### 9. 贝叶斯优化（BO）+ 随机森林（RF）组合模型案例题目

**题目：新型光伏材料配方优化问题**

* **问题背景**：某材料实验室研发高效光伏组件，其核心材料的转化率受 5 种原料的配比影响（如硅片纯度、掺杂元素比例）。每次实验需 3 天且成本约 2 万元，传统试错法已进行 50 次实验，转化率最高仅 22%，远低于目标 25%。
* **问题描述**：需通过少量实验找到最优配方，目标是最大化光伏材料的光电转化率，约束条件为原料成本≤500 元 / 片，材料稳定性（耐温范围 - 40℃至 85℃）达标。需构建代理模型减少实验次数，兼顾优化效率与成本。
* **数据情况**：提供已完成的 50 次实验数据，包括 5 种原料的配比（如硅纯度 99.99%-99.999%）、实验环境参数（温度、湿度）、测得的转化率（%）、稳定性测试结果，以及各原料的单价。

### 9. 贝叶斯优化（BO）+ 随机森林（RF）求解新型光伏材料配方优化代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.ensemble import RandomForestRegressor  from sklearn.model\_selection import cross\_val\_score  from skopt import gp\_minimize  from skopt.space import Real  from skopt.plots import plot\_convergence  from skopt.utils import use\_named\_args  # 设置随机种子，保证结果可复现  np.random.seed(42)  # 数据生成（模拟光伏材料实验数据）  def generate\_material\_data(n\_initial=50):  """生成光伏材料配方与性能的模拟实验数据"""  # 配方参数范围：  # x1: 硅纯度（99.99%-99.999%）  # x2: 掺杂元素A比例（0.01%-0.1%）  # x3: 掺杂元素B比例（0.005%-0.05%）  # x4: 薄膜厚度（100-500nm）  # x5: 退火温度（500-1000℃）    # 生成初始实验参数  data = pd.DataFrame()  data['x1'] = np.random.uniform(99.99, 99.999, n\_initial)  data['x2'] = np.random.uniform(0.01, 0.1, n\_initial)  data['x3'] = np.random.uniform(0.005, 0.05, n\_initial)  data['x4'] = np.random.uniform(100, 500, n\_initial)  data['x5'] = np.random.uniform(500, 1000, n\_initial)    # 生成光电转化率（%）：基于非线性函数模拟  # 基础转化率（与硅纯度正相关）  base\_eff = 18 + 2 \* ((data['x1'] - 99.99) / 0.009)  # 掺杂元素影响（存在最优比例）  dopant\_eff = 3 \* np.exp(-5 \* (data['x2'] - 0.05)\*\*2) + 2 \* np.exp(-10 \* (data['x3'] - 0.02)\*\* 2)  # 薄膜厚度和退火温度影响  film\_temp\_eff = 1.5 \* np.sin(data['x4']/500 \* np.pi) + 2 \* np.sin(data['x5']/1000 \* np.pi)  # 随机测量误差  noise = np.random.normal(0, 0.3, n\_initial)    data['efficiency'] = base\_eff + dopant\_eff + film\_temp\_eff + noise  data['efficiency'] = np.clip(data['efficiency'], 17, 23) # 限制在合理范围内    # 生成制造成本（元/片）  data['cost'] = 100 + 50\*(data['x1'] - 99.99)/0.009 + 1000\*data['x2'] + 2000\*data['x3'] + 0.1\*data['x4'] + 0.05\*data['x5']    # 生成稳定性指标（-40℃至85℃循环测试后的衰减率）  data['stability'] = 1 - (0.02 \* (data['x4'] < 200) + 0.03 \* (data['x5'] < 600) + np.random.normal(0, 0.01, n\_initial))  data['stability'] = np.clip(data['stability'], 0.85, 0.99)    return data  # 定义参数空间  param\_space = [  Real(99.99, 99.999, name='x1'), # 硅纯度  Real(0.01, 0.1, name='x2'), # 掺杂元素A比例  Real(0.005, 0.05, name='x3'), # 掺杂元素B比例  Real(100, 500, name='x4'), # 薄膜厚度  Real(500, 1000, name='x5') # 退火温度  ]  # 贝叶斯优化目标函数  def objective\_function(params, model, X\_train, y\_train, cost\_coef=0.1):  """  目标函数：最大化(光电转化率 - 成本惩罚)  加入稳定性约束  """  x1, x2, x3, x4, x5 = params    # 稳定性约束：薄膜厚度≥200nm，退火温度≥600℃  if x4 < 200 or x5 < 600:  return -np.inf # 不稳定配方惩罚    # 成本计算与惩罚  cost = 100 + 50\*(x1 - 99.99)/0.009 + 1000\*x2 + 2000\*x3 + 0.1\*x4 + 0.05\*x5  if cost > 500: # 成本上限约束  return -np.inf    # 用随机森林预测转化率  sample = np.array([[x1, x2, x3, x4, x5]])  pred\_eff = model.predict(sample)[0]    # 加入小噪声模拟实验误差  pred\_eff += np.random.normal(0, 0.1)    # 目标值：转化率 - 成本惩罚  return pred\_eff - cost \* cost\_coef  # 主优化流程  def optimize\_material():  # 生成初始实验数据  initial\_data = generate\_material\_data(n\_initial=60)  print(f"初始实验数据规模: {initial\_data.shape}")  print(f"初始最高转化率: {initial\_data['efficiency'].max():.2f}%")  print(f"初始最低成本: {initial\_data['cost'].min():.2f}元/片")    # 划分特征与目标  X\_train = initial\_data[['x1', 'x2', 'x3', 'x4', 'x5']].values  y\_train = initial\_data['efficiency'].values    # 训练随机森林代理模型  rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)  rf\_model.fit(X\_train, y\_train)    # 交叉验证评估模型性能  cv\_scores = cross\_val\_score(rf\_model, X\_train, y\_train, cv=5, scoring='neg\_mean\_absolute\_error')  print(f"随机森林交叉验证MAE: {np.mean(-cv\_scores):.3f}%")    # 定义带参数的目标函数  @use\_named\_args(param\_space)  def objective(\*\*params):  return -objective\_function(list(params.values()), rf\_model, X\_train, y\_train)    # 执行贝叶斯优化  print("\n开始贝叶斯优化...")  result = gp\_minimize(  objective,  param\_space,  n\_calls=40, # 额外实验次数  random\_state=42,  verbose=True,  n\_random\_starts=10 # 初始随机采样次数  )    # 提取最优配方  best\_params = result.x  best\_efficiency = -result.fun + 0.1 \* (100 + 50\*(best\_params[0] - 99.99)/0.009 +  1000\*best\_params[1] + 2000\*best\_params[2] +  0.1\*best\_params[3] + 0.05\*best\_params[4])    # 计算最优配方的成本和稳定性  best\_cost = 100 + 50\*(best\_params[0] - 99.99)/0.009 + 1000\*best\_params[1] + 2000\*best\_params[2] + 0.1\*best\_params[3] + 0.05\*best\_params[4]  best\_stability = 0.95 + 0.02\*(best\_params[3] > 300) + 0.02\*(best\_params[4] > 700) # 估计稳定性    print("\n优化结果:")  print(f"最优光电转化率: {best\_efficiency:.2f}%")  print(f"最优配方成本: {best\_cost:.2f}元/片")  print(f"估计稳定性: {best\_stability:.3f}")  print("最优配方参数:")  print(f"硅纯度: {best\_params[0]:.4f}%")  print(f"掺杂元素A比例: {best\_params[1]:.4f}%")  print(f"掺杂元素B比例: {best\_params[2]:.4f}%")  print(f"薄膜厚度: {best\_params[3]:.1f}nm")  print(f"退火温度: {best\_params[4]:.1f}℃")    # 可视化优化过程  plot\_results(initial\_data, result, best\_params, rf\_model)    return best\_params, best\_efficiency, best\_cost  # 结果可视化  def plot\_results(initial\_data, optimization\_result, best\_params, model):  """可视化优化过程和结果"""  plt.figure(figsize=(15, 12))    # 1. 优化收敛曲线  plt.subplot(2, 2, 1)  plot\_convergence(optimization\_result)  plt.title('贝叶斯优化收敛曲线')    # 2. 初始数据转化率分布  plt.subplot(2, 2, 2)  sns.histplot(initial\_data['efficiency'], kde=True)  plt.axvline(x=initial\_data['efficiency'].max(), color='r', linestyle='--', label=f'初始最大值: {initial\_data["efficiency"].max():.2f}%')  plt.axvline(x=-optimization\_result.fun + 0.1\*500, color='g', linestyle='--', label=f'优化后值: {(-optimization\_result.fun + 0.1\*500):.2f}%')  plt.title('初始实验转化率分布')  plt.xlabel('光电转化率（%）')  plt.legend()    # 3. 参数相关性热图  plt.subplot(2, 2, 3)  corr = initial\_data[['x1', 'x2', 'x3', 'x4', 'x5', 'efficiency', 'cost']].corr()  sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')  plt.title('参数相关性热图')    # 4. 最优配方与初始最优对比  plt.subplot(2, 2, 4)  initial\_best\_idx = initial\_data['efficiency'].idxmax()  initial\_best = initial\_data.iloc[initial\_best\_idx]    # 转换参数到相同尺度以便对比  params\_names = ['硅纯度', '掺杂A比例', '掺杂B比例', '薄膜厚度', '退火温度']  norm\_initial = [(initial\_best[col] - param\_space[i].low)/(param\_space[i].high - param\_space[i].low)  for i, col in enumerate(['x1', 'x2', 'x3', 'x4', 'x5'])]  norm\_best = [(best\_params[i] - param\_space[i].low)/(param\_space[i].high - param\_space[i].low)  for i in range(5)]    x = np.arange(len(params\_names))  width = 0.35  plt.bar(x - width/2, norm\_initial, width, label='初始最优')  plt.bar(x + width/2, norm\_best, width, label='优化后最优')  plt.xticks(x, params\_names, rotation=30)  plt.title('最优配方参数对比（归一化）')  plt.ylabel('归一化值（0-1）')  plt.legend()    plt.tight\_layout()  plt.show()  # 执行优化  if \_\_name\_\_ == "\_\_main\_\_":  optimize\_material()  ### 10. 蚁群算法（ACO）+ 粒子群优化（PSO）求解智能物流路径规划  ```python  import numpy as np  import matplotlib.pyplot as plt  import random  import seaborn as sns  from matplotlib.patches import Circle  # 设置随机种子，保证结果可复现  np.random.seed(42)  random.seed(42)  # 问题数据初始化  def initialize\_logistics\_data(n\_customers=20, n\_vehicles=3, vehicle\_capacity=500):  """初始化物流配送问题数据"""  # 配送中心坐标  depot = np.array([50, 50])    # 客户坐标（随机分布在10-90范围内）  customers = np.random.rand(n\_customers, 2) \* 80 + 10    # 客户需求（10-50单位）  demands = np.random.randint(10, 51, size=n\_customers)    # 时间窗口（[最早时间, 最晚时间]）  time\_windows = []  for i in range(n\_customers):  start = random.randint(60, 240) # 1-4小时  end = start + random.randint(60, 180) # 持续1-3小时  time\_windows.append([start, end])    # 服务时间（5-15分钟）  service\_times = np.random.randint(5, 16, size=n\_customers)    return {  'depot': depot,  'customers': customers,  'demands': demands,  'time\_windows': np.array(time\_windows),  'service\_times': service\_times,  'n\_customers': n\_customers,  'n\_vehicles': n\_vehicles,  'vehicle\_capacity': vehicle\_capacity,  'vehicle\_speed': 0.5 # 单位距离所需时间（分钟）  }  # 距离计算  def calculate\_distance(point1, point2):  """计算两点间欧氏距离"""  return np.sqrt(np.sum((point1 - point2)\*\* 2))  # 路径成本计算  def route\_cost(route, data):  """计算单条路径的总成本（距离+时间惩罚）"""  if not route:  return 0    # 路径点坐标（包括起点和终点）  points = [data['depot']] + [data['customers'][i] for i in route] + [data['depot']]  total\_distance = 0  total\_time = 0  time\_penalty = 0  current\_time = 0    # 计算距离和时间惩罚  for i in range(len(points) - 1):  # 距离成本  dist = calculate\_distance(points[i], points[i+1])  total\_distance += dist    # 时间计算  travel\_time = dist / data['vehicle\_speed']  current\_time += travel\_time    # 如果是客户点，检查时间窗口  if i > 0 and i < len(points) - 1:  customer\_idx = route[i-1]  # 早到惩罚  if current\_time < data['time\_windows'][customer\_idx][0]:  wait\_time = data['time\_windows'][customer\_idx][0] - current\_time  time\_penalty += wait\_time \* 0.5 # 早到惩罚系数  current\_time = data['time\_windows'][customer\_idx][0]  # 迟到惩罚（更严重）  elif current\_time > data['time\_windows'][customer\_idx][1]:  delay\_time = current\_time - data['time\_windows'][customer\_idx][1]  time\_penalty += delay\_time \* 2 # 迟到惩罚系数    # 加上服务时间  current\_time += data['service\_times'][customer\_idx]    # 总时间惩罚加入总成本  total\_cost = total\_distance + time\_penalty \* 0.1  return total\_cost, total\_distance, time\_penalty  # 蚁群算法（ACO）组件  class AntColonyOptimizer:  def \_\_init\_\_(self, data, n\_ants=30, alpha=1.0, beta=2.0, rho=0.1, Q=100):  self.data = data  self.n\_ants = n\_ants  self.alpha = alpha # 信息素重要性  self.beta = beta # 启发式信息重要性  self.rho = rho # 信息素蒸发率  self.Q = Q # 信息素增量常数    # 初始化信息素矩阵  n = data['n\_customers']  self.pherom</doubaocanvas> |