基于改进 CatBoost-LSTM 算法的电熔镁砂群炉需量预报

摘要

本文主要针对电熔镁砂生产过程需量预报问题展开研究。首先依据过去时刻的需量和功率实现需量单步预报。然后为避免需量尖峰导致不必要切断供电，本文基于峰值变化周期，通过改进的 CatBoost-LSTM 实现需量多步预报并对未来需量尖峰进行预警。

针对问题一，基于电熔镁砂生产过程需量与过去时刻的需量和功率的关系，本文建立了**单步需量预报模型**。首先通过滑动窗口计算需量作为观测值，然后通过标准化在求解过程中对功率和需量进行特征缩放。之后运用 **Lasso 回归**与 **CatBoost** 进行训练，利用**粒子群算法**优化权重并进行两个模型线性融合，并通过 RMSE 确定使用过去功率和需量进行预测的**最佳数据量**为 **3**，实现对需量的单步预报，最后利用**中值滤波**对预测结果进行修正。通过**时间序列交叉验证**判断模型未出现过拟合现象，通过测试集评价精度，得到模型的 RMSE、MAPE、RMAPE、R-Squard 分别为 **24.2233**、**0.1083**、**0.1039**、**0.918**.

针对问题二，本文建立了**基于差分峰值检测的周期规划模型**，以选取能够判断别尖峰的预报采样周期步长。首先初步分析需量的变化情况并进行**平滑性检验**，发现序列存在不平稳现象。然后通过 **ACF 检验**分析需量峰值的出现具有周期性。之后通过差分序列统计尖峰的分布情况，以峰值之间的平均时间间隔最大化为目标函数，将总时间间隔与峰值数量保持基本一致作为约束条件建立非线性规划模型，最后利用 SLSQP 求得**最优采样周期**为 **10** 个采样周期，并利用 **STL** 进行分析验证。

针对问题三，本文基于问题二将 10 个采样周期作为预报步长，建立**多步需量预报模型**。首先对 **LSTM**、**Seq2Seq** 、线性回归、CatBoost 进行训练，通过粒子群算法求得最优权重分别为 1、0.9983、0.9733、1，将四个模型进行交叉融合，进行需量多步预报并绘制真实值预报值对比曲线。最后通过计算 RMSE、RMAPE 、**TPR**、**TNP**、R-Squard 对预报精度进行评价，TPR 与 TNR 在测试集上均大于 **0.75**，R-Squard 均大于 **0.89**.

针对问题四，本文建立了**基于小波变换的尖峰预报模型**以判别需量尖峰。首先通过对尖峰特性进行分析并给出尖峰的判别算法。最后利用**小波变换**判断峰值，对问题三的多步预报模型进行改进，对测试数据样本进行尖峰预报并与实际结果进行比较，分析得到 **91.3%** 的预报结果能提前 1 分钟左右对未来时刻的尖峰做出准确的尖峰预警。

本文的亮点有：1. 采用滚动窗口实现时间序列交叉验证，多维客观评价预报精度；2. 集成深度学习和机器学习模型，实现时间序列森林算法，提高预报精度。

关键词：粒子群权重优化　CatBoost　Seq2Seq-LSTM　需量多步预报

目录

1 问题重述

1.1 问题背景

电熔镁砂是生产航天、航空、和其他工业领域不可缺少的耐火材料。其生产原料是菱镁矿石；生产设备是三相交流电熔镁炉；生产过程为经过备炉、起炉和正常熔炼将电能转换为热能，让生产原料受热熔化为氧化镁熔液，结束后溶液冷却成为电熔镁砂，最后进行破坨和分拣。但熔炼电熔镁砂耗电量巨大，电能成本在总成本的占比中超过60%，因此节能减排是生产企业的最关心的问题之一。而在生产过程中预报出电熔镁砂需量和变化趋势可有效帮助减少不必要的能源消耗[1]。

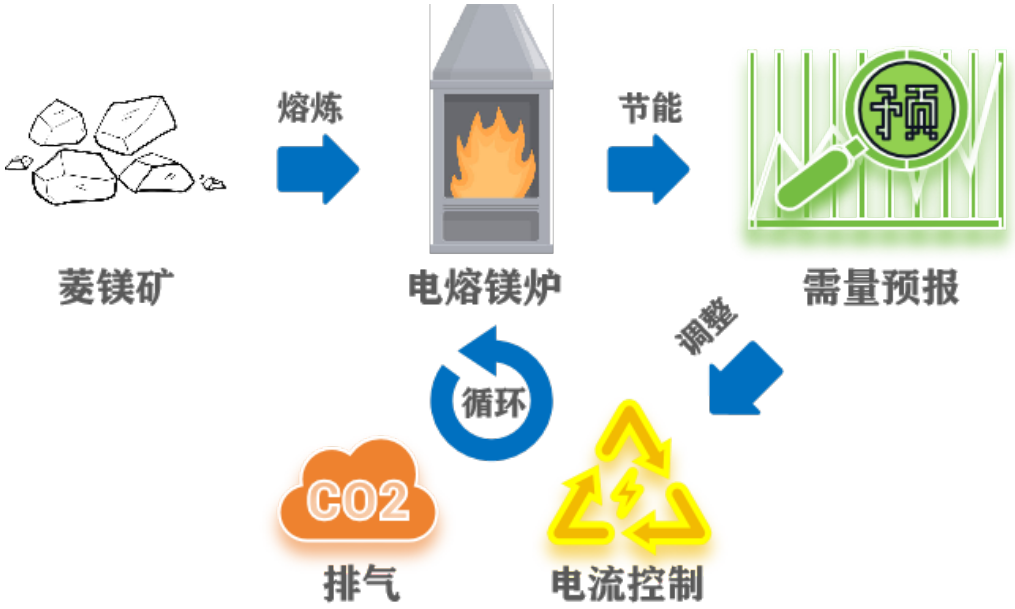

图 1　问题背景图

1.2 问题提出

已知电熔镁砂生产过程中需量与功率的训练数据和测试数据，要求解决如下问题：

问题一：当前时刻的电熔镁砂群炉需量表示该时刻和过去 29 个采样周期生产用电功率的均值。要求建立生产过程的需量预报模型，并评价需量预报值的精度。

问题二：在生产过程中由于阻抗减小但最优熔化电流值恒定，需量会出现先升后降的尖峰现象。建立需量多步预报模型来预估未来的需量变化，以根据趋势来判断尖峰，基于过去时刻出现的尖峰情况来分析需要的未来采样周期数目。

问题三：基于问题二，要求建立需量多步预报模型，对该模型的预报精度进行评价，并绘制关于预报值与真实值的对比曲线。

问题四：基于问题三，要求编写判别需量尖峰的算法，并病假需量尖峰预报精度。

2 模型假设

1. 假设附件中数据均真实可信，可以直接用于模型训练;
2. 假设过去时间戳对应的数据可以作为已知值用于预测;
3. 假设每个时刻的电流、熔池高度、杂质含量、气体等影响因素可以由功率反映;
4. 假设熔化电流系统反应时间忽略不计，到达尖峰区域立即可以采取断电措施。

c

3 符号说明

符号	说明
$\bar{P}(k)$	k 时刻的电熔镁砂需量
$p(k)$	第 k 时刻的功率
T	峰值变化周期
M	需量阈值
W_f	小波系数
μ	序列中检验到的峰值数量
S_t	趋势分量
$d(t)$	差分序列

注：表中未列出及重复的符号均以首次出现处为准。

4 问题一模型的建立及求解

4.1 问题一分析

问题一要求根据过去时刻的二功率和需量预报生产过程需量。首先我们通过滑动窗口法计算生产需量的观测值，为了避免数值计算问题并增加数值稳定性我们引入特征工程，在下面求解过程中进行利用标准化方法进行特征缩放。之后为了捕捉功率和需量序列的连续与离散特征，本文分别用 Lasso 回归和 CatBoost 进行预报，再通过 PSO 优化模型权重将两个模型进行线性融合。接着，我们引入使用过去功率和需量进行预测的数据量作为滞后值，通过统计 RMSE 随滞后值的变化情况，我们将预报模型中的滞后值设置为 3，并将由混合模型最终求得的预报值采用中值滤波进行去噪处理进行修正。最后利用时间序列交叉验证检验模型是否出现过拟合，并利用测试集求取预报精度，从两方面对模型进行评价。

问题一的思路图如下。

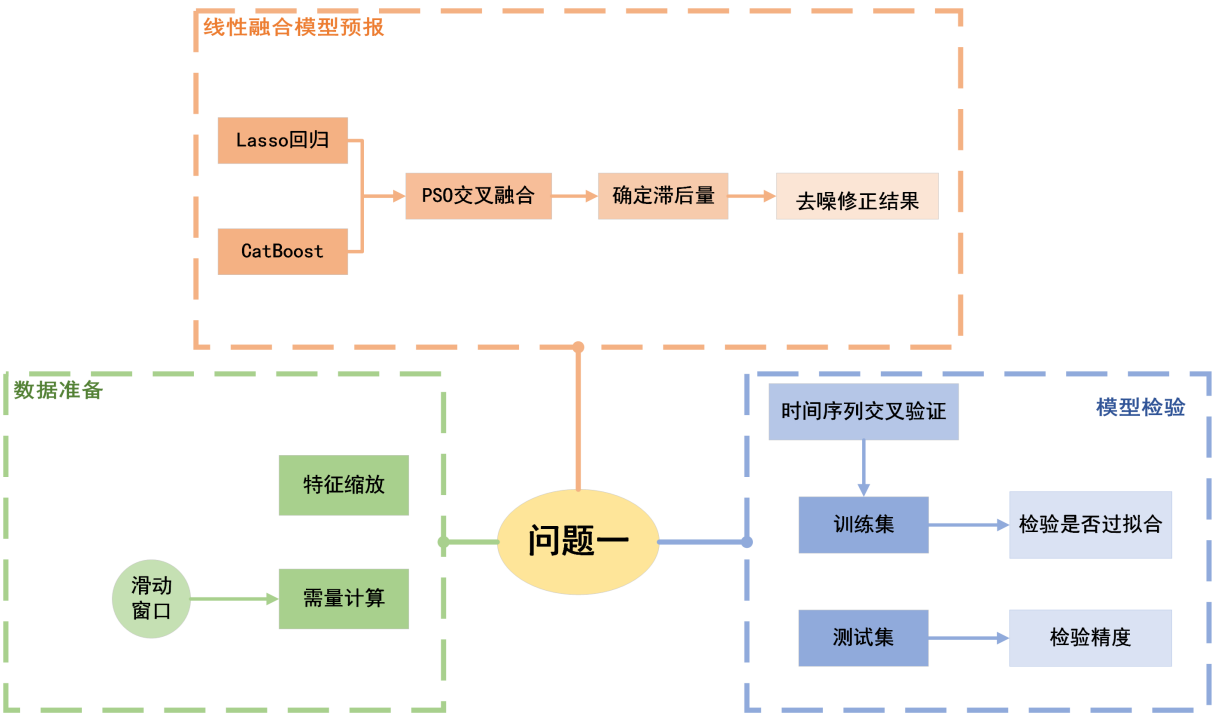

图 2　问题一思路图

4.2 数据准备

4.2.1 滑动窗口法计算需量

由题目可知电熔镁砂群炉需量是指该时刻和过去 29 个采样周期生产用电功率的均值。本文利用滑动窗口法进行计算。以功率时间序列的初始时间节点作为窗口放置的起

始位置，以 30 个时间步长为滑动窗口的大小，通过循环或迭代的方式实现窗口的不断滑动，在每个滑动窗口位置，利用如下公式求得电熔镁砂群炉需量的观测值：

$$\bar{P}(k)=\frac{1}{n}\sum_{j=0}^{n-1}p(k-j). \tag{1}$$

其中 $\bar{P}(k)$ 表示 k 时刻的电熔镁砂需量；$p(k-j)$ 表示第 $k-j$ 时刻的功率；n 表示 30 个采样周期长短。

4.2.2 标准化进行特征缩放

为了避免后面模型计算中出现“大数吃小数”的现象，本文对需量和功率两类特征数据进行如下标准化处理，对数据特征进行压缩，最后再将预报的结果按照同一特征缩放类进行还原。

$$x'=\frac{x-\mu}{\sigma}. \tag{2}$$

4.3 单步需量预报模型

已知当前需量与先前时刻的功率和需量有关，而且由于观测功率和需量的数据具有滞后性，本文建立预报模型，决定利用前一段时间内的观测功率和需量值来预报当前时刻的需量，之后再利用观测量进行下一时刻的预报。由于不同模型在预测时利用的序列特征不同，为提升预报的准确性，本文采用集成模型进行预报。

4.3.1 Lasso 回归

由于前一段时间内的观测功率和需量值之间存在函数关系，因此本文采用 Lasso 回归对生产需量进行预测，以处理部分变量间的多重共线性关系并提高预报精度，并捕捉时间序列的连续特征。

Lasso 回归原理是寻找合适的回归系数以使如下目标函数最小化，并通过引入 L1 正则化项作为惩罚项，实现特征选择，将不相关或荣誉的特征的系数缩减为 0，从而避免过拟合。

$$\min\ \mathrm{RRS}+\alpha\times\sum|\beta| \tag{3}$$

其中 RRS 表示残差平方和，即观测值与预报值之间差异的平方值；β 表示回归系数；$|\beta|$ 表示 L1 正则化项，即回归系数的绝对值之和；α 表示正则化参数，可控制正则化项的强度。

4.3.2 CatBoost

CatBoost 是由 Categorical 和 Boosting 组成，使用对称决策树作为基学习器，具有参数较少、支持类别型变量和高准确性等特点的 GBDT 框架[2]。它具有高效合理地处理

类别型特征的能力，能够依据特征变量与目标变量之间的有效信息，精准分析出特征变量对目标变量之间的影响。为了捕捉时间序列的离散特征，实现更加精准的预测，本文采用 CatBoost 对生产需量进行再次进行预报。

本文将类别按时刻不断进行细分，从而可将连续数据近似为离散数据。在求解中，处理类别特征是十分关键的步骤，本文采用如下 Greedy TS 进行处理，可以有效避免过拟合现象的发生。

$$\hat{x}_{i,k} = \frac{\sum_{j=1}^{n} [x_{i,j} = x_{i,k}] \cdot y_j}{\sum_{j=1}^{n} [x_{i,j} = x_{i,k}]} \tag{4}$$

其中 $x_{i,k}$ 表示第 k 个训练样本的第 i 个类别特征，y 表示根据类别条件估计的预期目标。当 $x_{i,k} = x_{j,k}$ 时，$[x_{j,k} = x_{i,k}] = 1$；当 $x_{i,k} \neq x_{j,k}$ 时，$[x_{j,k} = x_{i,k}] = 0$.

Catboost 的原理主要如下图所示。

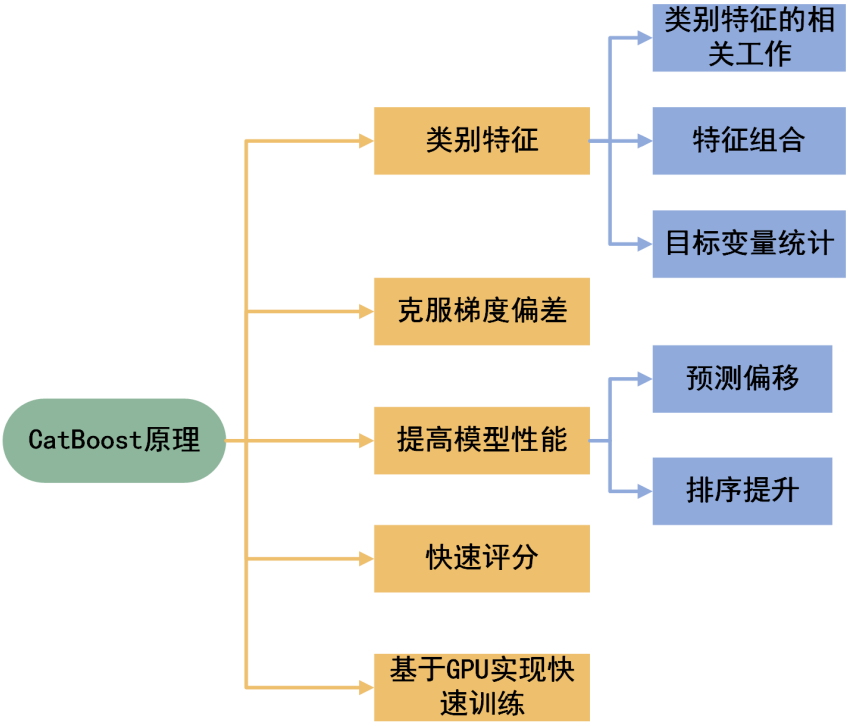

图 3　CatBoost 原理图

4.3.3 PSO 交叉融合

由于以上两个模型分别从序列的不同特征进行考虑，仅对连续或离散某一方面的特征较敏感，单一考虑一个模型可能会导致预测结果不稳定，因此本文通过 PSO 求得最优模型权重[10]，通过如下加权方式交叉融合上述模型，以增加整体预测的鲁棒性与准确性。

$$Y_{model,i} = \omega_1 \times Y_{Lasso,i} + \omega_2 \times Y_{CatBoost,i} \tag{5}$$

其中 ω_1 表示 Lasso 模型的权重；ω_2 表示 CatBoost 模型的权重；$Y_{model,k}$ 表示融合后模型预测的 k 时刻的需量值；$Y_{\text{Lasso},k}$ 表示 Lasso 回归模型预测的的 k 时刻的需量值；$Y_{\text{CatBoost},k}$ 表示 CatBoost 回归模型预测的的 k 时刻的需量值。

PSO(粒子群算法) 中每个粒子都代表问题的一个潜在最优解，具有适应度值、速度和位置三项指标特。在求解过程中，粒子会不断随自身及其他粒子的移动经验即通过跟踪个体极值 Pbest 和群体极值 Gbest 更新个体位置，进行动态调整，从而实现个体在可解空间中的寻优。

针对本题，本文将每个粒子的维度为权重个数即 2，规定权重范围为 [0,1]，将预报值与观测值的均方误差最小化作为目标函数。

$$\min \text{RMSE} = \sqrt{\frac{\sum_{k=1}^{N}\left(Y_{obs,i}-Y_{model,i}\right)^2}{n}} \tag{6}$$

其中 RMSE 表示模型预测值与观测值的均方根误差；N 表示附件训练集样本数据量即 4658；$Y_{obs,i}$ 表示型需量的观测值。

具体实现步骤如下图所示。

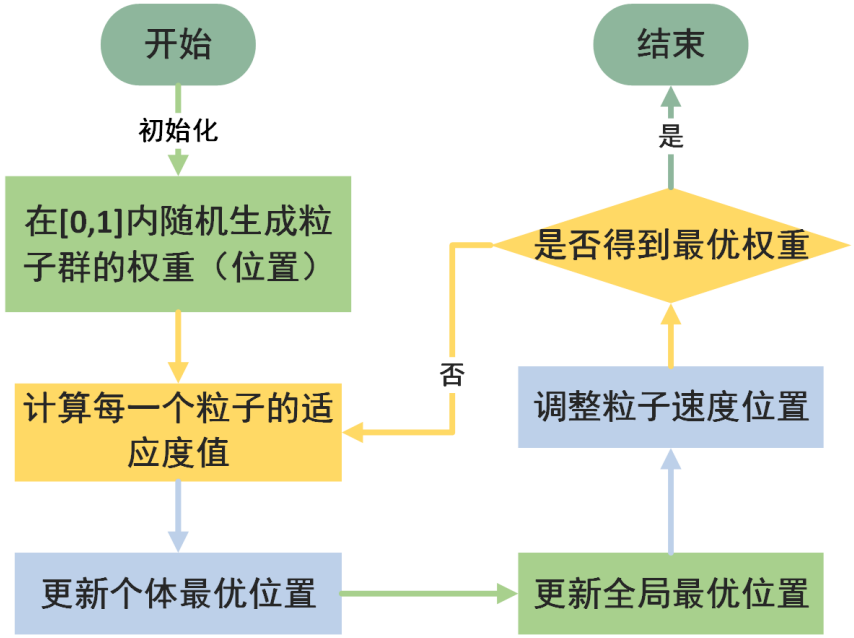

图 4　问题一 PSO 主要求解步骤

基于上述步骤，求得 Lasso 回归和 CatBoost 的权重分别为 0.8182、0.8586.

4.3.4 确定滞后量

滞后量可以用来描述当前值与过去观测值之间的关系，因此本文引入滞后量特征来捕捉生产需量在时间上的滞后效应，进而预报未来的需量。当滞后量过大即需要根据过去较长一段时间的数据进行预报，会造成模型的实用性减弱；而滞后量过小则会导致预

报精度下降。为确定最优滞后量，本文选取不同滞后量进行预报，以 RMSE 来衡量预报精度，得到 RMSE 随滞后值的变化情况如下。

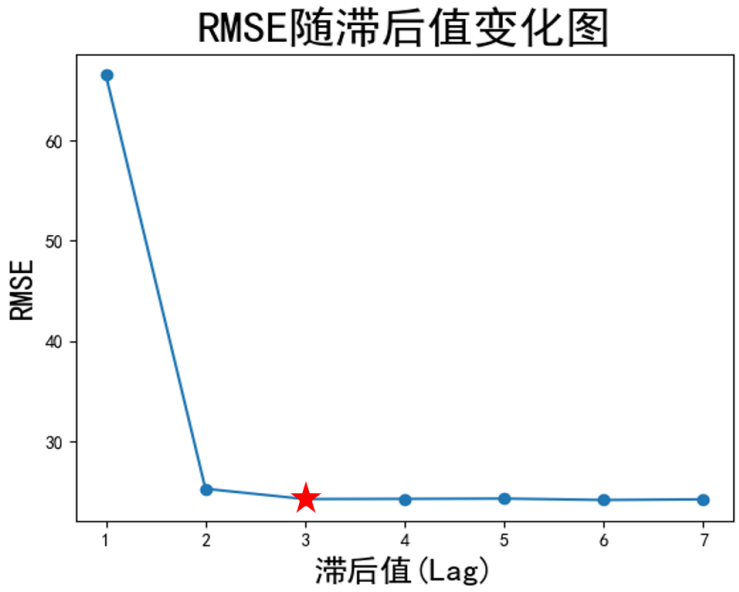

图 5　RMSE 随滞后值变化图

从图中可以看出、当滞后量从 1 增加到 3 时，预报精度上升较快；但当滞后量超过 3 时，预报精度上升趋势变缓。对此，本文确定最优滞后量为 3，即利用过去前三个时刻的功率和需量来预报当前时刻的需量。

4.3.5 去噪修正

需量预报结果可能存在噪声或异常值。由于需量是按时刻进行预报，可将结果看作时间序列，因此本文采用中值滤波对预报结果进行去噪处理，使结果更加平滑。它将通过计算序列中每一个数据的邻域的中值来平滑噪声，并将中值作为该数据的新值，从而对预报结果进行修正。

4.4 模型检验

4.4.1 时间序列交叉验证

为了检验模型是否过拟合，本文将附件中的训练集数据作为样本数据集，利用时间序列交叉验证对上述预报模型性能进行检验。在传统的交叉验证方法中，数据集通常会被打乱然后划分为训练集和测试集，然后用训练集训练模型，再用测试集评估模型的性能。然而，对于时间序列数据，时间的顺序是至关重要的，因此随机打乱数据并不适合时间序列模型的评估。时间序列交叉验证通过在时间轴上进行划分，确保测试集中的时

间点始终在训练集之后。这样可以更准确地模拟模型在未来数据上的预测能力，避免了将未来信息引入训练集造成过于乐观的评估。

本文采用的是时间序列中交叉验证中的滚动窗口交叉验证法。具体流程为设置一个固定大小的窗口，将该窗口内及已经过的的数据作为训练集，窗口之后数据作为测试集，每次迭代都会移动窗口以包含更多的数据。本文共将滚动分割次数设置为 5，则训练集数据量分别为 3720、2791、1860、930、371. 求解得到时间序列的交叉验证的结果如下表所示。

表 1　时间序列交叉验证结果

时间序列交叉验证	RMSE	MAPE	RMAPE	R-Squared
第 1 次	61.2211	0.21603	0.0021603	0.919928
第 2 次	24.3753	0.00068	0.0000068	0.919636
第 3 次	36.2924	0.00094	0.0000094	0.919668
第 4 次	24.8769	0.00066	0.0000066	0.919518
第 5 次	18.2520	0.00051	0.0000051	0.919795

从表中可以看出预报模型在五次验证结果上都表现良好，没有明显的性能差异，因此说明该模型未出现过拟合现象。

4.4.2 测试集检验精度

为了检验预测精度，本文将附件的测试集代入用训练集训练好的模型，求得评估预测模型性能的相关指标如下表所示。

表 2　模型评估结果

评估指标	RMSE	MAPE	RMAPE	R-Squared
数值大小	24.2233	0.1083	0.1039	0.9198

从表中可以看出均方误差 MSE、平均绝对百分比误差 MAPE、相对平均绝对百分比误差 RMAPE 均较小，R-Squared 接近于 1。明显地，该模型需量预报的精度较高。

5 问题二模型的建立与求解

5.1 问题二分析

问题二要求在多步预报模型中需要估计多少采样周期的需量才能判断尖峰。首先进行初步观察发现尖峰的出现可能存在周期性，然后通过平稳性检验和 ADF 检验确定了需量序列缺失存在周期。因此我们只需要求得峰值出现的周期，即可预报识别尖峰。本文采用差分峰值检测法得到尖峰出现的历史时刻，接着建立非线性目标规划模型求得最佳周期，最后借助 STL 进行检验。

问题一的思路图如下。

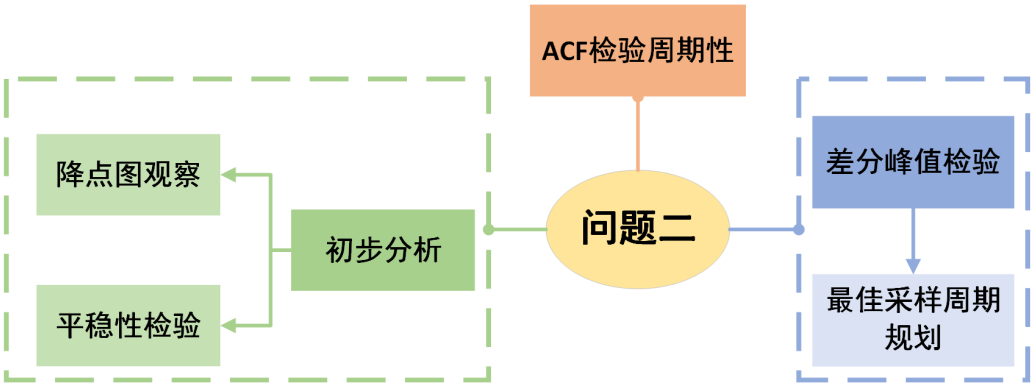

图 6　问题二思路图

5.1.1 初步分析

根据题目，当电熔镁砂群炉需量达到限幅值时，某台电熔镁炉的供电会被切断，从而会使电熔镁砂群炉需量下降。观察附件训练集的需量变化情况，选取一段时间为例其变换情况如下图所示。

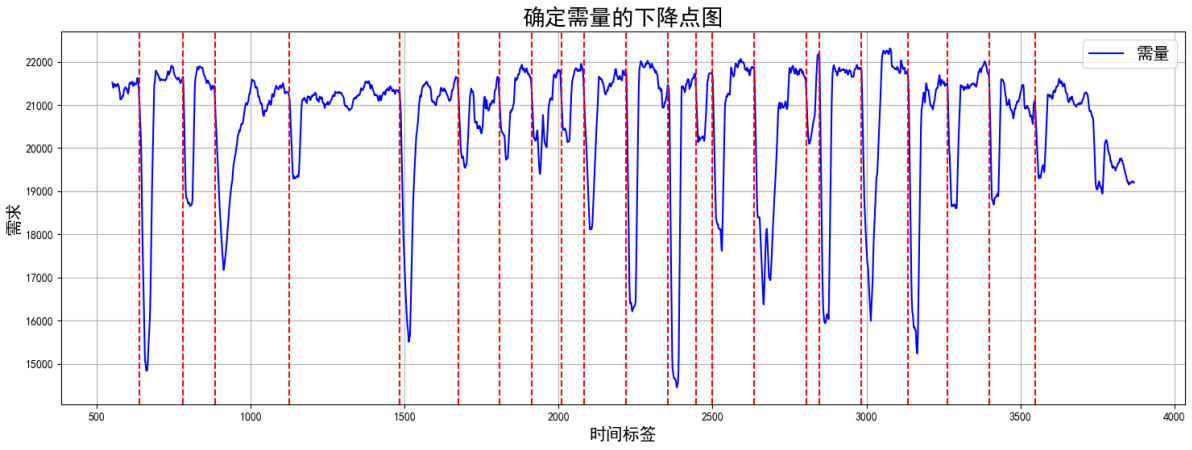

图 7　需量变化情况图

本文用红色虚线标注出了需量急剧下降时刻，直观地可以看出需量出现周期性骤降的现象。据此，本文猜测尖峰周期性出现，当出现时需量超过限幅值，供电切断引发功率骤降，从而需量也呈现断崖式下降的情况。

初步对需量序列进行平稳性检验，求得 ADF 统计量为-1.0351、p 值为 0.7403. 由于 p 值大于 0.05 的显著水平，说明该序列数据在单位根的存在下是非平稳的。数据确实可能具有趋势性和周期性。

5.2 ACF 检验周期性

为了进一步确定序列的周期性，本文利用 ACF（Autocorrelation Function，自相关函数）进行检验。ACF 是一种用于衡量时间序列数据中自相关性的函数。滞后版本是指表示时间序列中两个观察值之间的时间间隔，由于周期性会导致滞后版本之间出现高相关性，可以通过算时间序列数据与其自身滞后版本之间的相关性来确定序列的周期性。其中周期的长度对应于具有高自相关性的滞后值。自相关系数的公式如下：

$$ACF\left(k\right)=\frac{Cov\left[x\left(t\right),x\left(t-k\right)\right]}{Var\left[x\left(t\right)\right]} \tag{7}$$

其中 $ACF(k)$ 表示滞后期为 k 的自相关系数；$Cov\left[x\left(t\right),x\left(t-k\right)\right]$ 表示时间序列在时刻 t 和时刻 $t-k$ 处的协方差；$Var\left[x\left(t\right)\right]$ 表示时间序列在时刻 t 的方差。

求得自相关图如下所示，该图是以滞后项（lag）为横轴，对应的自相关系数为纵轴的图像。每个滞后项上的自相关系数表示该滞后项与原序列在该滞后项处的相关性。若在某个滞后项处自相关系数为正且显著不为零，表示在该滞后项处存在一个正相关的关系。若为负且显著不为零，表示在该滞后项处存在一个负相关的关系。若接近于零，说明在该滞后项处不存在明显的自相关关系。从图中可以看出序列具有周期性。

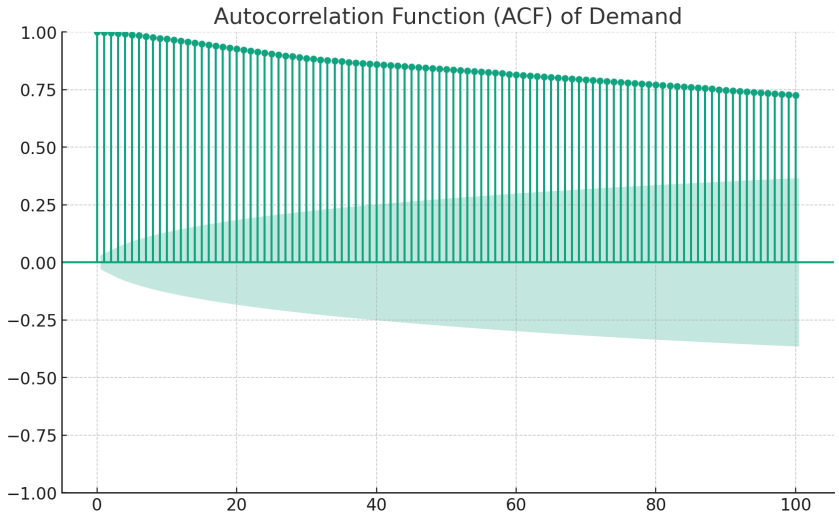

图 8　自相关图

5.3 基于差分峰值检测的周期规划模型

本文希望能根据历史数据确定峰值的出现周期，首先通过差分法检测峰值并进行记录。然后通过建立目标规划模型寻找最优周期。

5.3.1 差分峰值检测

根据图 7 可直观看出需量的时间序列的峰值特征明显且峰值形状较为简单，因此本文决定通过观察差分序列的变化来识别峰值点。通过如下公式生成差分序列 $D = [d(1), d(2), ..., d(N-1)]$，通过寻找差分序列中的由正变负的零交叉点，定位到峰值。

$$d(t) = \bar{P}(t+1) - \bar{P}(t) \tag{8}$$

其中 $\bar{P}(t+1)$ 表示 $t+1$ 时刻的需量；$\bar{P}(t)$ 表示 t 时刻的需量；$d(t)$ 表示两者差分。

经查找相关资料，本文规定最小需量峰值为 21000kw，从而对检测到的峰值进行约束。最终求得峰值数目为 64.

5.3.2 周期目标规划

当选取的周期过长时，预报准确率较低；而当选取的周期过短时，存在部分周期中未识别到峰值的情况。本文为了确定最佳周期即多步预报模型的预报步长，建立非线性目标规划模型。

目标函数：为保证尽可能预报识别到所有的尖峰，本文将目标函数设置为峰值之间的平均时间间隔最大化。

$$\max T = \sum_{i=0}^{\mu-1} \frac{1}{\mu} |t_{i+1} - t_i| \tag{9}$$

其中 T 表示峰值变化周期即预报步长；μ 表示通过差分峰值检测法检测到的尖峰数目；t_i 表示记录的第 i 个峰值时刻。

约束条件：为保证确保峰值之间的总时间间隔与峰值数量基本一致，设置如下约束

$$\left| \frac{t_\mu - t_0}{T} - \mu \right| < \varepsilon \tag{10}$$

其中 ε 表示可允许的误差项，本文设置为 3.

最终通过 SLSQP 求解上述规划模型，得到最佳周期为 10 个采样周期即 70s.

5.4 STL 验证

为检验所求周期的正确性与合理性，本文利用 STL[4] 进行分析。STL 主要由内循环和外循环两个部分构成。内循环主要进行季节性平滑以更新季节分量，其主要流程如下图所示。外循环主要负责为下一轮内循环计算并赋予稳健性权重，以减少异常值的影响。

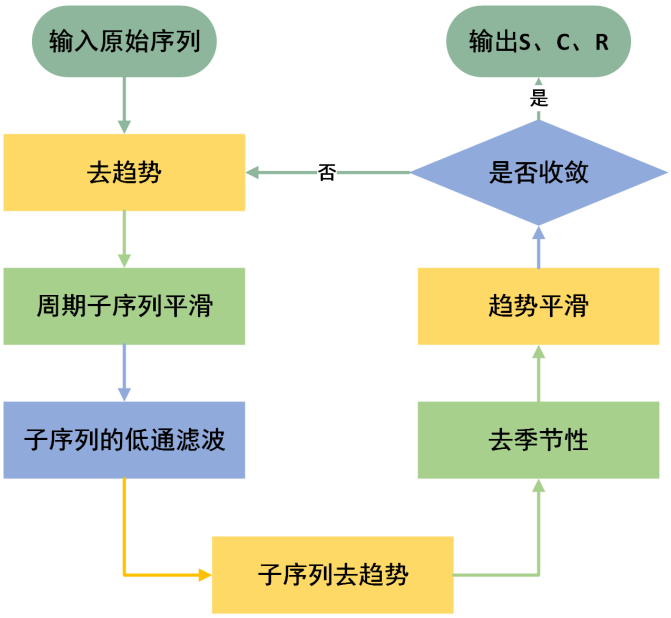

图 9　STL 内循环过程

本文将关于生产过程需量的时间序列通过 STL 进行分解，得到趋势、周期和残差三个成分，即

$$\bar{P}_t = f\left(T_t, S_t, R_t\right) \tag{11}$$

其中 T_t 表示周期分量；S_t 表示趋势分量；R_t 表示残差分量。

最终时间序列经 STL 分解得到三种成分图如下。趋势成分图的走向与分解之前相似，存在上升和下降的模式。从周期性图中可以看出，序列存在明显的周期性位置，确实存在每 10 个时间步长中的重复模式，这与我们选择的周期 10 相匹配，印证了结果的准确性。残差是数据中除去趋势和季节性后的随机部分，可以用来检测数据中的异常值或其他模式。

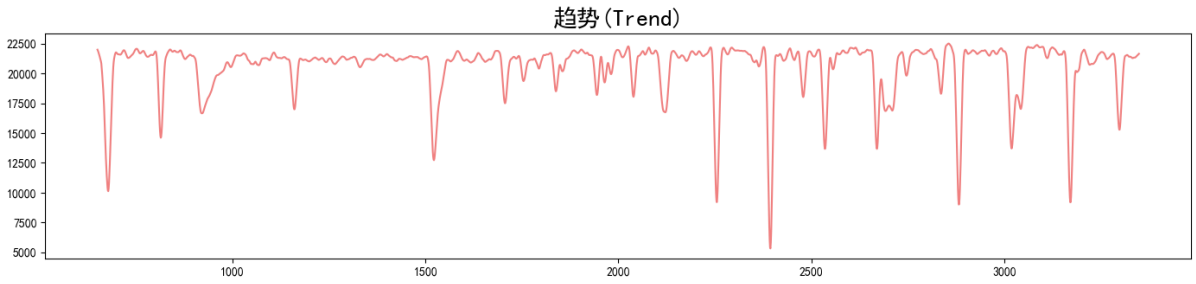

图 10　STL 分解结果图-趋势

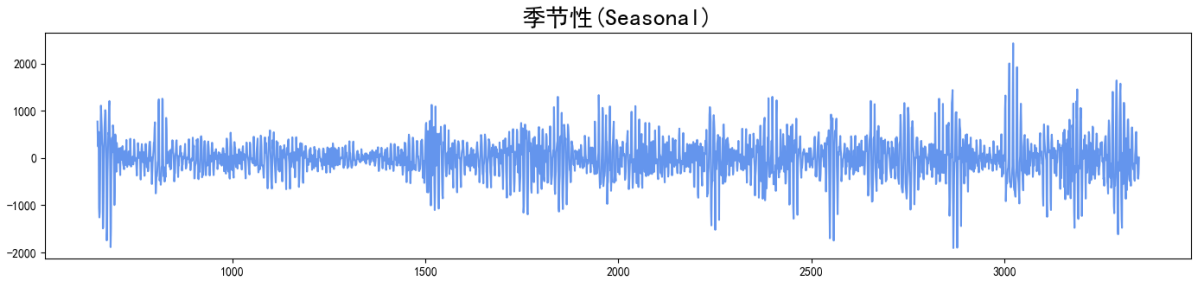

图 11　STL 分解结果图-季节性

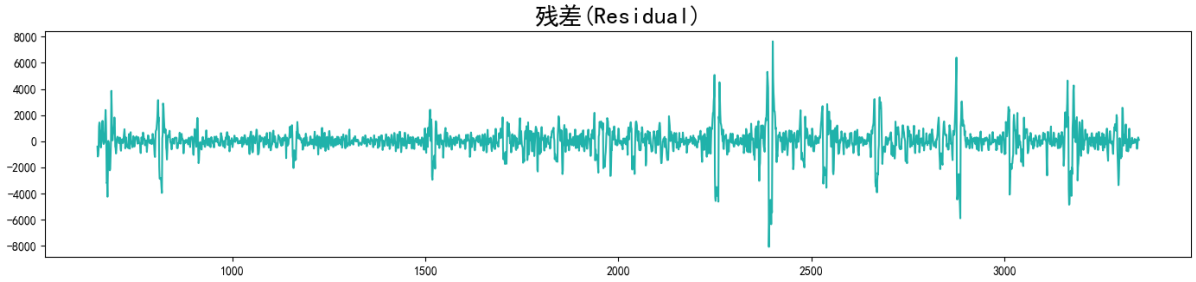

图 12　STL 分解结果图-残差

6 问题三模型的建立与求解

6.1 问题三分析

问题二求得的预报的多步长为 10 个采样周期，问题三要求在此基础上进行需量多步预报。首先，我们根据据电熔镁砂群炉需量含义建立多步需量预报模型，然后通过 LSTM、Seq2Seq、线性回归、CatBoost 分别进行预报，并通过 PSO 分配权重将四种学习模型进行集成。最后通过 TPR、TNR、RMSE、RMAPE、R-Squard 对预报精度进行检验。

问题三的思路图如下。

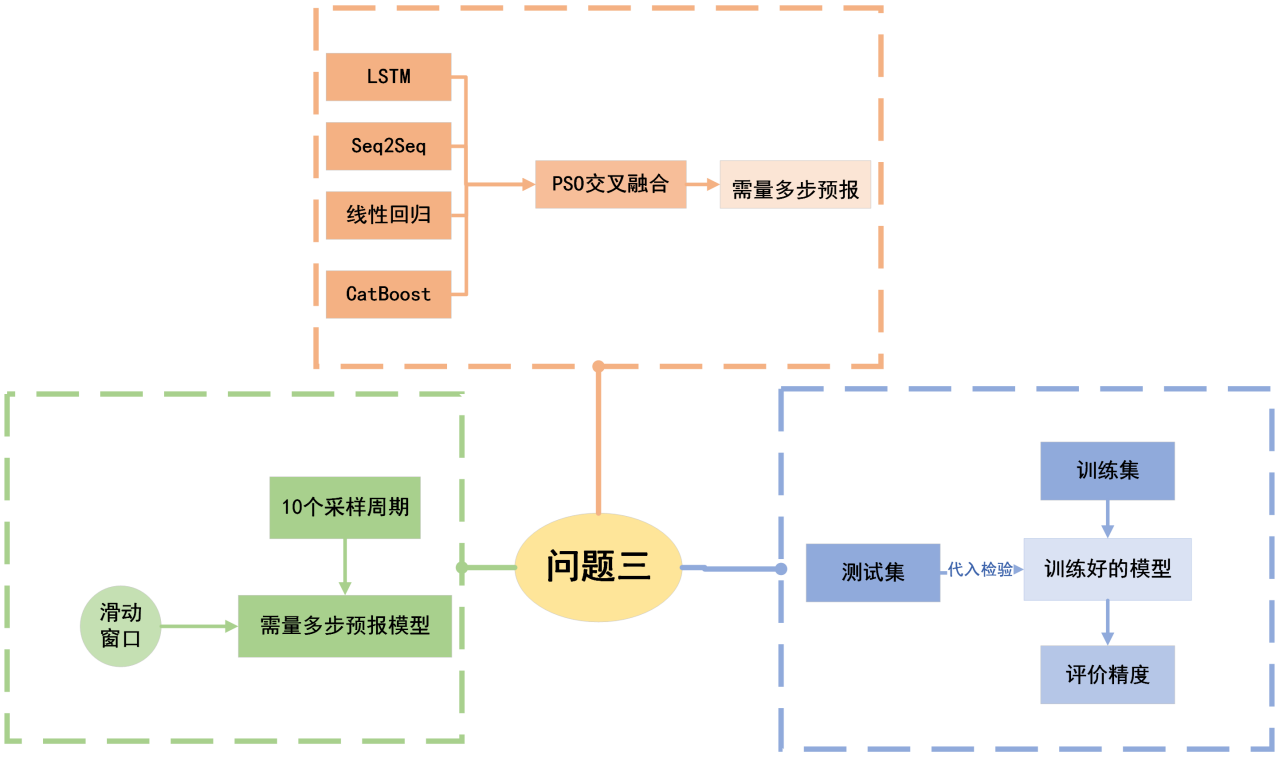

图 13　问题三思路图

6.2 多步需量预报模型

根据熔镁炉需量定义可得 T 时刻的需量为

$$\bar{P}(k+T)=\frac{1}{n}\sum_{j=0}^{n-1}p(k+T-j)=\bar{P}(k)-\frac{1}{n}\sum_{i=1}^{n}p(k-T+i)+\frac{1}{n}\sum_{i=1}^{n}p(k+i) \quad (12)$$

但是其中的群炉功率 $p(k+1)...p(k+n)$ 未知，因此本文建立多步预报模型[3] [8]，如下:

$$\bar{P}=\begin{pmatrix}\bar{P}(k)\\ \bar{P}(k)\\ \vdots\\ \bar{P}(k)\end{pmatrix}-\frac{1}{n}\begin{pmatrix}p(k-n+1)\\ \sum_{i=1}^{2}p(k-n+i)\\ \vdots\\ \sum_{i=1}^{n}p(k-n+i)\end{pmatrix}+\frac{1}{n}\begin{pmatrix}p(k+1)\\ \sum_{i=1}^{2}p(k+i)\\ \vdots\\ \sum_{i=1}^{n}p(k+i)\end{pmatrix} \quad (13)$$

其中 $\bar{P}=\left(\bar{P}(k+1),\bar{P}(k+2),...,\bar{P}(k+n)\right)^{T}$ 表示预报的需量。

6.3 集成学习求解

本文结合 LSTM、Seq2Seq、线性回归、CatBoost 四种模型，通过 PSO 分配权重进行集成，实现对需量的预报。

6.3.1 LSTM

LSTM（长短期记忆网络）是一种特殊的循环神经网络。长短期记忆网络通过遗忘门、输入门以及输出门这三个控制单元对梯度消失等问题进行处理和解决[5] [6]。长短期记忆网络主要由记忆细胞、遗忘门、输入门、输出门四个部分组成。记忆细胞 c_t、遗忘门 f_t、输入门 i_t、输出门 o_t 在 t 时刻输出如下：

$$\begin{aligned} c_t &= f_t c_{t-1} + i_t tanh\left(\omega_{x_t c} h_{t-1} + b_c\right), \\ {}_t &= \delta\left(\omega_{x_t f} x_t + \omega_{h_t f} h_{t-1} + \omega_{c_t f} c_{t-1} + b_f\right), \\ i_t &= \delta\left(\omega_{x_t i} x_t + \omega_{h_t i} h_{t-1} + \omega_{c_t i} c_{t-1} + b_i\right), \\ o_t &= \delta(\omega_{x_t o} x_t + \omega_{h_t o} h_{t-1} + \omega_{c_t o} c_{t-1} + b_o. \end{aligned} \tag{14}$$

其中 δ 表示激活函数，x_t 为当前状态下数据的输入，h_t 表示接收到的上一个节点的输入。上式中其他参数的含义如下表所示：

表 3　LSTM 模型参数表

	遗忘门	输入门	输出门	记忆单元
t 时刻的输出	f_t	i_t	o_t	c_t
与输入之间的权重	$\omega_{x_t f}$	$\omega_{x_t i}$	$\omega_{x_t o}$	$\omega_{x_t c}$
与隐藏层输出之间的权重	$\omega_{h_t f}$	$\omega_{h_t i}$	$\omega_{h_t o}$	$\omega_{h_t c}$
与记忆单元输出之间的权重	$\omega_{c_t f}$	$\omega_{c_t i}$	$\omega_{c_t o}$	-
偏差	b_f	b_i	b_o	b_c

此外，上述各参数和变量满足以下关系：

$$\begin{cases} h_t = o_t \tan h\left(c_t\right), \\ y_t = \delta\left(\omega_{h_t y} + b_y\right), \\ \tan h\left(\partial\right) = \frac{e^{\partial} - e^{-\partial}}{e^{\partial} + e^{-\partial}}, \\ \delta\left(\partial\right) = \frac{1}{1+e^{-\partial}}. \end{cases} \tag{15}$$

其中 y_t 为当前节点状态下的输出。上述 LSTM 神经网络非线性预测模型示意图如下。

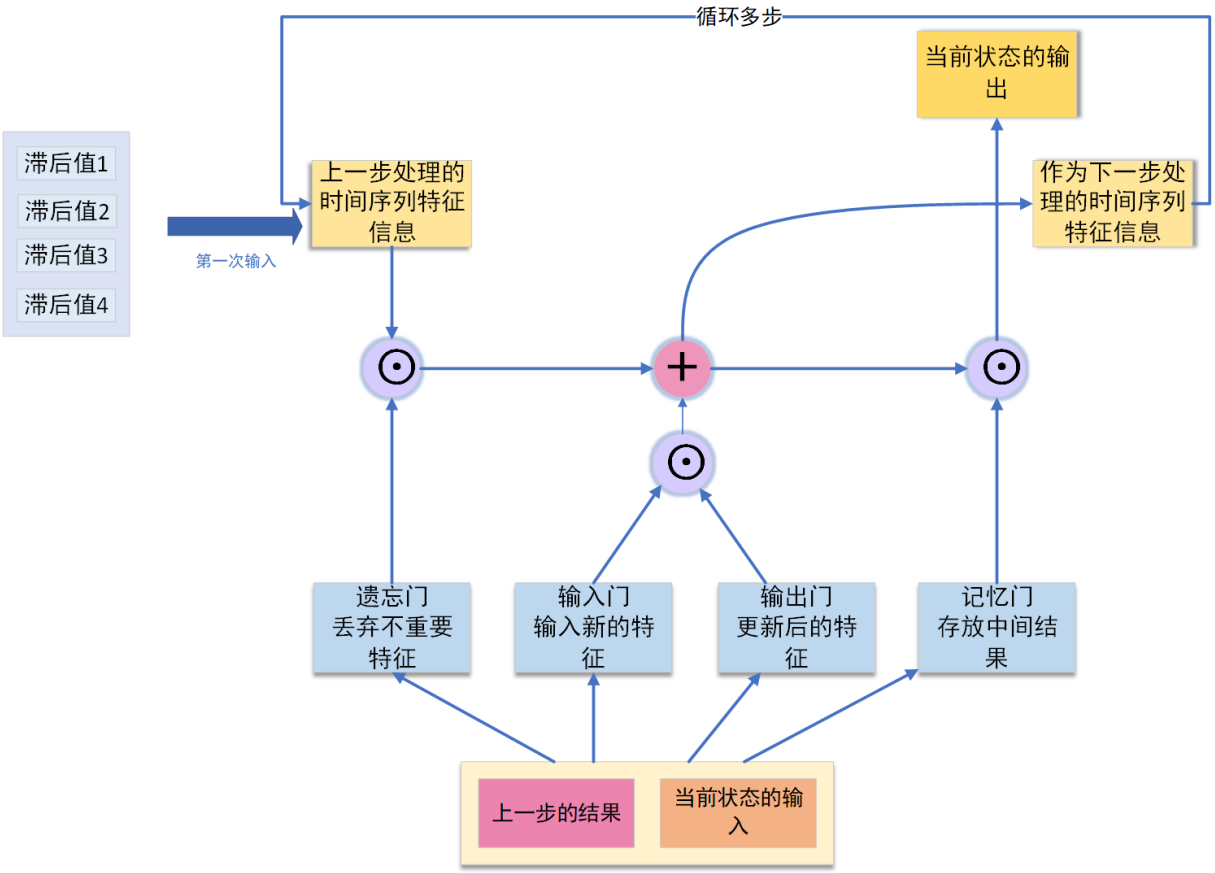

图 14　LSTM 神经网络示意图

6.3.2 Seq2Seq

Seq2Seq（Sequence-to-Sequence，序列到序列）主要由编码器和解码器两部分构成，每部分都可以看作为一个独立的 LSTM 模型[5]，本质上是循环神经网络。在编码器中，输入一个可变长度的时间序列数据，可以将其编码为指定长度的由编码器中的末尾 LSTM 的隐藏层状态和单元动态组成的中间向量；在解码器中，输入该中间向量，输出每个 LSTM 单元产生的预测结果。

6.3.3 线性回归

线性回归是一种常见的统计分析方法，用于建立自变量与因变量之间的线性关系模型。它的目标是通过拟合两者间的关系来预测因变量的变化。本文将一个周期 T 内多个连续时间步的需量作为因变量，将已知的滞后项的功率与需量作为自变量进行拟合，得到高维空间中的超平面进行多步预报。

6.3.4 PSO 交叉融合

与问题二类似，本文采用 PSO 优化权重分配给 LSTM、Seq2Seq、线性回归、CatBoost，交叉融合为集成模型进行多步预测，求解结果如下。

表 4　模型的权重分配

模型	LSTM	Seq2Seq	线性回归	CatBoost
权重	1	0.9983	0.9733	1

6.4 结果分析

首先利用附件训练集对上述模型进行训练，然后选取测试集的部分时间阶段，将其代入训练好的模型进行检验。由于本题是多步预报，将问题二求得的 10 个采样周期作为大小窗口，通过窗口滑动，利用上述模型算法对窗口内进行预报，会出现某一时刻的需量被预报多次的情况，本文将这多个预报值的平均值作为最终该时刻的需量预报值。以 21:43:10—21:54:50 时间段为例，预报值与观测值的比较图如下，明显看出预报曲线与观测曲线基本重合，预报结果较好。

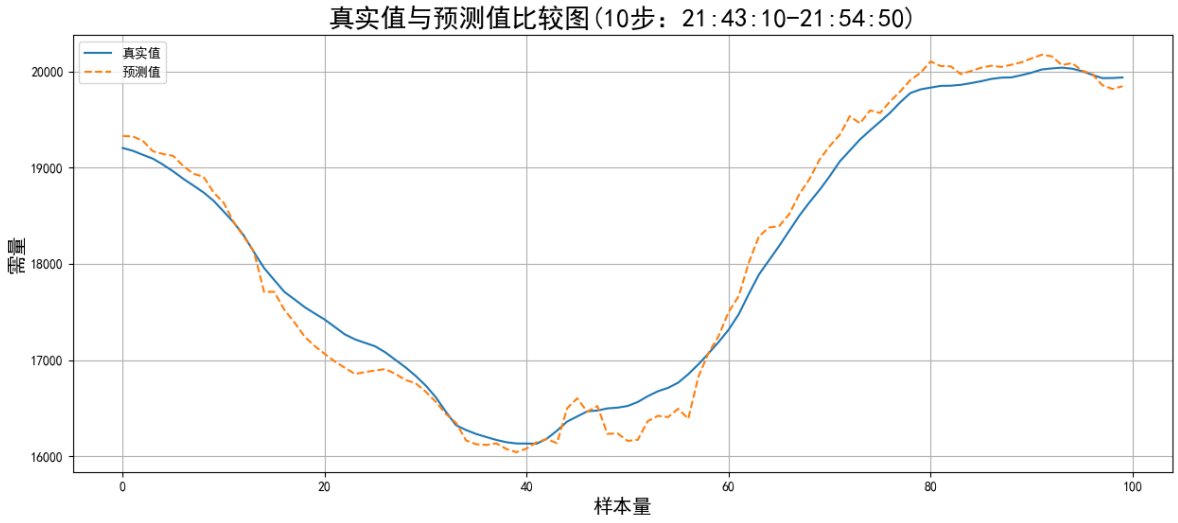

图 15　观测值与预报值比较图 (10 步：400)

6.5 预报精度评价

本文采用 TPR（True Positive Rate，灵敏度）、TNR（True Negative Rate，特异度）、RMSE、RMAPE、R-Sqaured 来对需量多步预报模型的预报精度进行评价。

其中，在计算 TPR 与 TNR 之前，首先需要通过计算真实值和预测值的相邻元素的差异将两者转换为二进制类别。将差异大于等于 0 的设为 1，作为正例，小于 0 的设为 0 作为负例。TPR 与 TNR 的具体计算公式如下：

$$TPR_i(k) = \frac{\sum_{j=0}^{N-1} TP_i(k-j)}{\sum_{j=0}^{N-1} TP_i(k-j) + \sum_{j=0}^{N-1} FP_i(k-j)} \tag{16}$$

其中 TP 表示观测的正例中被模型预测为正例的数量，FP 表示观测的正例中被模型预测为负例的数量。

$$TNR_i(k) = \frac{\sum_{i=0}^{N-1} TN_i(k-j)}{\sum_{j=0}^{N-1} TN_i(k-j) + \sum_{j=0}^{N-1} FN_i(k-j)} \tag{17}$$

其中 TN 表示观测的负例中被模型预测为负例的数量，FN 表示观测的负例中被模型预测为正例的数量。

求得各评价指标结果如下，从表中可以看出预测精度较高，模型较为准确。

表 5　模型评价指标

时间段	RMSE	RMAPE	TPR	TNR	R-Sqaured
21:54:50-22:06:37（500）	111.8774	4.4539	0.9649	0.9286	0.8779
22:53:10-23:04:50(1000)	123.0670	8.5494	0.7813	0.9104	0.8928
23:51:30-0:03:10(1500)	174.4222	6.0802	0.6818	0.8727	0.8966
0:49:50-1:01:30(2000)	197.2071	15.6640	0.8936	0.8936	0.8937
1:48:10- 1:59:50(2500)	100.5553	2.2898	0.8393	0.7907	0.9373
2:46:30- 2:58:10(3000)	93.9098	5.2218	0.7857	0.7857	0.9909

7 问题四模型的建立与求解

7.1 问题四分析

问题四要求根据上一问中需量多步预报结果，给出需量尖峰的判别算法，并对尖峰预报精度给出评价。首先可以对历史时刻的需量尖峰进行特性分析，寻找导致需量尖峰出现的因素，并作为尖峰判别条件。然后对问题三的多步预报模型进行改进，建立尖峰预报模型，对预测的数据采用小波变换来寻找波峰，并根据尖峰判别条件判定尖峰大致位置，并于实际尖峰位置进行对比，评价尖峰预报精度。

问题四的思路图如下。

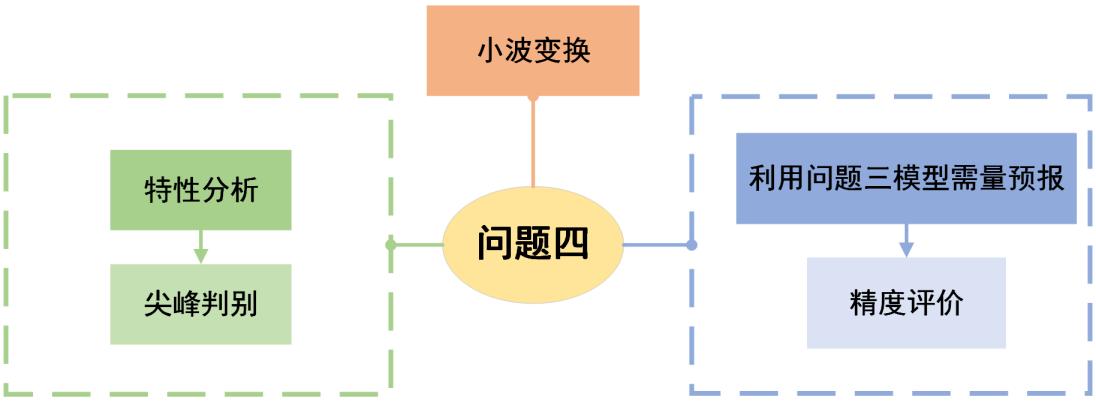

图 16　问题四思路图

7.2 基于小波变换的尖峰预报模型

7.2.1 尖峰判别条件

由分析可知，尖峰的出现是由于原料杂质成分含量增大和颗粒大小变化等因素会造成阻抗减小，导致熔化电流和功率增大，进而使需量上升，随后由熔化电流控制系统控制断电导致需量下降。为了更准确的判别，本文根据训练数据的历史需量变化情况进行分析并得出尖峰判别条件。

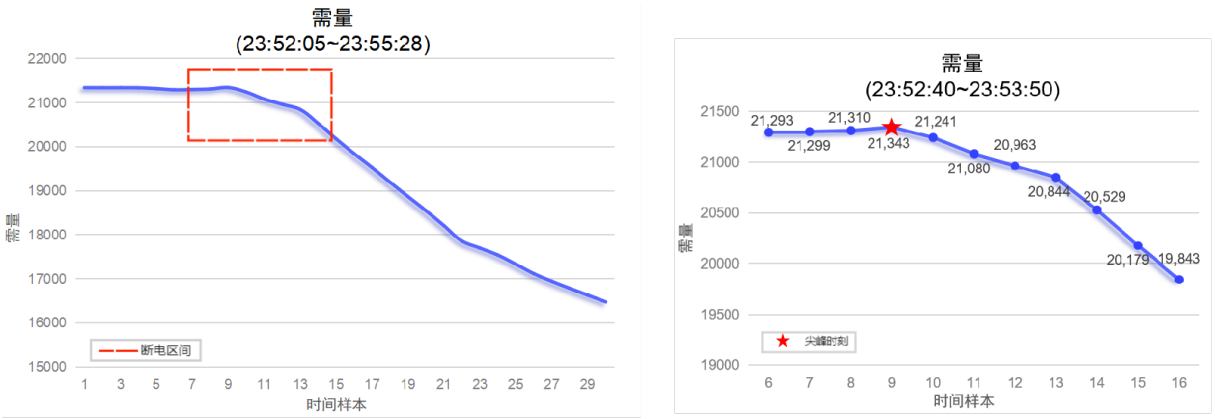

图 17　尖峰分析图

如上图所示，本文选取了 23:52:05 至 23:55:28 时间段训练数据的历史需量变化作为参照，其中可以发现，从红色区域的所在时刻开始需量开始迅速下降，可以判断这里熔化电流控制系统可能采取了断电操作，而在这之前可能存在某个尖峰区域。进一步缩小时间区间，对 23:52:40 至 23:53:50 时间段的具体需量值进行比较，可以发现在 23:52:54 时刻达到了需量的局部最大值，并且前面时间段的需量呈逐渐递增的趋势，这可能是由于电流、功率缓慢增大导致的。本文利用该尖峰的特性分析得到尖峰判别条件。

对于某点 x_i，其与前 h 个点的关系可定义为：

$$x_{i-j} < x_{i-j+1} \quad (j = 1, 2...h) \tag{18}$$

同时，所有这些点的值都必须超过一个给定的需量阈值 M：

$$x_{i-j} > M \quad (j = 0, 1, 2...h) \tag{19}$$

根据分析，本文采用 h=3，M=20500kw，对尖峰判别进行限制。

7.2.2 小波变换

给定一个信号 $f(t)$，其连续小波变换与一个母小波 $\psi(t)$ 的卷积为：

$$W_f(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t)\psi^* \left(\frac{t-b}{a}\right) dt \tag{20}$$

其中:

- $W_f(a,b)$ 是在尺度 a 和时间 b 处的小波系数。
- a 是尺度参数，决定了小波的伸缩。
- b 是平移参数，决定了小波的位置。
- ψ^* 是母小波的复共轭。

为了检测尖峰，我们选择了 Haar 小波作为母小波[7]，然后在信号的不同尺度和位置上计算小波系数，并去寻找局部最大值的小波系数，这些局部最大值超过了一个给定的阈值，是可能存在的尖峰点。

7.2.3 尖峰预报模型

结合尖峰判别算法，本文基于问题三进行了改进，在实现多步预测后对未来的预测需量进行尖峰判别和预报，并与实际情况进行对比，进而评价预报精度。

7.3 模型求解和结果分析

本文选取了 0:49:50 至 1:01:30 时间段测试数据作为数据样本并进行尖峰预报，预报结果如下。

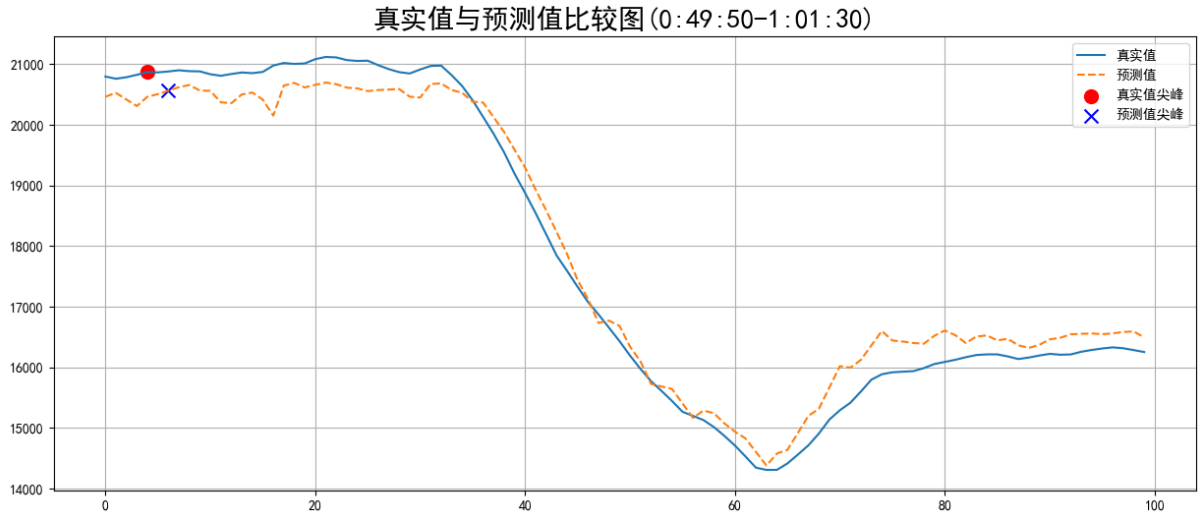

图 18　尖峰预报结果图

由图可以分析，尖峰预报位置与实际尖峰位置相差了 2 个采样周期（14 秒），但是在实际预报中本文采用的是十步预报模型，故大约能提前在 8 个采样周期（约 1 分钟）之前发出尖峰预警。在之后进行多次测试，91.3% 预报模型结果能保证在提前 1 分钟左右的时间进行尖峰预警，这段时间对于熔化电流控制系统而言足以提前为即将到来的尖峰做好准备并采取电流控制措施。

8 模型总结与评价

8.1 模型总结

本文首先采用线性加权融合 Lasso 回归模型和 CatBoost 模型建立单步需量预报模型，通过滚动时间窗口实现 5 折时间序列验证法防止模型过拟合，粒子群算法优化模型性能，并对预报值的准确度进行评价，之后采用自相关性检验和差分法求解需量尖峰数目和位置从而估计周期，然后基于第一问的模型，引入 LSTM、Seq2Seq 建立多步需量预报模型，经多指标检测验证模型预报效果良好，最后根据多步预报模型结合小波变换建立尖峰预报模型，比较尖峰位置与预测尖峰的位置差异，对模型精度做出评价。

8.2 模型优点

1) 问题一采用的 Lasso 回归模型可以较好提取时间序列的连续特征，CatBoost 则可以挖掘时间序列潜在的特征，通过线性加权将两个模型的优势结合起来，同时采用粒子群算法优化权值，进一步提升预报模型的精度。

2) 本文针对时间序列数据集的时间性和数据集不足的特点，采用滚动窗口法和时间序列验证法，可以扩充时间序列的数据集，有利于防止模型过拟合，为多维度评价模

型打下基础。

3) 问题三在第一问单步预报模型的基础上引入 LSTM 和 Seq2Seq 模型，重点针对预测多个值的情况，高质量地对时间序列特征进行处理，可以保留数据流动性和长期依赖的特征，也能很好防止梯度爆炸和梯度消失的问题，从而提高多步预测模型整体的精度。

4) 问题四建立尖峰预报模型，充分利用了小波变换提取时间序列的局部特征的优势，大幅度提升模型预报精度。

8.3 模型缺点

1) 模型计算量大，算法复杂度较高，模型计算耗时较久，后续可以通过模型减枝的操作进行优化，减少预报模型耗时的问题，实现实时预报。

2) 由于数据集只包含一个炉次，导致模型预报的适应度不高，模型训练效果不佳，后期通过补充数据集，可以完善模型，进一步提升模型精度。

8.4 模型改进

该模型参考电熔镁砂生产过程需量预报领域的专业知识，引入机器学习模型和深度学习模型，采用线性加权模型融合的思想，建立了基于改进 CatBoost-LSTM 算法的电熔镁砂群炉需量预报模型，保证一定预报精度的情况下，程序运行时间支持实时预报，可以考虑应用到实际场景中，帮助分析是否需要拉闸限电来控制生产过程的温度，从而为平衡产品质量、能耗和安全问题给出可参考性预报结果和分析。

参考文献

[1] 李潇睿, 班晓娟, 袁兆麟等. 工业场景下基于深度学习的时序预测方法及应用 [J]. 工程科学学报,2022,44(04):757-766.

[2] 张菁雯, 柴天佑, 李慷. 电熔镁砂生产用电需量多步智能预报方法 [J/OL]. 自动化学报:1-10.https://doi.org/10.16383/j.aas.c220659.

[3] 马晓君, 宋嫣琦, 常百舒等. 基于 CatBoost 算法的 P2P 违约预测模型应用研究 [J]. 统计与信息论坛,2020,35(07):9-17.

[4] 贺琪, 查铖, 宋巍等. 基于 STL 的海表面温度预测算法 [J]. 海洋环境科学,2020,39(06):918-925.

[5] 袁梅雪, 魏守科, 孙铭等. 基于小波去噪和 LSTM 的 Seq2Seq 水质预测模型 [J]. 计算机系统应用,2022,31(06):38-47.

[6] 姜山, 周秋鹏, 董弘川等. 考虑数据周期性及趋势性特征的长期电力负荷组合预测方法 [J]. 电测与仪表,2022,59(06):98-104.

[7] 何利健, 张锐, 林晓冬. 基于 DWT 和双通道 LSTM 的卫星电池阵电流预测方法 [J]. 中国科学院大学学报,2023,40(03):415-421.

[8] 杨杰, 柴天佑, 张亚军等. 数据与模型驱动的电熔镁群炉需量预报方法 [J]. 自动化学报,2018,44(08):1460-1474.

[9] Chai T, Zhang J, Yang T. Demand forecasting of the fused magnesia smelting process with system identification and deep learning[J]. IEEE Transactions on Industrial Informatics, 2021, 17(12): 8387-8396.

[10] Liao J, Zheng J, Chen Z. Research on the Fault Diagnosis Method of an Internal Gear Pump Based on a Convolutional Auto-Encoder and PSO-LSSVM[J]. Sensors, 2022, 22(24): 9841.

附录 A　问题结果

附录 B　问题一源代码

2.1　预测程序

```
import pandas as pd
import catboost as cb
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.ensemble import RandomForestRegressor
from scipy.optimize import minimize
from scipy.signal import medfilt

# Load Data
xls = pd.ExcelFile("E:/Desktop/RawData.xlsx")
train_data = xls.parse('train')
test_data = xls.parse('test')

# Compute Demand
n = 30
train_data['demand/需量'] = train_data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)
test_data['demand/需量'] = test_data['power/功率（kw）'].rolling(window=n).mean().shift(-n+1)

# Prepare Features with 5 lags for demand
num_lags = 3
for i in range(1, num_lags + 1):
    train_data[f'lag_demand_{i}'] = train_data['demand/需量'].shift(i)
    test_data[f'lag_demand_{i}'] = test_data['demand/需量'].shift(i)

lag_demand_columns = [f'lag_demand_{i}' for i in range(1, num_lags + 1)]

# Create lagged features for power
for i in range(1, num_lags + 1):
    train_data[f'lag_power_{i}'] = train_data['power/功率(kw)'].shift(i)
    test_data[f'lag_power_{i}'] = test_data['power/功率（kw）'].shift(i)

lag_power_columns = [f'lag_power_{i}' for i in range(1, num_lags + 1)]

# Combine all features
all_features = lag_demand_columns + lag_power_columns

train_data_cleaned = train_data.dropna()
X_train = train_data_cleaned[all_features]
```

```
y_train = train_data_cleaned['demand/需量']

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
test_data_cleaned = test_data.dropna()
X_test = test_data_cleaned[all_features]
X_test_scaled = scaler.transform(X_test)

# Train Models: Linear Regression, Time Series Forest, CatBoost, and Ridge Regression
linear_reg = LinearRegression()
time_series_forest = RandomForestRegressor(n_estimators=100, random_state=42)
catboost_model = cb.CatBoostRegressor(iterations=1000, learning_rate=0.1, depth=6)
ridge_reg = Ridge(alpha=0.5)

# Define the objective function for optimization
def objective_function(weights):
    weighted_predictions = np.dot(np.vstack((y_test_pred_linear, y_test_pred_tsforest,
        y_test_pred_catboost, y_test_pred_ridge)).T, weights)
    return mean_squared_error(test_data_cleaned['demand/需量'], weighted_predictions)

# Set the initial guess for weights
weights_init = np.ones(4) / 4

# Perform optimization to find optimal weights
bounds = [(0, 1)] * 4
result = minimize(objective_function, weights_init, bounds=bounds)
weights_opt = result.x


# Weighted blending of all models using optimal weights
blending_data = np.vstack((y_test_pred_linear, y_test_pred_tsforest, y_test_pred_catboost,
    y_test_pred_ridge)).T
weighted_blend_predictions = np.dot(blending_data, weights_opt)

# Denoise the predictions using Median Filtering
denoised_predictions = medfilt(weighted_blend_predictions, kernel_size=3)

# Calculate RMSE for the denoised predictions
rmse_denoised = mean_squared_error(test_data_cleaned['demand/需量'], denoised_predictions,
    squared=False)
print("RMSE after denoising with Median Filtering:", rmse_denoised)

# Define rolling window size and number of splits for time series cross-validation
rolling_window = 10 # 滚动预测窗口大小
n_splits = 5 # 时间序列交叉验证折数
```

```
def rolling_cross_validation(model, X, y):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    predictions = []

    for train_index, test_index in tscv.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        predictions.extend(y_pred)

    return predictions


linear_reg_predictions = rolling_cross_validation(linear_reg, X_train, y_train)
time_series_forest_predictions = rolling_cross_validation(time_series_forest, X_train, y_train)
catboost_predictions = rolling_cross_validation(catboost_model, X_train, y_train)
ridge_reg_predictions = rolling_cross_validation(ridge_reg, X_train, y_train)

# Save validation results to Excel
validation_results = pd.DataFrame({
    'linear_reg_predictions': linear_reg_predictions,
    'time_series_forest_predictions': time_series_forest_predictions,
    'catboost_predictions': catboost_predictions,
    'ridge_reg_predictions': ridge_reg_predictions
})

validation_results.to_excel("E:/Desktop/ValidationResults.xlsx", index=False)
```

2.2 滞后值计算程序

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# Initialize lists to store lag values and RMSE values
lags = []
rmse_values = []

# Iterate over lag values
```

```
for lag in range(1, 8):
    # Create lagged features for demand
    for i in range(1, lag + 1):
        train_data[f'lag_demand_{i}'] = train_data['demand/需量'].shift(i)
        test_data[f'lag_demand_{i}'] = test_data['demand/需量'].shift(i)

    lag_demand_columns = [f'lag_demand_{i}' for i in range(1, lag + 1)]

    # Combine all features
    all_features = lag_demand_columns + lag_power_columns

    train_data_cleaned = train_data.dropna()
    X_train = train_data_cleaned[all_features]
    y_train = train_data_cleaned['demand/需量']

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    test_data_cleaned = test_data.dropna()
    X_test = test_data_cleaned[all_features]
    X_test_scaled = scaler.transform(X_test)

    # Train Models: Linear Regression, Time Series Forest, CatBoost, and Ridge Regression
    linear_reg = LinearRegression().fit(X_train_scaled, y_train)
    time_series_forest = RandomForestRegressor(n_estimators=100,
        random_state=42).fit(X_train_scaled, y_train)
    catboost_model = cb.CatBoostRegressor(iterations=1000, learning_rate=0.1, depth=6)
    catboost_model.fit(X_train_scaled, y_train)
    ridge_reg = Ridge(alpha=0.5).fit(X_train_scaled, y_train)

    # Predict using all models
    y_test_pred_linear = linear_reg.predict(X_test_scaled)
    y_test_pred_tsforest = time_series_forest.predict(X_test_scaled)
    y_test_pred_catboost = catboost_model.predict(X_test_scaled)
    y_test_pred_ridge = ridge_reg.predict(X_test_scaled)

    # Linear Blending of all models
    blending_data = np.vstack((y_test_pred_linear, y_test_pred_tsforest, y_test_pred_catboost,
        y_test_pred_ridge)).T
    blend_train, blend_val, y_blend_train, y_blend_val = train_test_split(blending_data,
        test_data_cleaned['demand/需量'].iloc[-len(y_test_pred_linear):], test_size=0.2,
        random_state=42)
    blender = LinearRegression().fit(blend_train, y_blend_train)
    final_blend_predictions = blender.predict(blending_data)

    # Denoise the predictions using Median Filtering
    denoised_predictions = medfilt(final_blend_predictions, kernel_size=3)
```

```
    # Calculate RMSE for the denoised predictions
    rmse_denoised = mean_squared_error(test_data_cleaned['demand/需量'], denoised_predictions,
        squared=False)

    # Append lag and RMSE values to the lists
    lags.append(lag)
    rmse_values.append(rmse_denoised)

# Plot the RMSE values
plt.plot(lags, rmse_values, marker='o')
plt.xlabel('滞后值(Lag)',fontsize=18)
plt.ylabel('RMSE',fontsize=18)
plt.title('RMSE随滞后值变化图',fontsize=25)
plt.savefig("E:/Desktop/RMSE随滞后值变化图.png")
plt.show()
```

附录 C　问题二源代码

3.1　STL 分解程序

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import STL
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# 1. Load the data
data = pd.read_excel("E:/Desktop/RawData.xlsx")
data=data[500:3500]
# 2. Data Preprocessing: Remove initial and ending phase (let's say 5% from each side for
    simplicity)
portion_to_remove = 0.05
rows_to_remove = int(portion_to_remove * len(data))
filtered_data = data[rows_to_remove:-rows_to_remove]

# 3. Apply STL decomposition to the filtered power/功率(kw) data with a period of 10
stl = STL(filtered_data['power/功率(kw)'], period=10)
result = stl.fit()

# 4. Plot the STL decomposition results
plt.figure(figsize=(14, 10))

# Plotting the trend component
plt.subplot(3, 1, 1)
plt.plot(result.trend,color='lightcoral')
```

```
plt.title('趋势(Trend)',fontsize=20)

# Plotting the seasonal component
plt.subplot(3, 1, 2)
plt.plot(result.seasonal,color='cornflowerblue')
plt.title('季节性(Seasonal)',fontsize=20)

# Plotting the residual component
plt.subplot(3, 1, 3)
plt.plot(result.resid,color='lightseagreen')
plt.title('残差(Residual)',fontsize=20)

# Adjust the layout and show the plot
plt.tight_layout()
plt.savefig("E:/Desktop/TSL.png")
plt.show()
```

3.2 相关性检验程序

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
# 读取Excel文件
xls = pd.ExcelFile("E:/Desktop/RawData.xlsx")
train_data = xls.parse('train')

# 计算需量列均值
n = 30
train_data['demand/需量'] = train_data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)
data = train_data.loc[38:4142]

# 自相关性和偏相关性检验
plot_acf(data['demand/需量'])
plt.title('自相关',fontsize=20)
plt.savefig("E:/Desktop/自相关.png")
plt.show()

plot_pacf(data['demand/需量'])
plt.title('Partial Autocorrelation')
```

```
plt.show()

# 平稳性检验
adf_test = adfuller(data['demand/需量'].dropna())

print('平稳性检验结果：')
print('ADF统计量：', adf_test[0])
print('p值：', adf_test[1])

# 周期性检验
decomposition = seasonal_decompose(data['demand/需量'], period=365)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(4, 1, 1)
plt.plot(data['demand/需量'], color='b')
plt.title('Demand')
plt.subplot(4, 1, 2)
plt.plot(trend, color='r')
plt.title('Trend')
plt.subplot(4, 1, 3)
plt.plot(seasonal, color='g')
plt.title('Seasonal')
plt.subplot(4, 1, 4)
plt.plot(residual, color='purple')
plt.title('Residual')
plt.tight_layout()
plt.show()
```

3.3 周期求解程序

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from scipy.signal import find_peaks
from scipy.optimize import minimize

# 加载数据
data = pd.read_excel("E:/Desktop/RawData.xlsx")
data['time/时间戳'] = pd.to_datetime(data['time/时间戳'], format='%H:%M:%S', errors='coerce')

# 数据预处理
n = 30
```

```
data['demand/需量'] = data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)
data = data.loc[170:3870]

# 归一化数据
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data[['demand/需量']])

# 寻找峰值
distance = 1
peaks, _ = find_peaks(data_scaled[:, 0], distance=distance)

# 构建优化问题模型
def objective(x):
    t = x[0]
    return -np.mean(np.diff(peaks)/t)

def constraint(x):
    t = x[0]
    return np.sum(np.diff(peaks)/t) - len(peaks)

# 设置初始猜测值
x0 = [10]

# 定义约束条件
cons = {'type': 'eq', 'fun': constraint}

# 求解优化问题
res = minimize(objective, x0, constraints=cons, bounds=[(1, None)])

# 获取最优采样周期
sampling_period = res.x[0]
print(f"Optimal sampling period: {sampling_period}")

# 绘图
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['demand/需量'], label='Original Data', color='blue')
plt.plot(data.index[peaks], data['demand/需量'].iloc[peaks], label='Peaks', color='red',
    marker='o')
plt.xlabel('Time')
plt.ylabel('Demand')
plt.title('Demand Peaks')
plt.legend()
plt.grid(True)
plt.show()
```

附录 D　问题三源代码

4.1　LSTM 程序

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, RepeatVector
from catboost import CatBoostRegressor
from pyswarm import pso
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load Data
xls = pd.ExcelFile("E:/Desktop/RawData.xlsx")
data = xls.parse('train')
test_data = xls.parse('test')

# 计算需量
n = 30
data['demand/需量'] = data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)
test_data['demand/需量'] = test_data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)

# Data Preparation
data['demand_lag1'] = data['demand/需量'].shift(1)
data['demand_lag2'] = data['demand/需量'].shift(2)
data['power_lag1'] = data['power/功率(kw)'].shift(1)
data['power_lag2'] = data['power/功率(kw)'].shift(2)
train_data = data.dropna()

X_train = train_data[['demand_lag1', 'demand_lag2', 'power_lag1', 'power_lag2']]
y_train = train_data['demand/需量']

# Normalize Data
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))

# Build and train LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(X_train_scaled.shape[1], 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
model.fit(X_train_reshaped, y_train_scaled, epochs=100, batch_size=32)

# Initialize dictionaries to save predictions and their counts
predictions_dict = {}
count_dict = {}

start_point = int(input("Enter the starting point in the test data: "))

for i in range(100):
    sample = test_data.iloc[start_point + i: start_point + i + 2]
    actual_demand = sample['demand/需量'].values
    actual_power = sample['power/功率(kw)'].values

    # Constructing the input data for models
    input_data = np.array([[actual_demand[0], actual_demand[1], actual_power[0],
        actual_power[1]]])
    input_data_scaled = scaler_X.transform(input_data)
    input_data_reshaped = input_data_scaled.reshape(input_data_scaled.shape[0],
        input_data_scaled.shape[1], 1)

    # Predict using LSTM
    lstm_pred = model.predict(input_data_reshaped)
    lstm_pred_original = scaler_y.inverse_transform(lstm_pred)

    # Compute other models' predictions (assumed to be done, for brevity)
    # e.g., linear_pred, catboost_pred, etc.
    # For now, let's assume these are 0.
    linear_pred = 0
    catboost_pred = 0

    # Combine the predictions
    combined_pred = (lstm_pred_original[0][0] + linear_pred + catboost_pred)
    print(combined_pred)
    # Update the predictions_dict and count_dict
    idx = start_point + i
    if idx in predictions_dict:
        predictions_dict[idx] += combined_pred
        count_dict[idx] += 1
    else:
        predictions_dict[idx] = combined_pred
        count_dict[idx] = 1

# Compute average predictions after the loop
final_predictions_avg = [predictions_dict[key] / count_dict[key] for key in
    sorted(predictions_dict.keys())]
```

```
# Convert the final predictions back to original scale
final_predictions_original = np.array(final_predictions_avg).reshape(-1, 1)

# # Plotting the results
# plt.figure(figsize=(15,6))
# plt.plot(test_data['demand/需量'].iloc[start_point:start_point+100].values, label='Actual')
# plt.plot(final_predictions_original, label='Predicted', linestyle='--')
# plt.title('Comparison of Actual and Predicted Values')
# plt.legend()
# plt.show()
# Calculate RMSE
rmse =
    np.sqrt(mean_squared_error(test_data['demand/需量'].iloc[start_point:start_point+100].values,
    final_predictions_original))
print("RMSE:", rmse)

# Calculate RMAPE
rmape = np.mean(np.abs(final_predictions_original -
    test_data['demand/需量'].iloc[start_point:start_point+100].values) /
    test_data['demand/需量'].iloc[start_point:start_point+100].values) * 100
print("RMAPE:", rmape)

# Calculate R-squared
ss_total = np.sum((test_data['demand/需量'].iloc[start_point:start_point+100].values -
    np.mean(test_data['demand/需量'].iloc[start_point:start_point+100].values)) ** 2)
ss_residual = np.sum((test_data['demand/需量'].iloc[start_point:start_point+100].values -
    final_predictions_original.flatten()) ** 2)
r_squared = 1 - (ss_residual / ss_total)
print("R-squared:", r_squared)

# Calculate TPR and TNR
actual_values = test_data['demand/需量'].iloc[start_point:start_point+100].values
trend_actual = np.diff(actual_values) > 0
trend_predicted = np.diff(final_predictions_original.flatten()) > 0

tp = np.sum(np.logical_and(trend_actual, trend_predicted))
tn = np.sum(np.logical_not(np.logical_or(trend_actual, trend_predicted)))
total_positive = np.sum(trend_actual)
total_negative = len(trend_actual) - total_positive

tpr = tp / total_positive
tnr = tn / total_negative

print("TPR:", tpr)
print("TNR:", tnr)
```

4.2 集成模型求解程序

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, RepeatVector
from catboost import CatBoostRegressor
from pyswarm import pso
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from scipy.signal import medfilt
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# Load Data
xls = pd.ExcelFile("E:/Desktop/RawData.xlsx")
data = xls.parse('train')
test_data = xls.parse('test')

# 计算需量
n = 30
data['demand/需量'] = data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)
test_data['demand/需量'] = test_data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)

# Data Preparation
data['demand_lag1'] = data['demand/需量'].shift(1)
data['demand_lag2'] = data['demand/需量'].shift(2)
data['power_lag1'] = data['power/功率(kw)'].shift(1)
data['power_lag2'] = data['power/功率(kw)'].shift(2)
train_data = data.dropna()

X_train = train_data[['demand_lag1', 'demand_lag2', 'power_lag1', 'power_lag2']]
y_train = train_data['demand/需量']

# Normalize Data
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], 1, X_train_scaled.shape[1])

# Train LSTM
model = Sequential()
model.add(LSTM(50, input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
model.add(Dense(1))
```

```
model.compile(optimizer='adam', loss='mse')
model.fit(X_train_reshaped, y_train_scaled, epochs=200, verbose=0)

# Train Linear Regression
lin_reg = LinearRegression().fit(X_train_scaled, y_train_scaled)

# Train CatBoost
catboost = CatBoostRegressor(learning_rate=0.01, n_estimators=1000, verbose=0)
catboost.fit(X_train, y_train)

# Prepare data for Seq2Seq model
X_train_seq = X_train_scaled.reshape(X_train_scaled.shape[0], 1, X_train_scaled.shape[1])
# Prepare data for Seq2Seq model
y_train_seq = y_train_scaled.reshape((y_train_scaled.shape[0], 1, y_train_scaled.shape[1]))
# y_train_seq = np.repeat(y_train_scaled, repeats=1, axis=1)

# Build Seq2Seq model
model_seq = Sequential()
model_seq.add(LSTM(50, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])))
model_seq.add(RepeatVector(1))
model_seq.add(LSTM(50, return_sequences=True))
model_seq.add(Dense(1))
model_seq.compile(optimizer='adam', loss='mse')
model_seq.fit(X_train_seq, y_train_seq, epochs=10, verbose=0)

final_predictions = []

predictions_dict = {}
count_dict = {}
start_point = int(input("Enter the starting point in the test data: "))

for i in range(100):
    input_data = test_data.iloc[start_point + i - 2:start_point + i].copy()
    input_features = input_data[['demand/需量', 'power/功率(kw)']].values.flatten()
    input_scaled = scaler_X.transform([input_features])
    input_reshaped = input_scaled.reshape(input_scaled.shape[0], 1, input_scaled.shape[1])

    lstm_pred = model.predict(input_reshaped)
    lin_reg_pred = lin_reg.predict(input_scaled)
    catboost_pred = catboost.predict([input_features])
    lstm_pred_seq = model_seq.predict(input_reshaped)
    lstm_pred_original = scaler_y.inverse_transform(lstm_pred)
    lin_reg_pred_original = scaler_y.inverse_transform(lin_reg_pred)
#     lstm_pred_seq_original = scaler_y.inverse_transform(lstm_pred_seq)

    # PSO to combine predictions
    def objective_function(weights):
```

```
        combined_prediction = weights[0] * lstm_pred_seq[0][0] + weights[1] *
            lstm_pred_original[0][0] + weights[2] * lin_reg_pred_original[0][0] + weights[3] *
            catboost_pred[0]
        combined_prediction = combined_prediction/3
        return -combined_prediction

    lb = [0, 0, 0, 0]
    ub = [1,1, 1, 1]
    weights, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=50)

    combined_pred = weights[0] * lstm_pred_seq[0][0] + weights[1] * lstm_pred_original[0][0] +
        weights[2] * lin_reg_pred_original[0][0] + weights[3] * catboost_pred[0]
    combined_pred = combined_pred/3
    print(combined_pred)
    # Update the predictions_dict and count_dict
    idx = start_point + i
    if idx in predictions_dict:
        predictions_dict[idx] += combined_pred
        count_dict[idx] += 1
    else:
        predictions_dict[idx] = combined_pred
        count_dict[idx] = 1

# Compute average predictions after the loop
final_predictions_avg = [predictions_dict[key] / count_dict[key] for key in
    sorted(predictions_dict.keys())]

# Convert the final predictions back to original scale
final_predictions_original = np.array(final_predictions_avg).reshape(-1, 1)

# 去除异常值
def remove_outliers(data):
    filtered_data = data.copy()
    for i in range(1, len(data)-1):
        if abs(data[i] - data[i-1]) > 200 and abs(data[i] - data[i+1]) > 200:
            filtered_data[i] = (data[i-1] + data[i+1]) / 2
    return filtered_data

# 应用算法去除异常值
filtered_data = remove_outliers(final_predictions_original)

# Plotting the results
plt.figure(figsize=(15,6))
plt.plot(test_data['demand/需量'].iloc[start_point:start_point+100].values, label='真实值')
plt.plot(filtered_data, label='预测值', linestyle='--')
plt.title('真实值与预测值比较图(10步: 300至400)',fontsize=20)
plt.legend()
```

```
plt.show()
```

附录 E　问题四源代码

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, RepeatVector
from catboost import CatBoostRegressor
from pyswarm import pso
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from scipy.signal import medfilt
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# Load Data
xls = pd.ExcelFile("E:/Desktop/RawData.xlsx")
data = xls.parse('train')
test_data = xls.parse('test')

# 计算需量
n = 30
data['demand/需量'] = data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)
test_data['demand/需量'] = test_data['power/功率(kw)'].rolling(window=n).mean().shift(-n+1)

# Data Preparation
data['demand_lag1'] = data['demand/需量'].shift(1)
data['demand_lag2'] = data['demand/需量'].shift(2)
data['power_lag1'] = data['power/功率(kw)'].shift(1)
data['power_lag2'] = data['power/功率(kw)'].shift(2)
train_data = data.dropna()

X_train = train_data[['demand_lag1', 'demand_lag2', 'power_lag1', 'power_lag2']]
y_train = train_data['demand/需量']

# Normalize Data
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()
X_train_scaled = scaler_X.fit_transform(X_train)
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], 1, X_train_scaled.shape[1])
```

```
# Train LSTM
model = Sequential()
model.add(LSTM(50, input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.fit(X_train_reshaped, y_train_scaled, epochs=100, verbose=0)

# Train Linear Regression
lin_reg = LinearRegression().fit(X_train_scaled, y_train_scaled)

# Train CatBoost
catboost = CatBoostRegressor(learning_rate=0.01, n_estimators=100, verbose=0)
catboost.fit(X_train, y_train)

# Prepare data for Seq2Seq model
X_train_seq = X_train_scaled.reshape(X_train_scaled.shape[0], 1, X_train_scaled.shape[1])
# Prepare data for Seq2Seq model
y_train_seq = y_train_scaled.reshape((y_train_scaled.shape[0], 1, y_train_scaled.shape[1]))
# y_train_seq = np.repeat(y_train_scaled, repeats=1, axis=1)

# Build Seq2Seq model
model_seq = Sequential()
model_seq.add(LSTM(50, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])))
model_seq.add(RepeatVector(1))
model_seq.add(LSTM(50, return_sequences=True))
model_seq.add(Dense(1))
model_seq.compile(optimizer='adam', loss='mse')
model_seq.fit(X_train_seq, y_train_seq, epochs=10, verbose=0)

final_predictions = []

predictions_dict = {}
count_dict = {}
start_point = int(input("Enter the starting point in the test data: "))

for i in range(100):
    input_data = test_data.iloc[start_point + i - 2:start_point + i].copy()
    input_features = input_data[['demand/需量', 'power/功率(kw)']].values.flatten()
    input_scaled = scaler_X.transform([input_features])
    input_reshaped = input_scaled.reshape(input_scaled.shape[0], 1, input_scaled.shape[1])

    lstm_pred = model.predict(input_reshaped)
    lin_reg_pred = lin_reg.predict(input_scaled)
    catboost_pred = catboost.predict([input_features])
    lstm_pred_seq = model_seq.predict(input_reshaped)
    lstm_pred_original = scaler_y.inverse_transform(lstm_pred)
    lin_reg_pred_original = scaler_y.inverse_transform(lin_reg_pred)
```

```
#    lstm_pred_seq_original = scaler_y.inverse_transform(lstm_pred_seq)

    # PSO to combine predictions
    def objective_function(weights):
        combined_prediction = weights[0] * lstm_pred_seq[0][0] + weights[1] *
            lstm_pred_original[0][0] + weights[2] * lin_reg_pred_original[0][0] + weights[3] *
            catboost_pred[0]
        combined_prediction = combined_prediction/3
        return -combined_prediction

    lb = [0, 0, 0, 0]
    ub = [1,1, 1, 1]
    weights, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=50)

    combined_pred = weights[0] * lstm_pred_seq[0][0] + weights[1] * lstm_pred_original[0][0] +
        weights[2] * lin_reg_pred_original[0][0] + weights[3] * catboost_pred[0]
    combined_pred = combined_pred/3
    print(combined_pred)
    # Update the predictions_dict and count_dict
    idx = start_point + i
    if idx in predictions_dict:
        predictions_dict[idx] += combined_pred
        count_dict[idx] += 1
    else:
        predictions_dict[idx] = combined_pred
        count_dict[idx] = 1

# Compute average predictions after the loop
final_predictions_avg = [predictions_dict[key] / count_dict[key] for key in
    sorted(predictions_dict.keys())]

# Convert the final predictions back to original scale
final_predictions_original = np.array(final_predictions_avg).reshape(-1, 1)

# 去除异常值
def remove_outliers(data):
    filtered_data = data.copy()
    for i in range(1, len(data)-1):
        if abs(data[i] - data[i-1]) > 200 and abs(data[i] - data[i+1]) > 200:
            filtered_data[i] = (data[i-1] + data[i+1]) / 2
    return filtered_data

# 应用算法去除异常值
filtered_data = remove_outliers(final_predictions_original)

# Compute RMSE, MAPE, RMAPE
y_true = test_data['demand/需量'].iloc[start_point:start_point+100].values
```

```
rmse = np.sqrt(mean_squared_error(y_true, filtered_data))
mape = np.mean(np.abs((y_true - filtered_data) / y_true)) * 100
rmape = np.mean(np.abs((y_true - filtered_data) / (y_true + filtered_data))) * 200

# Compute TNR and TPR
threshold = np.median(filtered_data) # 使用预测值的中位数作为阈值

# 将真实值和预测值转换为二进制类别
y_true_binary = np.where(y_true <= threshold, 0, 1)
filtered_data_binary = np.where(filtered_data <= threshold, 0, 1)

# 计算混淆矩阵
tn = np.sum((y_true_binary == 0) & (filtered_data_binary == 0)) # 真负例
fp = np.sum((y_true_binary == 0) & (filtered_data_binary == 1)) # 假正例
fn = np.sum((y_true_binary == 1) & (filtered_data_binary == 0)) # 假负例
tp = np.sum((y_true_binary == 1) & (filtered_data_binary == 1)) # 真正例

# 计算 TNR 和 TPR
tnr = tn / (tn + fp)
tpr = tp / (tp + fn)

# 输出结果
print("RMSE:", rmse)
print("MAPE:", mape)
print("RMAPE:", rmape)
print("TNR:", tnr)
print("TPR:", tpr)

# Plotting the results
plt.figure(figsize=(15,6))
plt.plot(test_data['demand/需量'].iloc[start_point:start_point+100].values, label='真实值')
plt.plot(filtered_data, label='预测值', linestyle='--')
plt.title('真实值与预测值比较图(10步: 300至400)',fontsize=20)
plt.legend()
plt.show()
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, RepeatVector
from catboost import CatBoostRegressor
from pyswarm import pso
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from scipy.signal import medfilt
import pywt
```

```
# plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
# plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号

# # Load Data
# xls = pd.ExcelFile("E:/Desktop/RawData.xlsx")
# data = xls.parse('train')
# test_data = xls.parse('test')

# # Calculate demand
# n = 30
# data['demand/需量'] = data['power/功率(kw)'].rolling(window=n).mean().shift(-n + 1)
# test_data['demand/需量'] = test_data['power/功率(kw)'].rolling(window=n).mean().shift(-n + 1)

# # Data Preparation
# data['demand_lag1'] = data['demand/需量'].shift(1)
# data['demand_lag2'] = data['demand/需量'].shift(2)
# data['power_lag1'] = data['power/功率(kw)'].shift(1)
# data['power_lag2'] = data['power/功率(kw)'].shift(2)
# train_data = data.dropna()

# X_train = train_data[['demand_lag1', 'demand_lag2', 'power_lag1', 'power_lag2']]
# y_train = train_data['demand/需量']

# # Normalize Data
# scaler_X = MinMaxScaler()
# scaler_y = MinMaxScaler()
# X_train_scaled = scaler_X.fit_transform(X_train)
# y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
# X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], 1,
    X_train_scaled.shape[1])

# # Train LSTM
# model = Sequential()
# model.add(LSTM(50, input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])))
# model.add(Dense(1))
# model.compile(optimizer='adam', loss='mse')
# model.fit(X_train_reshaped, y_train_scaled, epochs=10, verbose=0)

# # Train Linear Regression
# lin_reg = LinearRegression().fit(X_train_scaled, y_train_scaled)

# # Train CatBoost
# catboost = CatBoostRegressor(learning_rate=0.01, n_estimators=100, verbose=0)
# catboost.fit(X_train, y_train)

# # Prepare data for Seq2Seq model
```

```
# X_train_seq = X_train_scaled.reshape(X_train_scaled.shape[0], 1, X_train_scaled.shape[1])
# y_train_seq = y_train_scaled.reshape((y_train_scaled.shape[0], 1, y_train_scaled.shape[1]))

# # Build Seq2Seq model
# model_seq = Sequential()
# model_seq.add(LSTM(50, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])))
# model_seq.add(RepeatVector(1))
# model_seq.add(LSTM(50, return_sequences=True))
# model_seq.add(Dense(1))
# model_seq.compile(optimizer='adam', loss='mse')
# model_seq.fit(X_train_seq, y_train_seq, epochs=10, verbose=0)

# final_predictions = []
# predictions_dict = {}
# count_dict = {}
# start_point = int(input("Enter the starting point in the test data: "))

# # ...[Rest of the prediction loop from the previous code]

# Peak Detection Functions
def is_increasing(sequence):
    for i in range(1, len(sequence)):
        if sequence[i] <= sequence[i - 1]:
            return False
    return True

# def detect_peaks_using_wavelet(predictions, threshold=20500, window=3):
#     coeffs = pywt.wavedec(predictions, 'db1', level=4)
#     coeffs[-1] = np.zeros_like(coeffs[-1])
#     coeffs[-2] = np.zeros_like(coeffs[-2])
#     denoised = pywt.waverec(coeffs, 'db1')
#     peak_indices = np.where(denoised >= threshold)[0]
#     peak_indices = [idx for idx in peak_indices if idx > window and
    is_increasing(predictions[idx-window:idx]) and all(predictions[idx-window:idx] <
    predictions[idx])]
#     return peak_indices
def detect_first_peak_using_wavelet(predictions, threshold=20500, window=3):
    coeffs = pywt.wavedec(predictions, 'db1', level=4)
    coeffs[-1] = np.zeros_like(coeffs[-1])
    coeffs[-2] = np.zeros_like(coeffs[-2])
    denoised = pywt.waverec(coeffs, 'db1')
    peak_indices = np.where(denoised >= threshold)[0]
    valid_peaks = [idx for idx in peak_indices if idx > window and
        is_increasing(predictions[idx-window:idx]) and all(predictions[idx-window:idx] <
        predictions[idx])]
    if valid_peaks:
        return valid_peaks[0] # Return the first valid peak
```

```
    return None




# Get the first peak for the real and predicted values
real_peak =
    detect_first_peak_using_wavelet(test_data['demand/需量'].iloc[start_point:start_point+100].values)
predicted_peak = detect_first_peak_using_wavelet(filtered_data)

# Calculate the time difference
if real_peak is not None and predicted_peak is not None:
    time_difference = abs(real_peak - predicted_peak)
    print(f"The time difference between the real and predicted peak is: {time_difference}
        sampling periods.")
else:
    print("A peak was not detected in either the real or predicted data.")



# # Detect peaks in predicted data
# predicted_peak_indices = detect_peaks_using_wavelet(final_predictions_original.flatten())
# # Detect peaks in real data
# actual_peak_indices =
    detect_peaks_using_wavelet(test_data['demand/需量'].iloc[start_point:start_point+100].values)

# # Calculate accuracy
# correctly_predicted = len(set(predicted_peak_indices) & set(actual_peak_indices))
# accuracy = correctly_predicted / len(predicted_peak_indices) if predicted_peak_indices else 0
# print(f"Accuracy of peak detection: {accuracy * 100:.2f}%")

# # Plotting the results
# plt.figure(figsize=(15,6))
# plt.plot(test_data['demand/需量'].iloc[start_point:start_point+100].values, label='真实值')
# plt.plot(filtered_data, label='预测值', linestyle='--')
# for peak in predicted_peak_indices:
#    plt.axvline(x=peak, color='r', linestyle='--', label='预测尖峰')
# for peak in actual_peak_indices:
#    plt.axvline(x=peak, color='g', linestyle='-', label='实际尖峰')
# plt.title('真实值与预测值比较图', fontsize=20)
# plt.legend()
# plt.show()



# Plotting the results
plt.figure(figsize=(15,6))
plt.plot(test_data['demand/需量'].iloc[start_point:start_point+100].values, label='真实值')
plt.plot(filtered_data, label='预测值', linestyle='--')
```

```
# Mark the first peak points for both real and predicted values
if real_peak is not None:
    plt.scatter(real_peak, test_data['demand/需量'].iloc[start_point + real_peak], color='red',
        s=100, label='真实值尖峰')
if predicted_peak is not None:
    plt.scatter(predicted_peak, filtered_data[predicted_peak], color='blue', s=100,
        label='预测值尖峰', marker='x')

plt.title('真实值与预测值比较图',fontsize=20)
plt.legend()
plt.savefig("E:/Desktop/真实值与预测值比较图(10步：2000至2100).png")
plt.show()
```