**【预测类】创新型算法汇总**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **编号** | **组合模型** | **主要优势** | **常见案例** | **适合赛题** |
| 1 | ARIMA + LSTM | 结合 ARIMA 的线性趋势捕捉能力与 LSTM 的非线性拟合能力，兼顾序列中的线性规律与复杂波动。 | 短期电力负荷预测、股票价格预测、商品销量预测（含线性趋势 + 非线性波动）。 | 含线性趋势 + 非线性波动预测问题 |
| 2 | 随机森林（RF） + 遗传算法（GA） | GA 优化 RF 的超参数（树数量、深度等），解决 RF 参数敏感问题，提升模型泛化能力与预测精度。 | 房价预测、疾病风险预测、客户流失预测（高维特征 + 参数依赖场景）。 | 高维特征 + 参数依赖场景 |
| 3 | CNN + LSTM | CNN 提取多变量数据的空间关联特征（如区域相关性），LSTM 捕捉时间序列动态，实现 “时空双维度” 建模。 | 交通流量预测（路段空间关联 + 时间变化）、气象预测（多站点数据时空耦合）、视频帧预测。 | 涉及多变量多维度的预测问题 |
| 4 | SVM + 粒子群优化（PSO） | PSO 优化 SVM 的核函数参数与惩罚因子，解决 SVM 参数调试复杂、易过拟合的问题。 | 信用评分、产品次品率预测、信用卡欺诈检测（小样本 + 高维度特征）。 | 小样本 + 高维度特征 |
| 5 | 灰色预测（GM (1,1)） + BP 神经网络 | GM 处理小样本线性趋势，BP 拟合非线性残差，适合数据量少、规律复杂的场景。 | 稀有金属价格预测、设备早期故障预测（小样本数据）、新兴市场需求预测。 | 小样本预测 |
| 6 | GRU + 注意力机制（Attention） | GRU 简化 LSTM 结构提升效率，Attention 聚焦关键时间步（如季节性节点），增强长序列预测能力。 | 长期气象预测（关注季节转换节点）、自然语言文本生成（关注上下文关键信息）。 | 长序列预测 |
| 7 | XGBoost + 贝叶斯优化 | 贝叶斯优化自动寻优 XGBoost 超参数（学习率、树深度等），避免手动调参盲目性，提升效率与精度。 | 电商销量预测、用户点击率（CTR）预测、疾病发病率预测（高维特征 + 强非线性）。 | 高维特征 + 强非线性 |
| 8 | 马尔可夫链 + 指数平滑法（ETS） | ETS 处理趋势与季节性，马尔可夫链捕捉状态随机转移（如突发需求），提升抗干扰能力。 | 零售商品短期需求预测（含促销 / 节日随机波动）、库存水平动态预测。 |  |
| 9 | 小波分析 + LSTM | 小波分析分解序列为不同频率成分（分离噪声与信号），LSTM 分别预测后合成，增强抗噪声能力。 | 汇率预测（受高频噪声影响）、地震波信号预测、机械振动故障预测。 | 含噪音的预测 |
| 10 | LightGBM + 堆叠集成（Stacking） | LightGBM 作为基模型，Stacking 整合多模型输出（如 LSTM、SVM），降低过拟合风险，提升稳定性。 | 用户流失预测、金融违约概率预测（需高稳健性场景）。 | 高稳健性场景预测 |
| 11 | 模糊神经网络（FNN） + GA | GA 优化 FNN 的隶属度函数与权重，增强对模糊信息（如 “中等风险”）的处理能力，提升非线性拟合。 | 水质污染指数预测（评价标准模糊）、化工反应参数优化预测。 | 模糊性预测 |
| 12 | SARIMA + GARCH | SARIMA 处理趋势与季节性，GARCH 捕捉波动率聚类（如价格突变），适合波动剧烈的序列。 | 股票波动率预测、大宗商品价格波动预测（高风险市场）。 | 高风险市场预测 |
| 13 | 决策树（DT） + AdaBoost | AdaBoost 迭代优化决策树，聚焦错分样本，将弱分类器升级为强分类器，提升预测稳健性。 | 汽车故障诊断、客户信用等级划分（需高容错性场景）。 | 需高容错性场景预测 |
| 14 | LSTM + SVR | LSTM 处理复杂非线性时序，SVR 修正残差（降低 LSTM 过拟合），实现 “主预测 + 精修正” 双阶段优化。 | 风速预测（可再生能源发电规划）、交通流短时预测（突发干扰修正）。 | 含突发情况的预测 |
| 15 | 混沌时间序列 + PSO-BP | 混沌理论解析序列内在随机性，PSO 优化 BP 神经网络权重，增强对混沌系统（如电力负荷）的拟合。 | 电力系统负荷预测（含混沌特性）、金融时间序列预测（非线性混沌特征）。 | 含混沌特性的预测 |

# **一、15种组合预测模型的详细解析**

### 1. ARIMA + LSTM

* **基本原理**：ARIMA 由自回归（AR）、差分（I）和移动平均（MA）构成。通过 d 阶差分将非平稳序列平稳化，再用 p 阶自回归项捕捉滞后相关性，q 阶移动平均项消除随机干扰，以此捕捉线性趋势与周期性。LSTM 作为特殊 RNN，凭借遗忘门、输入门和输出门，解决长序列梯度问题，捕捉非线性特征与长期依赖。二者结合，ARIMA 处理线性趋势，LSTM 拟合非线性波动，实现优势互补。
* **核心公式推导**
  + ARIMA (p,d,q)：，其中为 d 阶差分，是自回归系数，是移动平均系数，为白噪声。
  + LSTM 遗忘门：；输入门：；细胞状态更新：；细胞状态：；输出门：；隐藏状态：。
* **主要优势**：融合线性趋势捕捉与非线性拟合能力，能同时处理规则线性变化与复杂非线性波动，在混合特征数据中预测精度高、适应性强。
* **适合赛题**：短期电力负荷预测（含日常 / 季节性线性趋势与极端天气等非线性波动）、股票价格波动预测（长期宏观经济线性趋势与短期市场情绪等非线性波动）。

### 2. 随机森林（RF） + 遗传算法（GA）

* **基本原理**：RF 通过 Bootstrap 抽样构建多个决策树，每棵树节点随机选特征分裂，最后投票或平均输出，降低过拟合，提升泛化能力。GA 模拟生物进化，将 RF 超参数编码为染色体，经选择、交叉、变异迭代，筛选优参，解决 RF 参数敏感问题，提升性能。
* **核心公式推导**
  + RF 输出：分类用，回归用，其中为树数量，是第 k 棵树预测值。
  + GA 适应度函数：，，为超参数向量。
* **主要优势**：GA 的全局寻优能力优化 RF 超参数，避免手动调参盲目性，提升预测精度、稳定性与泛化能力，适合高维特征数据。
* **适合赛题**：房价预测（受多高维特征影响，关系复杂）、客户流失预警（受多种因素影响，参数对结果影响大）。

### 3. CNN + LSTM

* **基本原理**：CNN 以卷积操作为核心，卷积层用卷积核提取局部与空间关联特征，池化层降维保关键特征。LSTM 捕捉时序动态与长期依赖。结合后，CNN 提取空间特征，LSTM 处理特征序列的时间变化，实现时空联合建模。
* **核心公式推导**
  + CNN 卷积：，是第 l 层（i,j）输出，为卷积核权重，是激活函数。
  + LSTM 公式同前，输入为 CNN 提取的空间特征序列。
* **主要优势**：融合空间特征提取与时间动态捕捉能力，对时空关联数据预测效果远超单一模型。
* **适合赛题**：城市交通流量预测（路段空间关联且随时间变化）、区域气象变化预测（多站点数据空间关联且时序变化）。

### 4. 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，找最优超平面实现分类或回归，但性能依赖核参数与惩罚因子。PSO 模拟鸟群觅食，粒子代表 SVM 参数组合，跟踪自身与群体最优位置更新速度和位置，寻优参数，优化 SVM 性能。
* **核心公式推导**
  + SVM 回归：，是核函数，为拉格朗日乘子。
  + PSO 速度更新：；位置更新：，为惯性权重，是加速系数。
* **主要优势**：PSO 高效寻优 SVM 参数，发挥 SVM 在小样本、高维数据上的优势，提升预测精度与泛化能力。
* **适合赛题**：信用评分（样本量小、特征维度高）、产品次品率预测（受多高维因素影响，样本可能不平衡）。

### 5. 灰色预测（GM (1,1)） + BP 神经网络

* **基本原理**：GM (1,1) 对小样本数据累加生成近似指数序列，建一阶线性微分方程预测，再累减还原，擅长线性趋势预测，但非线性残差大。BP 神经网络通过反向传播调整权重和偏置，拟合非线性函数。结合后，GM (1,1) 初步预测，BP 拟合残差，修正非线性偏差。
* **核心公式推导**
  + GM (1,1)：原始序列，累加得，微分方程，参数，解为，还原。
  + BP 输出：，为残差，最终预测。
* **主要优势**：适合小样本数据，兼顾线性趋势与非线性波动，提升预测准确性。
* **适合赛题**：稀有资源价格预测（数据量少，受政策等非线性因素影响）、新兴市场需求预测（数据有限，需求波动非线性）。

### 6. GRU + 注意力机制（Attention）

* **基本原理**：GRU 简化 LSTM 结构，用更新门和重置门捕捉时序特征，提升效率。注意力机制为不同时间步分配权重，聚焦关键信息（如季节性节点）。结合后，GRU 高效处理序列，Attention 强化关键步影响，增强长序列预测能力。
* **核心公式推导**
  + GRU 更新门：；重置门：；候选隐藏状态；隐藏状态。
  + 注意力权重：，为相似度函数，是上下文向量。
* **主要优势**：高效处理长序列，聚焦关键信息，提升长期预测精度。
* **适合赛题**：长期气象预测（关注季节转换节点）、自然语言文本生成（关注上下文关键信息）。

### 7. XGBoost + 贝叶斯优化

* **基本原理**：XGBoost 基于梯度提升框架，用正则化控制复杂度，性能优但依赖超参数。贝叶斯优化用高斯过程建模参数与性能关系，高效搜索最优超参数（如学习率、树深度），避免手动调参盲目性。
* **核心公式推导**
  + XGBoost 目标函数：，，为叶节点数，为叶节点权重。
  + 贝叶斯优化期望改进：，为当前最优参数。
* **主要优势**：自动寻优超参数，提升模型效率与精度，适合高维、强非线性数据。
* **适合赛题**：电商销量预测（高维特征，受促销等强非线性因素影响）、用户点击率预测（特征多，非线性关系显著）。

### 8. 马尔可夫链 + 指数平滑法（ETS）

* **基本原理**：ETS 分解序列为趋势、季节性和误差项，平滑预测。马尔可夫链描述状态随机转移，捕捉突发波动（如促销需求）。结合后，ETS 处理趋势与季节性，马尔可夫链修正随机波动，提升抗干扰能力。
* **核心公式推导**
  + ETS(A,N,N)：，，为水平项，是平滑系数。
  + 马尔可夫链转移概率：，状态预测，为状态概率向量。
* **主要优势**：兼顾趋势 / 季节性与随机波动，提升模型稳定性与抗干扰性。
* **适合赛题**：零售商品短期需求预测（含促销 / 节日随机波动）、库存水平动态预测（受突发需求影响）。

### 9. 小波分析 + LSTM

* **基本原理**：小波分析用小波基函数将序列分解为不同频率子序列（分离噪声与信号），LSTM 分别预测子序列后重构，增强抗噪声能力。
* **核心公式推导**
  + 小波分解：，为小波函数。
  + 重构：，是 LSTM 预测的系数。
* **主要优势**：分离噪声与有效信号，提升含噪序列预测精度。
* **适合赛题**：汇率预测（受高频投机噪声影响）、机械振动故障预测（含噪声信号）。

### 10. LightGBM + 堆叠集成（Stacking）

* **基本原理**：LightGBM 用直方图算法和 Leaf-wise 生长策略高效处理数据，作为基模型。Stacking 用元模型（如逻辑回归）整合多基模型（含 LightGBM、LSTM 等）输出，降低过拟合，提升稳定性。
* **核心公式推导**
  + LightGBM 分裂增益：，为梯度和，为 Hessian 和。
  + Stacking 输出：，是第 k 个基模型预测值。
* **主要优势**：集成多模型优势，提升预测稳定性与精度。
* **适合赛题**：用户流失预测（需高稳健性）、金融违约概率预测（对稳定性要求高）。

### 11. 模糊神经网络（FNN） + GA

* **基本原理**：FNN 结合模糊逻辑与神经网络，用隶属度函数处理模糊信息（如 “中等风险”）。GA 优化 FNN 的隶属度函数参数与连接权重，增强非线性拟合能力。
* **核心公式推导**
  + FNN 输出：，为规则激活度，是规则输出。
  + GA 优化目标：，通过进化操作更新参数。
* **主要优势**：增强对模糊信息的处理能力，提升复杂系统预测精度。
* **适合赛题**：水质污染指数预测（评价标准模糊）、化工反应参数优化预测（参数关系模糊）。

### 12. SARIMA + GARCH

* **基本原理**：SARIMA 捕捉序列趋势、季节性和线性自相关。GARCH 描述波动率聚类特性（如价格突变），修正 SARIMA 对波动剧烈序列的偏差。
* **核心公式推导**
  + SARIMA (p,d,q)(P,D,Q) s：含季节差分、季节 AR和 MA。
  + GARCH(p,q)：，为条件方差。
* **主要优势**：兼顾趋势 / 季节性与波动率聚类，适合波动剧烈序列。
* **适合赛题**：股票波动率预测（波动剧烈且聚类）、大宗商品价格波动预测（高风险市场）。

### 13. 决策树（DT） + AdaBoost

* **基本原理**：决策树通过特征分裂构建模型，AdaBoost 迭代生成弱分类器（决策树），对错误样本加权，将弱分类器集成为强分类器，提升容错性与稳健性。
* **核心公式推导**
  + 决策树输出：，为树结构参数。
  + AdaBoost 权重更新：，，为第 t 轮误差。
* **主要优势**：聚焦错分样本，提升模型容错性与稳健性。
* **适合赛题**：汽车故障诊断（需高容错性）、客户信用等级划分（对错误分类敏感）。

### 14. LSTM + SVR

* **基本原理**：LSTM 处理复杂非线性时序，输出初步预测，SVR 拟合 LSTM 的预测残差，修正结果，降低过拟合，实现 “主预测 + 精修正”。
* **核心公式推导**
  + LSTM 预测：，残差。
  + SVR 残差预测：，最终。
* **主要优势**：结合非线性捕捉与残差修正能力，提升预测精度。
* **适合赛题**：风速预测（可再生能源规划，需修正湍流干扰）、交通流短时预测（受突发干扰）。

### 15. 混沌时间序列 + PSO-BP

* **基本原理**：混沌时间序列具有内在随机性、初值敏感性和短期可预测性等特点。其分析通过相空间重构技术，将一维的混沌时间序列映射到高维相空间，从而揭示序列中隐藏的确定性规律和内在结构，为后续预测提供更有效的信息。PSO-BP 则是结合了粒子群优化（PSO）和 BP 神经网络的模型，BP 神经网络能逼近复杂的非线性函数，但存在易陷入局部最优、收敛速度慢等问题。PSO 算法通过模拟粒子群的社会行为，对 BP 神经网络的初始权重和阈值进行优化，有效改善 BP 神经网络的性能，提高其收敛速度和预测精度。将两者结合，先用混沌时间序列分析对数据进行相空间重构，再利用 PSO-BP 模型对重构后的相空间数据进行预测，能充分挖掘混沌系统的规律，提升预测效果。
* **核心公式推导**
  + 相空间重构：对于给定的一维混沌时间序列，其中为数据长度，根据 Takens 定理，选择合适的嵌入维数和延迟时间，构建维相空间向量，其中。
  + PSO 优化 BP 神经网络：BP 神经网络的预测输出为，其中为输入层到隐层的权重，为隐层神经元输出，为偏置项，为激活函数。PSO 中粒子的位置代表 BP 神经网络的权重和阈值，通过速度和位置更新公式寻找最优参数。速度更新：；位置更新：，其中为惯性权重，、为加速系数，、为 [0,1] 随机数，为个体最优位置，为全局最优位置。优化目标为最小化预测误差，为实际值。
  + 最终预测：利用优化后的 PSO-BP 模型对相空间重构后的向量进行预测，得到。
* **主要优势**：混沌时间序列分析能有效解析具有混沌特性数据的内在规律，将一维序列转化为高维相空间，更利于捕捉数据的复杂关系。PSO 优化后的 BP 神经网络克服了自身缺陷，提高了收敛速度和预测精度，使其能更好地拟合混沌系统的非线性关系。两者结合，显著提升了对非线性混沌序列的预测能力，尤其在数据呈现出复杂、无规则但有内在确定性的情况下表现出色。
* **适合赛题**：电力系统负荷预测赛题，电力负荷受多种因素影响，具有明显的混沌特性，该组合模型能精准预测负荷变化；金融时间序列预测赛题，金融数据（如股票价格、汇率等）常表现出混沌特征，利用该模型可更好地捕捉其波动规律，提高预测准确性。

# 二、15种组合预测模型的数学建模案例题目

### 案例 1：基于 ARIMA-LSTM 模型的城市短期电力负荷预测

* **问题背景**：随着城市化进程加快和新能源接入电网比例提升，城市电力负荷波动日益复杂。短期电力负荷既受经济发展、居民生活习惯等因素影响呈现线性增长趋势，又会因极端天气（如高温、寒潮）、大型文体活动等突发情况产生非线性剧烈波动。精准预测短期电力负荷，对电力调度部门制定发电计划、保障电网稳定运行具有重要意义。
* **问题描述**：某省级电力公司需要对其管辖的核心城市进行未来 72 小时的电力负荷预测。要求预测模型能够同时捕捉电力负荷的线性趋势（如工作日与周末的常规负荷差异）和非线性波动（如夏季高温导致的空调负荷激增），并给出每小时的负荷预测值，为电网调度提供科学依据。
* **数据情况**：提供该城市过去 5 年的小时级电力负荷数据，包括负荷值（单位：MW）；同时提供对应的气象数据（温度、湿度、降雨量等）、节假日信息、大型活动举办记录等辅助数据。数据存在部分缺失值和异常值，需要进行预处理。

### 案例 1：ARIMA-LSTM 模型城市短期电力负荷预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import matplotlib.font\_manager as fm  from statsmodels.tsa.arima.model import ARIMA  from sklearn.preprocessing import MinMaxScaler  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error  import os  import warnings  # 忽略所有警告  warnings.filterwarnings('ignore')  os.environ['MKL\_NUM\_THREADS'] = '1' # 限制线程数，减少资源占用  def main():  try:  # 简化字体设置，使用默认字体避免问题  plt.rcParams["axes.unicode\_minus"] = False # 仅解决负号问题    # 生成示例数据（更少的数据点，更快处理）  print("正在生成示例数据...")  dates = pd.date\_range(start='2023-01-01', periods=300, freq='h') # 减少到300小时  np.random.seed(42)    # 更简单的数据模式  base\_load = 1000 + 150 \* np.sin(np.linspace(0, 15, 300))  trend = np.linspace(0, 100, 300)  noise = np.random.normal(0, 20, 300)  load = base\_load + trend + noise    data = pd.DataFrame({'time': dates, 'load': load})  data = data.set\_index('time')  print(f"生成数据完成，共 {len(data)} 条记录")    # 数据预处理  load\_data = data['load'].values.reshape(-1, 1)  scaler = MinMaxScaler(feature\_range=(0, 1))  load\_scaled = scaler.fit\_transform(load\_data)    # 划分数据集  train\_size = int(len(load\_scaled) \* 0.8)  train, test = load\_scaled[0:train\_size, :], load\_scaled[train\_size:, :]  print(f"数据划分：训练集 {len(train)} 条，测试集 {len(test)} 条")    # ARIMA模型（更简单的参数）  print("训练ARIMA模型...")  arima\_model = ARIMA(train, order=(2, 1, 1)) # 更简单的参数组合  arima\_result = arima\_model.fit()  arima\_pred = arima\_result.predict(  start=len(train),  end=len(train) + len(test) - 1  ).reshape(-1, 1)    # 计算残差  arima\_residual = test - arima\_pred    # 准备残差预测数据（使用更简单的特征）  def create\_simple\_dataset(dataset, look\_back=6): # 更短的回溯期  dataX, dataY = [], []  for i in range(len(dataset) - look\_back):  dataX.append(dataset[i:(i + look\_back), 0])  dataY.append(dataset[i + look\_back, 0])  return np.array(dataX), np.array(dataY)    look\_back = 6  if len(arima\_residual) <= look\_back + 24:  raise ValueError("数据量不足，请增加数据点数")    # 准备数据  trainX, trainY = create\_simple\_dataset(arima\_residual[:-24], look\_back)  testX, testY = create\_simple\_dataset(arima\_residual, look\_back)  print(f"残差预测数据准备：训练样本 {len(trainX)} 个")    # 使用线性回归替代LSTM（移除TensorFlow依赖）  print("训练残差修正模型...")  lr\_model = LinearRegression()  lr\_model.fit(trainX, trainY)  lr\_pred = lr\_model.predict(testX)    # 组合预测结果  arima\_part = arima\_pred[look\_back:look\_back + len(lr\_pred)]  final\_pred = arima\_part.flatten() + lr\_pred.flatten()    # 反归一化  final\_pred = scaler.inverse\_transform(final\_pred.reshape(-1, 1))  test\_actual = scaler.inverse\_transform(test[look\_back:look\_back + len(final\_pred)])    # 评估模型  mse = mean\_squared\_error(test\_actual, final\_pred)  rmse = np.sqrt(mse)  print(f'组合模型MSE: {mse:.4f}')  print(f'组合模型RMSE: {rmse:.4f}')    # 简化可视化  plt.figure(figsize=(8, 4))  plt.plot(test\_actual, label='实际值', alpha=0.7)  plt.plot(final\_pred, label='预测值', linestyle='--')  plt.title('预测结果对比')  plt.xlabel('时间(小时)')  plt.ylabel('负荷值')  plt.legend()  plt.tight\_layout()    # 尝试显示图像，失败则跳过  try:  plt.show()  except:  print("图像显示功能不可用，已跳过")    except Exception as e:  print(f"运行错误：{str(e)}")  return  if \_\_name\_\_ == "\_\_main\_\_":  main() |

### 案例 2：基于 RF-GA 模型的城市二手房价格预测

* **问题背景**：二手房市场是房地产市场的重要组成部分，其价格受多种因素影响，如房屋面积、房龄、地理位置、周边配套设施（学校、医院、商圈等）、政策调控等。准确预测二手房价格，可为购房者、售房者和房地产政策制定者提供决策支持。
* **问题描述**：某房地产中介公司计划开发一套二手房价格评估系统，需要基于房屋的各项特征预测其成交价格。由于影响因素众多且关系复杂，传统单一模型难以精准预测，需采用优化后的集成模型。要求模型能自动优化参数，提高预测精度，帮助中介公司更合理地为房源定价。
* **数据情况**：提供该城市过去 3 年的二手房成交数据，每条数据包含房屋的基本属性（面积、户型、房龄、朝向等）、地理位置信息（经纬度、所属行政区等）、周边配套设施距离、成交时间、成交价格等。数据量约 5 万条，存在部分特征缺失和极端值。

### 案例 2：RF-GA 模型城市二手房价格预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, MinMaxScaler  from sklearn.model\_selection import train\_test\_split  from sklearn.ensemble import RandomForestRegressor  from sklearn.metrics import mean\_absolute\_error  from geneticalgorithm import geneticalgorithm as ga  import joblib  # 数据加载与预处理  data = pd.read\_csv('second\_hand\_house.csv')  data = data.dropna()  # 类别特征编码  le = LabelEncoder()  for col in ['district', 'house\_type', 'orientation']:  data[col] = le.fit\_transform(data[col])  # 特征与目标变量  X = data.drop('price', axis=1)  y = data['price']  # 数据归一化  scaler = MinMaxScaler()  X\_scaled = scaler.fit\_transform(X)  # 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  # 定义目标函数（RF模型MAE）  def objective\_function(params):  n\_estimators = int(params[0])  max\_depth = int(params[1])  min\_samples\_split = int(params[2])  min\_samples\_leaf = int(params[3])    rf = RandomForestRegressor(  n\_estimators=n\_estimators,  max\_depth=max\_depth,  min\_samples\_split=min\_samples\_split,  min\_samples\_leaf=min\_samples\_leaf,  random\_state=42  )  rf.fit(X\_train, y\_train)  y\_pred = rf.predict(X\_test)  return mean\_absolute\_error(y\_test, y\_pred)  # 定义参数范围  varbound = np.array([  [50, 500], # n\_estimators  [5, 50], # max\_depth  [2, 20], # min\_samples\_split  [1, 10] # min\_samples\_leaf  ])  # 遗传算法优化  algorithm\_param = {  'max\_num\_iteration': 50,  'population\_size': 20,  'mutation\_probability': 0.1,  'elit\_ratio': 0.05,  'crossover\_probability': 0.5,  'parents\_portion': 0.3,  'crossover\_type': 'uniform',  'max\_iteration\_without\_improv': None  }  model\_ga = ga(function=objective\_function,  dimension=4,  variable\_type='int',  variable\_boundaries=varbound,  algorithm\_parameters=algorithm\_param,  function\_timeout=500)  model\_ga.run()  best\_params = model\_ga.best\_variable  # 最优RF模型训练与预测  best\_rf = RandomForestRegressor(  n\_estimators=int(best\_params[0]),  max\_depth=int(best\_params[1]),  min\_samples\_split=int(best\_params[2]),  min\_samples\_leaf=int(best\_params[3]),  random\_state=42  )  best\_rf.fit(X\_train, y\_train)  y\_pred = best\_rf.predict(X\_test)  # 评估模型  print(f'最优参数: {best\_params}')  print(f'MAE: {mean\_absolute\_error(y\_test, y\_pred)}')  # 保存模型  joblib.dump(best\_rf, 'rf\_ga\_house\_price.pkl')  joblib.dump(scaler, 'scaler\_house.pkl') |

### 案例 3：基于 CNN-LSTM 模型的城市区域交通流量预测

* **问题背景**：城市交通拥堵已成为制约城市发展的重要问题，精准的交通流量预测是智能交通系统的核心功能之一。交通流量具有明显的时空特性，某一区域的交通流量不仅与该区域历史流量相关，还与相邻区域的流量密切相关，同时随时间动态变化。
* **问题描述**：某市交通管理部门需要对城市核心商圈及周边 5 平方公里范围内的 100 个路口未来 24 小时的交通流量进行预测。要求模型能够同时考虑不同路口之间的空间关联（如主干道与支路的流量影响）和时间变化规律（如早晚高峰的流量特征），为交通信号配时优化和拥堵疏导提供依据。
* **数据情况**：提供过去 1 年的各路口每 15 分钟的交通流量数据（单位：辆 / 15 分钟），同时提供路口的位置关系图、道路等级、周边建筑功能（商业区、住宅区、办公区等）、节假日及天气数据。数据存在因设备故障导致的部分时段缺失。

### 案例 3：CNN-LSTM 模型城市区域交通流量预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense, Flatten  from sklearn.metrics import mean\_squared\_error  import os  # 数据加载与预处理  def load\_traffic\_data(file\_path):  data = pd.read\_csv(file\_path, parse\_dates=['time'], index\_col='time')  return data.values # 形状为 (时间步, 路口数量)  traffic\_data = load\_traffic\_data('traffic\_flow.csv')  scaler = MinMaxScaler(feature\_range=(0, 1))  traffic\_scaled = scaler.fit\_transform(traffic\_data)  # 构建时空序列数据  def create\_spatio\_temporal\_data(data, time\_steps=8, pred\_steps=1):  X, y = [], []  for i in range(len(data) - time\_steps - pred\_steps + 1):  X.append(data[i:i+time\_steps, :]) # 时空特征 (时间步, 路口数)  y.append(data[i+time\_steps:i+time\_steps+pred\_steps, :]) # 预测目标  return np.array(X), np.array(y).squeeze()  time\_steps = 8 # 8个15分钟，共2小时  X, y = create\_spatio\_temporal\_data(traffic\_scaled, time\_steps)  # 划分训练集和测试集  train\_size = int(len(X) \* 0.8)  X\_train, X\_test = X[:train\_size], X[train\_size:]  y\_train, y\_test = y[:train\_size], y[train\_size:]  # 重塑输入形状 (样本数, 时间步, 特征数)  X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], X\_train.shape[2])  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], X\_test.shape[2])  # 构建CNN-LSTM模型  model = Sequential()  # CNN层提取空间特征  model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=(time\_steps, X\_train.shape[2])))  model.add(MaxPooling1D(pool\_size=2))  model.add(Conv1D(filters=32, kernel\_size=2, activation='relu'))  # LSTM层提取时间特征  model.add(LSTM(50, return\_sequences=False))  model.add(Dense(y\_train.shape[1])) # 输出每个路口的预测值  model.compile(optimizer='adam', loss='mse')  # 模型训练  history = model.fit(X\_train, y\_train, epochs=30, batch\_size=32, validation\_data=(X\_test, y\_test), verbose=1)  # 模型预测  y\_pred = model.predict(X\_test)  # 反归一化  y\_pred\_actual = scaler.inverse\_transform(y\_pred)  y\_test\_actual = scaler.inverse\_transform(y\_test)  # 评估模型  mse = mean\_squared\_error(y\_test\_actual.flatten(), y\_pred\_actual.flatten())  print(f'时空模型MSE: {mse}')  # 可视化第一个路口的预测结果  plt.figure(figsize=(12, 6))  plt.plot(y\_test\_actual[:, 0], label='实际流量')  plt.plot(y\_pred\_actual[:, 0], label='预测流量')  plt.legend()  plt.savefig('traffic\_prediction.png')  plt.show()  # 保存模型  model.save('cnn\_lstm\_traffic.h5') |

### 案例 4：基于 SVM-PSO 模型的中小企业信用评分预测

* **问题背景**：中小企业是国民经济的重要组成部分，但融资难问题一直制约其发展。银行等金融机构需要对中小企业的信用风险进行评估，以决定是否放贷及放贷额度。中小企业信用评分受企业财务状况、经营年限、行业前景、负责人信用记录等多因素影响，且样本数据相对较少。
* **问题描述**：某商业银行计划构建中小企业信用评分模型，根据企业的各项指标预测其信用等级（分为 AAA、AA、A、BBB、BB、B、CCC 共 7 个等级）。要求模型能在小样本、高维特征情况下准确预测，帮助银行有效控制信贷风险。
* **数据情况**：提供该银行过去 5 年的中小企业贷款客户数据，每条数据包含企业的财务指标（资产负债率、净利润率、流动比率等 20 项）、经营数据（成立年限、员工人数、年销售额等）、行业类别、负责人信用记录及最终信用等级。数据量约 2000 条，部分企业存在财务数据不完整的情况。

### 案例 4：SVM-PSO 模型中小企业信用评分预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.model\_selection import train\_test\_split  from sklearn.svm import SVC  from sklearn.metrics import classification\_report, accuracy\_score  from pyswarm import pso  import joblib  # 数据加载与预处理  data = pd.read\_csv('small\_business\_credit.csv')  data = data.dropna()  # 信用等级编码  le = LabelEncoder()  data['credit\_rating'] = le.fit\_transform(data['credit\_rating'])  # 特征与目标变量  X = data.drop('credit\_rating', axis=1)  y = data['credit\_rating']  # 数据标准化  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  # 定义SVM参数优化目标函数  def svm\_objective(params):  C = 10 \*\*params[0] # 惩罚因子，取指数避免负数值  gamma = 10\*\* params[1] # 核函数参数    svm = SVC(C=C, gamma=gamma, kernel='rbf', random\_state=42)  svm.fit(X\_train, y\_train)  y\_pred = svm.predict(X\_train)  return 1 - accuracy\_score(y\_train, y\_pred) # 最小化错误率  # 参数范围 (-5, 5) 对应 (1e-5, 1e5)  lb = [-5, -5]  ub = [5, 5]  # PSO优化  best\_params, \_ = pso(svm\_objective, lb, ub, swarmsize=20, maxiter=50, debug=True)  best\_C = 10 \*\* best\_params[0]  best\_gamma = 10 \*\* best\_params[1]  # 最优SVM模型训练与预测  best\_svm = SVC(C=best\_C, gamma=best\_gamma, kernel='rbf', random\_state=42)  best\_svm.fit(X\_train, y\_train)  y\_pred = best\_svm.predict(X\_test)  # 评估模型  print(f'最优参数: C={best\_C}, gamma={best\_gamma}')  print(f'准确率: {accuracy\_score(y\_test, y\_pred)}')  print(classification\_report(y\_test, y\_pred))  # 保存模型  joblib.dump(best\_svm, 'svm\_pso\_credit.pkl')  joblib.dump(scaler, 'scaler\_credit.pkl')  joblib.dump(le, 'label\_encoder\_credit.pkl') |

### 案例 5：基于 GM (1,1)-BP 模型的稀有金属月度价格预测

* **问题背景**：稀有金属（如稀土、钨、钼等）在新能源、高端制造等领域应用广泛，其价格受资源储量、市场需求、政策调控、国际形势等因素影响，且由于交易不频繁，历史数据样本量较少，价格波动呈现非线性特征。
* **问题描述**：某稀有金属贸易公司需要对某类重点稀有金属未来 6 个月的月度价格进行预测。要求模型能够利用有限的历史数据，兼顾价格的长期线性趋势（如资源稀缺导致的价格上涨）和短期非线性波动（如突发政策影响），为公司的采购和销售策略提供指导。
* **数据情况**：提供该稀有金属过去 3 年的月度价格数据（单位：元 / 千克），同时提供相关的行业政策文件、全球产量及需求量统计数据、主要贸易国的进出口情况等。数据共 36 条，存在个别月份因特殊事件导致的价格异常波动。

### 案例 5：GM (1,1)-BP 模型稀有金属月度价格预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from sklearn.metrics import mean\_absolute\_percentage\_error  # 灰色预测GM(1,1)模型  class GM11:  def \_\_init\_\_(self):  self.a = None  self.b = None    def fit(self, x0):  x1 = np.cumsum(x0) # 累加生成  z1 = (x1[:-1] + x1[1:]) / 2.0 # 紧邻均值生成  B = np.vstack([-z1, np.ones\_like(z1)]).T  Y = x0[1:].reshape(-1, 1)    # 最小二乘法求解参数  params = np.dot(np.dot(np.linalg.inv(np.dot(B.T, B)), B.T), Y)  self.a, self.b = params[0][0], params[1][0]    def predict(self, x0, n\_pred):  x1\_initial = np.cumsum(x0)[-1]  predictions = []  for k in range(1, n\_pred + 1):  x1\_pred = (x0[0] - self.b / self.a) \* np.exp(-self.a \* k) + self.b / self.a  x0\_pred = x1\_pred - x1\_initial  predictions.append(x0\_pred)  x1\_initial = x1\_pred  return np.array(predictions)  # 数据加载与预处理  data = pd.read\_csv('rare\_metal\_price.csv', parse\_dates=['month'], index\_col='month')  price\_data = data['price'].values  # 划分训练集和预测期  train\_data = price\_data[:-6] # 留6个月作为测试  n\_pred = 6  # GM(1,1)模型训练与预测  gm = GM11()  gm.fit(train\_data)  gm\_pred = gm.predict(train\_data, n\_pred)  # 计算GM(1,1)残差  actual = price\_data[-6:]  gm\_residual = actual - gm\_pred  # 准备BP神经网络数据（用历史价格作为特征预测残差）  def create\_bp\_data(price, residual, look\_back=3):  X, y = [], []  for i in range(len(residual)):  X.append(price[-(look\_back + len(residual)) + i : -(len(residual) - i)])  y.append(residual[i])  return np.array(X), np.array(y)  look\_back = 3  X\_bp, y\_bp = create\_bp\_data(price\_data, gm\_residual, look\_back)  # 数据归一化  scaler\_X = MinMaxScaler()  scaler\_y = MinMaxScaler()  X\_bp\_scaled = scaler\_X.fit\_transform(X\_bp)  y\_bp\_scaled = scaler\_y.fit\_transform(y\_bp.reshape(-1, 1))  # BP神经网络构建与训练  bp\_model = Sequential()  bp\_model.add(Dense(16, input\_dim=look\_back, activation='relu'))  bp\_model.add(Dense(8, activation='relu'))  bp\_model.add(Dense(1))  bp\_model.compile(loss='mse', optimizer='adam')  bp\_model.fit(X\_bp\_scaled, y\_bp\_scaled, epochs=50, batch\_size=2, verbose=1)  # BP预测残差  residual\_pred\_scaled = bp\_model.predict(X\_bp\_scaled)  residual\_pred = scaler\_y.inverse\_transform(residual\_pred\_scaled).flatten()  # 组合预测结果  final\_pred = gm\_pred + residual\_pred  # 评估模型  mape = mean\_</doubaocanvas> |

### 案例 6：基于 GRU-Attention 模型的流域长期降水量预测

* **问题背景**：降水量是影响农业生产、水资源管理和防洪抗旱的重要因素。某流域的降水量具有明显的季节性和年际变化特征，且受厄尔尼诺、拉尼娜等气候现象影响，长期降水量预测难度较大，需要关注关键时间节点的气候特征。
* **问题描述**：某流域管理局需要对该流域未来 5 年的年度降水量进行预测。要求模型能够高效处理长序列的历史降水数据，重点关注影响降水量的关键季节转换节点（如厄尔尼诺现象发生的月份），提高长期预测的准确性，为流域水资源规划和防灾减灾提供依据。
* **数据情况**：提供该流域过去 50 年的月度降水量数据（单位：mm），同时提供全球气候指数（如厄尔尼诺指数、南方涛动指数等）、大气环流数据等。数据存在部分年份的个别月份数据缺失。

### 案例 6：GRU-Attention 模型流域长期降水量预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from tensorflow.keras.models import Model  from tensorflow.keras.layers import Input, GRU, Dense, Attention, Concatenate  from sklearn.metrics import mean\_squared\_error  import tensorflow as tf  # 数据加载与预处理  data = pd.read\_csv('basin\_precipitation.csv', parse\_dates=['date'], index\_col='date')  precip\_data = data['precipitation'].values.reshape(-1, 1)  # 数据归一化  scaler = MinMaxScaler(feature\_range=(0, 1))  precip\_scaled = scaler.fit\_transform(precip\_data)  # 构建序列数据  def create\_sequences(data, time\_steps=12, pred\_steps=1):  X, y = [], []  for i in range(len(data) - time\_steps - pred\_steps + 1):  X.append(data[i:i+time\_steps, 0])  y.append(data[i+time\_steps:i+time\_steps+pred\_steps, 0])  return np.array(X), np.array(y).squeeze()  time\_steps = 12 # 12个月的历史数据  X, y = create\_sequences(precip\_scaled, time\_steps)  # 划分训练集和测试集  train\_size = int(len(X) \* 0.8)  X\_train, X\_test = X[:train\_size], X[train\_size:]  y\_train, y\_test = y[:train\_size], y[train\_size:]  # 重塑输入形状 [samples, time steps, features]  X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  # 构建GRU-Attention模型  inputs = Input(shape=(time\_steps, 1))  gru\_out = GRU(64, return\_sequences=True)(inputs) # 返回所有时间步的输出用于注意力机制  # 注意力机制层  attention = Attention()([gru\_out, gru\_out])  attention = tf.reduce\_sum(attention, axis=1) # 聚合注意力权重  # 输出层  outputs = Dense(1)(attention)  model = Model(inputs=inputs, outputs=outputs)  model.compile(optimizer='adam', loss='mse')  # 模型训练  history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16, validation\_data=(X\_test, y\_test), verbose=1)  # 模型预测  y\_pred = model.predict(X\_test)  # 反归一化  y\_pred\_actual = scaler.inverse\_transform(y\_pred.reshape(-1, 1))  y\_test\_actual = scaler.inverse\_transform(y\_test.reshape(-1, 1))  # 评估模型  mse = mean\_squared\_error(y\_test\_actual, y\_pred\_actual)  print(f'GRU-Attention模型MSE: {mse}')  # 可视化结果  plt.figure(figsize=(12, 6))  plt.plot(y\_test\_actual, label='实际降水量')  plt.plot(y\_pred\_actual, label='预测降水量')  plt.legend()  plt.savefig('precipitation\_prediction.png')  plt.show()  # 保存模型  model.save('gru\_attention\_precip.h5') |

### 案例 7：基于 XGBoost - 贝叶斯优化模型的电商平台 “618” 大促销量预测

* **问题背景**：电商平台的大型促销活动（如 “618”“双十一”）期间，商品销量会出现爆发式增长，准确预测销量对平台库存管理、物流调度和营销策略制定至关重要。销量受商品价格、促销力度、用户评价、历史销量、竞争对手活动等多因素影响，呈现强非线性关系。
* **问题描述**：某大型电商平台需要对平台上 1000 款热销商品在即将到来的 “618” 大促期间（15 天）的日销量进行预测。要求模型能够自动优化超参数，充分利用高维特征数据，精准预测各商品的销量变化，避免库存积压或缺货。
* **数据情况**：提供各商品过去 3 年的日常销量及过往 “618”“双十一” 期间的销量数据，同时提供商品的属性信息（类别、价格、品牌、规格等）、促销方案（折扣力度、满减活动等）、用户评价数据（评分、好评率等）、平台流量数据等。数据量庞大，特征维度超过 50 维。

### 案例 7：XGBoost - 贝叶斯优化模型电商大促销量预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, MinMaxScaler  from sklearn.model\_selection import train\_test\_split, cross\_val\_score  from xgboost import XGBRegressor  from skopt import BayesSearchCV  from skopt.space import Real, Integer  from sklearn.metrics import mean\_squared\_error  import joblib  # 数据加载与预处理  data = pd.read\_csv('ecommerce\_sales.csv')  data = data.dropna()  # 类别特征编码  le = LabelEncoder()  for col in ['product\_category', 'brand']:  data[col] = le.fit\_transform(data[col])  # 特征与目标变量  X = data.drop(['sales', 'product\_id'], axis=1)  y = data['sales']  # 数据归一化  scaler = MinMaxScaler()  X\_scaled = scaler.fit\_transform(X)  # 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  # 定义XGBoost参数空间  param\_space = {  'n\_estimators': Integer(50, 500),  'max\_depth': Integer(3, 10),  'learning\_rate': Real(0.01, 0.3, 'log-uniform'),  'subsample': Real(0.6, 1.0),  'colsample\_bytree': Real(0.6, 1.0),  'gamma': Real(0, 5)  }  # 贝叶斯优化  xgb = XGBRegressor(objective='reg:squarederror', random\_state=42)  bayes\_search = BayesSearchCV(  estimator=xgb,  search\_spaces=param\_space,  cv=5,  n\_iter=30,  scoring='neg\_mean\_squared\_error',  random\_state=42,  verbose=1  )  bayes\_search.fit(X\_train, y\_train)  best\_xgb = bayes\_search.best\_estimator\_  # 模型预测  y\_pred = best\_xgb.predict(X\_test)  # 评估模型  print(f'最优参数: {bayes\_search.best\_params\_}')  print(f'MSE: {mean\_squared\_error(y\_test, y\_pred)}')  # 保存模型  joblib.dump(best\_xgb, 'xgb\_bayes\_sales.pkl')  joblib.dump(scaler, 'scaler\_sales.pkl') |

### 案例 8：基于马尔可夫链 - ETS 模型的连锁超市生鲜产品短期需求预测

* **问题背景**：生鲜产品具有易腐烂、保质期短的特点，连锁超市需要精准预测其短期需求，以合理制定采购计划，减少损耗。生鲜产品需求受日常消费习惯影响呈现一定趋势和季节性，同时受周末、节假日、促销活动等因素影响产生随机波动。
* **问题描述**：某连锁超市需要对旗下 10 家门店的 15 种生鲜产品（如蔬菜、水果、肉类等）未来 7 天的日需求量进行预测。要求模型能够兼顾产品的日常消费趋势、周度季节性，同时捕捉促销活动或节假日导致的需求突发波动，提高预测的抗干扰能力。
* **数据情况**：提供各门店过去 2 年的生鲜产品日销量数据，同时提供促销活动记录（促销时间、促销方式、折扣幅度等）、节假日信息、天气数据（影响顾客到店频率）等。数据存在因缺货导致的部分零销量记录。

### 案例 8：马尔可夫链 - ETS 模型超市生鲜需求预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from statsmodels.tsa.holtwinters import ExponentialSmoothing  from sklearn.preprocessing import MinMaxScaler  from sklearn.metrics import mean\_absolute\_error  from collections import Counter  # 数据加载与预处理  data = pd.read\_csv('fresh\_food\_demand.csv', parse\_dates=['date'], index\_col='date')  demand\_data = data['demand'].values  # 划分训练集和测试集  train\_size = int(len(demand\_data) \* 0.8)  train, test = demand\_data[:train\_size], demand\_data[train\_size:]  # ETS模型训练与预测  ets\_model = ExponentialSmoothing(train, trend='add', seasonal='add', seasonal\_periods=7)  ets\_result = ets\_model.fit()  ets\_pred = ets\_result.forecast(len(test))  # 马尔可夫链状态划分与转移矩阵构建  def markov\_chain(residuals, n\_states=5):  # 残差状态划分  scaler = MinMaxScaler(feature\_range=(0, n\_states-1))  scaled\_res = scaler.fit\_transform(residuals.reshape(-1, 1)).astype(int).flatten()    # 计算转移矩阵  transition = np.zeros((n\_states, n\_states))  for i in range(len(scaled\_res)-1):  transition[scaled\_res[i], scaled\_res[i+1]] += 1  transition = transition / transition.sum(axis=1, keepdims=True) # 归一化  return transition, scaler  # 计算训练集残差并构建马尔可夫链  train\_residuals = train - ets\_result.fittedvalues  transition\_matrix, scaler = markov\_chain(train\_residuals)  # 马尔可夫链预测残差  def predict\_residuals(ets\_pred, test\_actual, transition\_matrix, scaler, n\_states=5):  residuals = test\_actual - ets\_pred[:len(test\_actual)]  scaled\_res = scaler.transform(residuals.reshape(-1, 1)).astype(int).flatten()    # 基于最后一个状态预测下一状态  last\_state = scaled\_res[-1]  next\_state\_probs = transition\_matrix[last\_state]  pred\_state = np.argmax(next\_state\_probs)    # 状态反归一化得到残差预测  pred\_residual = scaler.inverse\_transform(np.array([[pred\_state]]))[0][0]  return pred\_residual  # 组合预测（滚动预测）  final\_pred = []  for i in range(len(test)):  # 每步更新ETS预测  current\_ets\_pred = ets\_result.forecast(i+1)[-1]  # 计算当前残差并更新马尔可夫预测  if i == 0:  prev\_residual = test[i] - current\_ets\_pred  pred\_res = prev\_residual # 初始残差  else:  pred\_res = predict\_residuals(ets\_pred[:i], test[:i], transition\_matrix, scaler)  final\_pred.append(current\_ets\_pred + pred\_res)  # 评估模型  mae = mean\_absolute\_error(test, final\_pred)  print(f'组合模型MAE: {mae}')  # 可视化结果  plt.figure(figsize=(12, 6))  plt.plot(test, label='实际需求')  plt.plot(final\_pred, label='预测需求')  plt.legend()  plt.savefig('fresh\_demand\_prediction.png')  plt.show() |

### 案例 9：基于小波分析 - LSTM 模型的人民币兑欧元汇率预测

* **问题背景**：汇率波动受宏观经济指标、国际贸易收支、利率政策、地缘政治等多种因素影响，短期汇率受投机交易等因素影响存在大量噪声，长期则呈现一定趋势。准确预测汇率对进出口企业规避汇率风险、制定经营策略具有重要意义。
* **问题描述**：某进出口企业需要对人民币兑欧元的未来 30 天的日汇率中间价进行预测。要求模型能够分离汇率序列中的高频噪声（如短期投机交易影响）和低频有效信号（如宏观经济趋势），提高预测精度，帮助企业提前锁定汇率成本。
* **数据情况**：提供过去 10 年的人民币兑欧元日汇率中间价数据，同时提供中国和欧元区的 GDP 增长率、通货膨胀率、利率、进出口贸易差额等宏观经济数据，以及重大国际事件（如贸易摩擦、政策调整）记录。数据存在个别交易日因节假日导致的缺失。

### 案例 9：小波分析 - LSTM 模型汇率预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import pywt  from sklearn.preprocessing import MinMaxScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import LSTM, Dense  from sklearn.metrics import mean\_squared\_error  # 数据加载与预处理  data = pd.read\_csv('exchange\_rate.csv', parse\_dates=['date'], index\_col='date')  rate\_data = data['rate'].values.reshape(-1, 1)  # 数据归一化  scaler = MinMaxScaler(feature\_range=(0, 1))  rate\_scaled = scaler.fit\_transform(rate\_data)  # 小波分解  def wavelet\_decompose(data, wavelet='db4', level=3):  coeffs = pywt.wavedec(data, wavelet, level=level)  return coeffs # 返回近似系数和细节系数  # 小波重构  def wavelet\_reconstruct(coeffs, wavelet='db4'):  return pywt.waverec(coeffs, wavelet)  # 分解序列  coeffs = wavelet\_decompose(rate\_scaled.flatten(), level=3)  n\_levels = len(coeffs)  # 为每个分量构建LSTM模型并预测  predictions = []  for i in range(n\_levels):  component = coeffs[i].reshape(-1, 1)    # 构建序列数据  def create\_seq(data, look\_back=10):  X, y = [], []  for j in range(len(data) - look\_back - 1):  X.append(data[j:j+look\_back, 0])  y.append(data[j+look\_back, 0])  return np.array(X), np.array(y)    look\_back = 10  X, y = create\_seq(component, look\_back)  X = X.reshape(X.shape[0], X.shape[1], 1)    # 划分训练集和测试集  train\_size = int(len(X) \* 0.8)  X\_train, X\_test = X[:train\_size], X[train\_size:]  y\_train, y\_test = y[:train\_size], y[train\_size:]    # 训练LSTM模型  model = Sequential()  model.add(LSTM(32, input\_shape=(look\_back, 1)))  model.add(Dense(1))  model.compile(loss='mse', optimizer='adam')  model.fit(X\_train, y\_train, epochs=20, batch\_size=16, verbose=0)    # 预测  y\_pred = model.predict(X\_test)  predictions.append(y\_pred)  # 小波重构预测结果  # 对齐各分量长度  min\_len = min(len(p) for p in predictions)  aligned\_preds = [p[:min\_len] for p in predictions]  # 重构  pred\_coeffs = []  for i in range(n\_levels):  # 低通分量（近似系数）需要补全长度  if i == 0:  pred\_coeff = np.zeros\_like(coeffs[i])  pred\_coeff[-min\_len:] = aligned\_preds[i].flatten()  else:  pred\_coeff = np.zeros\_like(coeffs[i])  pred\_coeff[-min\_len:] = aligned\_preds[i].flatten()  pred\_coeffs.append(pred\_coeff)  # 重构序列  final\_pred = wavelet\_reconstruct(pred\_coeffs)  final\_pred = final\_pred[-min\_len:].reshape(-1, 1)  # 反归一化  final\_pred = scaler.inverse\_transform(final\_pred)  test\_actual = scaler.inverse\_transform(rate\_scaled[-min\_len:])  # 评估模型  mse = mean\_squared\_error(test\_actual, final\_pred)  print(f'小波-LSTM模型MSE: {mse}')  # 可视化结果  plt.figure(figsize=(12, 6))  plt.plot(test\_actual, label='实际汇率')  plt.plot(final\_pred, label='预测汇率')  plt.legend()  plt.savefig('exchange\_rate\_prediction.png')  plt.show() |

### 案例 10：基于 LightGBM-Stacking 模型的互联网企业用户流失预测

* **问题背景**：用户是互联网企业的核心资源，用户流失会导致企业收入下降和市场份额萎缩。准确预测用户流失风险，有助于企业采取针对性挽留措施，降低流失率。用户流失受用户使用频率、产品体验、竞争对手吸引、价格因素等多方面影响，且不同用户群体的流失原因差异较大。
* **问题描述**：某互联网社交平台需要预测未来 1 个月内可能流失的用户（定义为连续 15 天未登录的用户）。要求模型能够集成多种基模型的优势，提高预测的稳健性和准确性，识别出高流失风险用户，为平台制定个性化挽留策略提供支持。
* **数据情况**：提供平台过去 2 年的用户行为数据，包括用户的登录频率、在线时长、互动次数、好友数量、付费情况等 30 余项特征，以及用户是否流失的标签。数据量约 100 万用户，存在部分用户行为数据缺失的情况。

### 案例 10：LightGBM-Stacking 模型用户流失预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.preprocessing import LabelEncoder, StandardScaler  from sklearn.model\_selection import train\_test\_split, KFold  from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier  from lightgbm import LGBMClassifier  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import roc\_auc\_score, classification\_report  import joblib  # 数据加载与预处理  data = pd.read\_csv('user\_churn.csv')  data = data.dropna()  # 类别特征编码  le = LabelEncoder()  for col in ['gender', 'subscription\_type']:  data[col] = le.fit\_transform(data[col])  # 特征与目标变量  X = data.drop(['user\_id', 'churn'], axis=1)  y = data['churn']  # 数据标准化  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)  # 定义基模型  base\_models = [  ('lightgbm', LGBMClassifier(n\_estimators=100, random\_state=42)),  ('rf', RandomForestClassifier(n\_estimators=100, random\_state=42)),  ('gbt', GradientBoostingClassifier(n\_estimators=100, random\_state=42))  ]  # Stacking第一层：生成基模型预测  def stacking\_train(X, y, base\_models, n\_folds=5):  kf = KFold(n\_splits=n\_folds, shuffle=True, random\_state=42)  meta\_features = np.zeros((X.shape[0], len(base\_models)))    for i, (name, model) in enumerate(base\_models):  for train\_idx, val\_idx in kf.split(X):  X\_tr, X\_val = X[train\_idx], X[val\_idx]  y\_tr = y[train\_idx]  model.fit(X\_tr, y\_tr)  meta\_features[val\_idx, i] = model.predict\_proba(X\_val)[:, 1]  return meta\_features  # 生成训练集元特征  meta\_train = stacking\_train(X\_train, y\_train, base\_models)  # 训练元模型（逻辑回归）  meta\_model = LogisticRegression()  meta\_model.fit(meta\_train, y\_train)  # 生成测试集元特征  def stacking\_predict(X\_train, X\_test, y\_train, base\_models):  meta\_test = np.zeros((X\_test.shape[0], len(base\_models)))  for i, (name, model) in enumerate(base\_models):  model.fit(X\_train, y\_train)  meta\_test[:, i] = model.predict\_proba(X\_test)[:, 1]  return meta\_test  meta\_test = stacking\_predict(X\_train, X\_test, y\_train, base\_models)  # 最终预测  y\_pred\_proba = meta\_model.predict\_proba(meta\_test)[:, 1]  y\_pred = meta\_model.predict(meta\_test)  # 评估模型  print(f'AUC: {roc\_auc\_score(y\_test, y\_pred\_proba)}')  print(classification\_report(y\_test, y\_pred))  # 保存模型  joblib.dump(base\_models, 'base\_models\_stacking.pkl')  joblib.dump(meta\_model, 'meta\_model\_stacking.pkl')  joblib.dump(scaler, 'scaler\_churn.pkl') |

### 案例 11：基于 FNN-GA 模型的湖泊水质污染指数预测

* **问题背景**：湖泊水质污染问题日益突出，影响生态环境和人类健康。水质污染指数（如 COD、氨氮、总磷等）的评价标准存在一定模糊性（如 “轻度污染”“中度污染” 的界定），且受工业废水排放、农业面源污染、降雨量、水体自净能力等多因素影响，非线性关系显著。
* **问题描述**：某环境监测部门需要对某重点湖泊未来 1 个月的每周水质污染指数进行预测。要求模型能够处理评价标准中的模糊信息，准确预测污染指数变化趋势，为污染防治措施的制定提供科学依据。
* **数据情况**：提供过去 5 年的每周水质监测数据（包括 COD、氨氮、总磷等污染指数），同时提供周边工业废水排放量、农业化肥使用量、降雨量、风速等数据。数据存在因监测设备故障导致的部分时段缺失，且部分污染指数的划分标准具有模糊性。

### 案例 11：FNN-GA 模型湖泊水质污染指数预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Activation  from sklearn.metrics import mean\_squared\_error  from geneticalgorithm import geneticalgorithm as ga  # 数据加载与预处理  data = pd.read\_csv('lake\_water\_quality.csv', parse\_dates=['date'], index\_col='date')  pollution\_data = data['pollution\_index'].values.reshape(-1, 1)  # 数据归一化  scaler = MinMaxScaler(feature\_range=(0, 1))  pollution\_scaled = scaler.fit\_transform(pollution\_data)  # 构建数据集  def create\_dataset(data, look\_back=4):  X, y = [], []  for i in range(len(data) - look\_back):  X.append(data[i:i+look\_back, 0])  y.append(data[i+look\_back, 0])  return np.array(X), np.array(y)  look\_back = 4 # 用过去4周数据预测下一周  X, y = create\_dataset(pollution\_scaled, look\_back)  # 划分训练集和测试集  train\_size = int(len(X) \* 0.8)  X\_train, X\_test = X[:train\_size], X[train\_size:]  y\_train, y\_test = y[:train\_size], y[train\_size:]  # 定义FNN模型结构（由GA优化参数）  def build\_fnn(hidden\_units, activation):  model = Sequential()  model.add(Dense(hidden\_units[0], input\_dim=look\_back))  model.add(Activation(activation))  for units in hidden\_units[1:]:  model.add(Dense(units))  model.add(Activation(activation))  model.add(Dense(1))  model.compile(loss='mse', optimizer='adam')  return model  # 定义GA优化目标函数  def objective\_function(params):  # 参数解析：隐藏层神经元数(2层)、激活函数类型(0:relu,1:sigmoid)、迭代次数  hidden1 = int(params[0])  hidden2 = int(params[1])  activation\_idx = int(params[2])  epochs = int(params[3])    activation = 'relu' if activation\_idx == 0 else 'sigmoid'  model = build\_fnn([hidden1, hidden2], activation)  model.fit(X\_train, y\_train, epochs=epochs, batch\_size=8, verbose=0)  y\_pred = model.predict(X\_test)  return mean\_squared\_error(y\_test, y\_pred)  # 参数范围  varbound = np.array([  [8, 64], # hidden1  [4, 32], # hidden2  [0, 1], # activation  [20, 100] # epochs  ])  # 遗传算法优化  algorithm\_param = {  'max\_num\_iteration': 30,  'population\_size': 15,  'mutation\_probability': 0.1,  'elit\_ratio': 0.1,  'crossover\_probability': 0.5,  'variable\_type': 'int'  }  model\_ga = ga(  function=objective\_function,  dimension=4,  variable\_type='int',  variable\_boundaries=varbound,  algorithm\_parameters=algorithm\_param  )  model\_ga.run()  best\_params = model\_ga.best\_variable  # 构建最优FNN模型  best\_hidden = [int(best\_params[0]), int(best\_params[1])]  best\_activation = 'relu' if best\_params[2] == 0 else 'sigmoid'  best\_epochs = int(best\_params[3])  final\_model = build\_fnn(best\_hidden, best\_activation)  final\_model.fit(X\_train, y\_train, epochs=best\_epochs, batch\_size=8, verbose=1)  # 模型预测  y\_pred = final\_model.predict(X\_test)  # 反归一化  y\_pred\_actual = scaler.inverse\_transform(y\_pred.reshape(-1, 1))  y\_test\_actual = scaler.inverse\_transform(y\_test.reshape(-1, 1))  # 评估模型  mse = mean\_squared\_error(y\_test\_actual, y\_pred\_actual)  print(f'FNN-GA模型MSE: {mse}')  # 可视化结果  plt.figure(figsize=(12, 6))  plt.plot(y\_test\_actual, label='实际污染指数')  plt.plot(y\_pred\_actual, label='预测污染指数')  plt.legend()  plt.savefig('water\_pollution\_prediction.png')  plt.show()  # 保存模型  final\_model.save('fnn\_ga\_water\_quality.h5')  joblib.dump(scaler, 'scaler\_water.pkl') |

### 案例 12：基于 SARIMA-GARCH 模型的黄金期货价格波动率预测

* **问题背景**：黄金期货是重要的投资和避险工具，其价格波动率受全球经济形势、地缘政治风险、通货膨胀预期、美元指数等因素影响，呈现出明显的聚类特性（即高波动率之后往往跟随高波动率，低波动率之后往往跟随低波动率），且具有季节性特征。
* **问题描述**：某金融投资机构需要对黄金期货未来 3 个月的日价格波动率进行预测。要求模型能够捕捉价格的季节性趋势、线性自相关以及波动率聚类特性，为投资组合风险管理和交易策略制定提供依据。
* **数据情况**：提供过去 10 年的黄金期货日收盘价数据，同时提供美元指数、全球主要经济体的利率、通货膨胀率、地缘政治事件记录等。数据量约 2500 条，存在部分交易日因节假日缺失的情况。

### 案例 12：SARIMA-GARCH 模型黄金期货价格波动率预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from statsmodels.tsa.statespace.sarimax import SARIMAX  from arch import arch\_model  from sklearn.metrics import mean\_squared\_error  import statsmodels.api as sm  # 数据加载与预处理  data = pd.read\_csv('gold\_futures.csv', parse\_dates=['date'], index\_col='date')  price\_data = data['price'].dropna()  # 计算收益率（波动率基础）  returns = 100 \* price\_data.pct\_change().dropna() # 百分比收益率  # 划分训练集和测试集  train\_size = int(len(returns) \* 0.8)  train\_returns, test\_returns = returns[:train\_size], returns[train\_size:]  # SARIMA模型拟合均值方程  # 选择最优参数（此处简化为手动指定，实际可通过AIC筛选）  sarima\_order = (1, 0, 1)  seasonal\_order = (1, 0, 1, 22) # 22个交易日为月度周期  sarima\_model = SARIMAX(train\_returns, order=sarima\_order, seasonal\_order=seasonal\_order)  sarima\_result = sarima\_model.fit(disp=False)  # 获取SARIMA残差，用于GARCH模型  sarima\_residuals = sarima\_result.resid  # GARCH模型拟合波动率  garch\_model = arch\_model(sarima\_residuals, vol='GARCH', p=1, q=1)  garch\_result = garch\_model.fit(disp='off')  # 组合预测：先预测均值，再预测波动率  # SARIMA预测均值  sarima\_pred = sarima\_result.get\_forecast(steps=len(test\_returns))  mean\_forecast = sarima\_pred.predicted\_mean  # GARCH预测波动率（条件方差）  garch\_forecast = garch\_result.forecast(horizon=len(test\_returns))  volatility\_forecast = np.sqrt(garch\_forecast.variance.values[-1, :])  # 最终预测（均值±1.96\*波动率，95%置信区间）  upper\_bound = mean\_forecast + 1.96 \* volatility\_forecast  lower\_bound = mean\_forecast - 1.96 \* volatility\_forecast  # 评估波动率预测效果（使用实际收益率的绝对值作为真实波动率近似）  actual\_volatility = np.abs(test\_returns)  mse\_vol = mean\_squared\_error(actual\_volatility, volatility\_forecast)  print(f'波动率预测MSE: {mse\_vol}')  # 可视化结果  plt.figure(figsize=(12, 6))  plt.plot(test\_returns.index, test\_returns, label='实际收益率')  plt.plot(test\_returns.index, mean\_forecast, label='均值预测', color='red')  plt.fill\_between(test\_returns.index, lower\_bound, upper\_bound, color='pink', alpha=0.3, label='95%置信区间')  plt.legend()  plt.savefig('gold\_volatility\_prediction.png')  plt.show() |

### 案例 13：基于 DT-AdaBoost 模型的新能源汽车电池故障诊断预测

* **问题背景**：新能源汽车电池的可靠性直接影响车辆的安全性和续航能力，电池故障的早期预测对保障行车安全、降低维修成本至关重要。电池故障受充放电次数、工作温度、电池老化程度、电压电流波动等因素影响，且故障样本在总样本中占比较小，容易被误判。
* **问题描述**：某新能源汽车制造商需要基于电池的实时运行数据预测未来 1 个月内是否会发生故障（分为轻微故障、严重故障、无故障三类）。要求模型能够聚焦容易错分的故障样本，提高预测的容错性和准确性，为电池维护提供预警。
* **数据情况**：提供该品牌 1 万辆新能源汽车过去 2 年的电池运行数据，包括每小时的电压、电流、温度、充放电次数、电池容量衰减率等，以及电池故障记录（故障类型、发生时间）。数据中故障样本占比约 5%，存在部分车辆的传感器数据异常情况。

### 案例 13：DT-AdaBoost 模型新能源汽车电池故障诊断预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import StandardScaler, LabelEncoder  from sklearn.model\_selection import train\_test\_split  from sklearn.tree import DecisionTreeClassifier  from sklearn.ensemble import AdaBoostClassifier  from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  from sklearn.model\_selection import cross\_val\_score  import joblib  # 数据加载与预处理  data = pd.read\_csv('ev\_battery\_fault.csv')  data = data.dropna()  # 故障类型编码  le = LabelEncoder()  data['fault\_type'] = le.fit\_transform(data['fault\_type']) # 0:无故障,1:轻微故障,2:严重故障  # 特征与目标变量  X = data.drop(['battery\_id', 'fault\_type'], axis=1)  y = data['fault\_type']  # 数据标准化  scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  # 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42, stratify=y)  # 构建DT-AdaBoost模型  base\_dt = DecisionTreeClassifier(max\_depth=3, random\_state=42) # 弱分类器  adaboost\_model = AdaBoostClassifier(  base\_estimator=base\_dt,  n\_estimators=50,  learning\_rate=0.1,  random\_state=42  )  # 模型训练  adaboost\_model.fit(X\_train, y\_train)  # 模型预测  y\_pred = adaboost\_model.predict(X\_test)  y\_pred\_proba = adaboost\_model.predict\_proba(X\_test)  # 评估模型  print(f'准确率: {accuracy\_score(y\_test, y\_pred)}')  print('混淆矩阵:')  print(confusion\_matrix(y\_test, y\_pred))  print('分类报告:')  print(classification\_report(y\_test, y\_pred, target\_names=['无故障', '轻微故障', '严重故障']))  # 交叉验证  cv\_scores = cross\_val\_score(adaboost\_model, X\_scaled, y, cv=5)  print(f'交叉验证准确率: {np.mean(cv\_scores)} ± {np.std(cv\_scores)}')  # 特征重要性可视化  feature\_importance = adaboost\_model.feature\_importances\_  features = X.columns  plt.figure(figsize=(10, 6))  plt.barh(features, feature\_importance)  plt.xlabel('特征重要性')  plt.title('AdaBoost特征重要性')  plt.savefig('battery\_feature\_importance.png')  plt.show()  # 保存模型  joblib.dump(adaboost\_model, 'dt\_adaboost\_battery\_fault.pkl')  joblib.dump(scaler, 'scaler\_battery.pkl')  joblib.dump(le, 'label\_encoder\_battery.pkl') |

### 案例 14：基于 LSTM-SVR 模型的风电场风速短期预测

* **问题背景**：风能是一种清洁可再生能源，风电场的发电量取决于风速，而风速具有强烈的非线性和随机性，受地形、气候、湍流等因素影响，短期风速预测难度较大。准确预测风速对风电场的功率调度、电网稳定运行具有重要意义。
* **问题描述**：某风电场需要对场内 10 台风机未来 48 小时的每 10 分钟风速进行预测。要求模型能够先捕捉风速的整体非线性变化趋势，再对预测残差进行修正，以降低过拟合风险，提高预测精度，为风电场的发电计划制定提供支持。
* **数据情况**：提供该风电场过去 3 年的每 10 分钟风速数据，同时提供风机的位置信息、地形数据、气象数据（温度、湿度、气压、风向等）。数据存在因设备维护导致的部分时段缺失，且风速受湍流影响存在高频波动。

### 案例 14：LSTM-SVR 模型风电场风速短期预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from sklearn.svm import SVR  from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import LSTM, Dense  import joblib  # 数据加载与预处理  data = pd.read\_csv('wind\_farm\_data.csv', parse\_dates=['time'], index\_col='time')  wind\_speed = data['wind\_speed'].values.reshape(-1, 1)  # 数据归一化  scaler = MinMaxScaler(feature\_range=(0, 1))  wind\_scaled = scaler.fit\_transform(wind\_speed)  # 构建序列数据  def create\_sequences(data, look\_back=12):  X, y = [], []  for i in range(len(data) - look\_back - 1):  X.append(data[i:i+look\_back, 0])  y.append(data[i+look\_back, 0])  return np.array(X), np.array(y)  look\_back = 12 # 用过去12个10分钟数据预测下一个10分钟  X, y = create\_sequences(wind\_scaled, look\_back)  # 划分训练集和测试集  train\_size = int(len(X) \* 0.8)  X\_train, X\_test = X[:train\_size], X[train\_size:]  y\_train, y\_test = y[:train\_size], y[train\_size:]  # 重塑LSTM输入形状 [samples, time steps, features]  X\_train\_lstm = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)  X\_test\_lstm = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)  # 第一步：LSTM模型预测  lstm\_model = Sequential()  lstm\_model.add(LSTM(64, input\_shape=(look\_back, 1), return\_sequences=False))  lstm\_model.add(Dense(32, activation='relu'))  lstm\_model.add(Dense(1))  lstm\_model.compile(loss='mse', optimizer='adam')  lstm\_model.fit(X\_train\_lstm, y\_train, epochs=30, batch\_size=32, verbose=1)  # LSTM预测及残差计算  lstm\_pred\_train = lstm\_model.predict(X\_train\_lstm).flatten()  lstm\_pred\_test = lstm\_model.predict(X\_test\_lstm).flatten()  train\_residual = y\_train - lstm\_pred\_train # 训练残差  test\_residual = y\_test - lstm\_pred\_test # 测试残差  # 第二步：SVR模型拟合残差  svr\_model = SVR(kernel='rbf', C=10, gamma=0.1)  svr\_model.fit(X\_train, train\_residual) # 用原始特征预测残差  # SVR残差预测  svr\_residual\_pred = svr\_model.predict(X\_test)  # 组合预测：LSTM预测 + SVR残差修正  final\_pred = lstm\_pred\_test + svr\_residual\_pred  # 反归一化  final\_pred\_actual = scaler.inverse\_transform(final\_pred.reshape(-1, 1))  y\_test\_actual = scaler.inverse\_transform(y\_test.reshape(-1, 1))  lstm\_pred\_actual = scaler.inverse\_transform(lstm\_pred\_test.reshape(-1, 1))  # 评估模型  print(f'LSTM模型MAE: {mean\_absolute\_error(y\_test\_actual, lstm\_pred\_actual)}')  print(f'LSTM-SVR组合模型MAE: {mean\_absolute\_error(y\_test\_actual, final\_pred\_actual)}')  # 可视化结果  plt.figure(figsize=(12, 6))  plt.plot(y\_test\_actual[:100], label='实际风速')  plt.plot(lstm\_pred\_actual[:100], label='LSTM预测', alpha=0.7)  plt.plot(final\_pred\_actual[:100], label='组合模型预测', alpha=0.7)  plt.legend()  plt.savefig('wind\_speed\_prediction.png')  plt.show()  # 保存模型  lstm\_model.save('lstm\_wind\_speed.h5')  joblib.dump(svr\_model, 'svr\_residual\_wind.pkl')  joblib.dump(scaler, 'scaler\_wind.pkl') |

### 案例 15：基于混沌时间序列 - PSO-BP 模型的股票市场指数预测

* **问题背景**：股票市场指数（如上证指数、深证成指）的波动具有混沌特性，即短期可预测但长期因初值敏感性而难以预测，受宏观经济、政策变化、市场情绪等多种因素影响，呈现复杂的非线性关系。
* **问题描述**：某证券研究机构需要对某股票市场指数未来 10 个交易日的收盘指数进行预测。要求模型能够通过相空间重构揭示指数波动的内在混沌规律，结合优化后的神经网络提高预测精度，为投资者提供参考。
* **数据情况**：提供该股票市场指数过去 10 年的日收盘指数数据，同时提供宏观经济指标（GDP 增长率、利率、汇率等）、政策公告、市场成交量、投资者情绪指标等。数据量约 2500 条，存在部分交易日因节假日缺失的情况，且指数波动呈现明显的混沌特性。

### 案例 15：混沌时间序列 - PSO-BP 模型股票市场指数预测代码

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from sklearn.metrics import mean\_squared\_error  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Activation  from pyswarm import pso  import joblib  from scipy.spatial.distance import pdist, squareform  # 数据加载与预处理  data = pd.read\_csv('stock\_index.csv', parse\_dates=['date'], index\_col='date')  index\_data = data['close\_index'].values.reshape(-1, 1)  # 数据归一化  scaler = MinMaxScaler(feature\_range=(0, 1))  index\_scaled = scaler.fit\_transform(index\_data)  # 相空间重构（混沌时间序列处理）  def phase\_space\_reconstruction(series, delay=5, dim=3):  """  相空间重构  series: 一维时间序列  delay: 延迟时间  dim: 嵌入维数  """  n = len(series)  m = n - (dim - 1) \* delay  if m <= 0:  raise ValueError("重构参数不合适，无法构建相空间")  X = np.zeros((m, dim))  for i in range(dim):  X[:, i] = series[i \* delay : i \* delay + m].flatten()  return X[:, :dim-1], X[:, -1] # 前dim-1列为特征，最后一列为目标  # 计算最优延迟时间（互信息法简化）  def find\_optimal\_delay(series, max\_delay=20):  delays = range(1, max\_delay+1)  mi = []  for d in delays:  x1 = series[:-d].flatten()  x</doubaocanvas> |