

中国地质大学（武汉）自动化学院 智能制造大数据技术报告-TEP 仿真实验

课 程: 智能制造大数据技术实践

学 号: 20211001578

班 级: 231216

姓 名: 王渝杰

指导老师: 胡文凯

2024年6月25日

目录

1 任务描述	1
1.1 TEP 模型介绍	1
1.2 实验要求	2
2 数据预处理和可视化仿真实验	3
2.1 数据基本统计分析	3
2.1.1 时序分析	3
2.1.2 数据正态性检验	4
2.1.3 箱线图	5
2.2 数据预处理	6
2.2.1 离群值检测	6
2.2.2 数据去噪	7
2.3 数据变化	9
2.4 主成分分析	11
3 参数预测仿真实验	13
3.1 关联关系分析	13
3.2 回归分析	15
3.2.1 预测变量和响应变量的选择	15
3.2.2 一元线性回归	15
3.2.3 多元线性回归	17
3.2.4 支持向量回归	17
3.2.5 随机森林回归	20
3.3 模型评价与比较	22
4 故障诊断仿真实验	23
4.1 数据生成	23
4.2 数据划分	23
4.3 故障诊断	23
4.3.1 数据预处理	23
4.3.2 支持向量机分类	24
4.3.3 随机森林分类	26
5 总结与感悟	28
5.1 总结	28
5.2 感悟	28
附录	30

TEP 仿真实验

1 任务描述

1.1 TEP 模型介绍

根据实际化学反应过程，美国 Eastma 公司开发了开放的化学模型模拟平台 -TEP 仿真平台。其生成数据具有时变性、强耦合、非线性等特点，被广泛应用于复杂系统的控制和故障诊断测试。TE 过程包括 5 部分，反应器、冷凝器、压缩机、分离器和解析塔，示意图如图 1 所示。

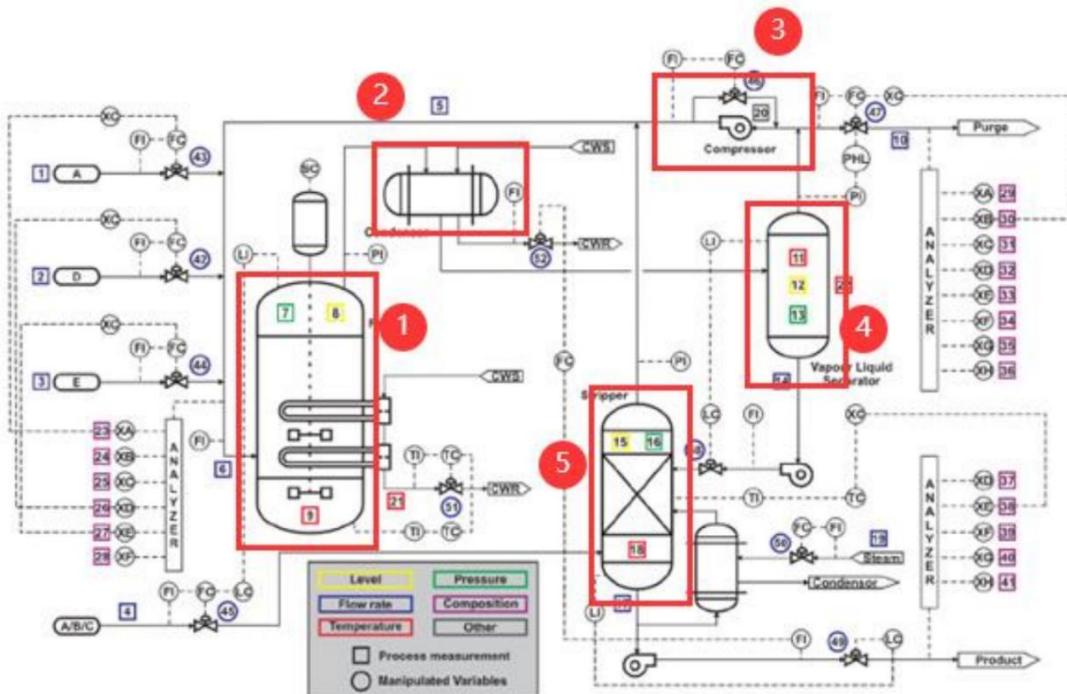
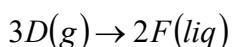
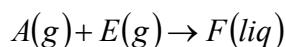
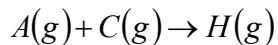
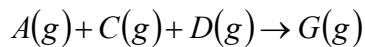


图 1 TE 过程示意图

TE 化工过程的反应起始物包括 A、C、D、E，目标生产物包括 G、H，副产物为 F。反应方程式如下：



实验中利用 Matlab 的 TEP_Model 生成不同故障的过程信号。该过程共有 11 个操纵变量 (XMV1~11)，41 个测量变量 (22 个过程变量，19 个成分变量)，

6 种运行模式和 20 中故障类型（15 种已知故障和 5 种未知故障）。故障类型如下表所示。

表 1 故障类型表

故障序号	故障原因
1-7	物料阶跃型变化
8-12	随机变换
13	慢偏移
14,15	粘住
16-20	未知故障

1.2 实验要求

(1) 任务一：数据预处理和可视化仿真实验

- ✓ 数据基本统计分析：通过统计分析和数据可视化，观测数据中存在哪些典型数据质量问题，比较变量之间的统计值差异，观察变量的分布特征。
- ✓ 数据预处理：根据所观察到的数据质量问题，采用合适的数据预处理方法，改善数据质量；比较不同方法在数据预处理中的差异，如对于一个有明显噪声的信号，可以采用不同的数据平滑方式，比较结果并绘出曲线，以及讨论参数改变对结果的影响。
- ✓ 数据变换：采用不同的归一化方式，对数据进行归一化，并绘出部分变量的信号变化曲线进行比较。
- ✓ 主成分分析：对数据进行降维处理，按贡献度阈值或者累计贡献度来筛选主成分，并对序列数据进行可视化。

(2) 任务二：参数预测仿真实验

- ✓ 关联关系分析：计算协方差 (Covariance)、相关系数 (Correlation Coefficient) 等相关性指标，用图将结果可视化；分析变量之间的相关性，找出具有较强关联关系的变量，并绘出 scatter matrix 进行辅助说明。
- ✓ 回归分析：选择某种成分变量（如 A 物料、C 物料等），根据关联关系分析，确定其相关过程测量变量；采用回归分析方法，建立其他解释变量关于该成分变量的预测模型；
- ✓ 采用合适的评价指标对模型质量进行评价，比较不同模型的差异以及模型参数对结果的影响。

(3) 任务三：故障诊断仿真实验

- ✓ 数据生成：设定不同故障类型（不少于 5 个），获得带有故障标签的数据集。

- ✓ 数据划分：对数据进行划分，包括训练集和测试集，在训练集上，训练分类器；在测试集上，测试分析已训练好的分类器的性能。
- ✓ 故障诊断：采用合适的分类器训练故障诊断模型。模型评估与选择：考虑交叉验证等评估方法，对故障诊断模型进行性能度量。

2 数据预处理和可视化仿真实验

本小节内容为预测模型建立之前的步骤，包括分析数据的统计分析，数据预处理，数据归一化以及主成分分析。

2.1 数据基本统计分析

2.1.1 时序分析

实验中共有 41 个测量变量，其中包括 22 个过程变量和 19 个成分变量。首先分别绘制过程变量和成分变量的时序图，观察各个变量的数据属于平稳型还是非平稳性，便于后续去噪方法的选择。



图 2 测量变量时序图

根据对图 2 的观察，将 41 个测量变量的数据分为平稳型和非平稳型两类，如表 2 所示。

表 2 数据类型划分结果表

平稳型	2, 3, 4, 5, 6, 8, 9, 12, 14, 15, 17, 19, 21, 22, 24, 25, 26, 27, 28, 30, 31, 34, 36, 37, 39, 41
非平稳型	1, 7, 10, 11, 13, 16, 18, 20, 23, 29, 32, 33, 35, 38, 40

2.1.2 数据正态性检验

本实验最终是要建立过程变量对成分变量的预测模型，很多机器学习预测算法要求特征的数据服从正态分布，因此在预处理环节需要进行数据正态性检验。本小节采用直方图和 Q-Q 图两种方式进行综合分析。

直方图通过将数据分组成连续的区间并展示每个区间的频数或频率，可以直观看到数据的分布形状。对于大型数据集，直方图提供了一种快速的方法来初步判断数据是否接近正态分布。Q-Q 图将样本数据的分位数与正态分布的理论分位数进行比较，如果数据点在 Q-Q 图上接近于一条直线，则数据接近于正态分布。

直方图和 Q-Q 图相结合，直方图通过展示数据的频率分布并与理论正态分布曲线进行比较，初步判断数据的正态性。QQ 图则通过比较样本分位数和理论分位数，提供更精确的正态性检验。综合使用这两种方法，可以更全面地理解数据的分布特征。41 个变量数据的直方图和 Q-Q 图分别如图 3 和图 4 所示。

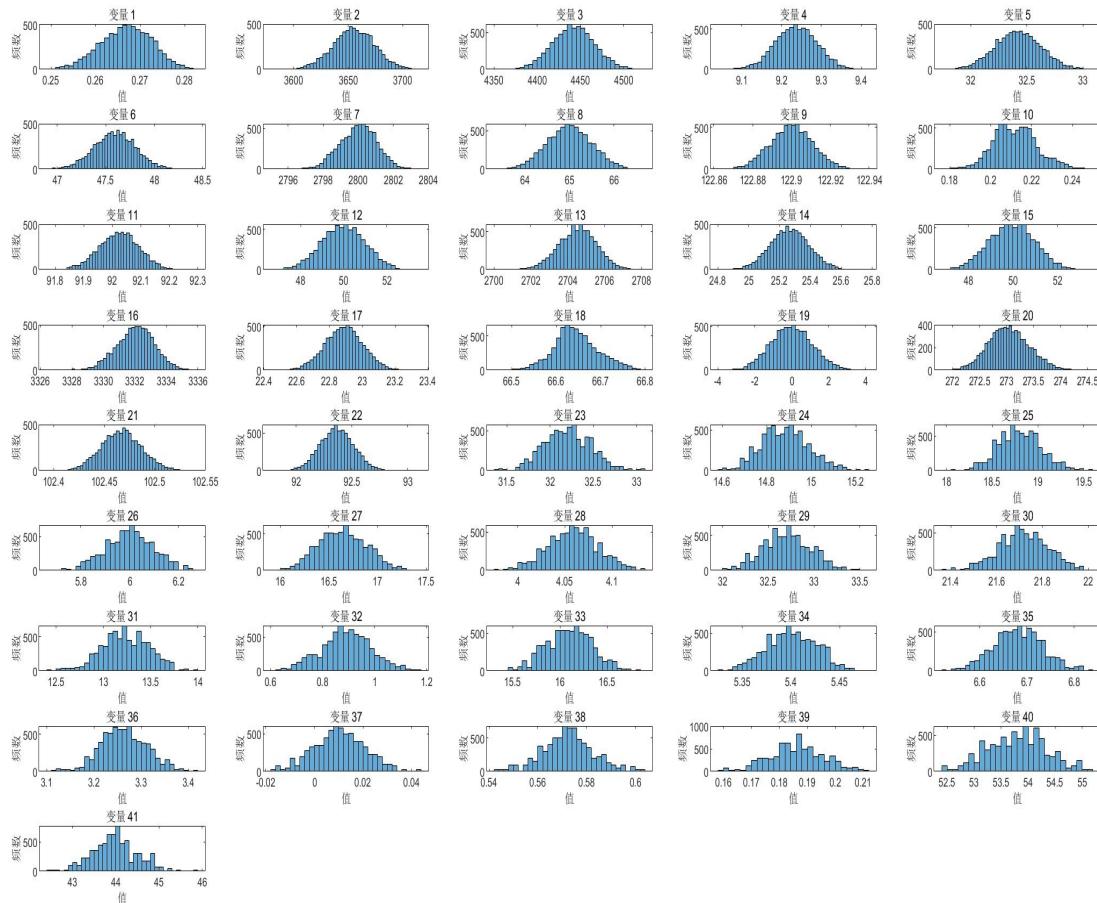


图 3 变量直方图

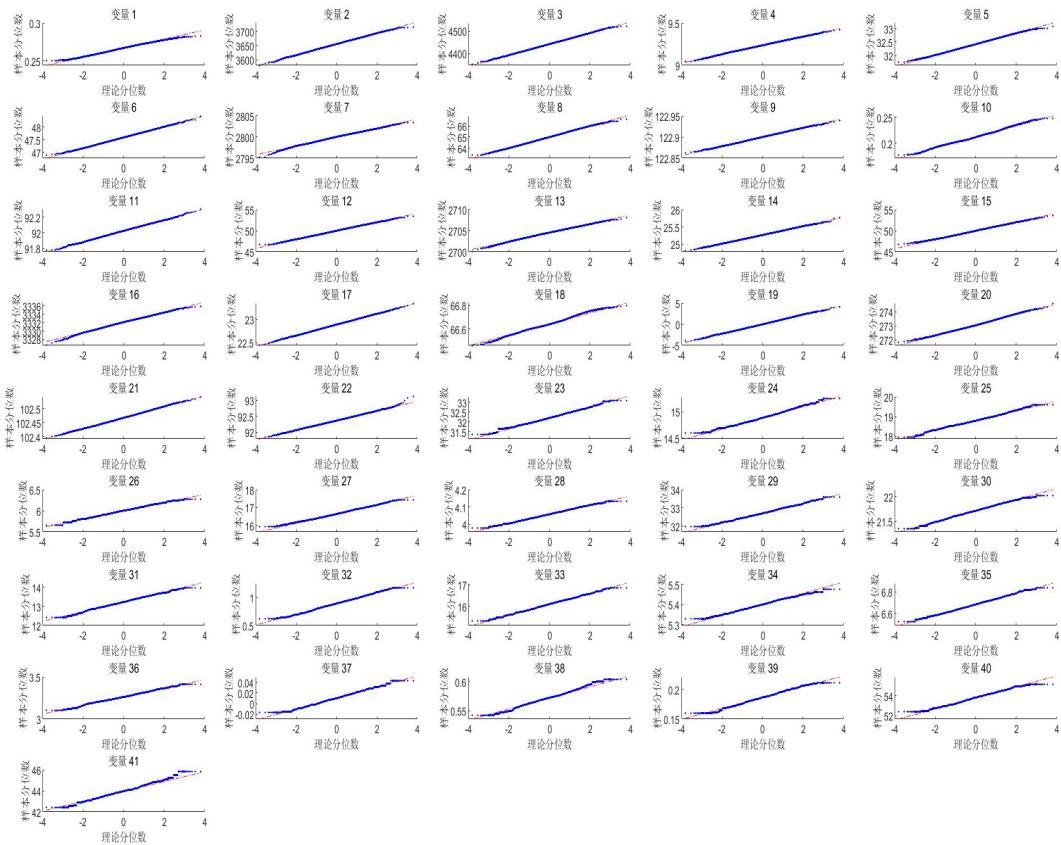


图 4 变量 Q-Q 图

综合观察图 3 和图 4，可以发现直方图中 41 个变量基本呈现中间向两边递减的趋势，Q-Q 图中 41 个变量均呈现出一条直线。两种图形都表明数据接近正态分布，结论更有说服力。

2.1.3 箱线图

在数据统计分析环节进行箱线图分析是非常重要的，因为箱线图能够提供数据集的直观概览，帮助我们快速识别和理解数据的特征及其分布情况。具体而言，箱线图分析具有以下几个优势：

- (1) **检测离群点：**箱线图能够有效地显示数据中的异常值或离群点，这些值可能需要进一步处理以避免影响后续的分析和建模。
- (2) **理解数据分布：**通过箱线图可以清晰地看到数据的分布情况，包括中位数、四分位数范围 (IQR)，以及数据的偏态和变异程度。
- (3) **比较数据集：**箱线图可以用于比较不同数据集或同一数据集中不同子集的分布情况，帮助我们发现不同组之间的差异和趋势。
- (4) **简化数据呈现：**箱线图以简单的图形方式概括了数据的五个统计量（最小值、第一四分位数、中位数、第三四分位数和最大值），使得数据的概览更加简洁明了。

41 个变量的箱线图如图 5 所示：

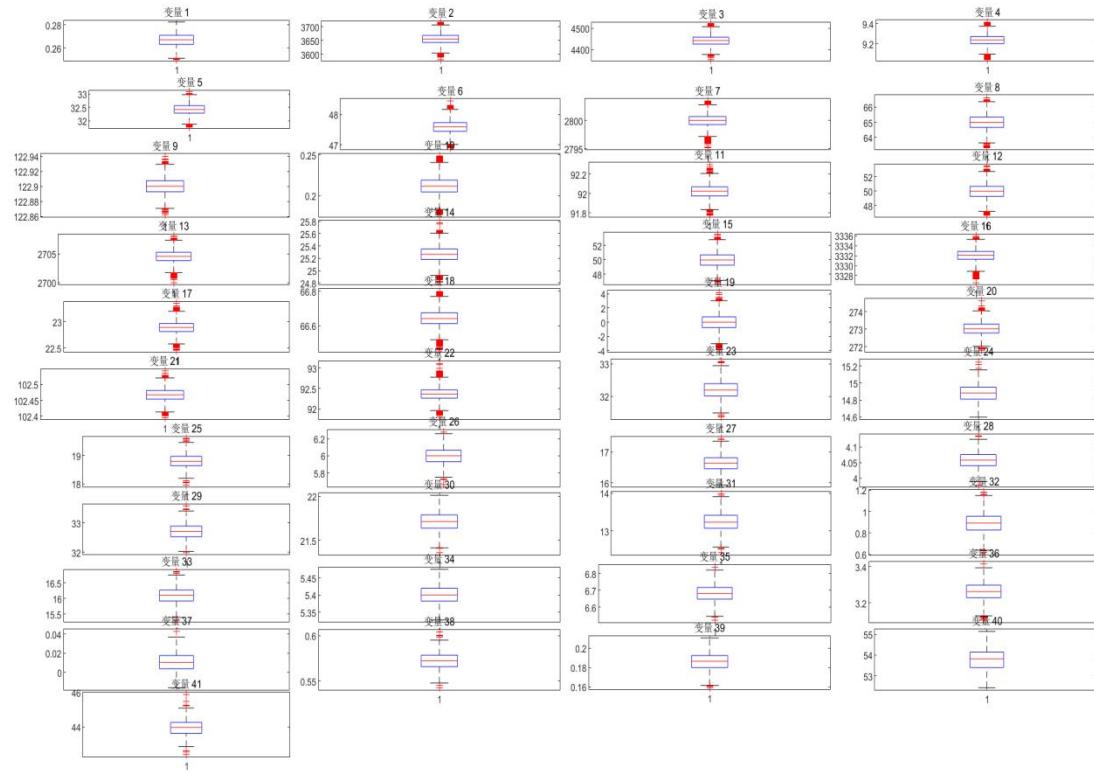


图 5 变量箱线图

根据图 5，我们可以观察各个变量的分布情况，并找出明显的离群点，进行初步的清理，以提高数据质量。

2.2 数据预处理

2.2.1 离群值检测

在原始数据集中，经常会出现一些离群值，这部分离群值的存在会影响后续预测模型的准确率，因此需要在数据预处理环节找出明显的离群值并出去。本小节首先以 A 物料的流量数据为例，分别使用箱线图、3-sigma 检测法、z-score 检测法进行离群值检测，对比分析。

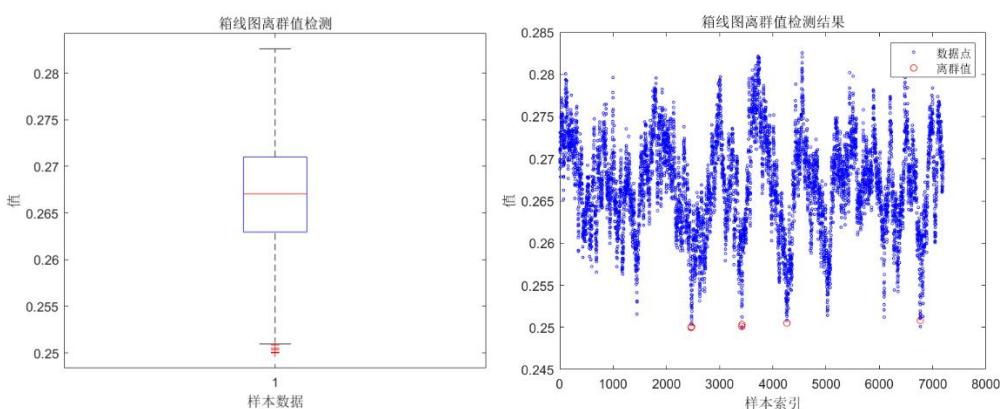


图 6 A 物料流量箱线图检测离群值

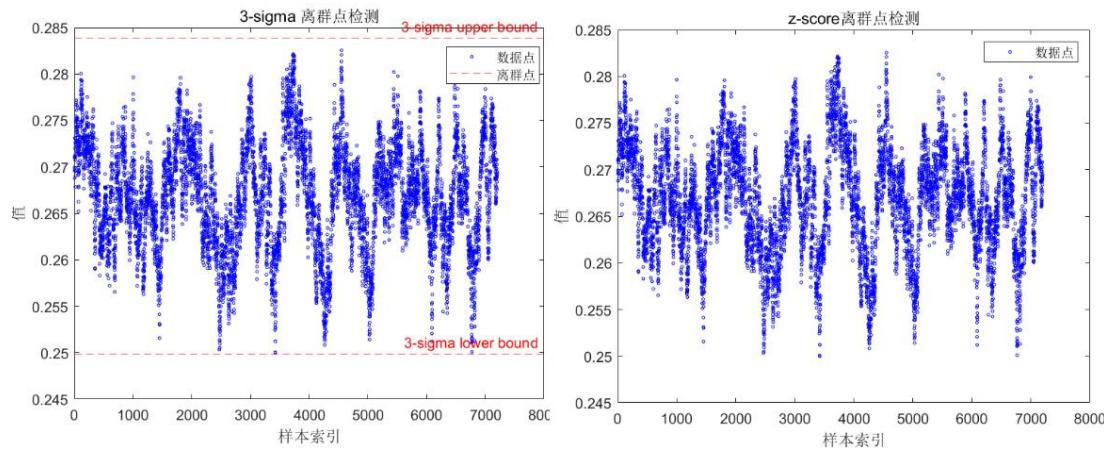


图 7 3-sigma 和 z_score 法检测离群点结果

对于 A 物料流量数据，三种离群点检测方法结果的可视化图如图 6-7 所示，整理后离群点数据如表 3 所示。

表 3 针对 A 物料流量的三种离群点检测方法对比表

方法	离群点
箱线图	0.2500、0.2501、0.2501、0.2503、0.2505、0.2508
3-sigma	无
z-score	无

经过对比分析，可以发现三种方法的结果较为相似，但箱线图的检测结果更加精细，因此本小节选择使用箱线图的方法对剩余变量的数据进行离群值检测。41 个测量变量的箱线图如下所示。

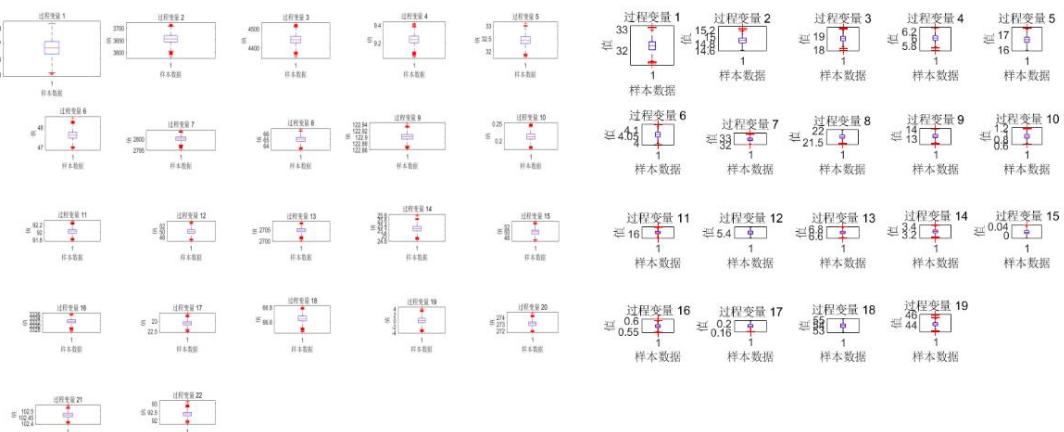


图 8 变量箱线图

2.2.2 数据去噪

在数据预处理环节进行数据去噪是至关重要的，因为原始数据往往包含各种形式的噪声，如异常值、缺失值、重复数据和测量误差。这些噪声会对后续的数据分析和建模过程产生负面影响，干扰模型的训练过程，导致模型的预测精度下降。通过去除噪声，可以提高模型的泛化能力和预测性能。

本文采用了小波去噪和滑动窗口去噪两种方法，这是两种常用的数据去噪技术，各有优点和适用场景。

(1) 小波去噪

小波去噪利用小波变换将信号分解到不同的尺度上，通过处理这些尺度上的系数来去除噪声。具体步骤包括：

① 小波分解：将信号分解成不同尺度的小波系数。

② 阈值处理：对高频小波系数（通常代表噪声成分）应用阈值处理（硬阈值或软阈值）。

③ 小波重构：将处理后的小波系数重构为去噪后的信号。

(2) 滑动窗口去噪

滑动窗口去噪通过将信号划分为一系列固定大小的窗口，窗口内的数据点进行平滑处理，常用的方法有移动平均、加权移动平均等。

(3) 方法对比

上述两种方法各有优缺点，一般来说，小波去噪适合突变型数据，滑动窗口去噪适合相对平稳型数据。

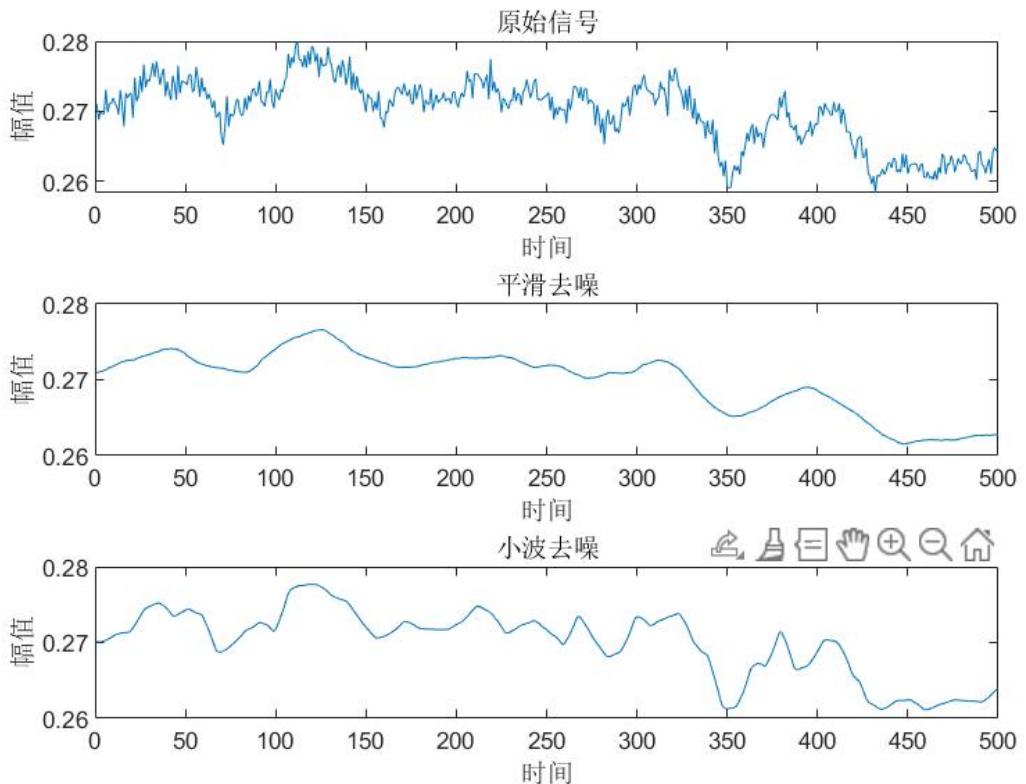


图 9 突变型数据去噪效果对比

以过程变量 1 为例，前 500 条数据属于突变型，分别使用两种方法进行去噪，结果如图 9 所示，可见小波去噪可以保留信号的尖峰，能够更好地保留信号细节，滑动窗口去噪可能会丢失部分细节。

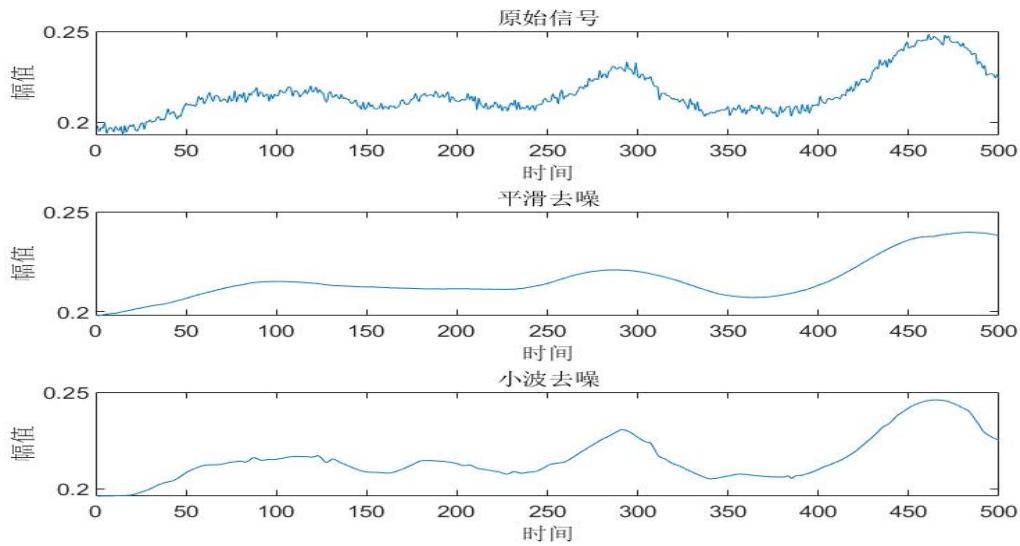


图 10 平稳性数据去噪效果对比

以过程变量 10 为例, 前 500 条数据相对属于平稳型, 分别使用两种方法进行去噪, 结果如图 10 所示, 可见滑动窗口去噪的效果优于小波去噪。

因此, 依据表 2 归纳的数据类型对突变型使用小波去噪法, 对相对平稳型使用滑动窗口去噪法分别进行去噪。去噪后的 41 个变量的时序图如图 11 所示。

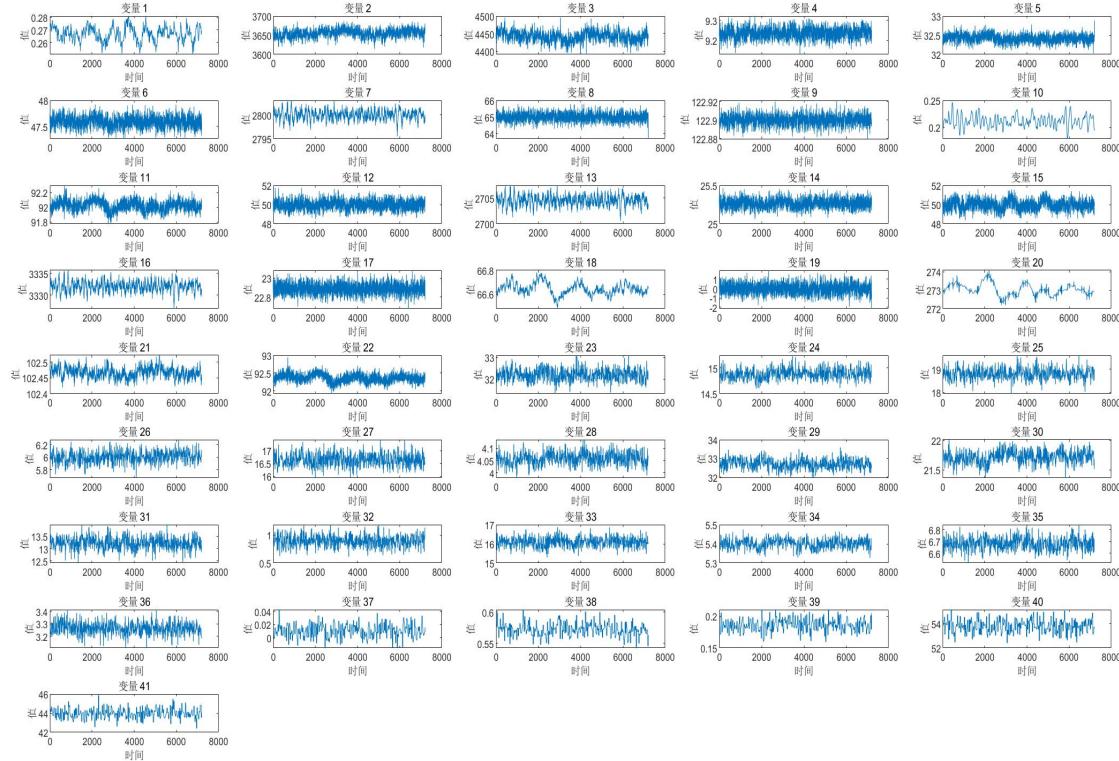


图 11 去噪后变量时序图

2.3 数据变化

通过对时序图的观察, 我们可以发现不同变量的量级有很大差异, 比如变量 1 的数值均为小数, 而变量 2 的数值则上千。当量级差异较大时, 量级较大的变

量会对模型产生主导作用，导致模型无法正确学习其他特征的信息。

数据归一化是数据预处理中的一个重要步骤，主要目的就是将不同量级的数据转换到相同的尺度上，从而提高模型的训练效果和预测精度。常用的有 z 分数归一化和最大最小归一化。

(1) z 分数归一化

z 分数归一化是通过将数据转换为均值为 0、标准差为 1 的标准正态分布，公式为：

$$z = \frac{x - \mu}{\sigma}$$

其中， z 表示原始数据， μ 表示数据的均值， σ 表示数据的标准差。z 分数归一化能保留数据的分布形态，不会改变数据的相对关系，适用于数据服从正态分布或近似正态分布的场景。

(2) 最大最小归一化

最大最小归一化将数据缩放到[0, 1]或[-1, 1]区间，公式为：

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

其中， x' 表示原始数据， x_{\max} 表示数据的最大值， x_{\min} 表示数据的最小值。该方法适用于数据范围已知且固定的场景。

(3) 方法对比

以 A 物料流量数据为例，分别使用 z 分数归一化和最大最小归一化，结果如图 12 所示。

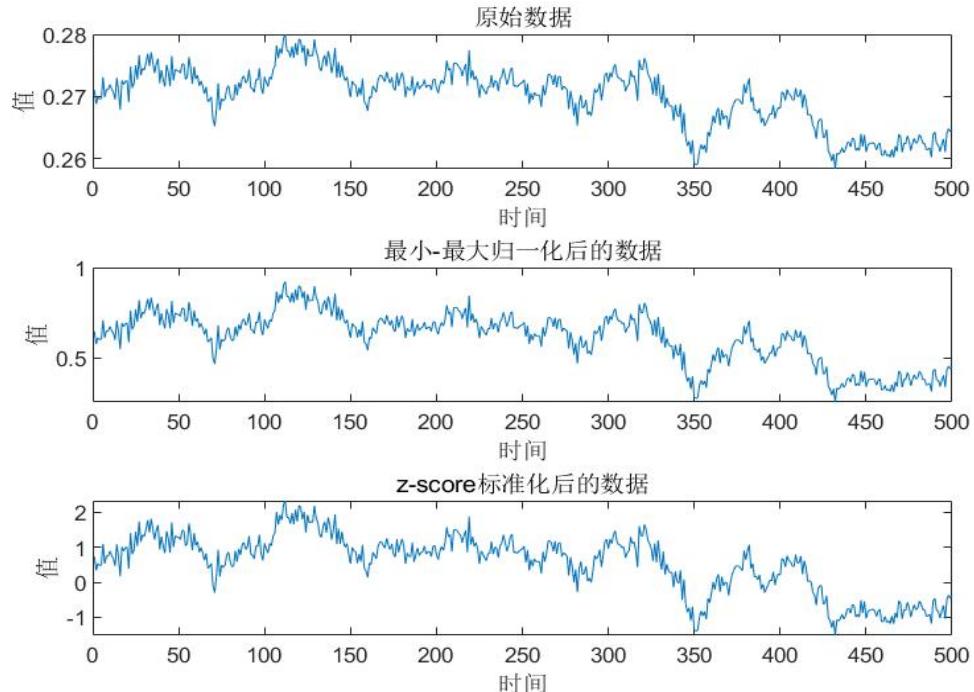


图 12 归一化方法对比

由图 12 可见，最大最小归一化将原始数据缩放到 0-1 之间，z 分数归一化将原始数据变换为均值为 0，标准差为 1。但两种方法都不会改变数据的形态。

在 2.1.2 小节中，我们对各个变量进行了正态性检验，得出均可视为正态分布的结论，并在 2.2.2 小节进行了数据去噪。综合这两点，实验中的数据更适合 z 分数归一化。

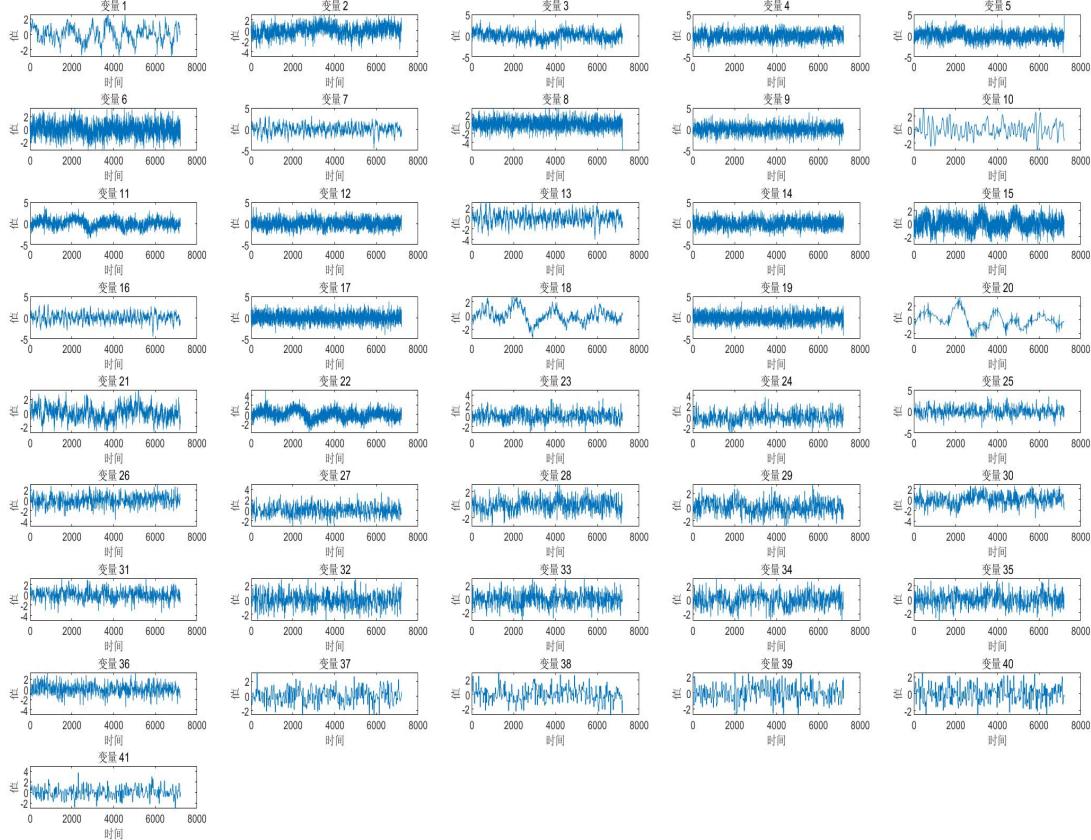


图 13 z 分数归一化后的时序图

z 分数归一化后的时序图如上图所示，可见所有变量的数据均变换为均值为 0，标准差为 1，便于后续预测模型的建立。

2.4 主成分分析

主成分分析（Principal Component Analysis, PCA）是一种统计方法，用于简化数据集，同时保留尽可能多的信息。它通过线性变换将原始数据投影到新的坐标系中，使得新坐标系中的变量（主成分）是原始变量的线性组合，并且这些主成分之间是不相关的。PCA 的主要目的是降维和数据压缩，同时最大化保留数据的方差。

PCA 的步骤如下：

- (1) 标准化：为了使得每个变量具有相同的重要性，需要对数据进行标准化处理。标准化通常包括减去均值并除以标准差，使得每个变量的均值为 0，标准差

为 1。

(2) 计算协方差矩阵

协方差矩阵描述了数据集中每对变量之间的线性关系。

(3) 特征值分解

对协方差矩阵进行特征值分解，得到特征值和特征向量。特征值表示数据在特征向量方向上的方差大小，特征向量则是新坐标系的轴。

(4) 选择主成分

根据特征值的大小选择主要的特征向量（即主成分）。通常选择前 k 个特征值对应的特征向量，保留最多的信息量。

(5) 转换数据

将原始数据投影到选定的特征向量上，得到降维后的数据。新数据集中的每一列是一个主成分。

通过主成分分析找到累计贡献度达到 80% 的主成分，一共筛选出 25 个主成分，时序图和贡献度排序图如图 14 和 15 所示。

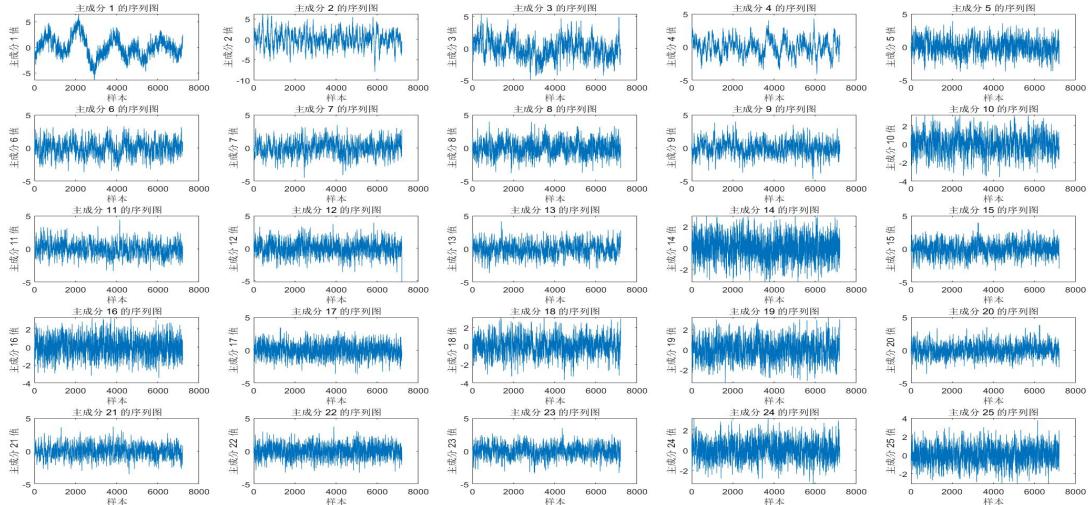


图 14 主成分时序图

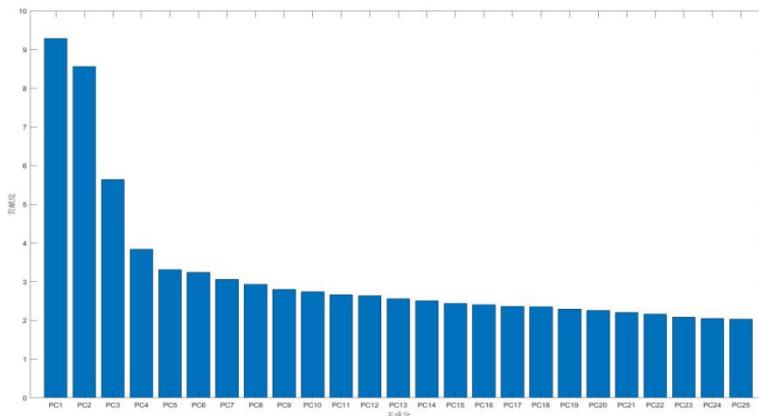


图 15 主成分贡献度排序图

3 参数预测仿真实验

在数据预处理和可视化环节我们对 41 个变量（包括 22 个过程变量和 19 个成分变量）的原始数据进行一系列处理，提高了数据质量，并将其变换为适合模型建立的形式。接下来将建立过程变量对成分变量的回归预测模型，首先通过关联分析筛选出与目标成分变量相关性相对较强的过程变量，将其作为预测变量分别使用一元线性回归、多元线性回归、支持向量回归以及随机森林回归的方法进行预测，并对不同方法的效果进行对比分析。

3.1 关联关系分析

实验中共有 22 个过程变量和 19 个成分变量，本小节采用皮尔逊相关系数来分析变量之间的相关性，根据分析结果确定预测模型的预测变量和响应变量。

皮尔逊相关系数（Pearson Correlation Coefficient）是用来衡量两个变量之间线性关系的强度和方向的统计指标，其值介于 -1 和 1 之间。对于地质钻探场景有以下适用性和优点：

- (1) 线性关系的测量：皮尔逊相关系数专注于测量变量之间的线性关系，对于钻探过程中常见的一些线性相关特性（如钻压与井深、转速与扭矩等）非常有效。
- (2) 数据要求：皮尔逊相关系数对数据的要求不高，只需要成对的样本数据。此外，它假设数据服从正态分布，虽然不是严格要求，但满足该假设时结果更可靠。
- (3) 处理多变量：钻探过程中涉及多个变量，皮尔逊相关系数可以方便地计算这些变量之间的两两相关性，从而识别出关键影响因素。

皮尔逊相关系数的计算公式如下：

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

其中， x_i 和 y_i 表示两个变量的样本值， \bar{x} 和 \bar{y} 表示两个变量的样本均值。相关性强弱的划分如下表：

表 4 相关性强弱划分

相关性系数	0-0.2	0.2-0.4	0.4-0.6	0.6-0.8	0.8-1.0
相关性强弱	极弱	弱	中等	强	极强

经过 MATLAB 计算，6 个指标相互之间的相关性系数热力图如图 16 所示。

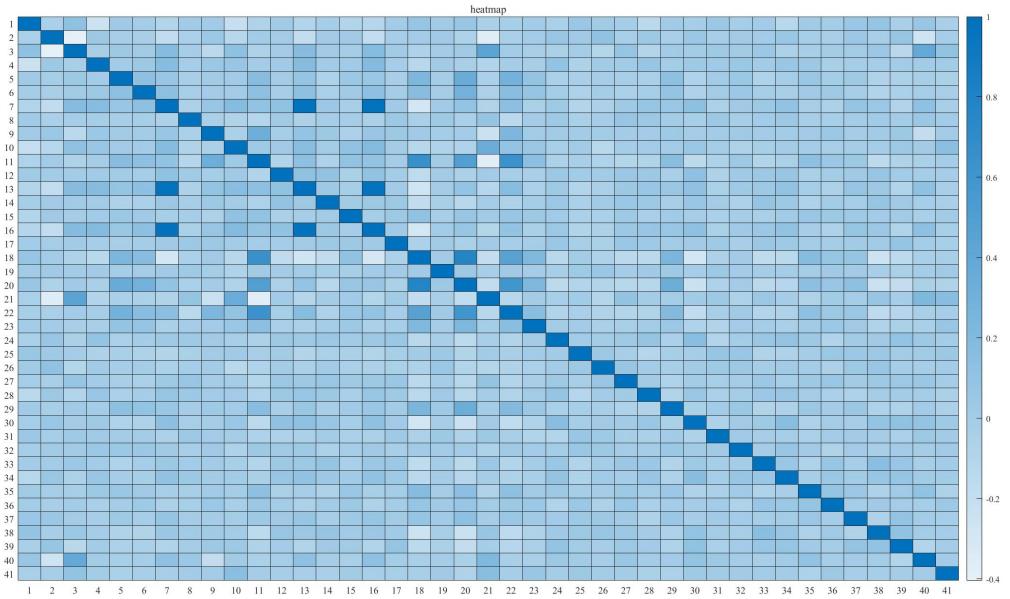


图 16 相关系数热力图

从图 16 中可找出具有较强相关性的变量对，如下表所示。

表 5 较强相关的变量对

变量对	相关系数
变量 13 与变量 7	0.97
变量 16 与变量 7	0.98
变量 16 与变量 13	0.98

对于表 5 中的变量对，接下来绘出 scatter matrix 进行辅助说明，如图 17 所示。从图中可看出三个变量之间的散点图都趋于直线，进一步说明了彼此间的强相关性。

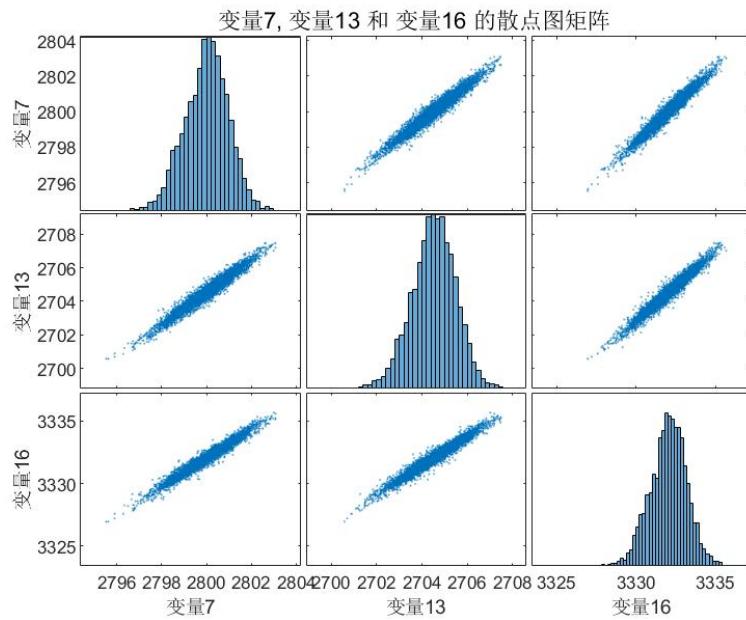


图 17 强相关变量间的散点图矩阵

3.2 回归分析

3.2.1 预测变量和响应变量的选择

本文选择成分变量 7 (数据集中第 29 列) 作为回归预测模型的响应变量，然后根据 3.1 小节的相关性分析选择预测变量，但 simulink 模型生成的数据是模拟化工现场的，过程变量与成分变量的相关系数较低，与成分变量 7 的相关性系数大于 0.1 的过程变量共有 5 个，如表 6 所示。

表 6 与响应变量的相关系数

变量	相关系数
过程变量 5	0.138155555
过程变量 11	0.156479439
过程变量 18	0.237745138
过程变量 20	0.328226148
过程变量 22	0.195666275

对于一元回归模型，将选择过程变量 20 作为预测变量，对于多元回归模型，则选择以上 5 个变量作为预测变量。

3.2.2 一元线性回归

一元线性回归 (Simple Linear Regression) 是一种统计方法，用于分析两个变量之间的关系。其目的是通过一条直线 (回归线) 来描述一个自变量 X 和一个因变量 Y 之间的线性关系。回归线的方程通常表示为：

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

其中，Y 和 X 分别为因变量和自变量， β_0 表示截距， β_1 表示斜率， ε 表示误差项。

建立一元线性回归模型的步骤如下：

(1) 划分训练集和测试集

将预处理后的数据以 7:3 划分为训练集和测试集，训练集用于拟合模型，测试机用于模型评估。

(2) 拟合模型

拟合一元线性回归模型主要是估计回归系数 β_0 和 β_1 ，常用的方法是最小二乘法，目的是找到使得所有观测值的残差平方和最小的回归系数。

公式如下：

$$\beta_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2}$$

$$\beta_0 = \bar{Y} - \beta_1 \bar{X}$$

(3) 模型预测

模型拟合好后，使用该模型对测试集进行预测，然后在模型评估中分析预测效果。

(4) 模型评估

本次实验中选择 R 方、均方误差 (MSE)、均方根误差 (RMSE) 以及平均绝对误差 (MAE) 对模型的拟合效果和预测效果进行分析。

训练集和测试集的各评价指标值如下表所示。

表 7 单元线性回归各评价指标值

评价指标	训练集	测试集
R 方	0.13689	0.035224
MSE	0.87962	0.92355
RMSE	0.93788	0.96102
MAE	0.75453	0.76514

训练集和测试集的拟合效果图，实际值与预测值的对比图分别如图 18 和 19 所示。

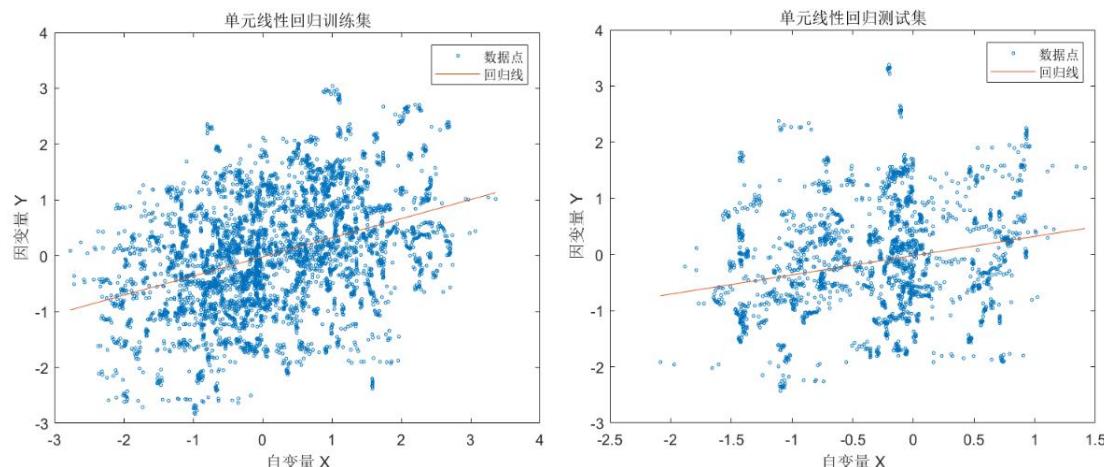


图 18 单元线性回归拟合效果图

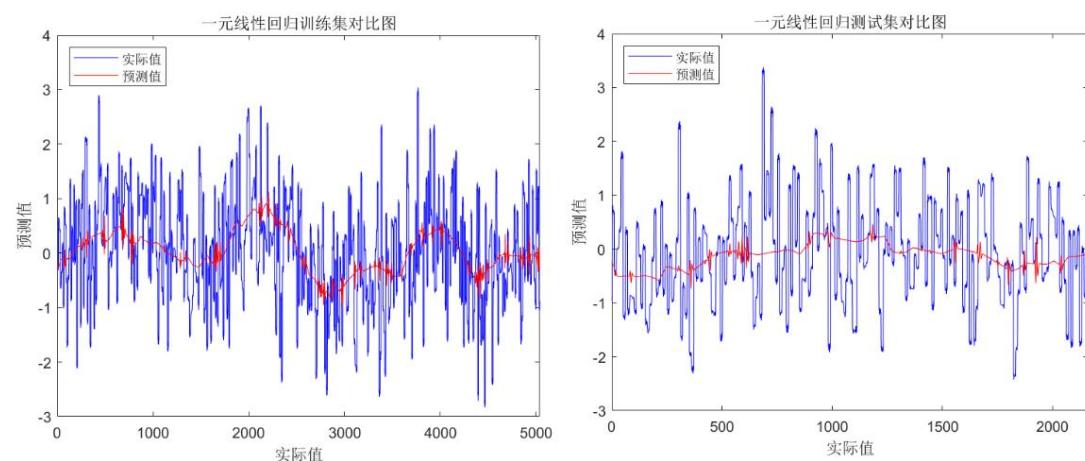


图 19 单元线性回归实际值与预测值对比图

根据表 7、图 18 和 19 的结果，可以发现一元线性回归无论是拟合效果还是预测结果都表现不佳，应该考虑使用别的预测模型。

3.2.3 多元线性回归

一元线性回归只能处理一个自变量，往往无法捕捉到多个因素共同作用下的复杂关系，且本实验中预测变量与响应变量间的相关性很低，因此预测效果很难达到预期。相对于一元线性回归，多元线性回归能够同时考虑多个自变量对因变量的影响，可以处理更多复杂的数据结构和关系，这使得模型更全面、更精确、适应性更强。

多元线性回归模型的标准形式可以表示为：

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \cdots \beta_p x_p + \varepsilon$$

多元线性回归的步骤和一元线性回归的步骤基本相同，此处不再赘述。训练集和测试集的各评价指标值如下表所示。

表 8 多元线性回归各评价指标值

评价指标	训练集	测试集
R 方	0.13775	0.033874
MSE	0.87874	0.92584
RMSE	0.93741	0.9622

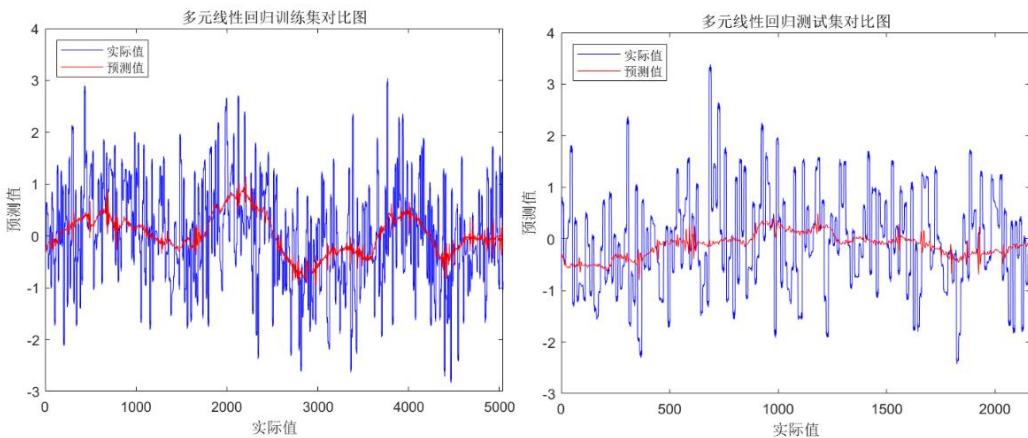


图 20 多元线性回归实际值与预测值对比图

3.2.4 支持向量回归

支持向量回归（Support Vector Regression）是一种基于支持向量机的回归方法，使用核函数来处理非线性回归问题。它通过映射数据到高维空间，在高维空间中寻找最佳拟合曲线。下面是对 SVR 的一些关键概念和实现步骤的介绍。

(1) 核函数

核函数是支持向量机中常用的核函数，定义如下：

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

SVR 的目标函数由两个部分组成：损失函数和正则化项。目标是找到最佳的 w 和 b 使得回归误差最小化，同时保持模型的复杂度尽量小。

$$\min \frac{1}{2} \|w\|^2 + C \sum (\xi_i + \xi_i^*)$$

其中， w 是模型的权重向量。 b 是偏置项。 ξ_i 和 ξ_i^* 是松弛变量，表示允许的误差。 C 是惩罚参数，控制模型复杂度和误差的权衡。

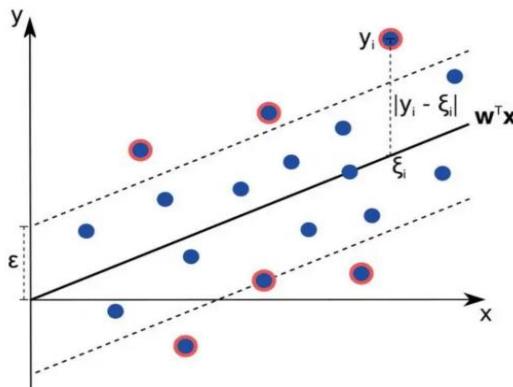


图 21 SVR 示意图

在化学反应过程中，预测成分变量是一个重要的任务。SVR 在这一场景中的应用契合度高，主要体现在以下几个方面：

- (1) **非线性关系处理能力：** 化学反应过程中的成分变量往往具有复杂的非线性关系。SVR 特别适合这种场景，因为它可以通过核方法（如 RBF 核）将数据映射到高维特征空间，在该空间中处理原本非线性的关系，从而实现更准确的预测。
- (2) **鲁棒性：** SVR 对异常值具有一定的鲁棒性，不敏感损失函数允许在一定范围内忽略误差，不会因为少数异常值而对模型造成重大影响。这在实际化学反应中非常重要，因为实验数据中常常存在一些异常值。
- (3) **高维数据处理：** 化学反应过程中可能涉及多个变量，导致数据维度较高。SVR 在处理高维数据方面表现优异，因为它的优化目标和约束条件只依赖于支持向量的数量，而不是特征的数量，这使得 SVR 在高维空间中仍能高效地进行计算。
- (4) **灵活性：** SVR 可以通过选择不同的核函数适应不同的数据分布和关系模型，具有很高的灵活性。

训练中，本文选择成分变量 7 作为响应变量，以 3.2.1 小节分析选择的 5 个预测变量建立成分变量预测模型，在原始数据集中选取 80% 作为训练集。训练中使用的超参数优化方法是网格搜索结合十折交叉验证。

- (1) 十折交叉验证

交叉验证是一种评估模型性能的技术，通过将数据集划分为多个子集，分别用于训练和验证，以获得模型在不同数据集上的性能表现。十折交叉验证的步骤如下：

- ① 将数据集划分为 10 个互斥的子集。
- ② 每次使用其中 9 个子集进行训练，剩下的 1 个子集进行验证。重复这一过程 10 次，每个子集都被用作一次验证集。
- ③ 计算每次验证的 R 方值，最后取 10 次验证的平均 R 方值作为该超参数组合的性能指标。

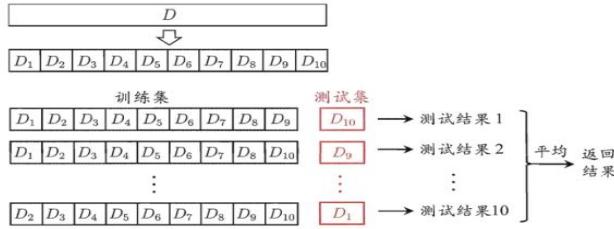


图 22 十折交叉验证示意图

(2) 网格搜索

网格搜索是一种系统地遍历预定义的超参数集合的方法，以选择出使模型表现最优的超参数组合。对于每个超参数组合，都会训练模型并评估其性能，最终选择最佳的超参数组合。网格搜索的步骤如下：

- ① 定义超参数范围，包括 C 的取值范围、 ε 的取值范围和核尺度（Kernel Scale）的取值范围。
- ② 遍历所有可能的超参数组合。
- ③ 对每个超参数组合，进行十折交叉验证，计算模型的平均 R 方指标。

网格搜索得出的超参数为：

表 9 SVR 超参数

超参数	取值
C	0.1
gamma	scale
kernel	rbf

训练集和测试集的各评价指标值如下表所示。

表 10 SVR 各评价指标值

评价指标	训练集	测试集
R 方	0.12410	0.12060
MSE	0.86156	0.88014
RMSE	0.92820	0.93816

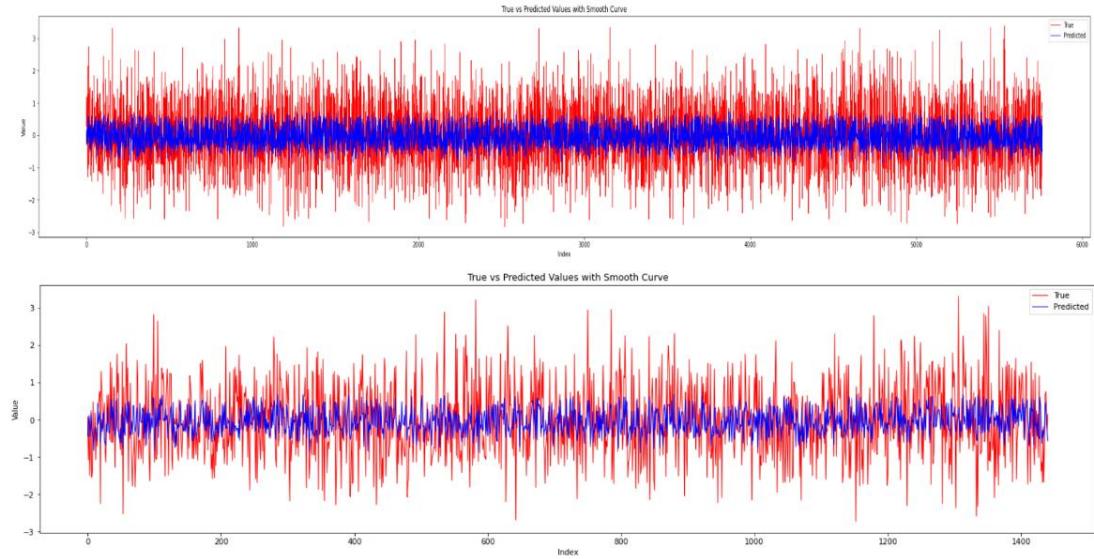


图 23 SVR 实际值与预测值对比图

观察表 10 和图 23 可以发现，SVR 在测试集上的预测效果优于一元线性和多元线性回归，体现出了它非线性的特点。

3.2.5 随机森林回归

随机森林回归（Random Forest Regression）是一种集成学习方法，通过构建多个决策树并结合其结果来进行预测，从而提高模型的准确性和稳定性。随机森林回归在处理非线性、复杂和高维数据时表现出色，广泛应用于各种回归问题。

随机森林回归的基本构件是决策树。决策树是一个递归的分割过程，将数据集分成更小的子集，同时在每个子集上建立预测模型。虽然单个决策树容易过拟合，但其集成方法可以缓解这一问题。随机森林通过构建多个决策树并将其结果结合起来进行预测的。每棵树的预测结果通过平均来决定最终的结果。

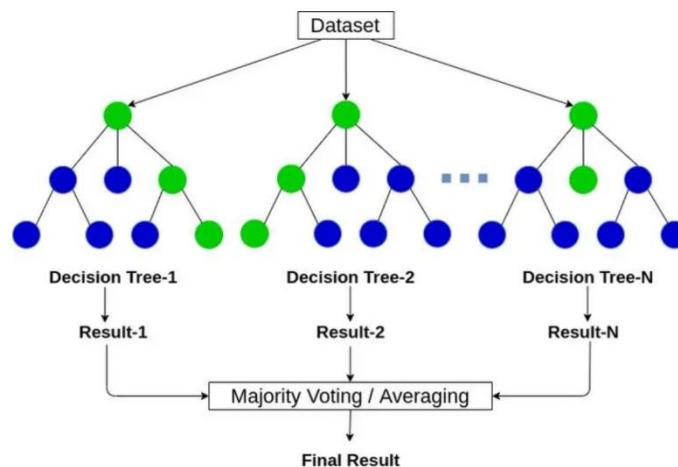


图 24 随机森林回归算法示意图

化工过程通常涉及多个变量（如温度、压力、反应时间、原料浓度等），这

些变量之间可能存在复杂的非线性相互作用。随机森林算法在化工过程中成分变量的预测中具有多项优势和高度契合性，主要体现在以下几个方面：

- (1) **处理非线性关系**: 化工过程中的成分变量通常具有复杂的非线性关系。随机森林通过集成多个决策树，可以很好地捕捉这些复杂关系，提供准确的预测结果
- (2) **高泛化能力**: 由于随机森林使用了多棵决策树的集成方法，并且每棵树都在不同的随机子集上进行训练，这减少了单个模型过拟合的风险，提高了整体模型的泛化能力。
- (3) **鲁棒性和抗噪性**: 化工数据通常会包含噪声和异常值，随机森林对这些噪声数据有较高的鲁棒性。即使存在一些异常值，也不会显著影响整体模型的预测性能。

模型训练和 SVR 一样，选择成分变量 7 作为响应变量，以 3.2.1 小节分析选择的 5 个预测变量建立成分变量预测模型，在原始数据集中选取 80% 作为训练集，20% 作为测试集。使用的超参数优化方法是网格搜索结合十折交叉验证。网格搜索结合十折交叉验证在上一小节中已介绍，此处不再赘述。

训练结束后，网格搜索得出的超参数为：

表 11 RF 超参数

超参数	取值
max_depth	52
min_samples_split	2
n_estimators	73

训练集和测试集的各评价指标值如下表所示。

表 12 RF 各评价指标值

评价指标	训练集	测试集
R 方	0.91073	0.40093
MSE	0.58692	0.08970
RMSE	0.76611	0.29951

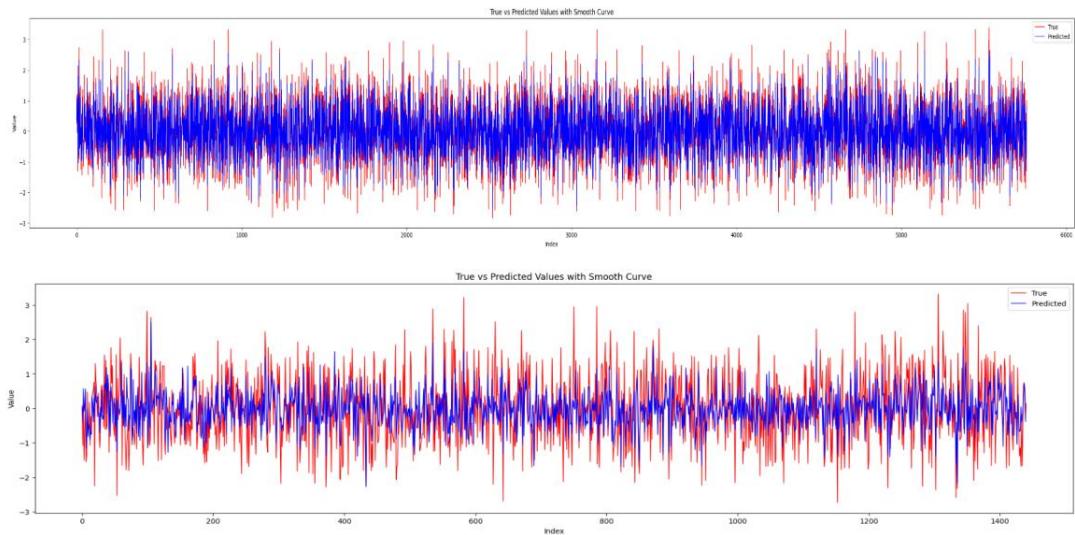


图 25 RF 实际值与预测值对比图

从表 12 和图 25 中可以看出随机森林算法的预测效果要明显好于前三种，尤其是 R 方值，达到了 0.4，拟合度大大提高。

3.3 模型评价与比较

在 3.2 节中，分别使用了一元线性回归、多元线性回归、支持向量回归和随机森林回归算法建立了成分变量预测模型。并使用 R 方、MSE 和 RMSE 指标对模型进行评价。不同模型在测试集上的表现如下表所示：

表 13 不同模型在测试集上的表现

	R 方	MSE	RMSE
一元线性	0.03522	0.92355	0.96102
多元线性	0.03387	0.92584	0.96220
支持向量	0.12060	0.88014	0.93816
随机森林	0.40093	0.08970	0.29951

分析表 13，可以得出如下结论：

- (1) 两种线性预测模型的效果都不太好，与化工过程的工艺特性有关，化工过程的变量间可能是非线性关系，线性模型不能准确描述过程变量和成分变量之间的关系。
- (2) 支持向量回归模型属于非线性模型，预测效果比前两种有明显提升，尤其是 R 方指标，拟合度更好。
- (3) 随机森林回归模型的表现最佳，在 R 方、MSE 和 RMSE 三个指标上均体现出大幅度的优势，拟合效果和预测效果不错。
- (4) 由于笔记本电脑的性能问题，实验中网格搜索的范围很有限，因此 SVR 和 RF 模型还可以在高性能计算机上进一步优化，得到效果更好的预测模型。

4 故障诊断仿真实验

化工过程涉及大量的化学物质和复杂的反应系统，故障的发生可能导致设备损坏、生产中断甚至危及人员安全。通过建立故障分类模型，可以及时识别和预测潜在故障，采取预防措施，显著提高生产过程的安全性。本节将采用合适的分类器训练故障诊断模型，考虑交叉验证等评估方法，对故障诊断模型进行性能度量。

4.1 数据生成

本次实验中在 simulink 模型中设定 7 种不同故障类型，获得带有故障标签的数据集，共 30715 条数据。

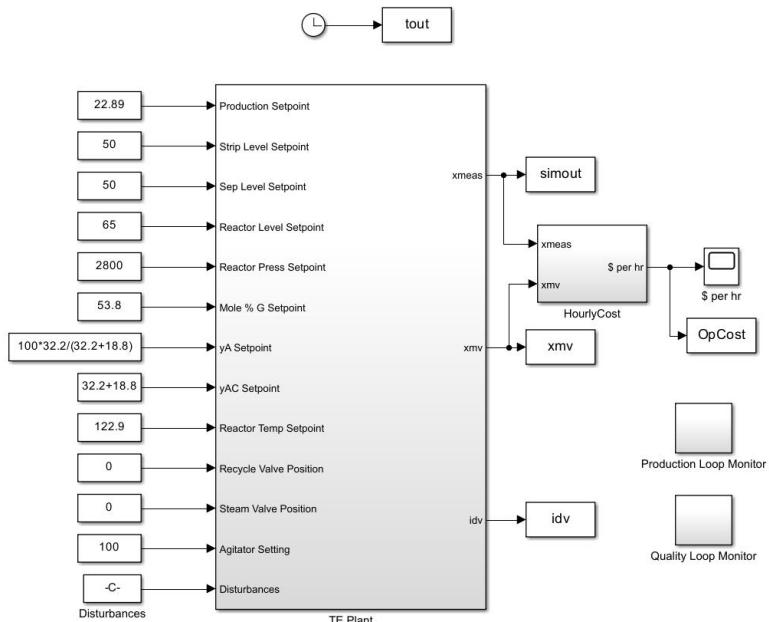


图 26 生成数据的模型

4.2 数据划分

首先将原始数据集的 30715 条数据进行打乱，然后按照 7:3 的比例将原数据集划分为训练集和测试集。在训练集上，训练分类器；在测试集上，测试分析已训练好的分类器的性能。

4.3 故障诊断

4.3.1 数据预处理

将原始数据集进行和回归分析一样的预处理，包括离群点检测、数据去噪和归一化，由于步骤一样，此处不再赘述。

4.3.2 支持向量机分类

支持向量机（Support Vector Machine, SVM）是一种常用于分类任务的监督学习算法。与 3.2.4 中的 SVR 将数据点包含在边界之间的方式不同，SVM 是通过找到一个最佳的超平面来最大化类间的间隔，从而实现分类，示意图如下。

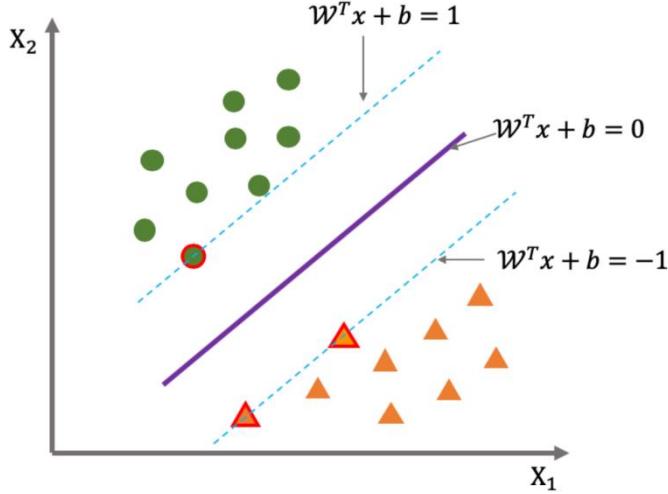


图 27 SVM 示意图

对模型进行训练后，我们采用混淆矩阵和 ROC 曲线两种方式对模型进行评价。混淆矩阵（Confusion Matrix）是一种用于评估分类模型性能的工具，特别适用于二分类和多分类任务。它通过显示预测结果与实际结果的对比，帮助我们全面了解模型在不同类别上的表现。

对于多分类问题，混淆矩阵是一个 $C \times C$ 的矩阵，其中 C 是类别的数量。矩阵中的每个元素 M_{ij} 表示实际为第 i 类且预测为第 j 类的样本数。基于混淆矩阵，有以下几个评价指标。

(1) 准确率

准确率表示模型正确分类的样本数占总样本数的比例。

$$\text{Accuracy} = \frac{\sum_{i=1}^C M_{ii}}{\sum_{i=1}^C \sum_{j=1}^C M_{ij}}$$

(2) 精确率

精确率表示模型预测为某类别的样本中实际属于该类别的比例。对于每个类别 i ，其精确率计算公式为：

$$\text{Precision} = \frac{M_{ii}}{\sum_{j=1}^C M_{ji}}$$

(3) 召回率

召回率表示实际属于某一类别的样本中被正确分类的比例。对于每个类别*i*，其召回率计算公式为：

$$\text{Recall} = \frac{M_{ii}}{\sum_{j=1}^C M_{ij}}$$

(4) 特异度

特异度表示实际不属于某一类别的样本中被正确分类为不属于该类别的比例。对于每个类别*i*，其特异度计算公式为：

$$\text{Specificity} = \frac{\sum_{j=1, j \neq i}^C \sum_{k=1, k \neq i}^C M_{jk}}{\sum_{j=1, j \neq i}^C \sum_{k=1}^C M_{jk}}$$

(5) F1 分数

F1 分数是精确率和召回率的调和平均值。对于每个类别*i*，其 F1 分数计算公式为：

$$F1Score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

训练集和测试集的混淆矩阵如图 28 所示，左边为训练集，右边为测试集。

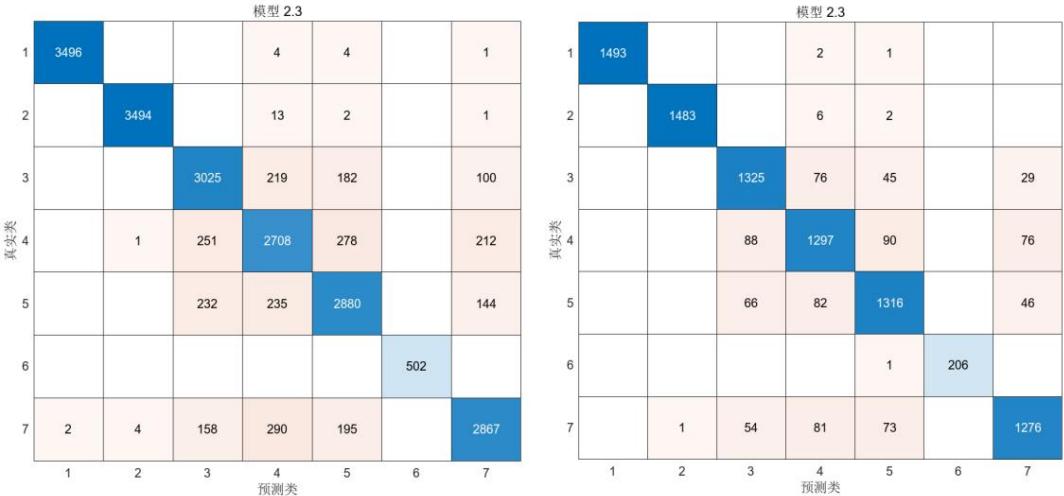


图 28 SVM 分类模型混淆矩阵图

对测试集的混淆矩阵计算上述 5 个评价指标的值，结果如下。

表 14 测试集各指标值

指标	值
准确率	91.06%
精确率	92.22%

召回率	92.14%
特异度	92.24%
F1 分数	92.17%

ROC (Receiver Operating Characteristic) 曲线是一种常用的评价分类模型性能的工具, 通过绘制不同阈值下的真正例率和假正例率得到的曲线。AUC 是 ROC 曲线下的面积, 用于量化 ROC 曲线。AUC 值的范围在 0 到 1 之间, 值越大表示模型性能越好。训练集和测试集的 ROC 曲线如图 29 所示。

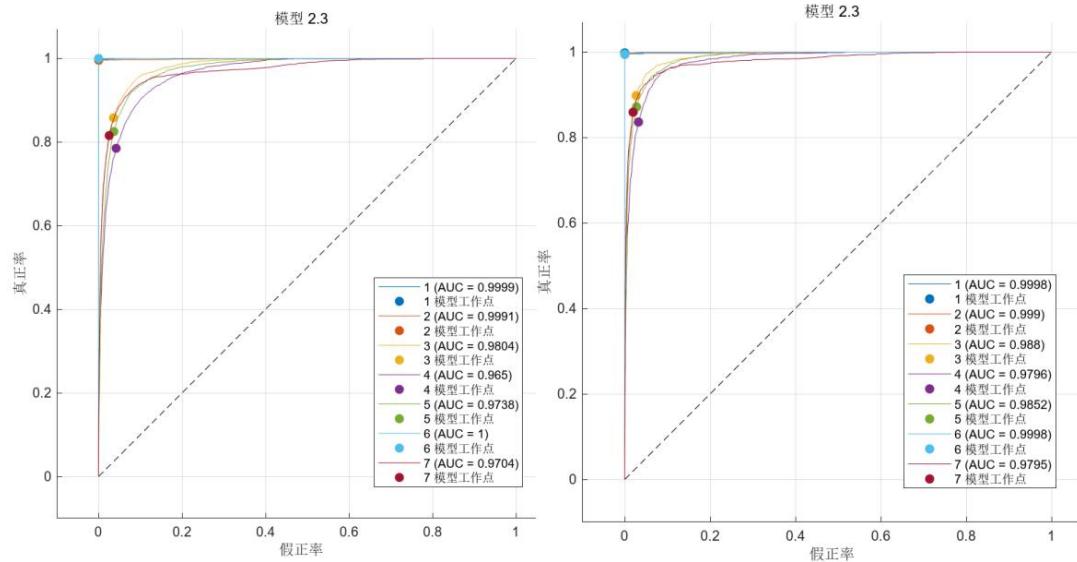


图 29 SVM 分类模型 ROC 曲线

图 29 显示, 训练集和测试集的 7 种分类 AUC 值均在 0.97 以上, 分类性能优秀, 接近完全正确分类。

结合混淆矩阵的相关指标和 ROC 曲线, SVM 分类模型的性能良好。

4.3.3 随机森林分类

随机森林分类算法的原理与 3.2.5 小节介绍的随机森林回归算法类似, 不同的是随机森林回归是将多个决策树的预测结果进行加权平均来预测, 随机森林分类则是将多个决策树的分类结果进行投票来进行预测的, 其余部分一致, 可看 3.2.5 处的内容, 此处不再赘述。

使用随机森林算法训练好, 同样使用和上一小节一样的评价指标对模型性能进行评价。混淆矩阵和 ROC 曲线如图 30, 31 所示。

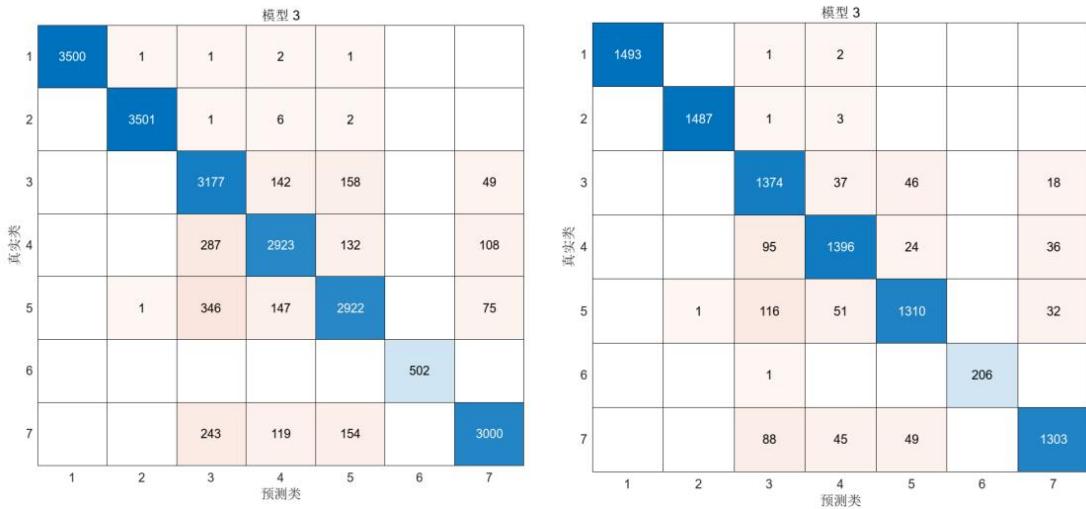


图 30 RF 分类模型混淆矩阵图

对测试集的混淆矩阵计算上述 5 个评价指标的值，结果如下。

表 15 RF 分类性能指标

指标	值
准确率	92.13%
精确率	93.29%
召回率	93.06%
特异度	92.77%
F1 分数	93.14%

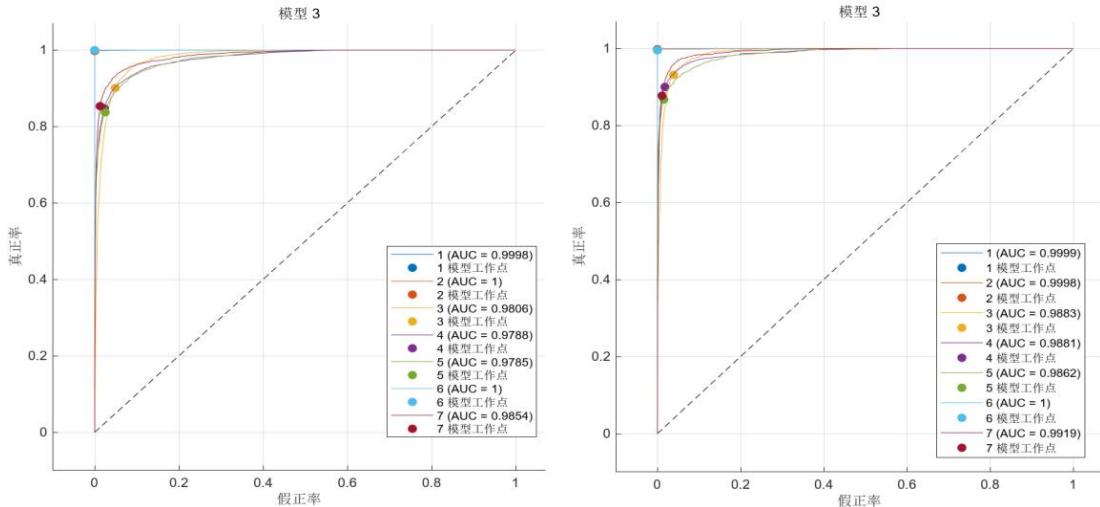


图 31 RF 分类模型 ROC 曲线

图 31 显示，训练集和测试集的 7 种分类 AUG 值均在 0.97 以上，分类性能优秀，接近完全正确分类。

结合混淆矩阵的相关指标和 ROC 曲线，随机森林分类模型的性能非常好，且优于 SVM 分类模型。

5 总结与感悟

5.1 总结

本次的大数据 TEP 仿真实验共分为数据预处理和可视化仿真实验、参数预测仿真实验、故障诊断仿真实验三大部分，从数据预处理到特征提取再到回归和分类模型的建立，层层递进，环环相扣。实验内容与大数据课程的课内理论知识关系密切，大大加深了我对理论部分的理解。

在数据预处理和可视化仿真部分，主要包括数据可视化、离群值检测、数据去噪、数据变化和主成分分析五个内容。可视化中使用了时序图、直方图、箱线图和 Q-Q 图；离群值检测中使用三种离群值检测方法进行对比分析；去噪中先对小波去噪和平滑去噪进行对比分析，然后分别对适用的变量进行去噪；数据变换中对比分析了 z 分数归一化和最大最小归一化，最后结合分析结果和工艺特性选择了 z 分数归一化；主成分分析中我回顾了算法步骤，通过累计贡献度找出了需要的主成分。

通过自己动手实验，我真切地感受到不同可视化方法的异同点，每一种图形都有自己的着重点和功能；将数据可视化后，可以直观地看到原始数据的分布情况和噪声的影响，接着利用数据去噪方法去除噪声，提高数据质量，便于后续预测模型的建立。

在参数预测仿真部分，主要有两部分内容，关联关系分析以及回归模型的建立。实验中模拟真实的化工过程，有 22 个过程变量和 19 个成分变量，直接使用 22 个过程变量去预测某个成分变量并不合理，维度较高且计算资源消耗大。因此首先需要对变量进行相关性分析，最终筛选出与响应变量相关性较强的 5 个过程变量作为预测变量。然后在回归分析部分本文建立了不同的回归预测模型进行对比，包括一元回归与多元回归，线性模型与非线性模型。最终通过对比分析发现多元非线性模型的性能明显优于其他模型，其中随机森林回归模型表现最佳，R 方值达到 0.4，MSE 值为 0.08。

在故障诊断仿真部分，主要是建立故障多分类预测模型，和之前的回归分析步骤类似，先对数据进行预处理，然后使用 SVM 和 RF 分别建立分类预测模型，并对模型的性能进行评估。最后通过对比分析得出随机森林分类模型表现最佳，准确率 92.13%，ROC 曲线中 AUG 值均在 0.99 左右。

综上所述，实验内容包括了大数据课程几乎所有重点知识，对数据挖掘知识的理解和应用有很大的帮助。

5.2 感悟

胡老师指导的大数据实践的 TEP 仿真部分让我受益匪浅，这是我这学期做

的最有趣的实验。

在内容上，实验刚开始的时候老师就在线上详细介绍了实验的内容和要点，并在指导书中清晰写明，非常直观的让我知道自己做的是哪一块的哪一部分。之前做别的实验时，总是感觉与讲课内容关系不是很密切，但是，本实验内容与理论部分关系非常密切，从数据预处理到特征提取再到模型建立，这都是在之前的理论课中讲述过的内容，并不会感到陌生，从认真听到动手做，这门实践课真正的做到了理论与实验的结合，也让我切身感受到了动手敲代码的重要性，学习算法又怎么能缺少动手实践呢。

实验中我印象深刻的有两部分，第一部分是数据预处理，之前我在网络上看到一句话说“大数据分析的大部分时间花在数据预处理上”，当时我还诧异为什么看似最简单的数据处理却需要这么多时间，直到自己动手操作才真正体会到数据预处理的重要性，数据类型多样，噪声类型不一，选择什么去噪方法和归一化方法是非常重要的。如果数据没有处理好，会严重影响后续预测模型的准确率。第二部分是预测模型的建立，回归分析实验中我刚开始用了 Matlab 的工具箱，虽然方便但是模型的性能并不好，工具箱的有些模型也不能用网格搜索寻找最优超参数，于是我重新在 jupyter notebook 中写了 SVR 和 RF 的回归预测模型，使用了课内学过的交叉验证和网格搜索，不过期间我也深刻体会到电脑配置在大数据分析中的重要性，我的笔记本电脑带不动大范围和网格搜索，只能一小段的去搜，因此最后只能得出相对较优的超参数。

在指导上，胡老师每次做实验时都会亲自到实验室答疑，您认真严谨的工作态度也感染了我，我不再摸鱼划水，而是专注实验，争取高效完成；此外，面对我们的各种疑问您都会耐心解答，并会抛出一些思考性的问题，培养我们的思考能力。更重要的是，您的平易近人让我们感受到亲切感，同时对这门课程也有了亲切感，学习更加上进。您的大数据理论课程和实践课程让我对数据挖掘产生了很大的兴趣，看到自己的模型一步一步使预测值接近真实值是非常有趣且有成就感的！

伴随着实践课的结束，我对所学知识也有了更深入的理解。未来在控制科学与工程的道路上，课程蕴含的知识和胡老师认真严谨的工作态度永远会引领我前行，带给我启迪！

附录

任务 1：数据预处理和可视化仿真实验程序（Matlab）

```
clear;clc;

%对 TEP 模型生成的数据进行保存
% save('gz7.mat','simout');

%% 加载仿真数据
load('simout.mat');

%索引数据
i = 1;
A_Feed = simout(:,i);

%% 变量时序图
% 21 个过程变量的时序图
figure;
for i = 1:22
    subplot(5, 5, i); % 创建 5 行 9 列的子图布局
    plot(simout(:, i)); % 绘制每一列的直方图
    title(['过程变量 ', num2str(i)]);
    xlabel('时间');
    ylabel('值');
end

% 19 个成分变量的时序图
figure;
for j = 23:41
    subplot(4, 5, j-22); % 创建 5 行 9 列的子图布局
    plot(simout(:, j)); % 绘制每一列的直方图
    title(['成分变量 ', num2str(j-22)]);
    xlabel('时间');
    ylabel('值');
end

%所有变量的时序图
```

```

figure;
for i = 1:41
subplot(9, 5, i); % 创建 5 行 9 列的子图布局
plot(simout(:, i)); % 绘制每一列的直方图
title(['变量 ', num2str(i)]);
xlabel('时间');
ylabel('值');
end
%% 数据可视化

subplot(2,2,1)
plot(A_Feed)
title('A 物料流量时间序列图') %显示数据的时序特征

%直方图
subplot(2,2,2);
histogram(A_Feed);
title('A 物料流量直方图'); %显示数据在各个区间的分布情况

%箱体图
subplot(2,2,3);
boxplot(A_Feed);
title('A 物料流量箱体图'); %显示分布情况, 观察离群点

%填充区二维绘图
subplot(2,2,4);
area(A_Feed);
title('A 物料流量填充区二维绘图'); %观察缺失值, 空白区域

%% 绘制直方图
% 绘制 22 个过程变量的直方图
figure;
for i = 1:22
subplot(5, 5, i); % 创建 5 行 9 列的子图布局
histogram(simout(:, i)); % 绘制每一列的直方图

```

```

title(['过程变量 ', num2str(i)]);
xlabel('值');
ylabel('频数');
end

% 绘制 19 个成分变量的直方图

figure;
for i = 23:41
subplot(4, 5, i-22); % 创建 5 行 9 列的子图布局
histogram(simout(:, i)); % 绘制每一列的直方图
title(['成分变量 ', num2str(i-22)]);
xlabel('值');
ylabel('频数');
end


figure;
t.TileSpacing = 'loose'; % 使用宽松布局，增大子图之间的间距
t.Padding = 'loose'; % 增大子图与图形边界之间的间距
for i = 1:41
subplot(9, 5, i); % 创建 5 行 9 列的子图布局
histogram(simout(:, i)); % 绘制每一列的直方图
title(['变量 ', num2str(i)]);
xlabel('值');
ylabel('频数');
end

%% QQ 图，用于观察数据是否服从正态分布

% 绘制 22 个过程变量的 QQ 图

figure;
for i = 1:22
subplot(5, 5, i); % 创建 5 行 9 列的子图布局
h = qqplot(simout(:, i)); % 绘制每一列的 QQ 图
% 调整 QQ 图数据点的大小
set(h, 'MarkerSize', 1); % h(1) 是数据点对象
title(['过程变量 ', num2str(i)]);
xlabel('理论分位数');

```

```

ylabel('样本分位数');

end

% 绘制 19 个成分变量的 QQ 图

figure;

for i = 23:41

subplot(4, 5, i-22); % 创建 5 行 9 列的子图布局
h = qqplot(simout(:, i)); % 绘制每一列的 QQ 图
% 调整 QQ 图数据点的大小
set(h, 'MarkerSize', 1); % h(1) 是数据点对象
title(['成分变量 ', num2str(i-22)]);
xlabel('理论分位数');
ylabel('样本分位数');
end


figure;
t.TileSpacing = 'loose'; % 使用宽松布局, 增大子图之间的间距
t.Padding = 'loose'; % 增大子图与图形边界之间的间距
for i = 1:41

subplot(9, 5, i); % 创建 5 行 9 列的子图布局
h = qqplot(simout(:, i)); % 绘制每一列的 QQ 图
% 调整 QQ 图数据点的大小
set(h, 'MarkerSize', 1); % h(1) 是数据点对象
title(['变量 ', num2str(i)]);
xlabel('理论分位数');
ylabel('样本分位数');
end

%% 绘制所有变量的箱线图

% 绘制 21 个过程变量的箱线图

figure;

for i = 1:22

subplot(5, 5, i); % 创建 5 行 9 列的子图布局
boxplot(simout(:, i));
title(['过程变量 ', num2str(i)]);
xlabel('样本数据');

```

```

ylabel('值');

end

%绘制 19 个成分变量的箱线图

figure;

for i = 23:41

subplot(4, 5, i-22); % 创建 5 行 9 列的子图布局

boxplot(simout(:, i));

title(['过程变量 ', num2str(i-22)]);

xlabel('样本数据');

ylabel('值');

end


figure;

t.TileSpacing = 'loose'; % 使用宽松布局，增大子图之间的间距

t.Padding = 'loose'; % 增大子图与图形边界之间的间距

for i = 1:41

subplot(11, 4, i); % 创建 5 行 9 列的子图布局

boxplot(simout(:, i));

title(['变量 ', num2str(i)]);

% xlabel('样本数据');

% ylabel('值');

end

%% 离群点检测方法对比（以 A_Feed 为例）

% 方法 1：绘制箱线图

figure;

boxplot(A_Feed);

title('箱线图离群值检测');

xlabel('样本数据');

ylabel('值');


% 获取箱线图数据

h = findobj(gca, 'Tag', 'Outliers');

outliersData = [];

```

```

for i = 1:length(h)
outliersData = [outliersData; h(i).YData];
end

% 显示离群值
disp('离群值:');
disp(outliersData);

% 可视化
figure;
plot(A_Feed, 'bo', 'MarkerSize', 2);
hold on;
plot(find(ismember(A_Feed, outliersData)), outliersData, 'ro', 'MarkerSize',
5);
title('箱线图离群值检测结果');
xlabel('样本索引');
ylabel('值');
legend('数据点', '离群值');
hold off;

%% 方法 2: 3-sigma 检测法
% 计算数据的均值和标准差
mu = mean(A_Feed);
sigma = std(A_Feed);

% 设定 3-sigma 阈值
lower_bound = mu - 3 * sigma;
upper_bound = mu + 3 * sigma;

% 检测离群点
outliers = (A_Feed < lower_bound) | (A_Feed > upper_bound);

% 显示离群点

```

```

disp('离群点:');
disp(A_Feed(outliers));

% 可视化
figure;
plot(A_Feed, 'bo', 'MarkerSize', 2);
hold on;
plot(find(outliers), A_Feed(outliers), 'ro', 'MarkerSize', 5);
yline(lower_bound, 'r--', '3-sigma lower bound');
yline(upper_bound, 'r--', '3-sigma upper bound');
title('3-sigma 离群点检测');
xlabel('样本索引');
ylabel('值');
legend('数据点', '离群点', 'Location', 'best');
hold off;

%% 方法 3: z-score 检测法
z_scores = zscore(A_Feed);

% 设定阈值
threshold = 3;

% 检测离群点
outliers_1 = abs(z_scores) > threshold;

% 显示离群点
disp('离群点:');
disp(A_Feed(outliers_1));

% 可视化
figure;
plot(A_Feed, 'bo', 'MarkerSize', 2);
hold on;
plot(find(outliers_1), A_Feed(outliers_1), 'ro', 'MarkerSize', 5);

```

```

title('z-score 离群点检测');
xlabel('样本索引');
ylabel('值');
legend('数据点', 'Location', 'best');
hold off;

%% 数据去噪 (过程变量 8, 相对平稳数据)
process_8 = simout(1:500,10);
process_8_Wde = wdenoise(process_8); %小波去噪
process_8_Smooth = smoothdata(process_8); %平滑去噪
% % 傅里叶变换去噪
% signal_fft = fft(process_8);
%
% % 设计滤波器
% n = 7201;
% f = (0:n-1)*(1/n); % 频率轴
% cutoff = 0.1; % 截止频率 (归一化)
% filter = (f < cutoff | f > (1 - cutoff)); % 布尔滤波器 (低通滤波器)
%
% % 应用滤波器
% signal_fft_filtered = signal_fft .* filter;
%
% % 逆傅里叶变换
% signal_denoised = ifft(signal_fft_filtered);

% 对比不同的去噪方法
figure
subplot(3,1,1);
plot(process_8);
title('原始信号');
xlabel('时间');
ylabel('幅值');


```

```

subplot(3,1,2);
plot(process_8_Smooth);
title('平滑去噪');
xlabel('时间');
ylabel('幅值');

subplot(3,1,3);
plot(process_8_Wde);
title('小波去噪');
xlabel('时间');
ylabel('幅值');

% subplot(4,1,4);
% plot(real(signal_denoised)); % 取实部
% title('傅里叶去噪后的信号');
% xlabel('时间');
% ylabel('幅值');

%% 数据去噪 (过程变量 1: A 物料流量, 突变数据)

%去噪方法 1: 小波去噪
A_Feed = A_Feed(1:500)
A_Feed_Wde = wdenoise(A_Feed);

%去噪方法 2: 平滑窗口滤波去噪
%对比不同窗口宽度下的去噪效果
A_Feed_Smooth = smoothdata(A_Feed);
A_Feed_Smooth_3 = smoothdata(A_Feed, 'movmean', 3);
A_Feed_Smooth_30 = smoothdata(A_Feed, 'movmean', 30);
A_Feed_Smooth_60 = smoothdata(A_Feed, 'movmean', 60);
figure
subplot(3,1,1);
plot(A_Feed_Smooth_3);
title('A_Feed_Smooth_3');

```

```

subplot(3,1,2);
plot(A_Feed_Smooth_30);
title('A_Feed_Smooth_30');

subplot(3,1,3);
plot(A_Feed_Smooth_60);
title('A_Feed_Smooth_60');

% % 傅里叶变换去噪
% signal_fft = fft(A_Feed);
%
% % 设计滤波器
% n = 7201;
% f = (0:n-1)*(1/n); % 频率轴
% cutoff = 0.1; % 截止频率（归一化）
% filter = (f < cutoff | f > (1 - cutoff)); % 布尔滤波器（低通滤波器）
%
% % 应用滤波器
% signal_fft_filtered = signal_fft .* filter;
%
% % 逆傅里叶变换
% signal_denoised = ifft(signal_fft_filtered);

% 对比不同的去噪方法
figure
subplot(3,1,1);
plot(A_Feed);
title('原始信号');
xlabel('时间');
ylabel('幅值');

subplot(3,1,2);

```

```

plot(A_Feed_Smooth);
title('平滑去噪');
xlabel('时间');
ylabel('幅值');

subplot(3,1,3);
plot(A_Feed_Wde);
title('小波去噪');
xlabel('时间');
ylabel('幅值');

% subplot(4,1,4);
% plot(real(signal_denoised)); % 取实部
% title('傅里叶去噪后的信号');
% xlabel('时间');
% ylabel('幅值');
%% 小波变换详细步骤
% 绘制原始信号
figure;
signal = A_Feed
subplot(3,1,1);
plot(signal);
title('原始信号');
xlabel('时间');
ylabel('幅值');

% 选择小波类型和分解层数
waveletType = 'db4'; % Daubechies 4 小波
decompositionLevel = 4;

% 进行小波分解
[C, L] = wavedec(signal, decompositionLevel, waveletType);

% 提取近似系数和细节系数

```

```

approximation = appcoef(C, L, waveletType, decompositionLevel); %提取近似系数
details = cell(1, decompositionLevel); %提取细节系数
for i = 1:decompositionLevel
details{i} = detcoef(C, L, i);
end

% 绘制近似系数
subplot(3,1,2);
plot(approximation);
title(['近似系数 (层数 = ' num2str(decompositionLevel) ')']);
xlabel('样本点');
ylabel('幅值');

% 重构信号
reconstructedSignal = waverec(C, L, waveletType);

% 绘制重构信号
subplot(3,1,3);
plot(reconstructedSignal);
title('重构信号');
xlabel('时间');
ylabel('幅值');

% 显示各个细节系数
figure;
for i = 1:decompositionLevel
subplot(decompositionLevel, 1, i);
plot(details{i});
title(['细节系数 (层数 = ' num2str(i) ')']);
xlabel('样本点');
ylabel('幅值');
end

%% 整理后对所有数据进行去噪

```

```

%整理出平稳数据和突变数据，分别需要平滑去噪和小波去噪

columns_to_process_smooth =
[2,3,4,5,6,8,9,12,14,15,17,19,21,22,24,25,26,27,28,30,31,34,36,37,39,41];
columns_to_process_wde = [1,7,10,11,13,16,18,20,23,29,32,33,35,38,40];

% 创建处理后的数据矩阵
qz_data = simout;

% 对特定列进行平滑去噪
for col = columns_to_process_smooth
qz_data(:, col) = smooth(simout(:,col));
end

% 对特定列进行小波去噪
for col = columns_to_process_wde
qz_data(:, col) = wdenoise(simout(:,col));
end

%% 变量时序图
% 22个过程变量的时序图
figure;
for i = 1:22
subplot(5, 5, i); % 创建 5 行 9 列的子图布局
plot(qz_data(:, i)); % 绘制每一列的直方图
title(['过程变量 ', num2str(i)]);
xlabel('时间');
ylabel('值');
end

% 19个成分变量的时序图
figure;
for j = 23:41
subplot(4, 5, j-22); % 创建 5 行 9 列的子图布局
plot(qz_data(:, j)); % 绘制每一列的直方图
title(['成分变量 ', num2str(j-22)]);
xlabel('时间');
ylabel('值');
end

```

```

end

% 所有变量的时序图
figure;
t.TileSpacing = 'loose'; % 使用宽松布局，增大子图之间的间距
t.Padding = 'loose'; % 增大子图与图形边界之间的间距
for i = 1:41
    subplot(9, 5, i); % 创建 5 行 9 列的子图布局
    plot(qz_data(:, i)); % 绘制每一列的直方图
    title(['变量 ', num2str(i)]);
    xlabel('时间');
    ylabel('值');
end

%% 数据变换：归一化处理（本小节用于对比）
% 最小-最大归一化
min_val = min(A_Feed);
max_val = max(A_Feed);
data_minmax = (A_Feed - min_val) / (max_val - min_val);

% z-score 标准化
mu = mean(A_Feed);
sigma = std(A_Feed);
data_zscore = (A_Feed - mu) / sigma;

% 绘制原始数据和归一化后的数据
figure;

% 绘制原始数据
subplot(3, 1, 1);
plot(A_Feed(1:500));
title('原始数据');
xlabel('时间');
ylabel('值');

```

```

% 绘制最小-最大归一化后的数据

subplot(3, 1, 2);
plot(data_minmax(1:500));
title('最小-最大归一化后的数据');
xlabel('时间');
ylabel('值');

% 绘制 z-score 标准化后的数据

subplot(3, 1, 3);
plot(data_zscore(1:500));
title('z-score 标准化后的数据');
xlabel('时间');
ylabel('值');

%% 整体数据归一化处理，采用 Z-score 归一化

% 初始化归一化后的矩阵

z_score_normalized_data = zeros(size(qz_data));

% 对每一列分别进行 Z 分数归一化

for col = 1:size(qz_data, 2)
col_data = qz_data(:, col);
mean_val = mean(col_data);
std_val = std(col_data);
z_score_normalized_data(:, col) = (col_data - mean_val) / std_val;
end

% 所有变量的时序图

figure;
t.TileSpacing = 'loose'; % 使用宽松布局，增大子图之间的间距
t.Padding = 'loose'; % 增大子图与图形边界之间的间距

for i = 1:41
subplot(9, 5, i); % 创建 5 行 9 列的子图布局
plot(z_score_normalized_data(:, i)); % 绘制每一列的直方图
title(['变量 ', num2str(i)]);
xlabel('时间');

```

```

ylabel('值');

end

%% % 使用 pca 函数进行 PCA
[coeff, score, latent, tsquared, explained, mu] = pca(z_score_normalized_data);

% 计算累计贡献度
cumulative_explained = cumsum(explained);

% 找到累计贡献度超过 80% 的主成分数量
num_components = find(cumulative_explained >= 80, 1);

% 选择前 num_components 个主成分
reduced_data = score(:, 1:num_components);

% 显示选择的主成分数量
disp(['选择的主成分数量: ', num2str(num_components)]);

% % 显示降维后的数据
% disp('降维后的数据: ');
% disp(reduced_data);

% 可视化降维后的主成分的序列图
figure;
t.TileSpacing = 'loose'; % 使用宽松布局，增大子图之间的间距
t.Padding = 'loose'; % 增大子图与图形边界之间的间距
for i = 1:num_components
    subplot(5, 5, i);
    plot(reduced_data(:, i));
    title(['主成分 ' num2str(i) ' 的序列图']);
    xlabel('样本');
    ylabel(['主成分 ' num2str(i) ' 值']);
end

figure

```

```

explained_pc = explained(1:num_components,1);

for i = 1:num_components
fprintf('第%d个主成分的贡献度为%f%%\n',i,explained_pc(i,1));
end

X =
categorical({'PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10','PC11','PC12','PC13','PC14','PC15','PC16','PC17','PC18','PC19','PC20','PC21','PC22','PC23','PC24','PC25'});
X =
reordercats(X,['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10','PC11','PC12','PC13','PC14','PC15','PC16','PC17','PC18','PC19','PC20','PC21','PC22','PC23','PC24','PC25']);
bar(X,explained_pc)
xlabel('主成分');
ylabel('贡献度')

```

任务 2：参数预测仿真实验程序（Matlab， Python）

```

%% 任务 2

% 计算协方差矩阵
cov_matrix = cov(qz_data);

% 计算相关系数矩阵
corr_matrix = corrcoef(qz_data);

% 显示协方差矩阵和相关系数矩阵
disp('协方差矩阵: ');
disp(cov_matrix);

disp('相关系数矩阵: ');
disp(corr_matrix);

% 可视化相关系数矩阵
figure;
heatmap(corr_matrix, 'FontSize', 10, 'FontName', 'Times New Roman');

```

```

title('heatmap');

% 设置相关系数阈值
threshold = 0.8;

% 找出绝对值大于阈值的相关性对
[row, col] = find(abs(corr_matrix) > threshold & abs(corr_matrix) < 1); % 排除自身的 1 相关性

% 显示具有较强关联关系的变量对
disp('具有较强关联关系的变量对: ');
for i = 1:length(row)
    fprintf('变量 %d 和变量 %d 的相关系数为 %.2f\n', row(i), col(i),
    corr_matrix(row(i), col(i)))
end

% 提取具有较强关联关系的变量
strong_corr_variables = unique([row; col]);

% 绘制散点矩阵，进行辅助说明
figure;

% 创建变量标签
variable_labels = {'变量 7', '变量 13', '变量 16'};

% 绘制散点图矩阵
[~, ax] = plotmatrix(qz_data(:, strong_corr_variables));

% 添加横纵坐标标签
num_variables = 3;
for i = 1:num_variables
    % 设置对角线下的横坐标标签
    ax(num_variables, i).XLabel.String = variable_labels{i};
    % 设置对角线左边的纵坐标标签
    ax(i, 1).YLabel.String = variable_labels{i};
end

```

```

% 添加标题
title('变量 7, 变量 13 和 变量 16 的散点图矩阵');

%% 回归分析 (一元回归,因变量为成分变量 7)
data_train = z_score_normalized_data(1:5040,:); %分开训练集和测试集
data_test = z_score_normalized_data(5041:7201,:);

X = data_train(:,20); % 自变量
Y = data_train(:,29); % 因变量

X_test = data_test(:,20); % 自变量
Y_test = data_test(:,29); % 因变量

% 拟合线性回归模型
model = fitlm(X, Y);

% 显示模型摘要
disp(model);

% 提取回归系数
coefficients = model.Coefficients.Estimate;
slope = coefficients(2); % 斜率
intercept = coefficients(1); % 截距

% 预测值
Y_pred = predict(model, X);
Y_pred_test = predict(model, X_test);

% 计算 R 方值
R_squared = corr(Y_pred, Y)^2;
R_squared_test = corr(Y_pred_test, Y_test)^2;

% 计算均方误差 MSE
MSE = mean((Y - Y_pred).^2)

```

```

MSE_test = mean((Y_test - Y_pred_test).^2)

% 计算均方根误差 RMSE
RMSE = sqrt(mean((Y - Y_pred).^2))
RMSE_test = sqrt(mean((Y_test - Y_pred_test).^2))

% 计算平均绝对误差 MAE
MAE = mean(abs(Y - Y_pred))
MAE_test = mean(abs(Y_test - Y_pred_test))

disp(['R 方_train: ', num2str(R_squared)]);
disp(['R 方_test: ', num2str(R_squared_test)]);
disp(['MSE_train: ', num2str(MSE)]);
disp(['MSE_test: ', num2str(MSE_test)]);
disp(['RMSE_train: ', num2str(RMSE)]);
disp(['RMSE_test: ', num2str(RMSE_test)]);
disp(['MAE: ', num2str(MAE)]);
disp(['MAE_test: ', num2str(MAE_test)]);

% 绘制回归线
figure;
plot(X, Y, 'o', 'DisplayName', '数据点', 'MarkerSize', 2); % 绘制数据点
hold on;
plot(X, Y_pred, '-', 'DisplayName', '回归线', 'MarkerSize', 5); % 绘制回归线
title('单元线性回归训练集');
xlabel('自变量 X');
ylabel('因变量 Y');
legend('show');

figure;
plot(X_test, Y_test, 'o', 'DisplayName', '数据点', 'MarkerSize', 2); % 绘制数
据点
hold on;

```

```

plot(X_test, Y_pred_test, '-','DisplayName','回归线','MarkerSize',5); % 绘制回归线

title('单元线性回归测试集');
xlabel('自变量 X');
ylabel('因变量 Y');
legend('show');

% 绘制预测值与实际值的散点图
figure;
plot(Y, 'b-');
hold on;
plot(Y_pred, 'r-');
title('一元线性回归训练集对比图');
xlabel('实际值');
ylabel('预测值');
legend('实际值', '预测值', 'Location', 'northwest');
hold off;
xlim([0 5040]); % 设置横轴范围

% 绘制预测值与实际值的散点图
figure;
plot(Y_test, 'b-');
hold on;
plot(Y_pred_test, 'r-');
title('一元线性回归测试集对比图');
xlabel('实际值');
ylabel('预测值');
legend('实际值', '预测值', 'Location', 'northwest');
hold off;
xlim([0 2161]); % 设置横轴范围

%% 多元线性回归, 因变量为成分变量 7

data_train = z_score_normalized_data(1:5040,:); % 分开训练集和测试集
data_test = z_score_normalized_data(5041:7201,:);

```

```

X = data_train(:,[5,11,18,20,22]); % 自变量
Y = data_train(:,29); % 因变量

X_test = data_test(:,[5,11,18,20,22]); % 自变量
Y_test = data_test(:,29); % 因变量

% 拟合线性回归模型
model = fitlm(X, Y);

% 显示模型摘要
disp(model);

% 提取回归系数
coefficients = model.Coefficients.Estimate;
slope = coefficients(2); % 斜率
intercept = coefficients(1); % 截距

% 预测值
Y_pred = predict(model, X);
Y_pred_test = predict(model, X_test);

% 计算 R 方值
R_squared = corr(Y_pred, Y)^2;
R_squared_test = corr(Y_pred_test, Y_test)^2;

% 计算均方误差 MSE
MSE = mean((Y - Y_pred).^2)
MSE_test = mean((Y_test - Y_pred_test).^2)

% 计算均方根误差 RMSE
RMSE = sqrt(mean((Y - Y_pred).^2))
RMSE_test = sqrt(mean((Y_test - Y_pred_test).^2))

% 计算平均绝对误差 MAE

```

```

MAE = mean(abs(Y - Y_pred))
MAE_test = mean(abs(Y_test - Y_pred_test))

% 显示评估指标
disp(['R 方_train: ', num2str(R_squared)]);
disp(['R 方_test: ', num2str(R_squared_test)]);
disp(['MSE_train: ', num2str(MSE)]);
disp(['MSE_test: ', num2str(MSE_test)]);
disp(['RMSE_train: ', num2str(RMSE)]);
disp(['RMSE_test: ', num2str(RMSE_test)]);
disp(['MAE: ', num2str(MAE)]);
disp(['MAE_test: ', num2str(MAE_test)]);

% 绘制回归线
figure;
plot(X, Y, 'o', 'DisplayName', '数据点', 'MarkerSize', 2); % 绘制数据点
hold on;
plot(X, Y_pred, '-', 'DisplayName', '回归线', 'MarkerSize', 5); % 绘制回归线
title('多元线性回归训练集');
xlabel('自变量 X');
ylabel('因变量 Y');
legend('show');

figure;
plot(X_test, Y_test, 'o', 'DisplayName', '数据点', 'MarkerSize', 2); % 绘制数据点
hold on;
plot(X_test, Y_pred_test, '-', 'DisplayName', '回归线', 'MarkerSize', 5); % 绘制回归线
title('多元线性回归测试集');
xlabel('自变量 X');
ylabel('因变量 Y');
legend('show');

```

```

% 绘制预测值与实际值的散点图

figure;
plot(Y, 'b-');
hold on;
plot(Y_pred, 'r-');
title('多元线性回归训练集对比图');
xlabel('实际值');
ylabel('预测值');
legend('实际值', '预测值', 'Location', 'northwest');
hold off;
xlim([0 5040]); % 设置横轴范围

% 绘制预测值与实际值的散点图

figure;
plot(Y_test, 'b-');
hold on;
plot(Y_pred_test, 'r-');
title('多元线性回归测试集对比图');
xlabel('实际值');
ylabel('预测值');
legend('实际值', '预测值', 'Location', 'northwest');
hold off;
xlim([0 2161]); % 设置横轴范围

#导入读取文件的库
import pandas as pd
#读取文件
file_path = r"D:\桌面\RF.xls" # 替换为你的 Excel 文件路径
data = pd.read_excel(file_path)
data_array = data.to_numpy()

import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVR

```

```
from sklearn.metrics import r2_score

# 划分训练集和测试集
X = data_array[:,0:5]
y = data_array[:,5]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 定义 SVR 模型
svr = SVR()

# 定义超参数网格
param_grid = {
    'kernel': ['rbf', 'poly','sigmoid'],
    'C': [0.1],
    'gamma': ['scale', 'auto']
}

# 使用网格搜索找到最佳超参数
grid_search = GridSearchCV(svr, param_grid, cv=5, scoring='r2')
grid_search.fit(X_train, y_train)

# 输出最佳超参数组合
print("Best Parameters:", grid_search.best_params_)

# 在训练集上进行预测
y_pred_train = grid_search.predict(X_train)
# 在测试集上进行预测
y_pred = grid_search.predict(X_test)

# 计算 R 方
r2_train = r2_score(y_train, y_pred_train)
print("训练集 R-squared:", r2_train)
r2 = r2_score(y_test, y_pred)
```

```
print("测试集 R-squared:", r2)

# 计算 MSE
MSE_train = np.mean((y_test - y_pred) ** 2)
MSE_test = np.mean((y_train - y_pred_train) ** 2)

# 计算 RMSE
RMSE_train = np.sqrt(MSE_train)
RMSE_test = np.sqrt(MSE_test)

print("训练集 MSE 为:", MSE_train)
print("测试集 MSE 为:", MSE_test)
print("训练集 RMSE 为:", RMSE_train)
print("测试集 RMSE 为:", RMSE_test)

#-----
# 设置画图的宽度和高度，单位为英寸
figure_width = 40
figure_height = 6

# 创建一个 figure
fig = plt.figure(figsize=(figure_width, figure_height))

# 绘制实际值
plt.plot(range(len(y_train)), y_train, color='red', label='True', linewidth=1)

# 绘制预测值
plt.plot(range(len(y_pred_train)), y_pred_train, color='blue', label='Predicted', linewidth=1)

# 添加图例
plt.legend()

# 显示图形
```

```

plt.xlabel('Index')
plt.ylabel('Value')
plt.title('True vs Predicted Values with Smooth Curve')
plt.show()
#-----
# 设置画图的宽度和高度，单位为英寸
figure_width = 25
figure_height = 6

# 创建一个 figure
fig = plt.figure(figsize=(figure_width, figure_height))

# 绘制实际值
plt.plot(range(len(y_test)), y_test, color='red', label='True', linewidth=1)

# 绘制预测值
plt.plot(range(len(y_pred)), y_pred, color='blue', label='Predicted', linewidth=1)

# 添加图例
plt.legend()

# 显示图形
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('True vs Predicted Values with Smooth Curve')
plt.show()

#随机森林回归（python）
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

```

```
# 划分数据集为训练集和测试集
X = data_array[:,0:5]
y = data_array[:,5]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 定义随机森林回归模型
rf_regressor = RandomForestRegressor()

# 定义超参数网格
param_grid = {
    'n_estimators': [73],
    'max_depth': [52],
    'min_samples_split': [2]
}

# 使用网格搜索寻找最优超参数
grid_search = GridSearchCV(estimator=rf_regressor, param_grid=param_grid, cv=5,
scoring='r2')
grid_search.fit(X_train, y_train)

# 输出最优超参数
print("Best Parameters:", grid_search.best_params_)

# 使用最优超参数的模型进行预测
best_rf_regressor = grid_search.best_estimator_
y_pred = best_rf_regressor.predict(X_test)
y_pred_train = best_rf_regressor.predict(X_train)

# 计算 R 方
r2_train = r2_score(y_train, y_pred_train)
print("训练集的 R-squared:", r2_train)
r2 = r2_score(y_test, y_pred)
```

```
print("测试集的 R-squared:", r2)
# 计算 MSE
MSE_train = np.mean((y_test - y_pred) ** 2)
MSE_test = np.mean((y_train - y_pred_train) ** 2)
# 计算 RMSE
RMSE_train = np.sqrt(MSE_train)
RMSE_test = np.sqrt(MSE_test)

print("训练集的 MSE 为:", MSE_train)
print("测试集的 MSE 为:", MSE_test)
print("训练集的 RMSE 为:", RMSE_train)
print("测试集的 RMSE 为:", RMSE_test)

#-----
# 设置画图的宽度和高度, 单位为英寸
figure_width = 40
figure_height = 6

# 创建一个 figure
fig = plt.figure(figsize=(figure_width, figure_height))

# 绘制实际值
plt.plot(range(len(y_train)), y_train, color='red', label='True', linewidth=1)

# 绘制预测值
plt.plot(range(len(y_pred_train)), y_pred_train, color='blue', label='Predicted', linewidth=1)

# 添加图例
plt.legend()

# 显示图形
plt.xlabel('Index')
plt.ylabel('Value')
```

```

plt.title('True vs Predicted Values with Smooth Curve')
plt.show()
#-----
# 设置画图的宽度和高度，单位为英寸
figure_width = 25
figure_height = 6

# 创建一个 figure
fig = plt.figure(figsize=(figure_width, figure_height))

# 绘制实际值
plt.plot(range(len(y_test)), y_test, color='red', label='True', linewidth=1)

# 绘制预测值
plt.plot(range(len(y_pred)), y_pred, color='blue', label='Predicted', linewidth=1)

# 添加图例
plt.legend()

# 显示图形
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('True vs Predicted Values with Smooth Curve')
plt.show()

```

任务 3：故障诊断仿真实验程序（Matlab）

```

%% 训练出来的随机森林

data_train = data_resort(1:21500,:);
data_test = data_resort(21501:30715,:);

% 使用训练好的模型对测试集进行预测
X_test = data_test(:, 1:41); % 1000 行 41 列的测试数据
Y_test = data_test(:, 42); % 1000 行的测试标签（假设二分类）

predictedLabels = Bagging.predictFcn(X_test);

```

```

% 绘制混淆矩阵
figure
confusionMat = confusionmat(Y_test, predictedLabels);
confusionchart(confusionMat)
title('Confusion Matrix');

% 计算准确率、召回率、精确率和 F1 分数
numClasses = size(confusionMat, 1);
accuracy = sum(diag(confusionMat)) / sum(confusionMat(:));

precision = zeros(numClasses, 1);
recall = zeros(numClasses, 1);
f1Score = zeros(numClasses, 1);
Specificity=zeros(numClasses, 1);

for i = 1:numClasses
    truePositive = confusionMat(i, i);
    falsePositive = sum(confusionMat(:, i)) - truePositive;
    falseNegative = sum(confusionMat(i, :)) - truePositive;
    precision(i) = truePositive / (truePositive + falsePositive);
    recall(i) = truePositive / (truePositive + falseNegative);
    f1Score(i) = 2 * (precision(i) * recall(i)) / (precision(i) + recall(i));
    Specificity(i) = trueNegative / (trueNegative + falsePositive);
end

% 输出相关指标
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
fprintf('Precision:\n');
disp(precision);
fprintf('Recall:\n');
disp(recall);
fprintf('F1 Score:\n');

```

```

disp(f1Score);

fprintf('Specificity:\n');
disp(Specificity);

%% 训练好的 SVM

data_train = data_resort(1:21500,:);
data_test = data_resort(21501:30715,:);

% 使用训练好的模型对测试集进行预测

X_test = data_test(:, 1:41); % 1000 行 41 列的测试数据
Y_test = data_test(:, 42); % 1000 行的测试标签（假设二分类）

predictedLabels = SVM.predictFcn(X_test);

% 绘制混淆矩阵

figure
confusionMat = confusionmat(Y_test, predictedLabels);
confusionchart(confusionMat)
title('Confusion Matrix');

% 计算准确率、召回率、精确率和 F1 分数

numClasses = size(confusionMat, 1);
accuracy = sum(diag(confusionMat)) / sum(confusionMat(:));

precision = zeros(numClasses, 1);
recall = zeros(numClasses, 1);
f1Score = zeros(numClasses, 1);
Specificity=zeros(numClasses, 1);

for i = 1:numClasses
    truePositive = confusionMat(i, i);
    falsePositive = sum(confusionMat(:, i)) - truePositive;
    falseNegative = sum(confusionMat(i, :)) - truePositive;
    trueNegative = sum(confusionMat(:, i)) - falseNegative;

```

```
precision(i) = truePositive / (truePositive + falsePositive);
recall(i) = truePositive / (truePositive + falseNegative);
f1Score(i) = 2 * (precision(i) * recall(i)) / (precision(i) + recall(i));
Specificity(i) = trueNegative / (trueNegative + falsePositive);

end

% 输出相关指标
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
fprintf('Precision:\n');
disp(precision);
fprintf('Recall:\n');
disp(recall);
fprintf('F1 Score:\n');
disp(f1Score);
fprintf('Specificity:\n');
disp(Specificity);
```