### 11. 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。

### 11. 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() |