### 4. 支持向量机（SVM）+ 遗传算法（GA）组合模型案例题目

**题目：锂电池生产参数优化与质量预测问题**

* **问题背景**：某锂电池工厂的电芯生产中，“匀浆 - 涂布 - 辊压” 工序的参数设置直接影响电芯容量和循环寿命。当前采用经验参数（如浆料固含量 50%、涂布速度 3m/min），导致电芯合格率仅 82%，且不同批次的容量偏差达 15%。工厂希望通过数据驱动方法优化参数，提升产品一致性。
* **问题描述**：需构建参数优化模型，目标包括：① 最大化电芯容量（≥3.6Ah）；② 最大化循环寿命（≥1000 次）；③ 最小化参数调整成本（如固含量每变化 1% 成本增加 200 元 / 批次）。需通过历史数据训练模型，找到最优参数组合（固含量 45%-55%、涂布速度 2-4m/min、辊压压力 10-20MPa 等）。
* **数据情况**：提供 1000 批次的生产数据，包括 5 个关键工艺参数的具体值、对应批次的电芯容量检测值、循环寿命测试结果、参数调整的成本记录，以及部分不合格品的缺陷类型（如析锂、鼓包）。

### 4. 支持向量机（SVM）+ 遗传算法（GA）求解锂电池生产参数优化代码

|  |
| --- |
| import numpy as np  import random  import matplotlib.pyplot as plt  from sklearn.svm import SVR  from sklearn.preprocessing import MinMaxScaler  from sklearn.metrics import mean\_squared\_error  # 数据生成与预处理  np.random.seed(42)  # 生成1000批次生产数据  n\_samples = 1000  # 5个工艺参数：固含量(45-55)、涂布速度(2-4)、辊压压力(10-20)、温度(80-120)、时间(30-60)  X = np.random.uniform(  low=[45, 2, 10, 80, 30],  high=[55, 4, 20, 120, 60],  size=(n\_samples, 5)  )  # 电芯容量（3.2-3.8Ah）- 模拟非线性关系  capacity = 3.2 + 0.01\*(X[:,0]-45) + 0.05\*(X[:,1]-2) + 0.02\*(X[:,2]-10) + \  0.005\*(X[:,3]-80) + 0.01\*(X[:,4]-30) - 0.001\*(X[:,0]-50)\*\*2 + \  np.random.normal(0, 0.03, n\_samples)  capacity = np.clip(capacity, 3.2, 3.8)  # 循环寿命（800-1200次）  cycle\_life = 800 + 5\*(X[:,0]-45) + 30\*(X[:,1]-2) + 10\*(X[:,2]-10) + \  2\*(X[:,3]-80) + 3\*(X[:,4]-30) - 0.5\*(X[:,1]-3)\*\*2 + \  np.random.normal(0, 10, n\_samples)  cycle\_life = np.clip(cycle\_life, 800, 1200)  # 数据归一化  scaler = MinMaxScaler()  X\_scaled = scaler.fit\_transform(X)  # 遗传算法优化SVM参数  class GA\_SVM\_Optimizer:  def \_\_init\_\_(self, X, y, pop\_size=30, generations=50):  self.X = X  self.y = y  self.pop\_size = pop\_size  self.generations = generations  # 参数范围：C(1-100), gamma(0.001-1)  self.param\_ranges = [(1, 100), (0.001, 1)]    # 初始化种群  def init\_population(self):  pop = []  for \_ in range(self.pop\_size):  # 随机生成参数  C = random.uniform(\*self.param\_ranges[0])  gamma = random.uniform(\*self.param\_ranges[1])  pop.append((C, gamma))  return pop    # 适应度函数（SVM的交叉验证性能）  def fitness(self, params):  C, gamma = params  # 构建SVM回归模型  svm = SVR(C=C, gamma=gamma, kernel='rbf')  # 简单交叉验证  n\_split = 5  mse\_list = []  for i in range(n\_split):  idx = np.arange(len(self.X))  np.random.shuffle(idx)  train\_idx = idx[:int(0.8\*len(idx))]  test\_idx = idx[int(0.8\*len(idx)):]  svm.fit(self.X[train\_idx], self.y[train\_idx])  y\_pred = svm.predict(self.X[test\_idx])  mse = mean\_squared\_error(self.y[test\_idx], y\_pred)  mse\_list.append(mse)  # 适应度为MSE的倒数  return 1 / np.mean(mse\_list)    # 选择操作  def select(self, pop, fitnesses):  total\_fit = sum(fitnesses)  probs = [f / total\_fit for f in fitnesses]  return random.choices(pop, probs, k=2)    # 交叉操作  def crossover(self, p1, p2):  # 算术交叉  alpha = random.uniform(0, 1)  c1 = (alpha\*p1[0] + (1-alpha)\*p2[0], alpha\*p1[1] + (1-alpha)\*p2[1])  c2 = ((1-alpha)\*p1[0] + alpha\*p2[0], (1-alpha)\*p1[1] + alpha\*p2[1])  return c1, c2    # 变异操作  def mutate(self, params, rate=0.1):  C, gamma = params  if random.random() < rate:  C \*= random.uniform(0.8, 1.2)  C = np.clip(C, \*self.param\_ranges[0])  if random.random() < rate:  gamma \*= random.uniform(0.8, 1.2)  gamma = np.clip(gamma, \*self.param\_ranges[1])  return (C, gamma)    # 优化主函数  def optimize(self):  pop = self.init\_population()  best\_params = pop[0]  best\_fitness = self.fitness(best\_params)    for gen in range(self.generations):  fitnesses = [self.fitness(params) for params in pop]  current\_best\_idx = np.argmax(fitnesses)  if fitnesses[current\_best\_idx] > best\_fitness:  best\_params = pop[current\_best\_idx]  best\_fitness = fitnesses[current\_best\_idx]    new\_pop = [best\_params] # 精英保留    while len(new\_pop) < self.pop\_size:  p1, p2 = self.select(pop, fitnesses)  c1, c2 = self.crossover(p1, p2)  c1 = self.mutate(c1)  c2 = self.mutate(c2)  new\_pop.append(c1)  if len(new\_pop) < self.pop\_size:  new\_pop.append(c2)    pop = new\_pop  if gen % 10 == 0:  print(f"迭代 {gen}, 最优适应度: {best\_fitness:.4f}")    return best\_params  # 主函数：优化容量和寿命的加权目标  if \_\_name\_\_ == "\_\_main\_\_":  # 加权目标（容量和寿命各占50%）  y = 0.5 \* capacity + 0.5 \* (cycle\_life / 100) # 归一化寿命    # 用GA优化SVM参数  ga\_svm = GA\_SVM\_Optimizer(X\_scaled, y)  best\_params = ga\_svm.optimize()  print(f"最优SVM参数: C={best\_params[0]:.2f}, gamma={best\_params[1]:.4f}")    # 训练最优SVM模型  best\_svm = SVR(C=best\_params[0], gamma=best\_params[1], kernel='rbf')  best\_svm.fit(X\_scaled, y)    # 搜索最优参数组合（在参数空间采样）  n\_search = 100  test\_params = np.random.uniform(  low=[45, 2, 10, 80, 30],  high=[55, 4, 20, 120, 60],  size=(n\_search, 5)  )  test\_scaled = scaler.transform(test\_params)  y\_pred = best\_svm.predict(test\_scaled)  best\_idx = np.argmax(y\_pred)  best\_production = test\_params[best\_idx]    print("最优生产参数组合:")  print(f"固含量: {best\_production[0]:.2f}%")  print(f"涂布速度: {best\_production[1]:.2f}m/min")  print(f"辊压压力: {best\_production[2]:.2f}MPa")  print(f"温度: {best\_production[3]:.2f}℃")  print(f"时间: {best\_production[4]:.2f}分钟")    # 可视化参数影响（固含量vs容量）  plt.figure(figsize=(10, 6))  plt.scatter(X[:,0], capacity, alpha=0.5, label='原始数据')  plt.xlabel('固含量 (%)')  plt.ylabel('电芯容量 (Ah)')  plt.title('固含量对电芯容量的影响')  plt.legend()  plt.show()  ```</dou |