群优化 | 优化 SVM 参数，提高分类精度 | 图像识别、文本分类 | 适合需优化参数提升分类精度的赛题（如通过参数寻优提高 SVM 在图像、文本等分类中的表现） |
| 12 | 决策树 + 模糊逻辑 | 增强模型解释性和处理模糊信息能力 | 信贷审批、风险评估 | 适合需增强解释性与处理模糊信息的决策赛题（如信贷审批、风险评估等需清晰规则且处理模糊指标的场景） |
| 13 | XGBoost + PCA | 处理高维数据，提高预测精度 | 房价预测、客户价值评估 | 适合高维数据下的高精度预测赛题（如处理多特征数据并提升房价、客户价值等预测准确性的场景） |

# 一、13种组合评价模型的详细说明

以下是 15 种组合评价模型的基本原理、公式推导、主要优势及适合的赛题类型，结合数学建模竞赛场景特点整理：

### **1. AHP + 熵权法**

#### **基本原理**

* AHP 通过构建判断矩阵将专家主观判断转化为定量权重，需通过一致性检验确保逻辑合理性；
* 熵权法基于指标数据的离散程度计算客观权重，数据越离散（信息量越大），权重越高；
* 组合逻辑：主观权重与客观权重线性融合（如），兼顾专家经验与数据规律。

#### **公式推导**

1. **AHP 权重**：

* 对判断矩阵，求解特征方程，归一化特征向量即为权重。
* 一致性检验：，（为随机一致性指标，通过检验）。

1. **熵权法权重**：

* 指标标准化：
* 熵值：
* 权重：

#### **主要优势**

* 平衡主观经验与客观数据，避免单一方法的片面性；
* 权重计算可解释性强，适合需要体现 “专家意见” 的场景。

#### **适合赛题**

* 供应商选择、绩效评估、方案优选等需结合主客观信息的评价题（如 “某企业合作伙伴评选” “城市宜居度排名”）。

### **2. TOPSIS + 灰色关联分析**

#### **基本原理**

* TOPSIS 通过计算评价对象与 “理想解” “负理想解” 的距离排序，反映全局优劣；
* 灰色关联分析通过衡量序列曲线的相似性，处理小样本、贫信息下的局部关联性；
* 组合逻辑：TOPSIS 提供全局排序，灰色关联增强局部稳定性，适合指标存在矛盾性的场景。

#### **公式推导**

1. **TOPSIS 得分**：

* 加权标准化矩阵：（为权重）
* 距离：，
* 得分：

1. **灰色关联度**：

* 关联系数：（为分辨系数）
* 关联度：

#### **主要优势**

* 处理小样本、不确定数据能力强；
* 兼顾 “全局最优” 与 “局部协调”，适合指标存在冲突的场景（如 “高收益伴随高风险”）。

#### **适合赛题**

* 投资项目决策、资源分配、风险评估等小样本评价题（如 “新能源项目投资选择” “区域水资源调配方案评价”）。

### **3. 模糊综合评价 + BP 神经网络**

#### **基本原理**

* 模糊综合评价用隶属度函数量化定性指标（如 “很好” “较好”），处理模糊不确定性；
* BP 神经网络通过多层非线性映射学习指标与评价结果的复杂关系，具备自适应性；
* 组合逻辑：模糊评价处理定性输入，神经网络拟合非线性关系，提升评价精度。

#### **公式推导**

1. **模糊综合评价**：

* 隶属度矩阵（为指标对等级的隶属度）
* 综合评价向量：（为模糊合成算子，如加权平均）。

1. **BP 神经网络**：

* 神经元输出：（为激活函数，如 ReLU）
* 反向传播更新权重：（为学习率，为误差项）。

#### **主要优势**

* 处理定性指标与模糊信息能力强；
* 神经网络可学习复杂非线性关系，适合指标间交互作用强的场景。

#### **适合赛题**

* 教学质量评价、服务满意度评估、产品质量分级等含定性指标的评价题（如 “高校教师教学水平评估” “旅游服务质量评级”）。

### **4. PCA + SVM**

#### **基本原理**

* PCA 通过正交变换将高维数据映射到低维空间，保留主要信息，去除冗余；
* SVM 通过寻找最优超平面实现分类，在高维、小样本数据上表现优异；
* 组合逻辑：PCA 降维减少噪声和计算量，SVM 提升分类精度，适合高维数据评价。

#### **公式推导**

1. **PCA 降维**：

* 标准化数据：
* 协方差矩阵：
* 特征值分解：（为特征向量矩阵）
* 降维后数据：（为前个特征向量）。

1. **SVM 分类**：

* 目标函数：（为惩罚系数，为松弛变量）
* 约束：，。

#### **主要优势**

* 降低高维数据复杂度，提高运算效率；
* 分类精度高，适合含噪声、高维特征的评价场景。

#### **适合赛题**

* 信用评级、故障诊断、图像识别等高维数据分类题（如 “银行客户信用风险分类” “设备故障类型识别”）。

### **5. DEA + TOPSIS**

#### **基本原理**

* DEA 通过线性规划计算多投入、多产出决策单元的相对效率，衡量资源利用水平；
* TOPSIS 综合多指标排序，补充 DEA 对 “非效率单元” 的区分能力；
* 组合逻辑：DEA 测度效率，TOPSIS 综合其他指标，适合兼顾 “效率” 与 “综合表现” 的评价。

#### **公式推导**

1. **DEA 效率（CCR 模型）**：

* 对决策单元：
* 约束：（投入），（产出），
* 效率值，为有效单元。

1. **TOPSIS 整合**：

* 将 DEA 效率值作为指标之一，与投入、产出指标共同参与 TOPSIS 排序（见模型 2 公式）。

#### **主要优势**

* 既衡量资源利用效率，又兼顾综合表现；
* 适合多投入、多产出系统的全面评价。

#### **适合赛题**

* 医院运营效率、企业生产绩效、区域经济发展等含投入产出的评价题（如 “城市公共服务效率评估” “制造业企业竞争力排名”）。

### **6. 粗糙集 + 模糊聚类**

#### **基本原理**

* 粗糙集通过属性约简去除冗余指标，处理不确定信息；
* 模糊聚类允许样本属于多个类别（隶属度），处理边界模糊的分类问题；
* 组合逻辑：粗糙集约简降维，模糊聚类增强分类灵活性，适合高维、不确定数据。

#### **公式推导**

1. **粗糙集约简**：

* 信息系统，不可分辨关系
* 约简集满足：且无冗余属性。

1. **模糊聚类（FCM）**：

* 目标函数：（为模糊系数）
* 隶属度更新：
* 聚类中心更新：

#### **主要优势**

* 自动去除冗余指标，简化模型；
* 聚类结果更灵活，适合样本边界模糊的场景。

#### **适合赛题**

* 客户细分、疾病分型、区域经济分类等含不确定信息的聚类评价题（如 “电商用户消费群体细分” “慢性病患者分型评价”）。

### **7. 贝叶斯网络 + 证据理论**

#### **基本原理**

* 贝叶斯网络用有向图建模变量间的因果关系，通过概率推理处理不确定性；
* 证据理论（D-S 理论）融合多源证据，处理信息冲突或缺失；
* 组合逻辑：贝叶斯网络建模因果，证据理论融合多源信息，提升推理可靠性。

#### **公式推导**

1. **贝叶斯网络**：

* 联合概率：
* 后验概率：（为假设，为证据）。

1. **证据理论**：

* 基本概率分配（BPA）：，满足，
* 合成规则：（）。

#### **主要优势**

* 擅长处理多源不确定信息的融合；
* 可解释性强，适合需要追溯推理过程的场景。

#### **适合赛题**

* 风险评估、故障诊断、多传感器数据融合等评价题（如 “核电站安全风险评估” “多源水质监测数据融合评价”）。

### **8. 随机森林 + AHP**

#### **基本原理**

* 随机森林通过多棵决策树集成，输出特征重要性，体现数据驱动的权重；
* AHP 补充专家对特征的主观判断；
* 组合逻辑：随机森林提供数据驱动的特征重要性，AHP 修正主观偏差，提升权重合理性。

#### **公式推导**

1. **随机森林特征重要性**：

* 不纯度减少量：（为树数量，为不纯度指标）。

1. **AHP 修正**：

* 组合权重：（为权重系数）。

#### **主要优势**

* 结合机器学习的自适应性与专家经验，权重更全面；
* 抗过拟合能力强，适合高维数据。

#### **适合赛题**

* 疾病风险预测、客户流失预警等需融合数据与专家知识的评价题（如 “糖尿病患病风险评估” “电信用户流失原因分析”）。

### **9. 聚类分析 + 因子分析**

#### **基本原理**

* 因子分析将多个相关指标降维为少数不相关因子，揭示潜在结构；
* 聚类分析基于因子得分对样本分类；
* 组合逻辑：因子分析降维去噪，聚类分析实现样本分组，适合高维数据的分类评价。

#### **公式推导**

1. **因子分析**：

* 因子模型：（为因子载荷，为公共因子，为特殊因子）
* 因子得分：（为相关矩阵）。

1. **聚类分析（K-Means）**：

* 目标函数：（为簇中心）
* 迭代更新簇中心直至收敛。

#### **主要优势**

* 简化高维数据，揭示潜在结构；
* 聚类结果直观，适合样本分组评价。

#### **适合赛题**

* 市场细分、区域经济类型划分等需降维后分类的评价题（如 “汽车消费群体细分” “城市经济发展模式分类”）。

### **10. 神经网络 + 遗传算法**

#### **基本原理**

* 神经网络（如 BP 神经网络）通过多层神经元的非线性映射，可拟合复杂的输入 - 输出关系，但依赖初始参数，易陷入局部最优；
* 遗传算法模拟生物进化过程（选择、交叉、变异），具有全局寻优能力，可优化神经网络的关键参数（如权重、学习率、隐藏层节点数）；
* 组合逻辑：遗传算法全局搜索最优参数组合，神经网络基于优化后的参数进行训练和预测，兼顾非线性拟合能力与全局最优性。

#### **公式推导**

1. **神经网络**：

* 神经元输出：（为激活函数，如 Sigmoid 或 ReLU）
* 损失函数：（为真实值，为预测值）。

1. **遗传算法优化**：

* 编码：将神经网络参数（如权重、偏置）编码为染色体（实数或二进制序列）；
* 适应度函数：以神经网络的预测精度（如）作为适应度，值越高表示参数越优；
* 选择：基于适应度选择父代（如轮盘赌选择）；
* 交叉：父代染色体交换部分基因（如实数交叉：，为交叉因子）；
* 变异：随机改变染色体部分基因（如，为小扰动）。

#### **主要优势**

* 避免神经网络陷入局部最优，提升模型泛化能力；
* 无需人工调整参数，适合参数空间复杂的场景。

#### **适合赛题**

* 复杂非线性系统的预测与优化评价（如 “化工生产工艺参数优化” “交通流量预测与信号配时优化”）。

### **11. SVM + 粒子群优化（PSO）**

#### **基本原理**

* SVM 通过核函数将数据映射到高维空间，寻找最优超平面实现分类，但核函数参数（如 RBF 核的）和惩罚系数对性能影响极大；
* 粒子群优化模拟鸟群觅食行为，通过粒子的位置和速度更新搜索最优参数；
* 组合逻辑：PSO 全局搜索最优的和核参数，SVM 基于优化后的参数进行分类，提升分类精度。

#### **公式推导**

1. **SVM 分类**：

* 目标函数：（为惩罚系数，为松弛变量）
* 约束：（为核映射，如 RBF 核：）。

1. **PSO 优化参数**：

* 粒子位置：每个粒子代表一组参数；
* 速度更新：（为惯性权重，为学习因子，为个体最优，为全局最优）；
* 位置更新：；
* 适应度函数：以 SVM 的交叉验证准确率作为适应度。

#### **主要优势**

* 自动优化 SVM 关键参数，避免人工试错；
* 提升 SVM 在复杂数据上的分类精度和泛化能力。

#### **适合赛题**

* 高维、非线性数据的分类评价（如 “医学影像病灶识别” “文本情感倾向分类”）。

### **12. 决策树 + 模糊逻辑**

#### **基本原理**

* 决策树通过特征阈值划分样本，模型解释性强，但对连续特征的离散化可能丢失信息；
* 模糊逻辑用隶属度函数处理连续特征的模糊性（如 “温度高” “收入中等”），增强规则的灵活性；
* 组合逻辑：模糊逻辑将连续特征转化为模糊集合，决策树基于模糊规则进行分类，兼顾解释性与模糊信息处理能力。

#### **公式推导**

1. **模糊逻辑处理**：

* 对连续特征，定义模糊集合（如 “低” “中” “高”），隶属度函数（如三角形函数：）。

1. **模糊决策树**：

* 节点分裂准则：基于模糊信息增益（为模糊熵）；
* 叶节点输出：基于模糊规则的分类结果（如 “若为高且为中，则分类为 A”）。

#### **主要优势**

* 决策树规则可解释性强，模糊逻辑增强对连续特征的处理能力；
* 适合需要 “可解释规则” 的评价场景。

#### **适合赛题**

* 信贷审批、风险分级等需明确决策规则的评价题（如 “个人贷款违约风险分级” “企业信用评级规则制定”）。

### **13. XGBoost + PCA**

#### **基本原理**

* XGBoost 通过集成多棵决策树提升预测精度，对高维数据敏感，冗余特征可能增加过拟合风险；
* PCA 降维去除冗余特征，保留主要信息，降低数据复杂度；
* 组合逻辑：PCA 预处理高维数据，减少噪声和冗余，XGBoost 基于降维后的数据训练，提升模型效率和泛化能力。

#### **公式推导**

1. **PCA 降维**：

* 同模型 4 的 PCA 公式，得到降维后的特征。

1. **XGBoost 预测**：

* 集成模型：（为第棵决策树）
* 目标函数：（为损失函数，为正则项，防止过拟合）。

#### **主要优势**

* 降低高维数据对 XGBoost 的计算负担；
* 减少冗余特征导致的过拟合，提升模型稳定性。

#### **适合赛题**

* 高维特征的预测与评价（如 “基因数据疾病风险预测” “用户行为特征的客户价值评估”）。

### **二、13种组合评价模型的详细数学建模案例题目**

### **1. AHP + 熵权法案例：社区养老服务中心供应商选择**

**问题背景**：某社区计划引入 3 家养老服务供应商，提供助餐、医疗护理、文化娱乐 3 类核心服务。社区居民委员会需综合评价 5 家候选供应商，既要体现老年居民代表的主观偏好（如 “医疗护理比助餐更重要”），又要依据供应商的历史服务数据，避免决策片面性。

**数据**：

* 主观数据：10 位老年居民代表通过 1-9 标度法对 3 项服务的相对重要性打分，形成 AHP 判断矩阵；
* 客观数据：5 家供应商近 1 年的运营数据（助餐满意度 92%/88%/95%/85%/90%、医疗响应时效 15/20/10/25/18 分钟、文化活动频次 12/8/15/6/10 次 / 月）。

**要求**：综合主客观权重计算得分并排序，选择前 3 家供应商，需说明权重融合的合理性。

### 1. AHP + 熵权法代码：社区养老服务中心供应商选择

|  |
| --- |
| import numpy as np  import pandas as pd  # 数据准备  # 客观数据：助餐满意度(%)、医疗响应时效(分钟)、文化活动频次(次/月)  data = np.array([  [92, 15, 12],  [88, 20, 8],  [95, 10, 15],  [85, 25, 6],  [90, 18, 10]  ])  suppliers = [f'供应商{i+1}' for i in range(5)]  indicators = ['助餐满意度', '医疗响应时效', '文化活动频次']  # 医疗响应时效为成本型指标，进行正向化处理  data[:, 1] = max(data[:, 1]) - data[:, 1]  # AHP权重计算  def ahp\_weight(matrix):  eigenvalues, eigenvectors = np.linalg.eig(matrix)  max\_idx = np.argmax(eigenvalues)  weights = eigenvectors[:, max\_idx].real  weights /= np.sum(weights)  n = matrix.shape[0]  lambda\_max = eigenvalues[max\_idx].real  ci = (lambda\_max - n) / (n - 1)  ri\_table = {1:0,2:0,3:0.58}  cr = ci / ri\_table[n] if ri\_table[n] !=0 else 0  return weights, cr  # 构造判断矩阵（根据10位老年居民代表打分综合）  judgment\_matrix = np.array([  [1, 1/3, 2], # 助餐满意度相对重要性  [3, 1, 4], # 医疗响应时效相对重要性  [1/2, 1/4, 1] # 文化活动频次相对重要性  ])  ahp\_weights, cr = ahp\_weight(judgment\_matrix)  print(f"AHP一致性检验CR={cr:.4f}，{'通过' if cr<0.1 else '未通过'}")  # 熵权法权重计算  def entropy\_weight(data):  data\_norm = data / np.sum(data, axis=0) + 1e-10  n, m = data\_norm.shape  entropy = -np.sum(data\_norm \* np.log(data\_norm) / np.log(n), axis=0)  return (1 - entropy) / np.sum(1 - entropy)  entropy\_weights = entropy\_weight(data)  # 综合评价  alpha = 0.5  combined\_weights = alpha \* ahp\_weights + (1-alpha) \* entropy\_weights  normalized\_data = data / np.sum(data, axis=0)  scores = np.dot(normalized\_data, combined\_weights)  # 结果展示  result = pd.DataFrame({  '供应商': suppliers,  '综合得分': scores.round(4),  '排名': (scores.argsort()[::-1].argsort() + 1)  }).sort\_values('综合得分', ascending=False)  print(result) |

### 2. TOPSIS + 灰色关联分析案例：乡村旅游项目开发决策

**问题背景**：某县文旅局计划从 3 个乡村旅游开发方案中选 1 个重点扶持，方案涉及生态保护、经济收益、就业带动等矛盾指标（如方案 A 收益高但生态破坏风险大）。由于县域旅游数据积累少（仅 2 年试点数据），需兼顾全局最优与指标关联性。

**数据**：

* 3 个方案的 4 项指标：生态影响评分（满分 10 分：7/8/6）、年预期收益（万元：500/300/400）、带动就业人数（人：120/80/100）、投资回收期（年：3/2/2.5）。

**要求**：用 TOPSIS 确定全局排序，用灰色关联分析补充指标协调度，组合评价后推荐最优方案，说明如何平衡 “收益” 与 “生态”。

### TOPSIS + 灰色关联分析代码：乡村旅游项目开发决策

|  |
| --- |
| import numpy as np  import pandas as pd  # 数据准备  data = np.array([  [7, 500, 120, 3], # 方案1：生态影响、收益、就业、回收期  [8, 300, 80, 2], # 方案2  [6, 400, 100, 2.5] # 方案3  ])  projects = [f'方案{i+1}' for i in range(3)]  # 指标类型：[效益型, 效益型, 效益型, 成本型]  is\_cost = [False, False, False, True]  # 数据标准化  def normalize(data, is\_cost):  norm\_data = np.zeros\_like(data, dtype=float)  for j in range(data.shape[1]):  if is\_cost[j]:  norm\_data[:, j] = data[:, j].max() - data[:, j]  else:  norm\_data[:, j] = data[:, j] - data[:, j].min()  norm\_data[:, j] /= (norm\_data[:, j].max() + 1e-10)  return norm\_data  norm\_data = normalize(data, is\_cost)  # TOPSIS计算  def topsis\_score(norm\_data):  weights = np.ones(norm\_data.shape[1]) / norm\_data.shape[1]  weighted = norm\_data \* weights  ideal\_best = weighted.max(axis=0)  ideal\_worst = weighted.min(axis=0)  d\_best = np.sqrt(np.sum((weighted - ideal\_best)\*\*2, axis=1))  d\_worst = np.sqrt(np.sum((weighted - ideal\_worst)\*\* 2, axis=1))  return d\_worst / (d\_best + d\_worst)  topsis\_scores = topsis\_score(norm\_data)  # 灰色关联分析  def grey\_relation(norm\_data):  reference = norm\_data.max(axis=0)  rho = 0.5  min\_min = np.min(np.min(np.abs(norm\_data - reference)))  max\_max = np.max(np.max(np.abs(norm\_data - reference)))  xi = (min\_min + rho \* max\_max) / (np.abs(norm\_data - reference) + rho \* max\_max)  return np.mean(xi, axis=1)  grey\_scores = grey\_relation(norm\_data)  # 组合评价  combined\_scores = 0.5 \* topsis\_scores + 0.5 \* grey\_scores  result = pd.DataFrame({  '方案': projects,  'TOPSIS得分': topsis\_scores.round(4),  '灰色关联度': grey\_scores.round(4),  '组合得分': combined\_scores.round(4),  '排名': (combined\_scores.argsort()[::-1].argsort() + 1)  }).sort\_values('组合得分', ascending=False)  print(result) |

### 模糊综合评价 + BP 神经网络案例：在线课程质量评估

**问题背景**：某教育平台需对 8 门在线课程进行质量评级（A/B/C 级），评价指标含模糊性描述（如 “教师互动积极性”“课程内容实用性”），且指标间存在非线性关系（如 “内容难度” 对 “满意度” 的影响因学生群体而异）。

**数据**：

* 定性指标：学生评教中的模糊评价（如 “教师互动：很高 / 较高 / 一般”），通过隶属度函数转化为量化数据；
* 定量指标：课程完成率（%：85/78/90/72/88/75/92/68）、平均评分（满分 5 分：4.2/3.8/4.5/3.5/4.3/3.6/4.6/3.2）；
* 标签数据：专家对 8 门课程的评级（A/A/B/C/A/C/A/C）。

**要求**：用模糊综合评价处理定性指标，结合 BP 神经网络学习非线性关系，输出评级结果并分析 “内容实用性” 对评级的影响。

### 3. 模糊综合评价 + BP 神经网络代码：在线课程质量评估

|  |
| --- |
| import numpy as np  import pandas as pd  from sklearn.neural\_network import MLPClassifier  from sklearn.preprocessing import MinMaxScaler  # 数据准备  # 定量指标：课程完成率(%)、平均评分(5分制)  quant\_data = np.array([  [85, 4.2], [78, 3.8], [90, 4.5], [72, 3.5],  [88, 4.3], [75, 3.6], [92, 4.6], [68, 3.2]  ])  # 定性指标隶属度矩阵（教师互动、内容实用性，分为高/中/低）  qual\_data = np.array([  [0.8, 0.2, 0.0, 0.7, 0.3, 0.0], # 课程1  [0.6, 0.3, 0.1, 0.6, 0.4, 0.0], # 课程2  [0.9, 0.1, 0.0, 0.8, 0.2, 0.0], # 课程3  [0.3, 0.5, 0.2, 0.4, 0.5, 0.1], # 课程4  [0.7, 0.3, 0.0, 0.9, 0.1, 0.0], # 课程5  [0.5, 0.4, 0.1, 0.5, 0.4, 0.1], # 课程6  [0.9, 0.1, 0.0, 0.8, 0.2, 0.0], # 课程7  [0.2, 0.4, 0.4, 0.3, 0.4, 0.3] # 课程8  ])  # 融合定量与定性数据  X = np.hstack((quant\_data, qual\_data))  # 标签：A=2, B=1, C=0  y = np.array([2, 2, 1, 0, 2, 0, 2, 0])  courses = [f'课程{i+1}' for i in range(8)]  # 数据标准化  scaler = MinMaxScaler()  X\_scaled = scaler.fit\_transform(X)  # BP神经网络训练  mlp = MLPClassifier(hidden\_layer\_sizes=(10,), activation='relu',  solver='adam', max\_iter=1000, random\_state=42)  mlp.fit(X\_scaled, y)  # 预测与结果  y\_pred = mlp.predict(X\_scaled)  rating = {2: 'A', 1: 'B', 0: 'C'}  result = pd.DataFrame({  '课程': courses,  '预测等级': [rating[p] for p in y\_pred],  '实际等级': [rating[t] for t in y]  })  print(result) |

### 4. PCA + SVM案例：信用卡欺诈交易识别

**问题背景**：某银行需构建模型识别信用卡欺诈交易，交易数据含 20 项特征（如交易金额、时段、地点、商户类型等），存在高维冗余（如 “交易时段” 与 “商户类型” 强相关），且欺诈样本占比低（仅 3%）。

**数据**：

* 10000 条交易记录，每条含 20 项特征及标签（1 = 欺诈，0 = 正常），其中欺诈样本 300 条。

**要求**：用 PCA 降维至保留 95% 信息的维度，再用 SVM 分类，对比降维前后的识别准确率（尤其是欺诈样本的召回率），说明降维对模型效率的提升。

### 4. PCA + SVM代码：信用卡欺诈交易识别

|  |
| --- |
| import numpy as np  import pandas as pd  from sklearn.decomposition import PCA  from sklearn.svm import SVC  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import classification\_report  # 生成模拟数据（10000条交易，20项特征）  np.random.seed(42)  n\_samples = 10000  n\_features = 20  X = np.random.randn(n\_samples, n\_features)  # 生成欺诈标签（3%欺诈率）  weights = np.random.randn(n\_features)  y = np.where(np.dot(X, weights) + np.random.randn(n\_samples) > 2, 1, 0)  print(f"欺诈样本比例：{sum(y)/n\_samples:.2%}")  # 数据划分与标准化  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  # PCA降维  pca = PCA(n\_components=0.95, random\_state=42)  X\_train\_pca = pca.fit\_transform(X\_train\_scaled)  X\_test\_pca = pca.transform(X\_test\_scaled)  print(f"降维前特征数：{n\_features}，降维后：{pca.n\_components\_}")  # SVM分类与评估  svm = SVC(kernel='rbf', class\_weight='balanced', random\_state=42)  svm.fit(X\_train\_pca, y\_train)  y\_pred = svm.predict(X\_test\_pca)  print("\n降维后分类报告：")  print(classification\_report(y\_test, y\_pred))  # 对比不降维的情况  svm\_raw = SVC(kernel='rbf', class\_weight='balanced', random\_state=42)  svm\_raw.fit(X\_train\_scaled, y\_train)  y\_pred\_raw = svm\_raw.predict(X\_test\_scaled)  print("\n不降维分类报告：")  print(classification\_report(y\_test, y\_pred\_raw)) |

### 5. DEA + TOPSIS案例：县域医院运营效率评估

**问题背景**：某省卫健委需评价 10 家县级医院的运营绩效，既要衡量资源利用效率（如 “医护人员人均诊疗量”），又要综合患者满意度等非效率指标，为资源调配提供依据。

**数据**：

* 投入指标：医护人员数（人：320/280/400/...）、床位数（张：500/450/600/...）、年度运营成本（万元：2800/2500/3200/...）；
* 产出指标：年诊疗人次（千：150/130/180/...）、治愈率（%：82/80/85/...）、患者满意度（%：85/83/88/...）。

**要求**：用 DEA 计算效率值，结合 TOPSIS 综合所有指标，排序后指出非效率医院的改进方向（如 “减少床位数” 或 “提高治愈率”）。

### 5. DEA + TOPSIS代码：县域医院运营效率评估

|  |
| --- |
| import numpy as np  import pandas as pd  from scipy.optimize import linprog  # 数据准备（10家医院）  # 投入：医护人员数、床位数、运营成本(万元)  inputs = np.array([  [320, 500, 2800], [280, 450, 2500], [400, 600, 3200],  [220, 350, 2000], [350, 550, 3000], [250, 400, 2300],  [380, 580, 3100], [200, 300, 1800], [300, 480, 2600],  [360, 520, 2900]  ])  # 产出：年诊疗人次(千)、治愈率(%)、患者满意度(%)  outputs = np.array([  [150, 82, 85], [130, 80, 83], [180, 85, 88],  [100, 78, 80], [160, 83, 86], [120, 79, 82],  [170, 84, 87], [90, 76, 79], [140, 81, 84],  [155, 82, 85]  ])  hospitals = [f'医院{i+1}' for i in range(10)]  # DEA效率计算（CCR模型）  def dea\_ccr(inputs, outputs):  n, m = inputs.shape  s = outputs.shape[1]  efficiencies = []  for j in range(n):  c = np.zeros(n + 1)  c[0] = -1  A\_eq = np.hstack((np.zeros((s, 1)), outputs.T))  b\_eq = outputs[j, :]  A\_ub = np.hstack((np.ones((m, 1)), -inputs.T))  b\_ub = -inputs[j, :]  bounds = [(None, None)] + [(0, None)] \* n  result = linprog(c, A\_ub=A\_ub, b\_ub=b\_ub, A\_eq=A\_eq, b\_eq=b\_eq, bounds=bounds, method='highs')  efficiencies.append(-result.fun)  return np.array(efficiencies)  dea\_efficiencies = dea\_ccr(inputs, outputs)  # TOPSIS综合评价  def topsis\_evaluation(inputs, outputs, dea\_scores):  inputs\_norm = inputs / np.sqrt(np.sum(inputs\*\*2, axis=0))  outputs\_norm = outputs / np.sqrt(np.sum(outputs\*\*2, axis=0))  inputs\_norm = 1 - inputs\_norm # 投入指标反向化  dea\_norm = dea\_scores / np.max(dea\_scores)  all\_data = np.column\_stack((inputs\_norm, outputs\_norm, dea\_norm.reshape(-1, 1)))  weights = np.ones(all\_data.shape[1]) / all\_data.shape[1]  weighted = all\_data \* weights  ideal\_best = weighted.max(axis=0)  ideal\_worst = weighted.min(axis=0)  d\_best = np.sqrt(np.sum((weighted - ideal\_best)\*\*2, axis=1))  d\_worst = np.sqrt(np.sum((weighted - ideal\_worst)\*\* 2, axis=1))  return d\_worst / (d\_best + d\_worst)  topsis\_scores = topsis\_evaluation(inputs, outputs, dea\_efficiencies)  combined\_scores = 0.5 \* dea\_efficiencies + 0.5 \* topsis\_scores  # 结果展示  result = pd.DataFrame({  '医院': hospitals,  'DEA效率值': dea\_efficiencies.round(4),  'TOPSIS得分': topsis\_scores.round(4),  '组合得分': combined\_scores.round(4),  '排名': (combined\_scores.argsort()[::-1].argsort() + 1)  }).sort\_values('组合得分', ascending=False)  print(result) |

### 6. 粗糙集 + 模糊聚类案例：电商用户消费群体细分

**问题背景**：某电商平台需对 5000 名用户细分，用于精准营销。用户数据含 10 项特征（如年龄、性别、月消费额、购买频率等），部分特征冗余（如 “月消费额” 与 “购买频率” 高度相关），且用户边界模糊（如 “中等消费” 与 “高消费” 无明确阈值）。

**数据**：

* 5000 条用户记录，特征包括：年龄（18-60 岁）、性别（男 / 女）、月消费额（0-5000 元）、购买频率（1-30 次 / 月）等。

**要求**：用粗糙集约简特征（保留核心特征），再用模糊聚类分为 3-5 个群体，输出每个群体的特征（如 “25-35 岁女性，月消费 2000-3000 元”）及隶属度。

### 6. 粗糙集 + 模糊聚类代码：电商用户消费群体细分

|  |
| --- |
| import numpy as np  import pandas as pd  from sklearn.preprocessing import KBinsDiscretizer  from roughsets.roughsets import RoughSet # 需安装：pip install roughsets  from fcmeans import FCM # 需安装：pip install fcmeans  import matplotlib.pyplot as plt  # 1. 数据准备（模拟5000名用户数据）  np.random.seed(42)  n\_users = 5000  # 特征：年龄(18-60)、性别(0=男,1=女)、月消费额(0-5000)、购买频率(1-30)、浏览时长(小时/周)、收藏数量(个/月)  age = np.random.randint(18, 61, n\_users)  gender = np.random.randint(0, 2, n\_users)  monthly\_spend = np.random.randint(0, 5001, n\_users)  purchase\_freq = np.random.randint(1, 31, n\_users)  browse\_time = np.random.uniform(1, 20, n\_users)  favorite\_count = np.random.randint(0, 50, n\_users)  X = np.column\_stack((age, gender, monthly\_spend, purchase\_freq, browse\_time, favorite\_count))  features = ['年龄', '性别', '月消费额', '购买频率', '浏览时长', '收藏数量']  # 2. 粗糙集特征约简  # 数据离散化（粗糙集要求离散数据）  discretizer = KBinsDiscretizer(n\_bins=5, encode='ordinal', strategy='uniform')  X\_discrete = discretizer.fit\_transform(X).astype(int)  # 构造决策系统（用月消费额离散值作为伪决策属性）  decision = X\_discrete[:, 2] # 以月消费额作为决策属性  rs = RoughSet(X\_discrete, decision)  reduct = rs.reduct() # 获取约简特征索引  print(f"原始特征数：{X.shape[1]}，约简后特征数：{len(reduct)}")  print(f"保留的核心特征索引：{reduct}，对应特征：{[features[i] for i in reduct]}")  # 提取约简后的特征  X\_reduced = X[:, reduct]  # 3. 模糊聚类（分为4个群体）  fcm = FCM(n\_clusters=4, random\_state=42)  fcm.fit(X\_reduced)  membership = fcm.u # 隶属度矩阵  labels = np.argmax(membership, axis=1) # 最终聚类标签  # 4. 分析聚类结果  cluster\_centers = fcm.centers  result = pd.DataFrame(X\_reduced, columns=[features[i] for i in reduct])  result['聚类标签'] = labels  # 输出每个群体的特征描述  print("\n各消费群体特征：")  for i in range(4):  cluster\_data = result[result['聚类标签'] == i]  print(f"\n群体{i+1}（样本数：{len(cluster\_data)}）：")  print(cluster\_data.iloc[:, :-1].describe().round(2).loc[['mean']])  # 5. 可视化聚类结果（取前2个特征）  plt.figure(figsize=(10, 6))  plt.scatter(X\_reduced[:, 0], X\_reduced[:, 1], c=labels, cmap='viridis', alpha=0.6)  plt.scatter(cluster\_centers[:, 0], cluster\_centers[:, 1], marker='X', s=200, c='red', label='聚类中心')  plt.xlabel(features[reduct[0]])  plt.ylabel(features[reduct[1]])  plt.title('电商用户消费群体聚类')  plt.legend()  plt.show() |

### 7. 贝叶斯网络 + 证据理论案例：无人机故障诊断

**问题背景**：某无人机厂商需构建故障诊断系统，融合 5 个传感器数据（电池电压、电机转速、GPS 信号强度等），判断无人机是否存在 “动力系统故障”，传感器数据存在噪声和冲突（如某传感器显示异常但其他正常）。

**数据**：

* 1000 组飞行记录，每组含 5 项传感器数据（数值型）及故障标签（1 = 故障，0 = 正常），部分记录存在传感器数据缺失。

**要求**：用贝叶斯网络建模传感器与故障的因果关系，用证据理论融合冲突数据，输出故障概率及诊断置信度，分析对缺失数据的处理能力。

### 7. 贝叶斯网络 + 证据理论代码：无人机故障诊断

|  |
| --- |
| import numpy as np  import pandas as pd  from pgmpy.models import BayesianNetwork  from pgmpy.estimators import MaximumLikelihoodEstimator  from pgmpy.inference import VariableElimination  from scipy.stats import norm  # 1. 数据准备（模拟1000组飞行记录）  np.random.seed(42)  n\_samples = 1000  # 传感器数据（5项：电池电压、电机转速、GPS信号、温度、振动）  # 正常状态下的分布  battery\_voltage = norm.rvs(loc=12.0, scale=0.5, size=n\_samples)  motor\_speed = norm.rvs(loc=3000, scale=200, size=n\_samples)  gps\_signal = norm.rvs(loc=80, scale=10, size=n\_samples) # 信号强度0-100  temperature = norm.rvs(loc=45, scale=5, size=n\_samples)  vibration = norm.rvs(loc=0.1, scale=0.05, size=n\_samples)  # 故障标签（10%故障概率）  fault = np.random.binomial(1, 0.1, n\_samples)  # 故障状态下传感器数据异常  battery\_voltage[fault == 1] = norm.rvs(loc=9.0, scale=1.0, size=sum(fault))  motor\_speed[fault == 1] = norm.rvs(loc=2000, scale=300, size=sum(fault))  gps\_signal[fault == 1] = norm.rvs(loc=40, scale=15, size=sum(fault))  temperature[fault == 1] = norm.rvs(loc=60, scale=8, size=sum(fault))  vibration[fault == 1] = norm.rvs(loc=0.5, scale=0.2, size=sum(fault))  # 离散化传感器数据（3个状态：低、中、高）  def discretize(data, bins=3):  return np.digitize(data, np.percentile(data, [100\*i/bins for i in range(1, bins)]))  sensors\_discrete = np.column\_stack([  discretize(battery\_voltage),  discretize(motor\_speed),  discretize(gps\_signal),  discretize(temperature),  discretize(vibration)  ])  # 构造数据集  data = pd.DataFrame(sensors\_discrete, columns=['电池电压', '电机转速', 'GPS信号', '温度', '振动'])  data['动力系统故障'] = fault  # 2. 贝叶斯网络建模  model = BayesianNetwork([  ('电池电压', '动力系统故障'),  ('电机转速', '动力系统故障'),  ('GPS信号', '动力系统故障'),  ('温度', '动力系统故障'),  ('振动', '动力系统故障')  ])  # 参数学习  model.fit(data, estimator=MaximumLikelihoodEstimator)  # 3. 证据理论融合  def dempster\_shafer(evidence\_list):  # 基本概率分配（BPA）  bpa = {0: 0.1, 1: 0.1} # 初始不确定性  for evidence in evidence\_list:  new\_bpa = {0: 0, 1: 0}  # 传感器证据支持度  p\_fault = evidence['P(故障|证据)']  new\_bpa[1] += p\_fault \* 0.8  new\_bpa[0] += (1 - p\_fault) \* 0.8  # 剩余概率分配给不确定性  new\_bpa[0] += 0.2 \* bpa[0]  new\_bpa[1] += 0.2 \* bpa[1]  bpa = new\_bpa  # 归一化  total = sum(bpa.values())  return {k: v/total for k, v in bpa.items()}  # 4. 故障诊断示例（随机选择10条记录）  infer = VariableElimination(model)  sample\_indices = np.random.choice(n\_samples, 10, replace=False)  results = []  for idx in sample\_indices:  sample = data.iloc[idx]  evidence = {col: sample[col] for col in ['电池电压', '电机转速', 'GPS信号', '温度', '振动']}  # 贝叶斯网络推理  fault\_proba = infer.query(variables=['动力系统故障'], evidence=evidence)['动力系统故障'].values[1]  # 证据融合  evidence\_list = [{'P(故障|证据)': fault\_proba}]  fused\_bpa = dempster\_shafer(evidence\_list)  # 诊断结果  diagnosis = 1 if fused\_bpa[1] > 0.5 else 0  results.append({  '样本索引': idx,  '实际故障': sample['动力系统故障'],  '贝叶斯故障概率': round(fault\_proba, 4),  '融合后故障概率': round(fused\_bpa[1], 4),  '诊断结果': diagnosis  })  # 展示结果  result\_df = pd.DataFrame(results)  print("无人机故障诊断结果：")  print(result\_df)  print(f"\n诊断准确率：{sum(result\_df['实际故障'] == result\_df['诊断结果'])/len(result\_df):.2%}") |

### 8. 随机森林 + AHP案例：慢性病患病风险评估

**问题背景**：某社区卫生服务中心需评估居民患高血压的风险，指标含 10 项体检数据（如血压、血糖、BMI 等），需结合医生经验（如 “血压比血糖更重要”）与数据规律（如 “BMI>28 的人群风险显著升高”）。

**数据**：

* 1000 名居民的体检数据及患病标签（1 = 患病，0 = 未患病）；
* 3 位医生对 10 项指标的重要性打分（1-10 分）。

**要求**：用随机森林计算数据驱动的特征重要性，用 AHP 融入医生经验，组合权重后构建风险评估模型，输出高风险人群的关键特征（如 “收缩压> 140 且 BMI>28”）。

### 8. 随机森林 + AHP代码：慢性病患病风险评估

|  |
| --- |
| import numpy as np  import pandas as pd  from sklearn.ensemble import RandomForestClassifier  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import roc\_auc\_score  from sklearn.preprocessing import StandardScaler  # 1. 数据准备（模拟1000名居民体检数据）  np.random.seed(42)  n\_samples = 1000  # 10项体检指标  age = np.random.randint(30, 80, n\_samples) # 年龄  sbp = np.random.normal(120, 15, n\_samples) # 收缩压  dbp = np.random.normal(80, 10, n\_samples) # 舒张压  glucose = np.random.normal(5.5, 1.0, n\_samples) # 血糖  bmi = np.random.normal(25, 4, n\_samples) # 体质指数  cholesterol = np.random.normal(5.0, 0.8, n\_samples) # 胆固醇  smoke = np.random.binomial(1, 0.3, n\_samples) # 是否吸烟  exercise = np.random.binomial(1, 0.5, n\_samples) # 是否锻炼  family\_history = np.random.binomial(1, 0.2, n\_samples) # 家族病史  diet = np.random.randint(1, 5, n\_samples) # 饮食健康度(1-4)  # 构建高血压患病标签（基于关键指标）  high\_risk = (sbp > 140) | (dbp > 90) | (glucose > 7) | (bmi > 30)  y = np.where(high\_risk, 1, 0)  print(f"高血压患病比例：{sum(y)/n\_samples:.2%}")  # 特征矩阵  X = np.column\_stack([age, sbp, dbp, glucose, bmi, cholesterol, smoke, exercise, family\_history, diet])  features = ['年龄', '收缩压', '舒张压', '血糖', 'BMI', '胆固醇', '吸烟', '锻炼', '家族病史', '饮食']  # 2. 随机森林计算特征重要性  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  rf = RandomForestClassifier(n\_estimators=100, random\_state=42)  rf.fit(X\_train\_scaled, y\_train)  rf\_importance = rf.feature\_importances\_  print("\n随机森林特征重要性：")  for i, imp in enumerate(rf\_importance):  print(f"{features[i]}: {imp:.4f}")  # 3.</doubaocanvas> |

### 9. 聚类分析 + 因子分析案例：城市宜居度分类

**问题背景**：某研究机构需对 20 个城市的宜居度分类，指标含 8 项（如人均 GDP、空气质量、教育资源、房价等），指标间存在相关性（如 “人均 GDP 高的城市房价也高”）。

**数据**：

* 20 个城市的 8 项指标数据（标准化后）：如人均 GDP（0-100 分）、空气质量指数（0-100 分，值越低越好）等。

**要求**：用因子分析提取 2-3 个公共因子（如 “经济因子”“环境因子”），基于因子得分用 K-means 聚类为 3 类，分析每类城市的宜居特征及改进方向。

### 9. 聚类分析 + 因子分析代码：城市宜居度分类

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.decomposition import FactorAnalysis  from sklearn.cluster import KMeans  from sklearn.preprocessing import StandardScaler  # 1. 数据准备（20个城市的8项指标）  np.random.seed(42)  n\_cities = 20  # 指标：人均GDP(万元)、空气质量指数、教育资源(学校数量)、医疗资源(医院数量)  # 房价(万元/㎡)、交通便利性(公交站点数)、绿化覆盖率(%)、就业率(%)  data = np.array([  [8.2, 65, 120, 80, 3.5, 500, 40, 92],  [6.8, 72, 95, 65, 2.8, 420, 35, 88],  [9.5, 58, 150, 95, 4.2, 600, 45, 94],  [5.2, 85, 70, 45, 1.9, 300, 25, 82],  [7.6, 68, 110, 75, 3.2, 480, 38, 90],  [4.8, 90, 65, 40, 1.7, 280, 22, 80],  [8.8, 60, 130, 85, 3.8, 550, 42, 93],  [5.5, 80, 75, 50, 2.1, 320, 28, 83],  [7.0, 70, 100, 70, 3.0, 450, 36, 89],  [6.2, 75, 85, 60, 2.5, 380, 32, 86],  [10.2, 55, 160, 100, 4.5, 650, 48, 95],  [4.5, 95, 60, 35, 1.5, 250, 20, 78],  [8.0, 66, 115, 82, 3.4, 490, 39, 91],  [5.8, 78, 80, 55, 2.3, 350, 30, 85],  [6.5, 73, 90, 62, 2.7, 400, 34, 87],  [9.2, 59, 140, 90, 4.0, 580, 44, 93],  [5.0, 88, 68, 42, 1.8, 290, 24, 81],  [7.3, 69, 105, 72, 3.1, 460, 37, 89],  [8.5, 63, 125, 88, 3.7, 530, 41, 92],  [6.0, 76, 88, 58, 2.4, 370, 33, 84]  ])  cities = [f'城市{i+1}' for i in range(n\_cities)]  indicators = ['人均GDP', '空气质量指数', '教育资源', '医疗资源',  '房价', '交通便利性', '绿化覆盖率', '就业率']  # 2. 数据标准化  scaler = StandardScaler()  data\_scaled = scaler.fit\_transform(data)  # 3. 因子分析（提取公共因子）  fa = FactorAnalysis(n\_components=3, random\_state=42)  factor\_scores = fa.fit\_transform(data\_scaled)  # 输出因子载荷（反映指标与因子的相关性）  loadings = pd.DataFrame(  fa.components\_.T,  index=indicators,  columns=[f'公共因子{i+1}' for i in range(3)]  )  print("因子载荷矩阵（指标与公共因子的相关性）：")  print(loadings.round(3))  # 4. 基于因子得分的聚类分析  kmeans = KMeans(n\_clusters=3, random\_state=42)  clusters = kmeans.fit\_predict(factor\_scores)  # 5. 结果分析  result = pd.DataFrame(data, columns=indicators)  result['城市'] = cities  result['聚类标签'] = clusters  # 输出各类别城市的特征  print("\n各类别城市的平均指标：")  for cluster in range(3):  cluster\_data = result[result['聚类标签'] == cluster]  print(f"\n类别{cluster+1}（城市数量：{len(cluster\_data)}）：")  print(cluster\_data.iloc[:, :-2].mean().round(2))  # 6. 可视化聚类结果（前2个因子）  plt.figure(figsize=(10, 6))  plt.scatter(factor\_scores[:, 0], factor\_scores[:, 1], c=clusters, cmap='viridis', s=100)  plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1],  marker='X', s=300, c='red', label='聚类中心')  plt.xlabel('公共因子1（经济与资源）')  plt.ylabel('公共因子2（环境与就业）')  plt.title('城市宜居度聚类结果')  plt.legend()  plt.grid(True)  plt.show() |

### 10. 神经网络 + 遗传算法案例：化工反应参数优化

**问题背景**：某化工厂需优化某产品的生产参数（反应温度、压力、催化剂用量），目标是提高产品合格率（当前约 85%），参数间存在复杂非线性关系（如温度过高会降低催化剂活性），且参数调整成本高，试验次数有限。

**数据**：

* 50 组试验数据：反应温度（80-120℃）、压力（0.5-2MPa）、催化剂用量（1-5g）及对应的合格率（%）。

**要求**：用 BP 神经网络拟合参数与合格率的关系，用遗传算法优化参数（温度、压力、用量），输出最优参数组合及预期合格率，对比人工试错的效率提升。

### 10. 神经网络 + 遗传算法代码：化工反应参数优化

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.neural\_network import MLPRegressor  from sklearn.preprocessing import MinMaxScaler  from sklearn.metrics import mean\_squared\_error  # 1. 数据准备（50组试验数据）  np.random.seed(42)  n\_samples = 50  # 反应参数：温度(80-120℃)、压力(0.5-2MPa)、催化剂用量(1-5g)  temperature = np.random.uniform(80, 120, n\_samples)  pressure = np.random.uniform(0.5, 2, n\_samples)  catalyst = np.random.uniform(1, 5, n\_samples)  # 合格率（基于参数的非线性函数，加入噪声）  def quality\_rate(temp, press, cat):  # 模拟真实反应的非线性关系  base = 60 + 0.5\*temp - 5\*press + 3\*cat - 0.02\*(temp\*\*2) + 2\*(press\*\*2) - 0.5\*(cat\*\*2)  noise = np.random.normal(0, 2, len(temp))  return np.clip(base + noise, 50, 100) # 合格率控制在50%-100%  qualified\_rate = quality\_rate(temperature, pressure, catalyst)  X = np.column\_stack((temperature, pressure, catalyst))  y = qualified\_rate / 100 # 归一化到0-1  # 2. 数据预处理  scaler\_X = MinMaxScaler()  scaler\_y = MinMaxScaler()  X\_scaled = scaler\_X.fit\_transform(X)  y\_scaled = scaler\_y.fit\_transform(y.reshape(-1, 1)).flatten()  # 3. 构建BP神经网络（作为适应度函数的评估模型）  mlp = MLPRegressor(hidden\_layer\_sizes=(10, 10), activation='relu',  solver='adam', max\_iter=1000, random\_state=42)  mlp.fit(X\_scaled, y\_scaled)  # 4. 遗传算法优化参数  def genetic\_algorithm(model, scaler\_X, scaler\_y, pop\_size=50, generations=100, mutation\_rate=0.1):  # 参数范围：温度(80-120)、压力(0.5-2)、催化剂(1-5)  bounds = np.array([[80, 120], [0.5, 2], [1, 5]])  n\_params = bounds.shape[0]    # 初始化种群  population = np.random.rand(pop\_size, n\_params)  for i in range(n\_params):  population[:, i] = population[:, i] \* (bounds[i, 1] - bounds[i, 0]) + bounds[i, 0]    best\_scores = []  best\_params = None  best\_score = 0    for gen in range(generations):  # 计算适应度（合格率）  X\_pop\_scaled = scaler\_X.transform(population)  y\_pred\_scaled = model.predict(X\_pop\_scaled)  fitness = scaler\_y.inverse\_transform(y\_pred\_scaled.reshape(-1, 1)).flatten()    # 记录最优解  current\_best\_idx = np.argmax(fitness)  if fitness[current\_best\_idx] > best\_score:  best\_score = fitness[current\_best\_idx]  best\_params = population[current\_best\_idx]  best\_scores.append(best\_score)    # 选择（轮盘赌）  fitness\_norm = fitness / np.sum(fitness)  selected\_idx = np.random.choice(pop\_size, size=pop\_size, p=fitness\_norm)  selected = population[selected\_idx]    # 交叉（单点交叉）  offspring = []  for i in range(0, pop\_size, 2):  parent1 = selected[i]  parent2 = selected[i+1] if i+1 < pop\_size else selected[0]  cross\_point = np.random.randint(1, n\_params)  child1 = np.hstack((parent1[:cross\_point], parent2[cross\_point:]))  child2 = np.hstack((parent2[:cross\_point], parent1[cross\_point:]))  offspring.append(child1)  offspring.append(child2)  offspring = np.array(offspring[:pop\_size])    # 变异  for i in range(pop\_size):  if np.random.rand() < mutation\_rate:  mutate\_idx = np.random.randint(n\_params)  offspring[i, mutate\_idx] = np.random.uniform(bounds[mutate\_idx, 0], bounds[mutate\_idx, 1])    population = offspring    return best\_params, best\_scores  # 运行遗传算法  best\_params, best\_scores = genetic\_algorithm(mlp, scaler\_X, scaler\_y)  # 5. 结果展示  print(f"最优参数组合：")  print(f"温度：{best\_params[0]:.2f}℃")  print(f"压力：{best\_params[1]:.2f}MPa")  print(f"催化剂用量：{best\_params[2]:.2f}g")  print(f"预测最高合格率：{np.max(best\_scores):.2f}%")  # 可视化优化过程  plt.figure(figsize=(10, 6))  plt.plot(best\_scores)  plt.xlabel('进化代数')  plt.ylabel('最高合格率（%）')  plt.title('遗传算法优化过程')  plt.grid(True)  plt.show() |

### 11. SVM + 粒子群优化案例：医学影像病灶识别

**问题背景**：某医院需用 SVM 识别肺部 CT 影像中的结节（良性 / 恶性），影像特征含 7 项（如结节大小、密度、边缘光滑度等），SVM 的核参数（\gamma）和惩罚系数（C）对识别精度影响极大，人工调参耗时且效果差。

**数据**：

* 1000 份 CT 影像的 7 项特征及诊断结果（1 = 恶性，0 = 良性）。

**要求**：用粒子群优化搜索最优C和\gamma，训练 SVM 模型，对比默认参数与优化后参数的识别准确率（尤其是恶性结节的检出率），分析参数对结果的影响。

### 11. SVM + 粒子群优化代码：医学影像病灶识别

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.svm import SVC  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import classification\_report, confusion\_matrix  # 1. 数据准备（1000份CT影像特征）  np.random.seed(42)  n\_samples = 1000  # 7项特征：结节大小、密度、边缘光滑度、位置深度、增强程度、形状不规则度、钙化程度  X = np.random.randn(n\_samples, 7)  # 模拟特征与标签的关系（恶性结节=1，良性=0）  weights = np.array([0.8, 1.2, -0.9, 0.5, 1.0, 1.5, -0.7])  logits = np.dot(X, weights) + np.random.randn(n\_samples) \* 0.5  y = (logits > 0).astype(int) # 恶性结节占比约50%  features = ['结节大小', '密度', '边缘光滑度', '位置深度', '增强程度', '形状不规则度', '钙化程度']  # 2. 数据划分与标准化  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  # 3. 粒子群优化SVM参数（C和gamma）  def pso\_optimize\_svm(X, y, pop\_size=20, iterations=50, c1=2, c2=2, w=0.5):  # 参数范围：C(1e-3~1e3)，gamma(1e-3~1e3)（取对数空间）  def to\_params(position):  c = 10 \*\* position[0]  gamma = 10 \*\* position[1]  return c, gamma    # 适应度函数（5折交叉验证准确率）  def fitness(position):  c, gamma = to\_params(position)  svm = SVC(C=c, gamma=gamma, kernel='rbf', class\_weight='balanced', random\_state=42)  scores = cross\_val\_score(svm, X, y, cv=5, scoring='accuracy')  return np.mean(scores)    # 初始化粒子  dim = 2 # 优化参数维度：C和gamma  positions = np.random.uniform(-3, 3, (pop\_size, dim)) # 对数空间  velocities = np.random.uniform(-0.1, 0.1, (pop\_size, dim))  pbest\_pos = positions.copy()  pbest\_val = np.array([fitness(pos) for pos in positions])  gbest\_idx = np.argmax(pbest\_val)  gbest\_pos = pbest\_pos[gbest\_idx]  gbest\_val = pbest\_val[gbest\_idx]    # 迭代优化  history = [gbest\_val]  for \_ in range(iterations):  for i in range(pop\_size):  # 更新速度和位置  r1, r2 = np.random.rand(2)  velocities[i] = w \* velocities[i] + c1\*r1\*(pbest\_pos[i] - positions[i]) + c2\*r2\*(gbest\_pos - positions[i])  positions[i] += velocities[i]  # 边界限制  positions[i] = np.clip(positions[i], -3, 3)    # 更新个体最优  current\_val = fitness(positions[i])  if current\_val > pbest\_val[i]:  pbest\_val[i] = current\_val  pbest\_pos[i] = positions[i].copy()    # 更新全局最优  current\_gbest\_idx = np.argmax(pbest\_val)  if pbest\_val[current\_gbest\_idx] > gbest\_val:  gbest\_val = pbest\_val[current\_gbest\_idx]  gbest\_pos = pbest\_pos[current\_gbest\_idx].copy()  history.append(gbest\_val)    return to\_params(gbest\_pos), history  # 运行PSO优化  best\_params, pso\_history = pso\_optimize\_svm(X\_train\_scaled, y\_train)  print(f"优化后的SVM参数：C={best\_params[0]:.4f}, gamma={best\_params[1]:.4f}")  # 4. 优化后的SVM模型评估  svm\_opt = SVC(C=best\_params[0], gamma=best\_params[1], kernel='rbf', class\_weight='balanced', random\_state=42)  svm\_opt.fit(X\_train\_scaled, y\_train)  y\_pred = svm\_opt.predict(X\_test\_scaled)  # 对比默认参数模型  svm\_default = SVC(kernel='rbf', class\_weight='balanced', random\_state=42)  svm\_default.fit(X\_train\_scaled, y\_train)  y\_pred\_default = svm\_default.predict(X\_test\_scaled)  # 5. 结果展示  print("\n优化后模型分类报告：")  print(classification\_report(y\_test, y\_pred, target\_names=['良性', '恶性']))  print("\n默认参数模型分类报告：")  print(classification\_report(y\_test, y\_pred\_default, target\_names=['良性', '恶性']))  # 可视化优化过程  </doubaocanvas> |

### 12. 决策树 + 模糊逻辑案例：个人消费贷款审批

**问题背景**：某银行需制定个人消费贷款审批规则，指标含 6 项（如月收入、征信评分、负债比等），需明确 “若月收入> X 且征信评分 > Y，则批准贷款” 的规则，同时处理连续特征的模糊性（如 “月收入中等”）。

**数据**：

* 5000 条贷款申请记录：月收入（3000-50000 元）、征信评分（300-850 分）、负债比（0-100%）等，及审批结果（1 = 批准，0 = 拒绝）。

**要求**：用模糊逻辑将连续特征转为模糊集合（如 “月收入：低 / 中 / 高”），构建模糊决策树，输出可解释的审批规则，确保规则覆盖率 > 80% 且错误率 < 5%。

### 12. 决策树 + 模糊逻辑代码：个人消费贷款审批

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.preprocessing import KBinsDiscretizer  from sklearn.tree import DecisionTreeClassifier, plot\_tree  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import classification\_report, confusion\_matrix  # 1. 数据准备（5000条贷款申请记录）  np.random.seed(42)  n\_samples = 5000  # 6项指标：月收入(元)、征信评分(300-850)、负债比(%)、工作年限(年)、年龄(岁)、贷款金额(元)  monthly\_income = np.random.randint(3000, 50001, n\_samples)  credit\_score = np.random.randint(300, 851, n\_samples)  debt\_ratio = np.random.uniform(0, 100, n\_samples)  work\_years = np.random.randint(0, 31, n\_samples)  age = np.random.randint(18, 66, n\_samples)  loan\_amount = np.random.randint(10000, 500001, n\_samples)  # 生成审批标签（1=批准，0=拒绝）  # 核心逻辑：收入高、征信好、负债低更易批准  approve\_prob = (monthly\_income / 50000) + (credit\_score / 850) - (debt\_ratio / 100) + (work\_years / 30)  approve\_prob = np.clip(approve\_prob, 0, 2) / 2 # 归一化到0-1  y = np.where(np.random.rand(n\_samples) < approve\_prob, 1, 0)  print(f"贷款批准率：{sum(y)/n\_samples:.2%}")  # 特征矩阵  X = np.column\_stack([monthly\_income, credit\_score, debt\_ratio, work\_years, age, loan\_amount])  features = ['月收入', '征信评分', '负债比', '工作年限', '年龄', '贷款金额']  # 2. 模糊逻辑处理连续特征  def fuzzy\_discretize(data, n\_bins=3, feature\_names=None):  """将连续特征离散化为模糊集合（低/中/高）"""  discretizer = KBinsDiscretizer(n\_bins=n\_bins, encode='ordinal', strategy='quantile')  X\_disc = discretizer.fit\_transform(data).astype(int)    # 生成模糊集合名称  fuzzy\_names = []  for i in range(data.shape[1]):  bins = discretizer.bin\_edges\_[i]  names = [f'{feature\_names[i]}\_低', f'{feature\_names[i]}\_中', f'{feature\_names[i]}\_高']  fuzzy\_names.extend(names)    # 构建模糊特征矩阵（one-hot编码）  X\_fuzzy = np.zeros((data.shape[0], data.shape[1] \* n\_bins), dtype=int)  for i in range(data.shape[0]):  for j in range(data.shape[1]):  X\_fuzzy[i, j\*n\_bins + X\_disc[i, j]] = 1  return X\_fuzzy, fuzzy\_names  X\_fuzzy, fuzzy\_features = fuzzy\_discretize(X, feature\_names=features)  # 3. 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_fuzzy, y, test\_size=0.3, random\_state=42)  # 4. 构建模糊决策树  dt = DecisionTreeClassifier(max\_depth=5, min\_samples\_leaf=20, random\_state=42)  dt.fit(X\_train, y\_train)  # 5. 模型评估  y\_pred = dt.predict(X\_test)  print("\n贷款审批分类报告：")  print(classification\_report(y\_test, y\_pred, target\_names=['拒绝', '批准']))  # 6. 提取决策规则  plt.figure(figsize=(20, 10))  plot\_tree(dt, feature\_names=fuzzy\_features, class\_names=['拒绝', '批准'],  filled=True, rounded=True, fontsize=10)  plt.title('模糊决策树审批规则')  plt.show()  # 7. 规则示例解析  print("\n关键审批规则示例：")  print("1. 若 月收入\_高 且 征信评分\_高 且 负债比\_低 → 批准")  print("2. 若 月收入\_低 且 征信评分\_低 → 拒绝")  print("3. 若 月收入\_中 且 工作年限\_高 且 负债比\_中 → 批准") |

### 13. XGBoost + PCA案例：用户流失风险预测

**问题背景**：某视频平台需预测用户是否会取消会员订阅，用户特征含 50 项（如日均观看时长、互动评论数、付费历史等），存在大量冗余特征（如 “周均观看时长” 与 “日均观看时长” 强相关），导致模型过拟合。

**数据**：

* 10000 名会员的 50 项特征及流失标签（1 = 流失，0 = 留存）。

**要求**：用 PCA 降维至 10 个主成分，输入 XGBoost 模型，对比降维前后的预测精度（AUC 值）和训练时间，分析关键流失因素（如 “连续 3 天未登录”）。

### 13. XGBoost + PCA代码：用户流失风险预测

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.decomposition import PCA  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import roc\_auc\_score, classification\_report  import xgboost as xgb  # 1. 数据准备（10000名会员的50项特征）  np.random.seed(42)  n\_samples = 10000  n\_features = 50  # 生成高维特征（模拟用户行为：观看时长、评论数、付费次数等）  X = np.random.randn(n\_samples, n\_features)  # 核心特征（与流失强相关）  core\_features = np.random.choice(n\_features, 5, replace=False)  for i, idx in enumerate(core\_features):  if i < 3: # 正向特征（值越高越不易流失）  X[:, idx] = np.random.normal(5, 1, n\_samples)  else: # 负向特征（值越高越易流失）  X[:, idx] = np.random.normal(2, 1, n\_samples)  # 生成流失标签（1=流失，0=留存）  # 核心逻辑：核心特征决定大部分流失概率  流失概率 = 0.5 - 0.1\*X[:, core\_features[0]] - 0.1\*X[:, core\_features[1]] - 0.1\*X[:, core\_features[2]] + \  0.1\*X[:, core\_features[3]] + 0.1\*X[:, core\_features[4]]  流失概率 = np.clip(流失概率, 0, 1)  y = np.where(np.random.rand(n\_samples) < 流失概率, 1, 0)  print(f"用户流失率：{sum(y)/n\_samples:.2%}")  # 2. 数据标准化  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # 3. PCA降维  pca = PCA(n\_components=0.95, random\_state=42) # 保留95%信息  X\_pca = pca.fit\_transform(X\_scaled)  print(f"原始特征数：{n\_features}，降维后特征数：{pca.n\_components\_}")  # 4. 划分数据集  X\_train\_raw, X\_test\_raw, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3, random\_state=42)  X\_train\_pca, X\_test\_pca = train\_test\_split(X\_pca, test\_size=0.3, random\_state=42)  # 5. XGBoost模型训练与评估  def train\_xgboost(X\_train, X\_test, y\_train, y\_test, title):  model = xgb.XGBClassifier(  objective='binary:logistic',  n\_estimators=100,  max\_depth=5,  learning\_rate=0.1,  random\_state=42  )  model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  y\_prob = model.predict\_proba(X\_test)[:, 1]  auc = roc\_auc\_score(y\_test, y\_prob)  print(f"\n{title} AUC：{auc:.4f}")  print("分类报告：")  print(classification\_report(y\_test, y\_pred, target\_names=['留存', '流失']))  return model, auc  # 基于原始特征的XGBoost  model\_raw, auc\_raw = train\_xgboost(X\_train\_raw, X\_test\_raw, y\_train, y\_test, "原始特征XGBoost")  # 基于PCA降维特征的XGBoost  model\_pca, auc\_pca = train\_xgboost(X\_train\_pca, X\_test\_pca, y\_train, y\_test, "PCA降维XGBoost")  # 6. 结果对比  plt.figure(figsize=(8, 5))  plt.bar(['原始特征', 'PCA降维'], [auc\_raw, auc\_pca], color=['blue', 'orange'])  plt.ylim(0.5, 1.0)  plt.ylabel('AUC值（越大越好）')  plt.title('XGBoost模型AUC对比')  plt.grid(axis='y', alpha=0.3)  plt.show()  # 7. 特征重要性分析（降维后模型）  if hasattr(model\_pca, 'feature\_importances\_'):  plt.figure(figsize=(10, 6))  xgb.plot\_importance(model\_pca, max\_num\_features=10, title='PCA降维后特征重要性')  plt.show() |