空调制造过程中制冷剂灌注量的预测与优化建模

摘　要

现代空调系统依赖精确冷媒灌注量以提升效率。传统方法受限，现代技术采用质量流量计与智能预测模型，确保高精度灌注。高效的制冷剂灌注量的预测与优化模型有利于推动空调制造智能化、绿色化，对行业发展和环境保护都具有深远的影响。

首先，进行**探索性数据分析**（**EDA**），通过初步观察影响脉冲量的 12 个生产参数中 f3 为固定值、f9 为订单批次，故不考虑其对脉冲量的影响，其余 10 个变量均为数值型变量，进行数据预处理，绘制频率分布直方图以及正态和均匀分布检验。结果显示，其中 f5、f6 和 f10 服从**正态分布**，f7、f8、f11 和 f12 服从**均匀分布**，故推测影响脉冲量的变量为 f1（保压三秒之后的检测压力）、f2（真空泵预抽真空状态后的抽空压力）和 f4（由冷媒管道返回的灌注过程最小压力）。

针对问题一，采用基于**随机搜索**的**超参数调优**来构建 **XGBoost** 模型，以分析 10 个生产参数对脉冲量的影响。与随机森林和决策树模型相比，**XGBoost** 模型的 $\mathbf{R^2}$ 值表现更为优越。随后，引入 **SHAP** 算法对变量影响程度进行解释，结果表明，自变量 **f4**、**f2** 和 **f1** 的重要性排名位居前三，这与**探索性数据分析**（**EDA**）的结果一致。

针对问题二，基于问题一的前提下，选择自变量 **f4**、**f2** 和 **f1** 进行建模。使用基于**轮廓系数**的 **K-Means** 聚类算法对变量数据进行划分，最佳聚类数为 **3**。对分类后的数据使用 **CatBoost** 模型进行建模，数据划分后的 $\mathbf{R^2}$ 分别为 **0.82** 和 **0.8**，相较于 **XGBoost** 模型的 $\mathbf{R^2}$ 平均提升 **5%**。其中，第三类数据占比仅为 1.37%，因此不予建模分析。

针对问题三，在标准灌注量的附近，脉冲量的取值范围为(11570, 11600)。根据脉冲量对数据进行划分，并通过绘制**三维立体图**标注不同颜色进行分析。结合问题一和问题二的建模结果，本文采用的方案是通过多层 **DBSCAN** 算法，寻找在脉冲量范围内聚集最多点的区域，从而确定生产参数的取值范围。

综上所述，本文通过构建改进的梯度提升树（**GBTD**）模型，对空调制造过程中制冷剂灌注量进行预测与优化建模。然而，为了获得更优的方案，建议结合**模型**、**经验**和**大语言模型**（**LLM**）进行综合分析和优化。

关键词：EDA　XGBoost　SHAP　K-Means　CatBoost　DBSCAN　LLM

目录

一、 问题重述

随着空调系统的普及与技术进步，冷媒灌注量的精确控制成为提升系统效率与可靠性的关键。传统依靠人工经验的方法存在诸多局限，如精度低、易受环境影响等。

现代空调制造中，通过引入质量流量计及数据驱动的智能预测方法，实现了冷媒灌注量的高精度测量与预测。某空调厂家在冷媒灌注过程中，结合真空预处理与数据回溯技术，进一步确保灌注质量。同时，建立灌注量预测模型，以优化生产参数，提高生产效率与成本效益，引领空调制造向智能化、绿色化方向发展。

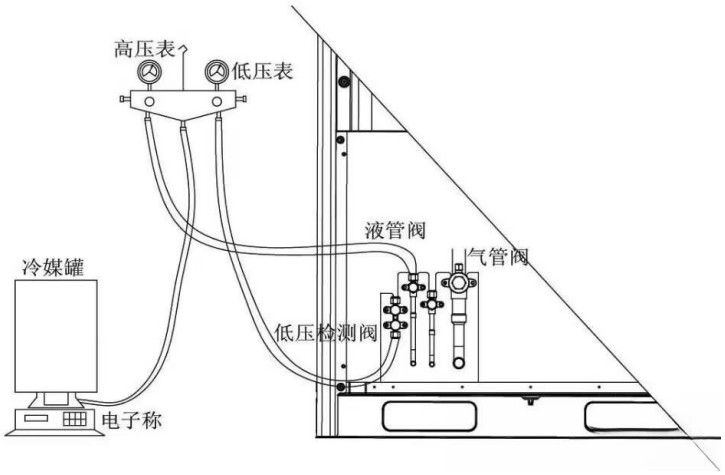

图 1 冷媒灌注示意图

问题一：基于提供的 7000 个样机的数据，对 12 个生产参数进行相关性分析，找出与脉冲量最相关的变量。再根据这些变量对脉冲量的影响程度进行特征重要性排序，并特别指出前三个影响最显著的自变量。详细解释变量筛选的过程及其合理性，即为何选择这些变量作为关键特征。

问题二：结合第一个问题的结果，从 12 个生产参数中选取不超过 4 个最相关的参数，利用这些参数构建一个定量预测模型，用于预测脉冲量。详细描述建模过程，包括数据预处理、模型选择、训练过程等，并通过对比不同的评估指标来说明该模型的预测优势。

问题三：寻找并分析影响脉冲量的生产参数，当这些参数处于哪些具体取值或取值范围时，能够使得灌注量控制在标准灌注量附近（即脉冲量取值在 11570 到 11600 之间）。

二、 问题分析

2.1 问题一的分析

对于问题一，在特征选择和重要性排序的任务中，首先对 7000 个样机的数据进行预处理，包括处理缺失值和异常值，并进行数据标准化或归一化，以确保数据的一致性和质量。

在特征重要性排序方面，利用 XGBoost 模型进行训练，XGBoost 是一种强大的梯度提升决策树算法，能够捕捉复杂的非线性关系。通过计算模型的 SHAP 值，可以评估每个特征对脉冲量预测的具体影响。SHAP 值提供了每个特征在模型中的重要性和贡献度，准确地识别出对脉冲量影响最大的自变量。

最终，基于 SHAP 值的分析结果，确定了对脉冲量预测最有显著影响的前三个自变量。这一过程结合了相关性分析和模型解释，确保了特征选择的科学性和有效性。

2.2 问题二的分析

对于问题二，在从 12 个生产参数中选取不超过 4 个最相关的参数并构建定量预测模型的过程中，利用 XGBoost 模型的 SHAP 值来确认这些特征的重要性，确保所选参数对模型的预测能力有显著贡献。

在选择回归模型时，选择复杂的模型如 XGBoost 回归、CatBoost 回归或 LightGBM 回归等梯度提升树模型，具体取决于数据的复杂性。使用选定的前 4 个特征作为输入进行模型训练，将数据分为训练集和验证集以确保模型的泛化能力。训练过程中，需要调整超参数以优化模型性能，并在验证集上进行测试，记录预测结果与实际值之间的差异。

最后，通过计算均方误差（MSE）、均方根误差（RMSE）、平均绝对误差（MAE）等评估指标来评价模型的预测能力。比较不同模型的评估指标，选择表现最优的模型。这一过程通过选取相关性高的参数并构建精确的预测模型，能够有效提升脉冲量预测的准确性和可靠性。

2.3 问题三的分析

为了寻找并分析影响脉冲量的生产参数，并确定在何种具体取值或范围内能够将脉冲量控制在 11570 到 11600 之间，，通过边界条件分析，利用模型预测功能，探讨不同生产参数取值范围对脉冲量的影响。可以应用优化算法（如网格搜索或随机搜索）系统性地探索参数空间，寻找能够使脉冲量保持在 11570 到 11600 范围内的最佳参数设置。

最后，在实际生产中根据分析结果调整参数设置，并验证这些调整是否能够有效控制脉冲量在目标范围内。同时，根据实验反馈对模型进行调整或重新训练，以进一步提高预测准确性。通过这一系列步骤，可以有效识别并优化使脉冲量保持在目标范围内的生产参数。

三、 模型假设

假设 1：假设生产参数订单批次对脉冲量没有影响；
假设 2：真空泵预抽真空状态后的抽空压力不为负值；
假设 3：本文数据预处理、变量选择合适；
假设 4：在建立预测模型时，模型的预测结果准确；
假设 4：在问题三中使用的方案合理。

四、 符号说明

符号	说明
X_i	第 i 个生产参数
x_i	某生产参数的变量值
μ_i	第 i 个生产参数的均值
σ_i	第 i 个生产参数的标准差
x'	某生产参数的标准化变量值
$x_{\max}$	某生产参数的最大变量值
$x_{\min}$	某生产参数的最小变量值

五、 探索性数据分析

5.1 数据预处理

针对 7000 个样机的 12 个生产参数进行变量相关性分析和变量选择，其中生产参数 f3（开始灌注时系统设置压力）为固定值 3，且假设生产参数 f9（订单批次）对于脉冲量没有影响，故排除生产参数 f3 和 f9 对脉冲量的影响。对其余变量进行异常值检验，绘制其中生产参数 f1（保压三秒之后的检测压力）和生产参数 f2（真空泵预抽真空状态后的抽空压力）的散点图如下所示：

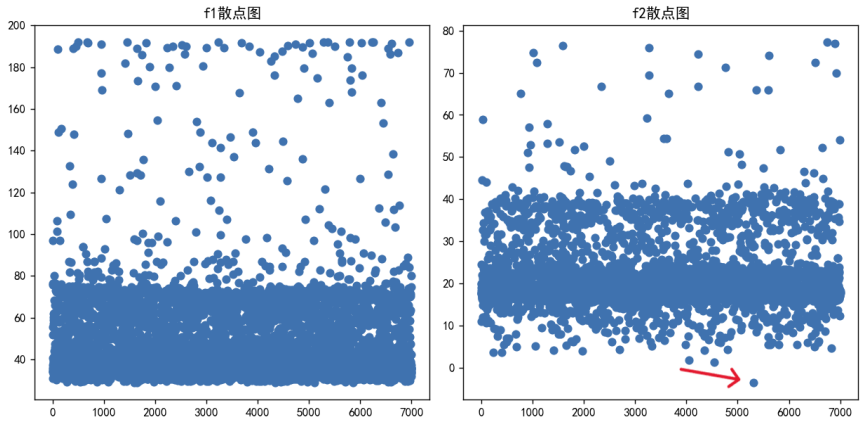

图 2 f1 f2 散点图

根据生产参数 f1 和 f2 的散点图，可以观察到在真空泵预抽真空状态下的抽空压力 f2 中存在负值。考虑到生产参数 f2 应该仅接近于 0 而不应小于 0，因此将这些负值视为异常值并进行剔除。剔除后，其余数据的分布符合实际生产作业中的情况。

5.2 初步分析

首先，通过绘制其余 10 个生产参数的频率分布直方图如下图所示：

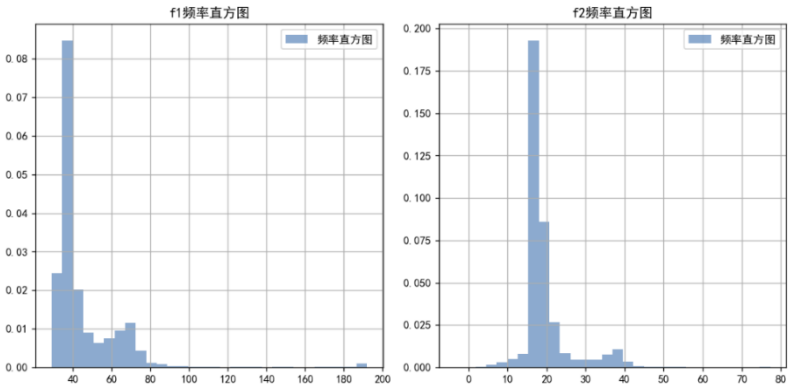

图 3 f1 f2 频率分布直方图

由生产参数 f1 和生产参数 f2 的频率分布直方图显示，数据明显不符合正态分布与均匀分布，可能与目标特征脉冲量有一定的影响。

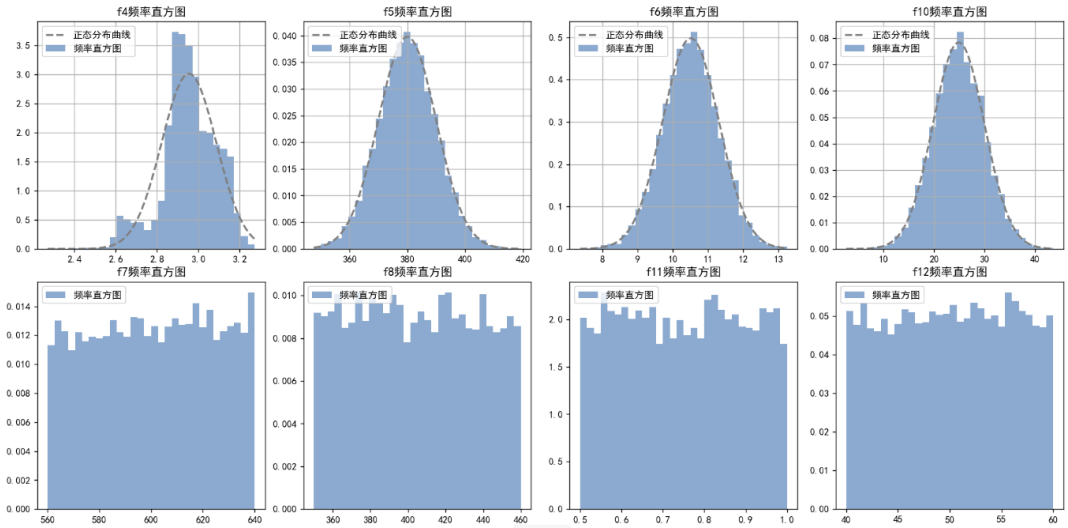

图 4 其余变量的频率分布直方图

根据其余变量的频率分布直方图，对生产参数 f4（由冷媒管道返回的灌注过程最小压力）、f5（电压）、f6（电流）、f7（制冷功率）、f8（制热功率）、f10（厂房环境温度）和 f11（室内风速）进行 Anderson-Darling 正态分布检验与 K-S 均匀分布检验。

Anderson-Darling 正态分布检验基于观测数据与理论分布之间的累积分布函数

（CDF）的差异进行度量。它计算出一个统计量，该统计量反映了观测数据在理论分布下的累积分布函数的拟合程度。

K-S 均匀分布检验是一种用于检验样本数据是否服从某一理论分布，或者两个样本是否来自同一总体分布的方法。其原理是通过比较样本数据的累积分布函数（CDF）与理论分布的 CDF（或两个样本数据的 CDF）之间的差异来进行判断，检验统计量如下式所示：

$$\begin{cases} A^2 = -n - \dfrac{1}{n}\sum_{i=1}^{n}\left[(2i-1)\left(\ln\left(F\left(x_i\right)\right)+\ln\left(1-F\left(x_{n-i+1}\right)\right)\right)\right] \\ D = \max\left(\left(\left|S\left(x_i\right)-f\left(x_i\right)\right|\right),\left(\left|S\left(x_{i-1}\right)-f\left(x_i\right)\right|\right)\right) \end{cases} \tag{1}$$

对 10 个生产参数进行 Anderson-Darling 正态分布检验与 K-S 均匀分布检验，结果如下：

$$\begin{cases} f\left(x\middle|\mu,\sigma^2\right) = \dfrac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}, X_i = 5,6,10 \\ f\left(x\middle|a,b\right) = \dfrac{1}{b-a}, X_i = 7,8,11,12 \end{cases} \tag{2}$$

经检验，生产参数 f5（电压）、生产参数 f6（电流）、生产参数 f10（厂房环境温度）服从正态分布，生产参数 f7（生产线检测时长平均制冷功率）、生产参数 f8（生产线检测时长平均制热功率）、生产参数 f11（室内风速）、生产参数 f12（室内湿度）服从均匀分布。

根据数据分布结构的分析，在 10 个生产参数中，3 个生产参数符合正态分布，4 个生产参数符合均匀分布。因此，影响目标变量的可能性较大的是剩余的 3 个生产参数，即 f1、f2 和 f4。为了深入理解这些变量对目标变量的影响，将采用 XGBoost 对数据进行建模并应用 SHAP 算法，以识别出潜在的关键因素。

六、 模型的建立与求解

6.1 问题一模型的建立与求解

6.1.1 算法介绍

优化的 XGBoost 算法

XGBoost 算法是一种基于 GBDT 的改进算法，GBDT（梯度提升树）是一种以决策树为弱学习器的集成学习方法,具有易训练、可解释性高等优点，在垃圾邮件检测、广告投放、销售预测、医疗数据分析等领域有广泛的应用。XGBoost 是首个支持在图形处理单元(Graphics Processing Units，GPU)环境下训练 GBDT 模型的开源系统,相

对于 GBDT，XGBoost 精度更高，灵活性更强、模型复杂度更容易控制、学习速率更快，广泛应用于各类分类和回归问题。其核心思想是通过不断地分裂生成决策树来拟合上一阶段的预测残差，最终预测值为每个叶节点计算值的加权。

给定具有n个样本的数据集，每个独立变量作为输入X_i，每个变量含有 15 个特征，每个样本的对应变量为y_i,一个具有K个树的模型预测值$\hat{y}_i$如下：

$$\hat{y}_i = \sum_{K=1}^{K} f_K(X_i), f_K \in F \tag{3}$$

其中K表示树的数量，F是回归树的集合空间，$f(x)$是一个回归树,$f_k(X_i)$表示样本X_i,在第k棵树中所在叶子的权重：

$$F = \{ f(X) = \mu_l(X) \} \tag{4}$$

其中$l(X)$表示第X个样本的叶节点，μ表示叶节点的得分。第t次迭代后的预测结果如下所示：

$$\hat{y}_i^i = \hat{y}_i^{t-1} + f_t(X_i) \tag{5}$$

因此，XGBoost 的目标函数定义如下公式所示：

$$J(f_t) = \sum_{n=1}^{n} \left(y_i, \hat{y}_i^{t-1} + f_t(X_i) + \Omega(f_t) \right) \tag{6}$$

其中L是损失函数，$\Omega(f_t)$表示模型复杂度。

$$\Omega(f_t) = \gamma \cdot T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j 2 \tag{7}$$

其中T表示叶子节点总数，$w_j 2$表示叶子节点得分L_2正则化项，γ表示惩罚系数。由此，上述公式可以通过二阶泰勒展开式化简，得到最终的目标函数：

$$J(f_t) = \sum_{i=1}^{n} \left[g_i w_l(X_i) + \frac{1}{2} h_i w_l^2(X_i) \right] + \gamma \cdot T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \tag{8}$$

$$g_i = \frac{\partial L\left(y_i, \hat{y}^{t-1}\right)}{\partial \hat{y}^{t-1}} \tag{9}$$

$$h_i = \frac{\partial 2L\left(y_i, \hat{y}^{t-1}\right)}{\partial \hat{y}^{t-1}} \tag{10}$$

由于 XGBoost 模型的预测精度受到众多超参数的显著影响，因此需要一种高效的优化方法来寻找超参数的最佳解。以下是基于随机搜索的超参数调优流程，用于优化 XGBoost 模型的超参数：

定义超参数空间：首先确定需要优化的超参数及其可能的取值范围。例如，XGBoost 的超参数空间可能包括学习率（learning_rate）、树的数量（n_estimators）、最大深度（max_depth）等。

设置随机采样次数：设定在超参数空间中进行随机采样的次数。这一步决定了搜索的广度与深度，一般情况下采样次数越多，找到最佳超参数的可能性也越大。

随机抽样超参数组合：从定义好的超参数空间中随机选择若干组超参数组合。这些组合将用于训练和评估 XGBoost 模型。

训练和评估模型：对于每一组随机抽样的超参数组合，训练对应的 XGBoost 模型，并在验证集上评估其性能。常用的评估指标包括准确率、F1-score、均方误差等，根据任务的不同选择适合的指标。

选择最佳模型：根据模型在验证集上的表现选择最佳的超参数组合。最终，使用这些最佳的超参数组合训练最终模型，并进行进一步的测试和应用。

使用 XGBoost 算法可以有效地学习脉冲量与空调样机生产过程中相关参数数据之间的关系。本文将采用优化后的 XGBoost 算法来构建这一模型，从而提升模型的预测精度和可靠性。

SHAP 算法

SHAP 是一个解释个体预测的算法，该算法基于博弈论和局部解释，用于对模型的输出提供可解释性，其核心思想是博弈中个体的边缘收益，通过计算在合作中个体的贡献来确定该个体的重要程度，从而评估每一个特征的贡献。一些传统的可解释性方法虽然可以直观地反映出特征对模型的影响程度，但不具备 SHAP 这种反应特征与最终预测结果关系的能力，因此 SHAP 算法广泛应用于机器学习的可解释性分析。

对于 XGBoost 模型，输入 N (包含 n 个特征)用于预测输出值 $v(N)$，在 SHAP 算法中，每个特征对 $v(N)$ 的贡献(φ_i 记为特征 i 的贡献)是基于其边际贡献分配的，其通过下式计算:

$$\varphi_i = \sum_{S \in N\{i\}} \frac{|S|!(n-|S|-1)}{n!}\left[v(S \cup \{i\}) - v(S)\right] \tag{11}$$

本文采用 SHAP 来分析影响脉冲量的特征重要性，对特征进行整体分析并将结果可视化，直观得到各个指标特征对脉冲量的正负影响，还探索了单个特征以及两个特征之间的交互如何影响模型的输出。

6.1.2 模型的对比与建模

由于不考虑生产参数 f3（开始灌注时系统设置压力）和生产参数 f9（订单批次）对于脉冲量的影响，在数据预处理阶段已将这两个生产参数进行剔除，剩余变量在导入模型时先进行标准化的处理，本文所采用的数据标准化的处理方法为 min-max 标准化方法，具体的处理方法如式所示：

$$x' = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \tag{12}$$

本文采用 R^2，或决定系数，作为衡量模型拟合优度的指标。它表示模型解释的变异比例。公式如下：

$$R^2 = 1 - \frac{RSS}{TSS} \tag{13}$$

其中：

残差平方和（RSS）：$\sum(y_i - \hat{y}_i)^2$，即实际值与预测值之间的平方差。

总平方和（TSS）：$\sum(y_i - \bar{y})^2$，即实际值与实际值均值之间的平方差。

R^2 的值范围从 0 到 1，值越接近 1 表示模型拟合越好。

模型对比

XGBoost 与随机森林和决策树相比，具有以下优势：

精度：XGBoost 通常提供比决策树和随机森林更高的预测精度，特别是在复杂数据集上。

训练速度：XGBoost 通过支持并行计算和高效的优化算法，通常训练速度快于随机森林和决策树。

过拟合控制：XGBoost 内置了正则化机制（L1 和 L2），有效防止过拟合，而随机森林和决策树没有这种直接的正则化手段。

特征重要性：XGBoost 提供了详细的特征重要性评分，帮助更好地解释模型预测，而随机森林和决策树虽然也提供特征重要性，但通常不如 XGBoost 直观。

处理缺失值：XGBoost 可以在训练过程中处理缺失值，而随机森林和决策树通常需要先处理缺失值或使用插补方法。

模型灵活性：XGBoost 支持多种损失函数和评估指标，适应性强，能够处理回归、分类、排序等多种任务。随机森林和决策树在这方面则不如 XGBoost 灵活。

总的来说，XGBoost 在处理复杂问题时通常能提供更高的性能和更强的灵活性，但其模型复杂性也较高，需要更多的调优和计算资源。

对比不同随机采样数据下，XGBoost 模型与随机森林和决策树的运行结果如图所示：

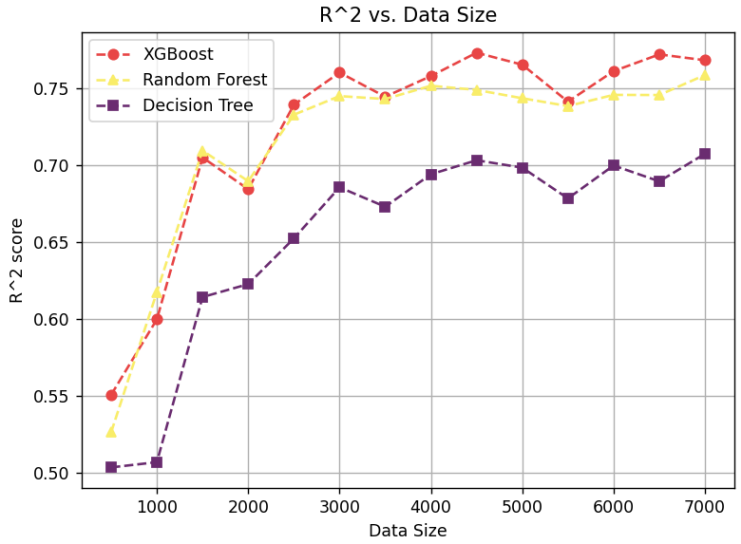

图 5 运行结果对比图

根据实验对比结果，XGBoost 模型明显优于随机森林和决策树。在实际运行中，XGBoost 不仅表现出更高的效率，还在效果上优于随机森林。因此，本文选择使用 XGBoost 模型。

基于随机搜索的调优方法，本实验选取的超参数即其搜索空间如下表所示。

表 1 实验超参数设置

超参数	数值
每棵树的特征子样本比例	0.8658
节点分裂所需的最小损失减少量	0.0638
学习率（步长）	0.0350
树的最大深度	9
树的数量（迭代次数）	289
用于训练每棵树的样本比例	0.7809

使用 XGBoost 模型对生产参数建模后的学习曲线如下图所示：

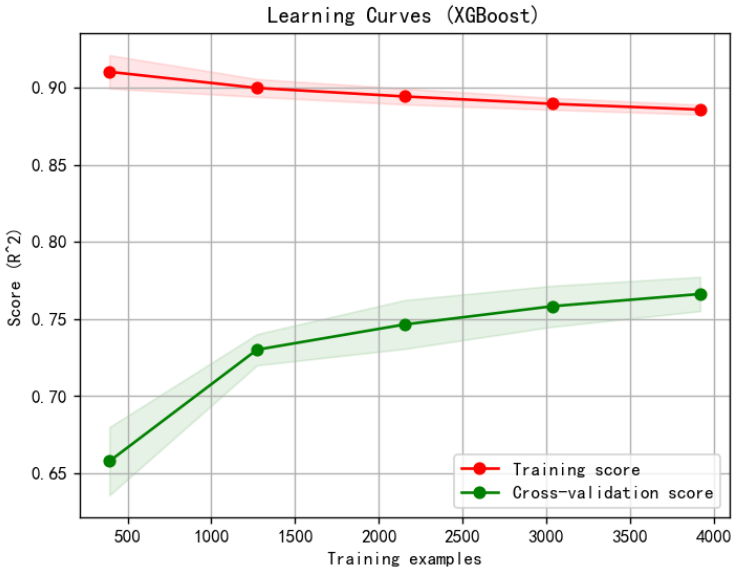

图 6 XGBoost 学习曲线图

6.1.3 问题的求解

由于传统的特征重要性计算方法存在一定局限性，因此利用 SHAP 值来计算特征重要性，即利用每个特征 SHAP 值绝对值的平均值表示该特征的重要性。为了更直观的判断给出了各参数的重要性排名。

每个参数在整体样本上的 SHAP 绝对值取平均值来代表该参数的重要性，因此 SHAP 均值越大，则参数越重要。由下示右图可知生产参数 f4 的 SHAP 值绝对值的平均值较大，对脉冲量的影响程度更高，生产参数 f2 和生产参数 f1 的影响次之。

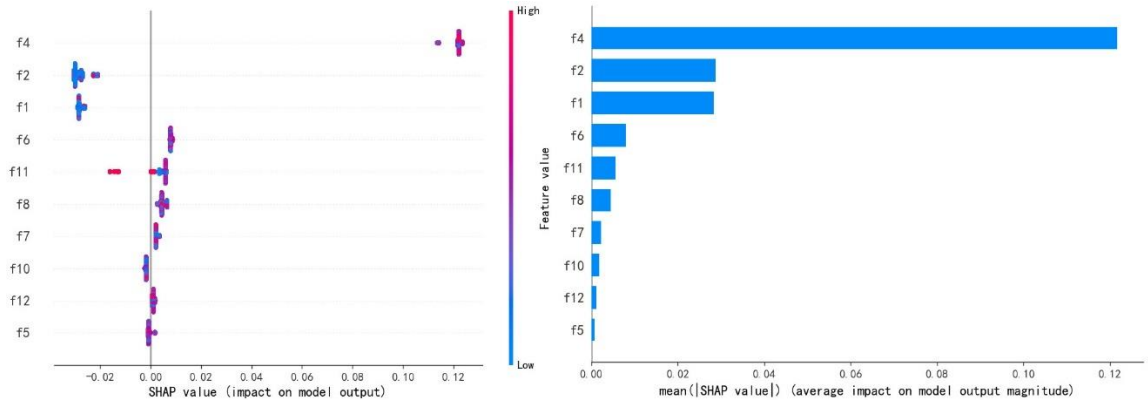

图 7 SHAP 值

在整体样本上，进行不同特征 SHAP 值的分布展示，上示左图中每一行代表一个特征，横坐标为每个特征的 SHAP 值。一个点代表一个样本，颜色越红说明特征本身

数值越大，颜色越蓝说明特征本身数值越小。可知生产参数 f4 与脉冲量成正相关，生产参数 f2 和生产参数 f1 与脉冲量成负相关。

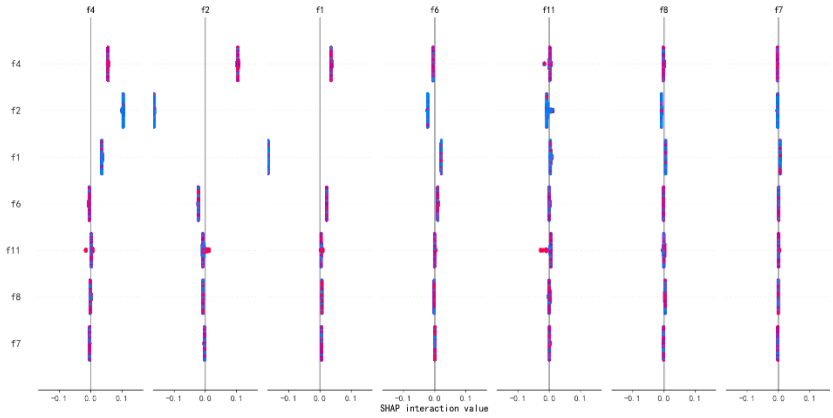

图 8 参数间的 SHAP 交互图

为分析特征之间的交互对模型分类结果的影响，这里计算量特征之间的 SHAP 交互值，并绘制了上图，可以直观的看出，生产参数 f4 和 f2、f1 之间有一定的相关性。

综上，最终筛选出与脉冲量变化有关的因素，影响程度前三的是生产参数 f4（由冷媒管道返回的灌注过程最小压力）、f2（真空泵预抽真空状态后的抽空压力）、f1（保压三秒之后的检测压力）。

6.2 问题二模型的建立与求解

6.2.1 算法介绍

CatBoost

CatBoost 使用梯度提升树作为基本学习器，是一种集成学习方法，通过逐步构建多个决策树来改进预测性能。每棵树都试图纠正前一棵树的错误。能够高效合理地处理类别型特征（Categorical Features），这是其与其他 GBDT 框架（如 XGBoost、LightGBM）相比的一个重要优势。采用了 Ordered Boost 的方法，通过考虑样本的观测历史来计算目标统计量（Target Statistics, TS），从而避免梯度估计的偏差，并减少预测偏移，提高算法的准确性和泛化能力。

模型的预测值是通过累加所有决策树的贡献来计算的：

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \tag{14}$$

其中，$f_k(x_i)$ 是第 k 个决策树对第 i 个样本的预测贡献，K 是决策树的总数，x_i 是第 i 个样本的特征向量。

在每一步k，CatBoost 会找到一个新的决策树 f_k，以最大化损失函数的负梯度，这可以视为以下优化问题：

$$f_k = \arg\min_f \sum_{i=1}^{N} L\left(y_i, \hat{y}_i^{(k-1)} + f\left(x_i\right)\right) \tag{15}$$

其中，L是损失函数，N是训练样本的数量，$\hat{y}_i^{(k-1)}$是在加入 f_k 之前，使用前$k-1$个第i个样本的预测值。

K-means 聚类

K-means 聚类算法是一种广泛使用的聚类算法，属于无监督学习范畴。对于给定的样本集，按照样本之间的距离大小，将样本集划分为 K 个簇，使得簇内的点尽量紧密地连在一起，而簇间的距离尽量大。通过迭代优化聚类中心和样本点的分配，实现对数据集的聚类分析。

使用欧氏距离的平方作为样本点之间的距离度量。对于两个样本点x_i和x_j，其欧氏距离的平方为：

$$d\left(x_i, x_j\right) = \sum_{k=1}^{m}\left(x_{ki} - x_{kj}\right)^2 = \left\|x_i - x_j\right\|_2^2 \tag{16}$$

其中，m是特征维度，x_i和x_j分别是m维的样本点。

每个簇的聚类中心（质心）是该簇中所有样本点的均值。对于第j个簇S_j，其聚类中心Z_j的计算公式为：

$$Z_j = \frac{1}{N}\sum_{i=1}^{N_j} X_i, X_i \in S_j \tag{17}$$

其中，N_j是簇S_j中的样本个数，X_i是簇S_j中的样本点。

算法的目标是最小化聚类集中每一个样本点到该类中心的距离的平方之和，即 SSE（Sum of Squared Error，误差平方和）：

$$J = \sum_{j=1}^{k}\sum_{i=1}^{N_j}\left\|X_i - Z_j\right\|^2, X_i \in S_j \tag{18}$$

SSE 值越小，表示数据点越接近于它们的质心，聚类效果也越好。

轮廓系数

轮廓系数（Silhouette Coefficient）是一种用于评估聚类效果的指标，通过比较每个数据点与其所在聚类内的点的平均距离（内聚度）和该点与最近的其他聚类中的点

的平均距离（分离度）来评估聚类效果。综合考虑了数据点在聚类中的紧密程度和分离程度。

$$s(i)=\frac{b(i)-a(i)}{\max\left(a(i),b(i)\right)} \tag{19}$$

$s(i)$表示第 i 个数据点的轮廓系数；$a(i)$表示第 i 个数据点与其所在聚类内其他数据点的平均距离，反映了数据点在其所属聚类内的紧密程度（内聚度）；$b(i)$表示第 i 个数据点与最近的其他聚类中所有数据点的平均距离的最小值，反映了数据点与其最近聚类之间的分离程度。

6.2.2 问题求解

结合问题一，由生产参数 f4、f2 和 f1 的相关数据构建定量预测模型。首先绘制三个生产参数的三维可视化如下图所示：

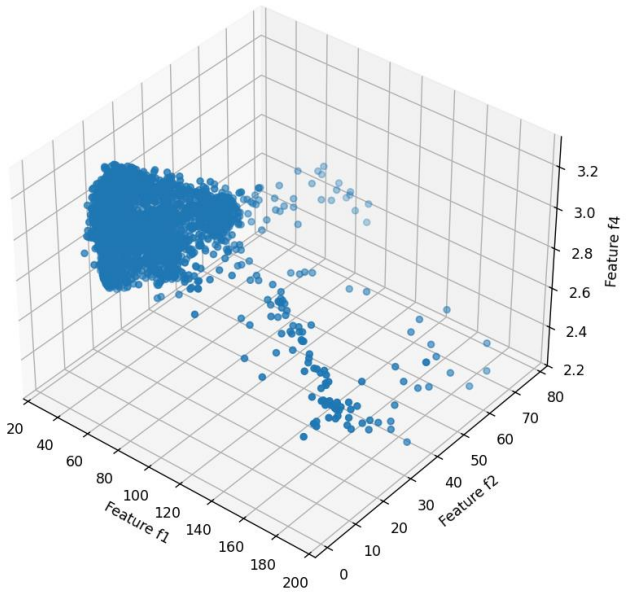

图 9 生产参数三维图

观察到三个生产参数的分布可能存在一定的类别关系，故采用基于轮廓系数的 K-means 算法进行聚类分析。

根据轮廓系数来确定最佳的聚类数量，由图 8 所示，最佳聚类数量为 3。并使用 K-means 算法进行实际的聚类分析，从而确保聚类结果的有效性和可靠性。聚类结果如图 9 所示。

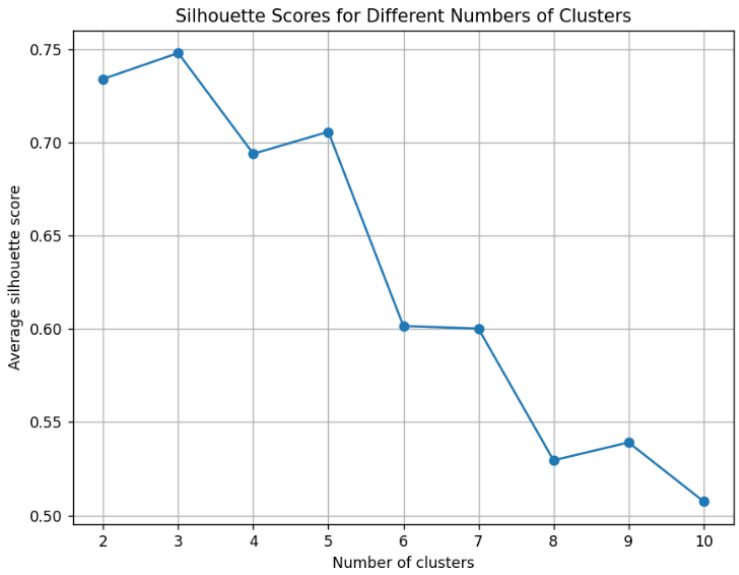

图 10 轮廓系数图

聚类结果如下图所示，共分为三个类别，其中由于黄色类别的数据占比仅为 1.37%，故不予其进行建模分析。

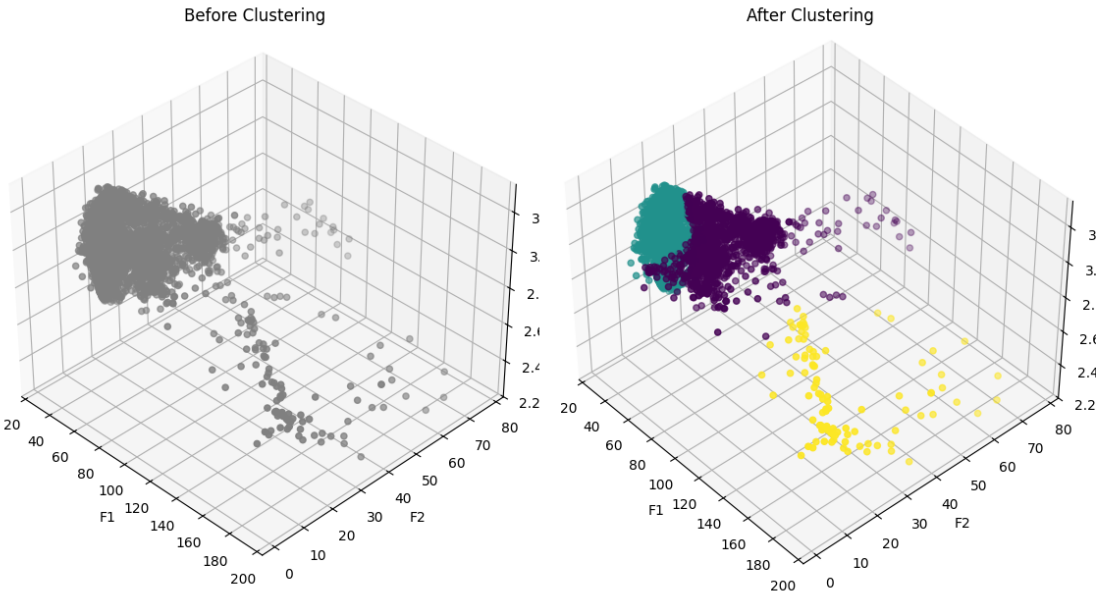

图 11 K-means 聚类

对数据进行聚类划分后，采用基于随机搜索进行超参数优化的 CatBoost 模型对数据进行建模，相较于 XGBoost 模型，CatBoost 模型在大数据上表现更有优势。

对比二者的 R^2、MAE 和 RMSE 如下表所示：

表 2 模型对比

数据	Data1		Data2	
指标	CatBoost	XGBoost	CatBoost	XGBoost
R^2	**0.8216**	0.8009	**0.8063**	0.7391
MAE	13.1200	13.6683	11.5078	12.5211
RMSE	20.0813	20.6462	18.8152	21.2918

问题二会根据分类建立两个预测模型，在进行预测时先要使用 K-means 进行类别的划分，在对其使用 CatBoost 模型进行预测。

6.3 问题三模型的建立与求解

6.3.1 算法介绍

DBSCAN

DBSCAN（Density-Based Spatial Clustering of Applications with Noise）是一种基于密度的空间聚类算法，它能够将具有足够高密度的区域划分为簇，并能在带有噪声的数据集中发现任意形状的簇。DBSCAN 能够识别出噪声点，并且不需要事先指定簇的数量。

算法相关步骤为：

（1）初始化：将所有点标记为未访问。

（2）选择点：从未访问的点中随机选择一个点 P，并将其标记为已访问。

（3）检查邻居：如果 P 的 ε-邻域内至少包含 MinPts 个点，则创建一个新的簇，并将 P 添加到该簇中。然后，对 P 的 ε-邻域内每个点 Q 执行以下操作：如果 Q 尚未被访问，则将其标记为已访问，并检查 Q 是否为核心点。如果 Q 是核心点，则将其添加到簇中，并递归地检查其 ε-邻域内的所有点。

（4）重复：重复步骤 2 和 3，直到所有点都被访问。

（5）噪声点：未被归入任何簇的点被视为噪声点。

6.3.2 问题求解

根据脉冲量取值范围(11570,11600)来确定生产参数的范围，基于前两问，继续对生产参数 f1、f2 和 f4 进行三维可视化，并使用颜色区分在脉冲量范围内的点，如下图所示：

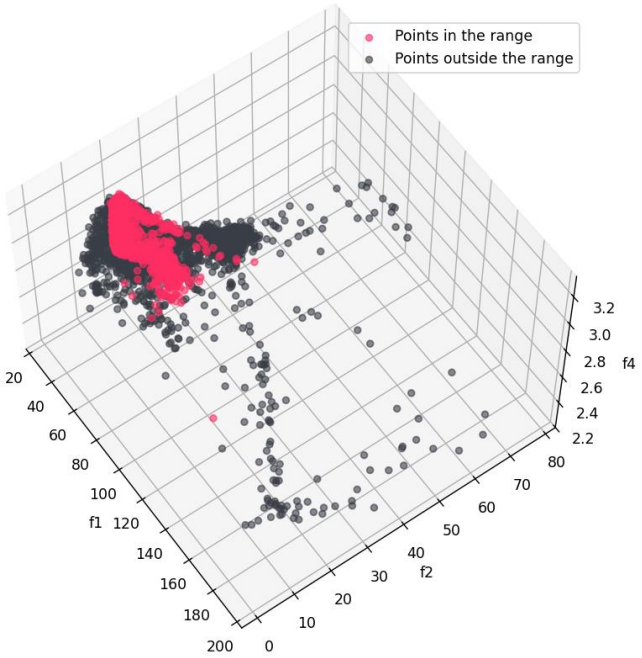

图 12 参数三维区分图

根据图示，在脉冲量范围内的点被其他范围的点夹在中间。为确定生产参数的取值范围，本文采用次优方案：首先对两个范围的数据进行 DBSCAN（密度）聚类，以缩小范围。然后，通过调整聚类的最小样本数，选择最佳的范围，从而确定生产参数的最终取值范围。

首先，进行 DBSCAN（密度）聚类并只保留最大的一类以缩小范围，如下图所示：

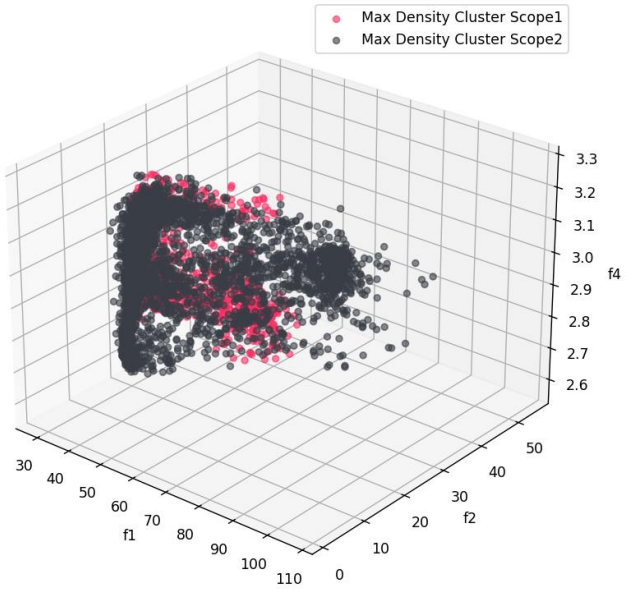

图 13 DBSCAN 聚类图

接着，通过调整范围 1 的最小密度值来缩小生产参数点的范围，如下图所示：

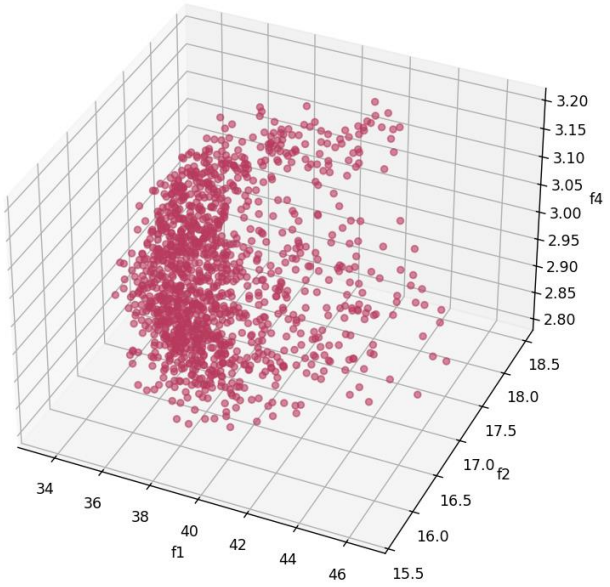

图 14 最小密度 DBSCAN 聚类图

最后通过选择一个最小且优的范围确定生产参数的范围，结果如下表所示：

表 3 参数范围

生产参数	范围
f1（保压三秒之后的检测压力）	(36, 37)
f2（真空泵预抽真空状态后的抽空压力）	(16.5, 17)
f4（由冷媒管道返回的灌注过程最小压力）	(2.85, 3.05)

七、 模型的评价、改进与推广

7.1 模型的优点

- XGBoost 和 CatBoost 模型在处理复杂数据特征和大规模数据集时表现出色。
- 两者都提供了强大的特征选择和自动化调优能力。
- CatBoost 特别适用于处理类别特征，而 XGBoost 在处理各种数据类型时更为灵活。

7.2 模型的缺点

- XGBoost 对于大数据集可能需要较长的训练时间。

- CatBoost 的计算需求较高，可能会对硬件资源有较大要求。
- 两者都可能过拟合，如果特征选择不当或参数调优不足。

7.3 模型的改进

- 集成多种模型方法，例如将 XGBoost 和 CatBoost 进行组合使用，以利用各自的优点。
- 结合特征工程和降维技术优化数据输入，提高模型效率。

7.4 模型的推广

- 将XGBoost和CatBoost模型结合人工经验和大语言模型（LLM），如ChatGPT，进行综合决策。
- 通过自动化工具和模型融合技术推广，进一步提高模型的普适性和适应性。
- 将这些模型应用于更广泛的领域，利用实际应用中的反馈进行迭代改进。

八、 参考文献

[1] 易中彪.基于 LSTM-XGBoost 组合的脱硫效率模型预测[J].电工技术,2024,(02):32-36.

[2] 黄宇玲.基于特征选择和 Stacking 集成学习算法的森林蓄积量估测研究[D].浙江农林大学,2019.

[3] 彭白雪,陈清华,季家东.基于 XGBoost 和 SHAP 的制冷系统故障分析[J].低温与超导,2024,52(07):89-96.

[4] 李占山,刘兆赓.基于 XGBoost 的特征选择算法[J].通信学报,2019,40(10):101-108.

[5] 周闯,张琦锦,郭映映,等.主成分分析法在合肥市空气质量评估中的应用[J].大气与环境光学学报,2024,19(04):479-488.

[6] Safari A ,Sabahi M ,Oshnoei A .ResFaultyMan: An intelligent fault detection predictive model in power electronics systems using unsupervised learning isolation forest[J]. Heliyon,2024,10(15):e35243-e35243.

[7] Chen W ,Xu X ,Liu W .Combined PMF modelling and machine learning to identify sources and meteorological influencers of volatile organic compound pollution in an industrial city in eastern China[J].Atmospheric Environment,2024,334120714-120714.

附录

附录 1
介绍：支撑材料的文件列表
EDA ----图片 --------f1-2.png --------f4-12.png --------散点图.png ----data.xlsx ----均匀分布检验.py ----散点图.py ----正态分布检验.py ----频率分布图.py **问题一** ----XGBoost --------data.xlsx --------XGBoost.py ----图片 --------shap1.png --------shap2.png --------shap3.png --------shap4.png --------合并 1.jpg --------合并 2.jpg --------学习曲线.png --------对比可视化.png ----模型对比 --------data.xlsx --------XGBoost.py --------决策树.py --------模型对比可视化 --------随机森林.py **问题二** **----**CATKM **--------**catboost_info --------CAT.py --------cluster_0.xlsx --------cluster_1.xlsx --------cluster_2.xlsx

--------data.xlsx
--------KM.py
--------XGB.py
--------可视化.py
--------对比.py
----图片
--------可视化.png
--------合并.png
--------聚类.png
--------轮廓系数.png

问题三
----图片
--------双聚类.png
--------可视化.png
--------结果.png
----data.xlsx
----双聚类.py
----可视化.py
----密度聚类.py
----结果.py

附录 2

介绍：代码

EDA 均匀分布检验.py

```
# -*- coding: utf-8 -*-

from sklearn.preprocessing import MinMaxScaler
from scipy import stats
import pandas as pd

# 读取数据
data = pd.read_excel("data2.xlsx")

# 标准化数据到 [0, 1] 范围
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data[['f1', 'f2', 'f4', 'f5', 'f6', 'f7', 'f8', 'f10', 'f11', 'f12']])

# 执行 Kolmogorov-Smirnov 检验
```

```
for i, col in enumerate(['f1', 'f2', 'f4', 'f5', 'f6', 'f7', 'f8', 'f10', 'f11', 'f12']):
    ks_statistic, p_value = stats.kstest(data_scaled[:, i], 'uniform')

    # 判断结果
    alpha = 0.025   # 显著性水平
    if p_value < alpha:
        pass

    else:
        print('-' * 20)
        print(f"变量 {col} 服从均匀分布")
        print('-' * 20)
```

EDA 散点图.py

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False

# 读取数据
df = pd.read_excel("data2.xlsx")

# 绘制 1 行 2 列的子图
fig, axs = plt.subplots(1, 2, figsize=(10, 5))   # 创建一个 1 行 2 列的子图布局
for idx, i in enumerate(['f1', 'f2']):
    data = df[i].dropna()   # 去掉 NaN 值
    x = np.arange(len(data))
    y = data
    axs[idx].scatter(x, y, alpha=1, color='#3f72af')
    axs[idx].set_title(f'{i}散点图')

plt.tight_layout()   # 自动调整子图布局
plt.show()
```

EDA 正态分布检验.py

```
---------------------------------------------------------------------------------------------------------
# -*- coding: utf-8 -*-

from scipy import stats
import pandas as pd

# 读取数据
data = pd.read_excel("data2.xlsx")

# 提取特征列
#f5 f6 f10
for col in ['f1', 'f2', 'f4', 'f5', 'f6', 'f7', 'f8', 'f10', 'f11', 'f12']:
    sample_data = data[col].values
    # 执行 Anderson-Darling 检验
    result = stats.anderson(sample_data, dist='norm')
    #print(f"Anderson-Darling Statistic: {result.statistic}")
    #print(f"Critical Values: {result.critical_values}")
    #print(f"Significance Level: {result.significance_level}")
    # 判断结果

    alpha = 0.05
    if result.statistic > result.critical_values[2]:  # 选择合适的临界值
        pass

    else:
        print('-' * 20)
        print(f"变量 {col} 服从正态分布。")
        print('-' * 20)

---------------------------------------------------------------------------------------------------------
                                        EDA 频率分布图.py
---------------------------------------------------------------------------------------------------------
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd
import numpy as np

plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
```

```
df = pd.read_excel("data2.xlsx")

fig, axs = plt.subplots(2, 4, figsize=(18, 10))  # 创建一个 2 行 4 列的子图布局
for idx, i in enumerate(['f4', 'f5', 'f6', 'f10', 'f7', 'f8', 'f11', 'f12']):
    data = df[i]
    # 计算正态分布曲线参数
    mean = np.mean(data)
    std_dev = np.std(data)
    xmin, xmax = np.min(data), np.max(data)
    x = np.linspace(xmin, xmax, 100)
    p = stats.norm.pdf(x, mean, std_dev)
    # 绘制频率直方图和正态分布曲线
    row = idx // 4
    col = idx % 4
    axs[row, col].hist(data, bins=30, density=True, alpha=0.6, color='#3f72af')
    axs[row, col].set_title(f'{i}频率直方图')
    if idx == 0 or idx == 1 or idx == 2 or idx == 3:
        axs[row, col].plot(x, p, 'gray', linewidth=2, linestyle='--')
        axs[row, col].legend(['正态分布曲线', '频率直方图'], loc='upper left')
        axs[row, col].grid()
    else:
        # 设置标题和标签
        # axs[row, col].set_xlabel('数据值')
        # axs[row, col].set_ylabel('密度')
        # 显示图例
        axs[row, col].legend(['频率直方图'], loc='upper left')
plt.tight_layout()  # 自动调整子图布局
plt.show()

# ['f1', 'f2']
fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # 创建一个 1 行 2 列的子图布局
for idx, i in enumerate(['f1', 'f2']):
    data = df[i]
    # 计算正态分布曲线参数
    mean = np.mean(data)
    std_dev = np.std(data)
    xmin, xmax = np.min(data), np.max(data)
    x = np.linspace(xmin, xmax, 100)
    p = stats.norm.pdf(x, mean, std_dev)
    # 绘制频率直方图和正态分布曲线
    col = idx  # 因为是一行两列，所以列数直接用 idx
    axs[col].hist(data, bins=30, density=True, alpha=0.6, color='#3f72af')
```

```
        axs[col].set_title(f'{i}频率直方图')
        # 设置标题和标签
        # axs[col].set_xlabel('数据值')
        # axs[col].set_ylabel('密度')
        # 显示图例
        axs[col].legend(['频率直方图'], loc='upper right')
        axs[col].grid()

plt.tight_layout()  # 自动调整子图布局
plt.show()
```

问题一 XGBoost.py

```
# -*- coding: utf-8 -*-

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import learning_curve
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import randint, uniform
import matplotlib.pyplot as plt
import xgboost as xgb
import pandas as pd
import numpy as np
import shap

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

# 读取数据
df = pd.read_excel("data.xlsx")
# 分离特征和目标变量
X = df.drop(columns='target')
y = df['target']

# 初始化 MinMaxScaler 对特征进行归一化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
y_scaled = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3,
```

```
random_state=42)

# 定义参数分布
param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': randint(3, 10),
    'learning_rate': uniform(0.01, 0.1),
    'gamma': uniform(0, 0.5),
    'subsample': uniform(0.5, 0.5),
    'colsample_bytree': uniform(0.5, 0.5)
}
# 创建 XGBoost 回归器对象
xgb_reg = xgb.XGBRegressor()
# 创建 RandomizedSearchCV 对象
random_search = RandomizedSearchCV(
    estimator=xgb_reg,  # 需要优化的模型，这里是 XGBRegressor 实例
    param_distributions=param_dist,  # 超参数分布的字典，用于指定要搜索的超参数范围
    n_iter=100,  # 随机搜索的迭代次数，即从参数分布中随机选择多少组参数组合进行评估
    cv=5,  # 交叉验证的折数，即将数据划分为 5 份进行交叉验证
    scoring='r2',  # 评估模型性能的指标，这里使用 R² 评分（决定系数），用于回归任务
    verbose=1,  # 控制输出的详细程度，1 表示输出一些基本的信息
    random_state=42,  # 随机种子，用于保证每次运行时的结果一致
    n_jobs=-1  # 并行运行的作业数，-1 表示使用所有可用的 CPU 核心
)
# 执行随机搜索
random_search.fit(X_train, y_train)
# 使用最优参数的模型进行预测和评估
best_xgb_model = random_search.best_estimator_
y_pred = best_xgb_model.predict(X_test)

# 计算模型的 R^2 分数
score = best_xgb_model.score(X_test, y_test)
print("R^2 score on test data: ", round(score, 4))
print("Best parameters found: ", random_search.best_params_)

explainer = shap.TreeExplainer(best_xgb_model)
shap_values = explainer.shap_values(X)  # 传入特征矩阵 X，计算 SHAP 值
# 可视化第一个 prediction 的解释
shap.initjs()
```

```
shap.force_plot(explainer.expected_value, shap_values[0, :], X.iloc[0, :])
shap.summary_plot(shap_values, X)
shap.summary_plot(shap_values, X, plot_type="bar")
shap_interaction_values = explainer.shap_interaction_values(X)
shap.summary_plot(shap_interaction_values, X)


# 绘制学习曲线
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score (R^2)")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring='r2')
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt


# Plot learning curve for the best XGBoost model
title = "Learning Curves (XGBoost)"
cv = 5
plot_learning_curve(random_search.best_estimator_, title, X_train, y_train, cv=cv, n_jobs=-1)
```

```
plt.show()
```

问题一 XGBoost.py

```
# -*- coding: utf-8 -*-

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import randint, uniform
import matplotlib.pyplot as plt
import xgboost as xgb
import pandas as pd

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

for num in range(1, 14):
    # 读取数据
    df = pd.read_excel("data2.xlsx")
    df = df.sample(n=num * 500, random_state=42)
    # 分离特征和目标变量
    X = df.drop(columns='target')
    y = df['target']
    # 初始化 MinMaxScaler 对特征进行归一化
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
    y_scaled = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3,
random_state=42)
    # 定义参数分布
    param_dist = {
        'n_estimators': randint(100, 1000),
        'max_depth': randint(3, 10),
        'learning_rate': uniform(0.01, 0.1),
        'gamma': uniform(0, 0.5),
        'subsample': uniform(0.5, 0.5),
        'colsample_bytree': uniform(0.5, 0.5)
    }
    # 创建 XGBoost 回归器对象
```

```
    xgb_reg = xgb.XGBRegressor()
    # 创建 RandomizedSearchCV 对象
    random_search = RandomizedSearchCV(
        estimator=xgb_reg,
        param_distributions=param_dist,
        n_iter=100,
        cv=5,
        scoring='r2',
        verbose=1,
        random_state=42,
        n_jobs=-1)
    # 执行随机搜索
    random_search.fit(X_train, y_train)
    # 使用最优参数的模型进行预测和评估
    best_xgb_model = random_search.best_estimator_
    y_pred = best_xgb_model.predict(X_test)
    # 计算模型的 R^2 分数
    score = best_xgb_model.score(X_test, y_test)
    print(f"{num * 500} data: ", round(score, 4))

df = pd.read_excel("data2.xlsx")
df = df.sample(n=6999)
# 分离特征和目标变量
X = df.drop(columns='target')
y = df['target']
# 初始化 MinMaxScaler 对特征进行归一化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
y_scaled = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3,
random_state=42)
# 定义参数分布
param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': randint(3, 10),
    'learning_rate': uniform(0.01, 0.1),
    'gamma': uniform(0, 0.5),
    'subsample': uniform(0.5, 0.5),
    'colsample_bytree': uniform(0.5, 0.5)
}
# 创建 XGBoost 回归器对象
```

```
xgb_reg = xgb.XGBRegressor()
# 创建 RandomizedSearchCV 对象
random_search = RandomizedSearchCV(
        estimator=xgb_reg,
        param_distributions=param_dist,
        n_iter=100,
        cv=5,
        scoring='r2',
        verbose=1,
        random_state=42,
        n_jobs=-1)
# 执行随机搜索
random_search.fit(X_train, y_train)
# 使用最优参数的模型进行预测和评估
best_xgb_model = random_search.best_estimator_
y_pred = best_xgb_model.predict(X_test)
# 计算模型的 R^2 分数
score = best_xgb_model.score(X_test, y_test)
print(f"All data: ", round(score, 4))
```

问题一 决策树.py

```
# -*- coding: utf-8 -*-

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import randint, uniform
import matplotlib.pyplot as plt
import pandas as pd

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

for num in range(1, 14):
        # 读取数据
        df = pd.read_excel("data2.xlsx")
        df = df.sample(n=num * 500, random_state=42)
        # 分离特征和目标变量
        X = df.drop(columns='target')
```

```
    y = df['target']
    # 初始化 MinMaxScaler 对特征进行归一化
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
    y_scaled = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3,
random_state=42)
    # 定义参数分布（决策树）
    param_dist = {
        'max_depth': randint(3, 10),
        'min_samples_split': randint(2, 20),
        'min_samples_leaf': randint(1, 20),
        'max_features': uniform(0.5, 0.5)
    }
    # 创建决策树回归器对象
    dt_reg = DecisionTreeRegressor()
    # 创建 RandomizedSearchCV 对象
    random_search = RandomizedSearchCV(
        estimator=dt_reg,
        param_distributions=param_dist,
        n_iter=100,
        cv=5,
        scoring='r2',
        verbose=1,
        random_state=42,
        n_jobs=-1)
    # 执行随机搜索
    random_search.fit(X_train, y_train)
    # 使用最优参数的模型进行预测和评估
    best_dt_model = random_search.best_estimator_
    y_pred = best_dt_model.predict(X_test)
    # 计算模型的 R^2 分数
    score = best_dt_model.score(X_test, y_test)
    print(f"{num * 500} data: ", round(score, 4))

df = pd.read_excel("data2.xlsx")
# 分离特征和目标变量
X = df.drop(columns='target')
y = df['target']
# 初始化 MinMaxScaler 对特征进行归一化
scaler = MinMaxScaler()
```

```
X_scaled = scaler.fit_transform(X)
y_scaled = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3, random_state=42)
# 定义参数分布（决策树）
param_dist = {
    'max_depth': randint(3, 10),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20),
    'max_features': uniform(0.5, 0.5)
}
# 创建决策树回归器对象
dt_reg = DecisionTreeRegressor()
# 创建 RandomizedSearchCV 对象
random_search = RandomizedSearchCV(
    estimator=dt_reg,
    param_distributions=param_dist,
    n_iter=100,
    cv=5,
    scoring='r2',
    verbose=1,
    random_state=42,
    n_jobs=-1)
# 执行随机搜索
random_search.fit(X_train, y_train)
# 使用最优参数的模型进行预测和评估
best_dt_model = random_search.best_estimator_
y_pred = best_dt_model.predict(X_test)
# 计算模型的 R^2 分数
score = best_dt_model.score(X_test, y_test)
print("All data: ", round(score, 4))
```

问题一 模型对比可视化.py

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
# 数据
plt.figure()
data_sizes = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000,
```

```
6500, 7000]
r2_scores = [0.5506, 0.5996, 0.7048, 0.6842, 0.7391, 0.76, 0.7443, 0.7577, 0.7729, 0.7651, 0.7411, 0.7609, 0.7718, 0.768]
plt.plot(data_sizes, r2_scores, marker='o', linestyle='--', color='#e84545', label='XGBoost')
data_sizes = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000]
r2_scores = [0.5263, 0.6173, 0.7093, 0.6894, 0.7326, 0.7447, 0.7428, 0.7514, 0.7487, 0.7433, 0.7382, 0.7455, 0.7454, 0.7584]
plt.plot(data_sizes, r2_scores, marker='^', linestyle='--', color='#f9ed69', label='Random Forest')
data_sizes = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000]
r2_scores = [0.5034, 0.5068, 0.6141, 0.6225, 0.6522, 0.6854, 0.6728, 0.6939, 0.703, 0.6982, 0.6782, 0.6996, 0.6893, 0.7072]
plt.plot(data_sizes, r2_scores, marker='s', linestyle='--', color='#6a2c70', label='Decision Tree')

# Linear Regression
#  显示图形
plt.xlabel('Data Size')
plt.ylabel('R^2 score')
plt.title('R^2 vs. Data Size')
plt.tight_layout()
plt.grid(True)
plt.legend()
plt.show()
```

问题一 随机森林.py

```
# -*- coding: utf-8 -*-

from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import randint, uniform
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import pandas as pd

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

for num in range(1, 14):
    # 读取数据
    df = pd.read_excel("data2.xlsx")
    df = df.sample(n=num * 500, random_state=42)
    # 分离特征和目标变量
    X = df.drop(columns='target')
    y = df['target']
    # 初始化 MinMaxScaler 对特征进行归一化
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)
    y_scaled = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3,
random_state=42)
    # 定义参数分布
    param_dist = {
        'n_estimators': randint(100, 1000),
        'max_depth': randint(3, 10),
        'min_samples_split': randint(2, 20),
        'min_samples_leaf': randint(1, 20),
        'max_features': uniform(0.5, 0.5)
    }
    # 创建随机森林回归器对象
    rf_reg = RandomForestRegressor()
    # 创建 RandomizedSearchCV 对象
    random_search = RandomizedSearchCV(
        estimator=rf_reg,
        param_distributions=param_dist,
        n_iter=100,
        cv=5,
        scoring='r2',
        verbose=1,
        random_state=42,
        n_jobs=-1)
    # 执行随机搜索
    random_search.fit(X_train, y_train)
    # 使用最优参数的模型进行预测和评估
    best_rf_model = random_search.best_estimator_
    y_pred = best_rf_model.predict(X_test)
    # 计算模型的 R^2 分数
```

```
        score = r2_score(y_test, y_pred)
        print(f"{num * 500}data: ", round(score, 4))

df = pd.read_excel("data2.xlsx")
# 分离特征和目标变量
X = df.drop(columns='target')
y = df['target']
# 初始化 MinMaxScaler 对特征进行归一化
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
y_scaled = scaler.fit_transform(y.values.reshape(-1, 1)).flatten()
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3,
random_state=42)
# 定义参数分布
param_dist = {
            'n_estimators': randint(100, 1000),
            'max_depth': randint(3, 10),
            'min_samples_split': randint(2, 20),
            'min_samples_leaf': randint(1, 20),
            'max_features': uniform(0.5, 0.5)
}
# 创建随机森林回归器对象
rf_reg = RandomForestRegressor()
# 创建 RandomizedSearchCV 对象
random_search = RandomizedSearchCV(
            estimator=rf_reg,
            param_distributions=param_dist,
            n_iter=100,
            cv=5,
            scoring='r2',
            verbose=1,
            random_state=42,
            n_jobs=-1)
# 执行随机搜索
random_search.fit(X_train, y_train)
# 使用最优参数的模型进行预测和评估
best_rf_model = random_search.best_estimator_
y_pred = best_rf_model.predict(X_test)
# 计算模型的 R^2 分数
score = r2_score(y_test, y_pred)
print("All data: ", round(score, 4))
```

问题二 CAT.py

```
# -*- coding: utf-8 -*-

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import uniform, loguniform
from catboost import CatBoostRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import pandas as pd
import numpy as np

# 读取数据
df = pd.read_excel("cluster_1.xlsx")
X = df[['f1', 'f2', 'f4']]
y = df['target']

# 数据归一化
scaler_X = MinMaxScaler()
X_scaled = scaler_X.fit_transform(X)

scaler_y = MinMaxScaler()
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()

# 数据分割
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3, random_state=42)

# 创建 CatBoost 模型
model = CatBoostRegressor(silent=True)

# 参数搜索空间
param_dist = {
    'iterations': [50, 100, 200],    # Discrete values
    'learning_rate': loguniform(1e-3, 1e-1),    # Log-uniform distribution for learning rate
    'depth': [6, 8, 10, 12, 14],    # Discrete values for depth
    'l2_leaf_reg': uniform(1, 10),    # Uniform distribution from 1 to 11
    'subsample': uniform(0.7, 0.3),    # Uniform distribution from 0.7 to 1.0
    'border_count': [32, 64, 128, 256]    # Discrete values for border count
}
```

```
# 设置随机搜索
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=100,  # 迭代次数
    scoring='neg_mean_squared_error',  # 对于回归任务使用负均方误差
    cv=5,  # 交叉验证折数
    random_state=42,
    n_jobs=-1
)

# 执行随机搜索
random_search.fit(X_train, y_train)

# 输出最佳参数
print(f"最佳参数: {random_search.best_params_}")

# 使用最佳参数的模型进行预测
best_model = random_search.best_estimator_
y_pred_scaled = best_model.predict(X_test)

# 逆归一化预测结果和真实值
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
y_pred_original = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).flatten()

# 计算评估指标
mae = mean_absolute_error(y_test_original, y_pred_original)
rmse = np.sqrt(mean_squared_error(y_test_original, y_pred_original))
r2_score = best_model.score(X_test, y_test)

print(f"R^2: {r2_score:.4f}")
print(f"MAE: {mae:.4f}")
print(f"RMSE: {rmse:.4f}")
```

问题二 KM.py

```
# -*- coding: utf-8 -*-

from sklearn.metrics import silhouette_score
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
```

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 读取数据
df = pd.read_excel("data.xlsx")

# 提取特征
X = df[['f1', 'f2', 'f4']]

# 设置要测试的簇数范围
range_n_clusters = list(range(2, 11))  # 从 2 到 10

silhouette_avgs = []

for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X)
    cluster_labels = kmeans.labels_

    # 计算轮廓系数
    silhouette_avg = silhouette_score(X, cluster_labels)
    silhouette_avgs.append(silhouette_avg)
    print(f'Number of clusters = {n_clusters}, Average silhouette score = {silhouette_avg}')

# 绘制轮廓系数图
plt.figure(figsize=(8, 6))
plt.plot(range_n_clusters, silhouette_avgs, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Average silhouette score')
plt.title('Silhouette Scores for Different Numbers of Clusters')
plt.grid()
plt.show()

# 根据最佳轮廓系数选择聚类数
best_n_clusters = range_n_clusters[np.argmax(silhouette_avgs)]
print(f'Best number of clusters: {best_n_clusters}')

# 使用最佳簇数进行最终的 K-means 聚类
kmeans = KMeans(n_clusters=best_n_clusters)
kmeans.fit(X)
```

```
df['cluster'] = kmeans.labels_

# 创建图形和 3D 坐标轴
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# color: viridis plasma inferno magma RdGy
# 按照聚类标签绘制不同颜色的点
scatter = ax.scatter(df['f1'], df['f2'], df['f4'], c=df['cluster'], cmap='viridis', marker='o')

# 设置标签
ax.set_xlabel('F1')
ax.set_ylabel('F2')
ax.set_zlabel('F4')

# 显示图形
plt.colorbar(scatter, ax=ax, label='Cluster')
plt.show()

# 保存每个类别的数据到不同的 Excel 文件
for cluster_label in df['cluster'].unique():
    # 选择当前类别的数据
    cluster_data = df[df['cluster'] == cluster_label]

    # 创建文件名
    filename = f'cluster_{cluster_label}.xlsx'

    # 保存到 Excel 文件
    cluster_data.to_excel(filename, index=False)
    print(f'Saved data for cluster {cluster_label} to {filename}')
```

问题二 XGB.py

```
# -*- coding: utf-8 -*-

from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import randint, uniform
import matplotlib.pyplot as plt
```

```
import xgboost as xgb
import pandas as pd
import numpy as np

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

# 读取数据
df = pd.read_excel("cluster_0.xlsx")
# 分离特征和目标变量
X = df.drop(columns='target')
y = df['target']

# 初始化 MinMaxScaler 对特征进行归一化
scaler_X = MinMaxScaler()
X_scaled = scaler_X.fit_transform(X)

scaler_y = MinMaxScaler()
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.3,
random_state=42)

# 定义参数分布
param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': randint(3, 10),
    'learning_rate': uniform(0.01, 0.1),
    'gamma': uniform(0, 0.5),
    'subsample': uniform(0.5, 0.5),
    'colsample_bytree': uniform(0.5, 0.5)
}

# 创建 XGBoost 回归器对象
xgb_reg = xgb.XGBRegressor()

# 创建 RandomizedSearchCV 对象
random_search = RandomizedSearchCV(
    estimator=xgb_reg,
    param_distributions=param_dist,
    n_iter=100,
    cv=5,
```

```
        scoring='r2',
        verbose=1,
        random_state=42,
        n_jobs=-1
)

# 执行随机搜索
random_search.fit(X_train, y_train)

# 使用最优参数的模型进行预测和评估
best_xgb_model = random_search.best_estimator_
y_pred_scaled = best_xgb_model.predict(X_test)

# 逆归一化预测结果和真实值
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
y_pred_original = scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).flatten()

# 计算 MAE 和 RMSE
mae = mean_absolute_error(y_test_original, y_pred_original)
rmse = np.sqrt(mean_squared_error(y_test_original, y_pred_original))

# 计算模型的 R^2 分数
score = best_xgb_model.score(X_test, y_test)

print('-' * 10)
print("R^2: ", round(score, 4))
print(f"MAE: {mae:.4f}")
print(f"RMSE: {rmse:.4f}")
print('-' * 10)

print("Best parameters found: ", random_search.best_params_)
```

问题二 可视化.py

```
# -*- coding: utf-8 -*-

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import pandas as pd

# 读取数据
```

```
df = pd.read_excel("data.xlsx")

# 提取特征
X = df[['f1', 'f2', 'f4']]

# 创建 3D 图形
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# 绘制散点图
ax.scatter(X['f1'], X['f2'], X['f4'])

# 设置标签
ax.set_xlabel('Feature f1')
ax.set_ylabel('Feature f2')
ax.set_zlabel('Feature f4')

# 显示图形
plt.show()
```

问题二 对比.py

```
# -*- coding: utf-8 -*-

from sklearn.metrics import silhouette_score
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 读取数据
df = pd.read_excel("data.xlsx")

# 提取特征
X = df[['f1', 'f2', 'f4']]

# 设置要测试的簇数范围
range_n_clusters = list(range(2, 11))  # 从 2 到 10

silhouette_avgs = []
```

```
for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X)
    cluster_labels = kmeans.labels_

    # 计算轮廓系数
    silhouette_avg = silhouette_score(X, cluster_labels)
    silhouette_avgs.append(silhouette_avg)
    print(f'Number of clusters = {n_clusters}, Average silhouette score = {silhouette_avg}')

# 绘制轮廓系数图
plt.figure(figsize=(12, 6))

# 聚类前的三维图
ax1 = plt.subplot(1, 2, 1, projection='3d')
ax1.scatter(X['f1'], X['f2'], X['f4'], c='gray', marker='o')
ax1.set_xlabel('F1')
ax1.set_ylabel('F2')
ax1.set_zlabel('F4')
ax1.set_title('Before Clustering')

# 根据最佳轮廓系数选择聚类数
best_n_clusters = range_n_clusters[np.argmax(silhouette_avgs)]
print(f'Best number of clusters: {best_n_clusters}')

# 使用最佳簇数进行最终的 K-means 聚类
kmeans = KMeans(n_clusters=best_n_clusters)
kmeans.fit(X)
df['cluster'] = kmeans.labels_

# 聚类后的三维图
ax2 = plt.subplot(1, 2, 2, projection='3d')
scatter = ax2.scatter(df['f1'], df['f2'], df['f4'], c=df['cluster'], cmap='viridis', marker='o')
ax2.set_xlabel('F1')
ax2.set_ylabel('F2')
ax2.set_zlabel('F4')
ax2.set_title('After Clustering')
# plt.colorbar(scatter, ax=ax2, label='Cluster')

# 显示图形
```

```
plt.tight_layout()
plt.show()
```

问题三 双聚类.py

```
# -*- coding: utf-8 -*-

from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 读取数据
df1 = pd.read_excel("data.xlsx", sheet_name='Sheet1')
df2 = pd.read_excel("data.xlsx", sheet_name='Sheet2')

# 提取特征
X1 = df1[['f1', 'f2', 'f4']]
X2 = df2[['f1', 'f2', 'f4']]

# 标准化特征
scaler1 = StandardScaler()
X1_scaled = scaler1.fit_transform(X1)

scaler2 = StandardScaler()
X2_scaled = scaler2.fit_transform(X2)

# 应用 DBSCAN 聚类
dbscan1 = DBSCAN(eps=0.5, min_samples=5)
labels1 = dbscan1.fit_predict(X1_scaled)

dbscan2 = DBSCAN(eps=0.5, min_samples=5)
labels2 = dbscan2.fit_predict(X2_scaled)

# 计算每个簇的点数
unique_labels1, counts1 = np.unique(labels1, return_counts=True)
unique_labels2, counts2 = np.unique(labels2, return_counts=True)

# 找到点数最多的簇标签
```

```
max_cluster_label1 = unique_labels1[np.argmax(counts1)]
max_cluster_label2 = unique_labels2[np.argmax(counts2)]

# 过滤出最大簇的数据
filtered_data1 = X1[labels1 == max_cluster_label1]
filtered_data2 = X2[labels2 == max_cluster_label2]

# 创建一个新的图形
fig = plt.figure()

# 添加 3D 坐标轴
ax = fig.add_subplot(111, projection='3d')

# 绘制第一个数据集的最大簇
scatter1 = ax.scatter(filtered_data1['f1'], filtered_data1['f2'], filtered_data1['f4'], c='#ff2e63', label='Max Density Cluster Scope1', alpha=0.6)

# 绘制第二个数据集的最大簇
scatter2 = ax.scatter(filtered_data2['f1'], filtered_data2['f2'], filtered_data2['f4'], c='#393e46', label='Max Density Cluster Scope2', alpha=0.6)

# 设置坐标轴标签
ax.set_xlabel('f1')
ax.set_ylabel('f2')
ax.set_zlabel('f4')

# 添加图例
plt.legend()

# 显示图形
plt.show()
```

问题三 可视化.py

```
# -*- coding: utf-8 -*-

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import pandas as pd

# 读取数据
```

```
df1 = pd.read_excel("data.xlsx", sheet_name='Sheet1')
df2 = pd.read_excel("data.xlsx", sheet_name='Sheet2')

# 提取特征
X1 = df1[['f1', 'f2', 'f4']]
X2 = df2[['f1', 'f2', 'f4']]

# 创建一个新的图形
fig = plt.figure()

# 添加 3D 坐标轴
ax = fig.add_subplot(111, projection='3d')

# 绘制第一个数据集的点
scatter1 = ax.scatter(X1['f1'], X1['f2'], X1['f4'], c='#ff2e63', label='Points in the range', alpha=0.6)

# 绘制第二个数据集的点
scatter2 = ax.scatter(X2['f1'], X2['f2'], X2['f4'], c='#393e46', label='Points outside the range', alpha=0.6)

# 设置坐标轴标签
ax.set_xlabel('f1')
ax.set_ylabel('f2')
ax.set_zlabel('f4')

# 添加图例
plt.legend()

# 显示图形
plt.show()
```

问题三 密度聚类.py

```
# -*- coding: utf-8 -*-

from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import pandas as pd
```

```
# 读取数据
df1 = pd.read_excel("data.xlsx", sheet_name='Sheet1')
df2 = pd.read_excel("data.xlsx", sheet_name='Sheet2')

# 提取特征
X1 = df1[['f1', 'f2', 'f4']]
X2 = df2[['f1', 'f2', 'f4']]

# 标准化特征
# scaler = StandardScaler()
# X1_scaled = scaler.fit_transform(X1)
# X2_scaled = scaler.transform(X2)  # 使用相同的 scaler 对第二组数据进行标准化

# 应用 DBSCAN 聚类
dbscan1 = DBSCAN(eps=0.5, min_samples=10)
labels1 = dbscan1.fit_predict(X1)

dbscan2 = DBSCAN(eps=0.5, min_samples=10)
labels2 = dbscan2.fit_predict(X2)

# 创建一个新的图形
fig = plt.figure(figsize=(14, 12))

# 添加子图 1：第一个数据集的聚类前的散点图
ax1 = fig.add_subplot(221, projection='3d')
ax1.scatter(X1['f1'], X1['f2'], X1['f4'], c='gray', label='Original Data', alpha=0.6)
ax1.set_xlabel('f1')
ax1.set_ylabel('f2')
ax1.set_zlabel('f4')
ax1.set_title('Scope1 Original Data')

# 添加子图 2：第一个数据集的聚类后的散点图
ax2 = fig.add_subplot(222, projection='3d')
scatter1 = ax2.scatter(X1['f1'], X1['f2'], X1['f4'], c=labels1, cmap='viridis',
label='Clusters', alpha=0.6)
ax2.set_xlabel('f1')
ax2.set_ylabel('f2')
ax2.set_zlabel('f4')
ax2.set_title('Scope1 DBSCAN Clusters')

# 添加子图 3：第二个数据集的聚类前的散点图
```

```
ax3 = fig.add_subplot(223, projection='3d')
ax3.scatter(X2['f1'], X2['f2'], X2['f4'], c='gray', label='Original Data', alpha=0.6)
ax3.set_xlabel('f1')
ax3.set_ylabel('f2')
ax3.set_zlabel('f4')
ax3.set_title('Scope2 Original Data')

# 添加子图4：第二个数据集的聚类后的散点图
ax4 = fig.add_subplot(224, projection='3d')
scatter2 = ax4.scatter(X2['f1'], X2['f2'], X2['f4'], c=labels2, cmap='viridis', label='Clusters', alpha=0.6)
ax4.set_xlabel('f1')
ax4.set_ylabel('f2')
ax4.set_zlabel('f4')
ax4.set_title('Scope2 DBSCAN Clusters')

# 添加图例
#fig.colorbar(scatter1, ax=ax2, shrink=0.5, aspect=5)
#fig.colorbar(scatter2, ax=ax4, shrink=0.5, aspect=5)

# 显示图形
plt.tight_layout()
plt.show()
```

问题三 结果.py

```
# -*- coding: utf-8 -*-

from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 读取数据
df1 = pd.read_excel("data.xlsx", sheet_name='Sheet1')

# 提取特征
X1 = df1[['f1', 'f2', 'f4']]
```

```
# 标准化特征
scaler = StandardScaler()
X1_scaled = scaler.fit_transform(X1)

# 应用 DBSCAN 聚类
dbscan = DBSCAN(eps=0.5, min_samples=200)
labels = dbscan.fit_predict(X1_scaled)

# 计算每个簇的点数
unique_labels, counts = np.unique(labels, return_counts=True)

# 找到点数最多的簇标签
max_cluster_label = unique_labels[np.argmax(counts)]

# 过滤出最大簇的数据
filtered_data = X1[labels == max_cluster_label]

# 创建一个新的图形
fig = plt.figure()

# 添加 3D 坐标轴
ax = fig.add_subplot(111, projection='3d')

# 绘制三维散点图，使用最大簇标签作为颜色
scatter = ax.scatter(filtered_data['f1'], filtered_data['f2'], filtered_data['f4'], c='#b83b5e',
alpha=0.6)

# 设置坐标轴标签
ax.set_xlabel('f1')
ax.set_ylabel('f2')
ax.set_zlabel('f4')

# 显示图形
plt.show()
```

全部代码均符合 PEP8 规范 **(>_<)**