



中国研究生创新实践系列大赛

“华为杯”第十七届中国研究生

数学建模竞赛

学　　校		浙江大学
参赛队号		20103350031
队员姓名	1.	郭亦宗
	2.	冯斌
	3.	杨梓锋

中国研究生创新实践系列大赛

“华为杯”第十七届中国研究生

数学建模竞赛

题 目 基于数据挖掘技术的汽油辛烷值优化研究

摘 要：

辛烷值（RON）是反映汽油燃烧性能的最重要指标。然而，现有技术在对催化裂化汽油进行脱硫和降烯烃过程中，普遍降低了汽油辛烷值，造成了经济损失和环境压力。随着信息技术的发展，数据挖掘技术已逐步应用于化工过程，亟需探索数据挖掘技术下的辛烷值模型和优化方法。围绕如何建立辛烷值模型和优化方法这一问题，本文串联起每一子问题，提出了考虑工艺实际、创新性的解决方案并验证可行。首先，对附件 3 的 285 和 313 号两个样本按照整定方法进行预处理，并将处理后的数据放入附件 1 供后续研究。然后，设计了两阶段整定算法对附件 1 的 325 个样本 367 个变量预处理，提出 4 种代表性的特征选择和降维方法并求解，进而提出综合筛选模型以融合各方法的优点。进一步改进综合筛选模型，将非操作变量和操作变量分开单独建立与辛烷值之间的关系模型，并求解自变量之间的相关度，满足代表性和独立性的需求，最终得到 30 个主要影响变量。在预处理和求解主要影响变量后，建立改进的长短期记忆网络（LSTM）预测模型预测产品辛烷值，从而预测辛烷值损失。利用遗传（GA）算法改进 LSTM 的参数选择，并对比了其他 3 种典型的预测方法，确定了该模型在辛烷值损失预测方面的优势。建立脱硫率的预测模型标准含硫量，采用 GA 和 PSO 两种算法对 325 个样本进行优化，得到操作条件。最终，依据前一步的优化模型和操作变量调整幅度限制，针对 133 号样本采用均匀步长调整策略以保证生产的稳定性。结合辛烷值、硫含量预测模型以及 t 分布随机邻域嵌入（t-SNE）的非线性降维方法，绘制了二维和三维的辛烷值和硫含量的变化轨迹实现了模型的可视化展示。

在问题一中，先只针对附件 3 的 285 和 313 号两个样本按照所给的 5 种方法进行整定，求解发现只有产品辛烷值和辛烷值损失与附件 1 给定的数据略有差别，考虑到化工采样方法可能不同，这一微小差别能够满足工程需要，因此采用附件 1 数据进行后续研究。

在问题二中，在问题一的基础上首先对附件 1 的样本数据进行整定，设计了两阶段整定算法，迭代 7 次后将 325 个样本、367 个变量筛选为 252 个样本、230 个变量。然后，提出综合筛选模型，将特征选择和降维方法中代表性的相关系数矩阵法、距离相关系数法、递归特征消除（RFE）、主成分分析（PCA）这 4 种方法分别求解产品辛烷值与其余 229 个变量的相关性，并综合 4 种方法的优点。进一步的，通过将非操作变量和操作变量分别单独建立与产品辛烷值的相关性模型，得到改进的综合筛选模型，增强筛选变量的代表性；寻找 229 个自变量之间的相关度，剔除相关度高的变量，增强筛选变量的独立性。最终，在 Python 的环境下编写程序，得到硫含量、原料辛烷值、饱和烃、烯烃、S. 1、K−101B 右

排气温度等共 30 个主要影响变量，其中 5 个非操作变量，25 个操作变量。

在问题三中，将问题二得到的 30 个主要影响变量作为批处理，建立 LSTM 预测模型，选取训练集：测试集=80%：20%，分析了输入输出维度和结构，并建立了 LSTM 预测模型的评价指标。然后，采用 GA 算法改进 LSTM 的参数选择，进一步增强预测精度。同时，为验证预测算法的有效性，对 RF、SVR、XGBoost 这三种代表性的预测方法在 Python 环境下实现，与 LSTM 预测结果进行对比。最终，GA 改进的 LSTM 模型预测精度能够满足工程要求，而且较其他三种方法其 RMSE、MAPE 指标均具有优势，结果如下表：

	LSTM	XGBoost	SVR	RF
RMSE	0.19269	0.21558	0.28495	0.26148
MAPE	0.49425	0.52689	0.59863	0.56217

在问题四中，由于优化过程需要约束含硫量，因此也需要按照相似的方法建立硫含量的预测模型。运用转化的思想，通过建立预测精度更高工程上更实用的脱硫率预测模型来表征含硫量，每一样本的脱硫率预测精度在 0.21%-0.67%之间，满足工程要求。建立 GA 算法下的适应度函数，并将含硫量的约束推导为障碍函数置于目标函数。综合考虑算法求解性能与工程实际意义，改善 GA 算法的初始种群设计，使得 GA 算法不但具有良好的搜索能力，而且具有响应速度和梯度优势。为保证优化过程符合工程实时工况的短时特性，设置迭代上限为 4 时，325 个样本有 252 个样本辛烷值损失降幅超过 30%，而有 73 个样本未超过 30%，达标率为 77.54%。这是由于部分样本迭代次数达到程序设定的上限，若将迭代次数改为 5 次，则达标率为 90.64%，但时间会更长，因此会存在达标率和迭代时间的权重问题，实际工程可根据情况进行权重赋值。求解得到 325 个样本的操作条件，并对 66 号样本单独举例分析了其操作条件以及 GA 的优化过程。同时，通过 66 号样本展示 PSO 和 GA 算法在辛烷值优化的优劣，GA 下的辛烷值损失降幅大于 PSO 下的辛烷值损失降幅，这是由于 PSO 容易陷入局部最优，造成较快收敛，而 GA 的速度较慢。

在问题五中，在问题四得出的操作变量优化结果的基础上，对 133 号样本等步长均匀地调整操作变量 1519 次，绘制了辛烷值和硫含量随操作变量调整次数的变化曲线，调整完步使得辛烷值损失降幅达 50%，硫含量下降，进一步验证了前述模型的正确性。随后采用 t-SNE 非线性降维方法，将 25 维操作变量降至二维，绘制二维、三维图以表征辛烷值和硫含量的变化轨迹并指出变化方向。

上述全部过程的结果与图表详见正文。

本文的创新点和亮点有：第一，考虑化学工艺流程，结合实际，保证理论实际相结合；第二，建立了问题之间的联系，使得整个问题具备整体性；第三，在筛选主要变量的过程中，提出综合筛选模型并进一步改进，综合了 4 种特征选择和降维方法的优点，使得筛选效果更佳；第四，考虑筛选变量的代表性和独立性，将非操作变和操作变量分别单独与辛烷值建立相关性模型，并寻找自变量间的相关性；第五，对比了其他 3 种预测方法，凸显 LSTM 预测模型在辛烷值预测时的优势；第六，通过转化为脱硫率表征较为准确的硫含量，并建立了以辛烷值的 GA 优化模型；第七，对比 PSO 和 GA 两种算法在辛烷值优化中的优势；第八，采用 t−SNE 降维方法，相较于 PCA 等线性降维方法能够更好地区分和判断辛烷值和硫含量的变化轨迹和变化方向。

关键词：辛烷值，整定，综合筛选模型，改进 LSTM 预测模型，GA 优化算法，变化轨迹

目 录

图 录

表 录

一、问题重述

1.1 研究背景

节能减排这一环境友好型目标自“十一五”规划纲要提出以来，取得了较大进展，众多企业及学者都在各自研究领域对节能减排目标作出了贡献。然而，汽油作为汽车排出温室气体的主要源头，特别是催化裂化汽油在车用汽油的比例较高，对节能减排造成了不小的压力[1]。因此，通过寻求合适、有效的方法，以保证汽油的清洁化生产、实现绿色工艺具有重要意义。在化工领域，汽油的清洁化重点围绕降低汽油中的硫、烯烃含量，同时尽量保持其辛烷值。

在我国，原油对外依存度超过 70%，且大部分是中东地区的含硫和高硫原油。原油中的重油通常占比 40-60%，这部分重油难以直接利用。为了有效利用重油资源，我国大力发展以催化裂化为核心的重油轻质化工艺技术，将重油转化为汽油、柴油和低碳烯烃，超过70%的汽油是由催化裂化生产得到，因此成品汽油中 95%以上的硫和烯烃来自催化裂化汽油。故必须对催化裂化汽油进行精制处理，以满足对汽油质量要求。目前，汽油精制技术已逐渐成熟，主要包含汽油加氢精制技术、汽油吸附脱硫技术 S-ZORB、萃取脱硫技术等。其中，汽油吸附脱硫技术 S-ZORB 凭借脱硫率高(脱硫后可达 10μg/g 以下,满足国 V 标准)、辛烷值损失小、氢耗低、能耗低的优点,已经成为中国石化汽油质量升级的主要技术手段,并在国内汽油脱硫生产中得到迅速推广。同时，本文的数据来源也均为 S-ZORB 装置所提供，因此在下文的建模过程中会考虑 S-ZORB 的工艺流程对辛烷值优化的影响。

辛烷值（RON）是反映汽油燃烧性能的最重要指标,并作为汽油的商品牌号（例如 89#、92#、95#）。现有技术在对催化裂化汽油进行脱硫和降烯烃过程中，普遍降低了汽油辛烷值。辛烷值每降低 1 个单位，相当于损失约 150 元/吨。以一个 100 万吨/年催化裂化汽油精制装置为例，若能降低 RON 损失 0.3 个单位，其经济效益将达到四千五百万元。然而，传统的辛烷值建模方法一般是通过数据关联或机理建模的方法来实现的。文献[2]研究表明，提高反应温度和增大进料空速有利于提高芳烃的选择性,提高辛烷值含量；文献[3]提出对催化裂化轻循环油（LCO）加氢-催化裂化组合，以生产高辛烷值的汽油。由于炼油工艺过程的复杂性以及设备的多样性，操作变量之间具有高度非线性和相互强耦联的关系，而且传统的数据关联模型中变量相对较少、机理建模对原料的分析要求较高，上述传统优化方法对过程优化的响应不及时，所以效果并不理想。本文也将立足于探索新的汽油辛烷值优化方法，以对经济、环境的发展均产生理论支撑。

近年来，随着信息技术的发展，数据挖掘技术已逐步应用于化工过程。数据挖掘技术是从数据中发现所蕴含的规律，是通过建立数据驱动的数学模型来优化操作条件的重要方法[4]。因此，亟需通过数据挖掘技术更好的解决汽油辛烷值的建模和优化问题，通过大量历史数据寻求相关性强的操作变量，设计精确的预测模型，并在此基础上不断优化各操作变量，在保证汽油产品的含硫量较低的情况下尽量减小辛烷值损失，以提高汽油品质，促进环境友好型建设。

1.2 问题的提出

本文立足于 S-ZORB 化工技术，以题目给定的催化裂化汽油精制装置采集的 325 个数据样本为分析对象，通过数据挖掘技术来建立汽油辛烷值（RON）损失的预测模型，并设计优化模型计算每个样本的优化操作条件，在保证汽油产品脱硫效果的前提下尽量降低汽油辛烷值损失在 30%以上。下面将对该数学问题进行提出和描述。

整个数学问题是递进描述的。参考近 4 年的工业数据，首先需要理解原料性质、产品性质、待生吸附剂性质、再生吸附剂性质以及操作变量的物理意义，并对 325 个样本数据

进行预处理，以降低设备测量误差等不可控因素的影响，提高数据的可用性。在预处理的基础上需要建立降低辛烷值损失模型，面对几百个操作变量的情况下，工程上需要降维处理，找到主要影响因素，在此过程中不仅要考虑历史数据还要考虑工艺流程的实际情况，才能建立满足工程应用需求的辛烷值优化模型。在前两步的基础上选取主要影响变量，利用数据挖掘技术建立辛烷值损失预测模型，并验证模型的准确性，得到一般性的数学模型。至此，根据原始数据经过预处理、降维形成了辛烷值损失模型，进一步的，需要利用该预测模型对辛烷值损失进行优化，根据优化结果调整操作变量数值，实现含硫量达标的同时尽量减小辛烷值的损失。最终，将优化方案可视化展示，形成汽油辛烷值和硫含量的变化轨迹，对石化公司产生一定的借鉴作用。基于上述问题的提出，本文将创新性的从数据预处理、特征选择与降维、建立辛烷值预测模型、优化辛烷值的操作条件、可视化展示变化轨迹等几个方面递进式的展开论述和编程证明，为化工企业提供可靠有效的辛烷值建模和优化方法，起到借鉴作用。

因而可以概括的说，该问题的本质是一个利用数据挖掘技术对汽油辛烷值实现精确预测，并对其进行优化的多变量、多过程的数学问题。

问题一：化学工艺的数据采集往往会伴随着数据的不准确，因此需要通过整定预处理改善不良数据。操作变量的不良数据主要包括如下类型：对于只含有部分时间点的位点且残缺数据较多、数据全部为空值的位点、部分为空值的位点、超出原始变量操作范围、不符合 3σ 准则。问题一主要是针对附件 3 中的 285 号和 313 号两个样本进行整定。

问题二：面对 367 个变量，不可能在辛烷值模型的建立时将全部的变量考虑进去，而是需要通过特征选择和降维的方法对变量进行筛选，筛选得到主要影响变量，从而简化模型维度，并保证筛选后的变量具有独立性和代表性。然而，325 个样本和 367 个变量不一定都具备筛选的条件，部分样本和部分变量需要利用整定方法重新进行整定。在整定处理后需要建立综合筛选模型，对多种代表性方法建模，综合特征选择和降维方法的优点，对各变量有效筛选在 30 个以内，并与相关文献对比验证模型结果的正确性。

问题三：在经历了问题一和问题二的建模和求解后，递进到问题三辛烷值损失预测模型的建立。由于已经对数据进行了整定预处理，并且筛选得到了主要变量，则在建立预测模型时只根据主要影响变量建立即可。面对大量样本大量数据的情况，需要寻找适合于辛烷值预测的数据挖掘技术，选择训练集和测试集对预测模型进行验证，同时需要提出能够改进深度学习预测模型参数选择的算法，以保证预测模型的准确度。同时，考虑到原料辛烷值是定值，而辛烷值损失并不是变量，所以更好的思路是首先对产品辛烷值进行预测，再将两者作差，可以实现对辛烷值损失预测模型的建立。辛烷值损失预测模型也是问题四的基础，其精度要求也是优化结果是否准确的关键，需要在建模时减小误差。

问题四：问题三建立了辛烷值的预测模型，对辛烷值通过数据挖掘技术实现了精确的建模，而下一步则是要根据辛烷值模型对其操作变量进行优化调整，使得满足工程需要。本题的要求是产品硫含量不大于 5μg/g，辛烷值（RON）损失降幅大于 30%。也就是说，在对全部 325 个样本对应的主要影响变量优化后，能够满足上述要求，主要影响变量向最优的方向调整就是操作条件。因此，需要寻找合适的优化算法，尽量在这一算法下使得全部样本都可以满足该指标要求。同时，需要注意原料性质等非操作变量是不可被优化的，只有操作变量可以被优化，这也是和工程实际相契合。

问题五：问题四是问题五的基础，在找到合适的优化模型后，由于各变量均按照各自的步长进行调整。工业装置为了平稳生产，往往只能逐步调整到位，因此有必要将产品辛烷值、含硫量与各主要影响变量的关系通过可视化图像的方式加以展现，也起到给予实际操作人员以操作方案的作用。因此，需要根据优化模型建立辛烷值和含硫量的变化轨迹。

二、问题分析

2.1 问题一的分析

问题一是只针对附件 3 的 285 和 313 号两个样本的数据整定问题，主要难点在于根据样本整定方法确定每一种数据异常类型的操作办法。附件 1 的 325 个样本的整定将放在问题二中分析和求解。

查阅相关文献及研究工艺流程后，探索得到如下具体的数据处理方法：

（1）对于只含有部分时间点的位点，如果其残缺数据较多，无法补充，将此类位点删除。本文依据 Bijlsma 提出的 80%准则判定数据缺失的多少，即当某一位点的所有时间点非缺失部分低于总量的 80%时，建议删除该位点。

（2）删除数据全部为空值的位点；

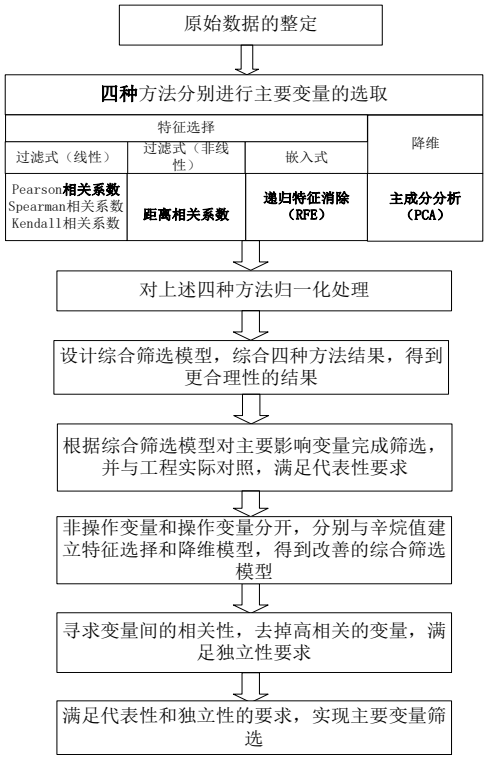
（3）285 和 313 两个样本用其两个小时数据的平均值代替；

（4）根据工艺要求与操作经验，总结出原始数据变量的操作范围，然后采用最大最小的限幅方法剔除一部分不在此范围的样本；

（5）根据拉依达准则（3σ 准则）去除异常值。

2.2 问题二的分析

问题二面向的是附件 1 的全部 325 个样本各自对应的 367 变量，需要从中筛选主要变量，并作为问题三和问题四的预测和优化变量。而在筛选变量时首先需要对全部的样本数据进行整定，包含问题一所述的五种整定类型。整定完成后提出特征选择和降维的方法，并建立综合筛选模型，以综合各方法的优势。进一步的，为保证筛选变量更具有代表性，改善综合筛选模型，将非操作变量和操作变量分开单独建立辛烷值的相关度模型，得到改进综合筛选模型下的变量。同时，这些变量之间可能存在高相关，为保证独立性，需要对改进综合筛选模型得到的变量进行进一步的筛选，利用 Pandas 相关度分析，保留高相关度变量中对工艺流程影响更大的变量。经过上述几个过程，得到了题目需要的主要影响变量。与文献得到的结论进行对比，验证模型的正确性。为清晰化表述，设计如下的流程图展示：



2.3 问题三的分析

问题三是问题一和问题二的递进，需要在求解得到的主要影响变量的基础上，建立产品辛烷值损失的预测模型。本文选择 LSTM 网络对辛烷值损失进行预测，为保证预测精度，减小误差，先对产品辛烷值进行预测再与原料辛烷值作差得到辛烷值损失的预测结果。

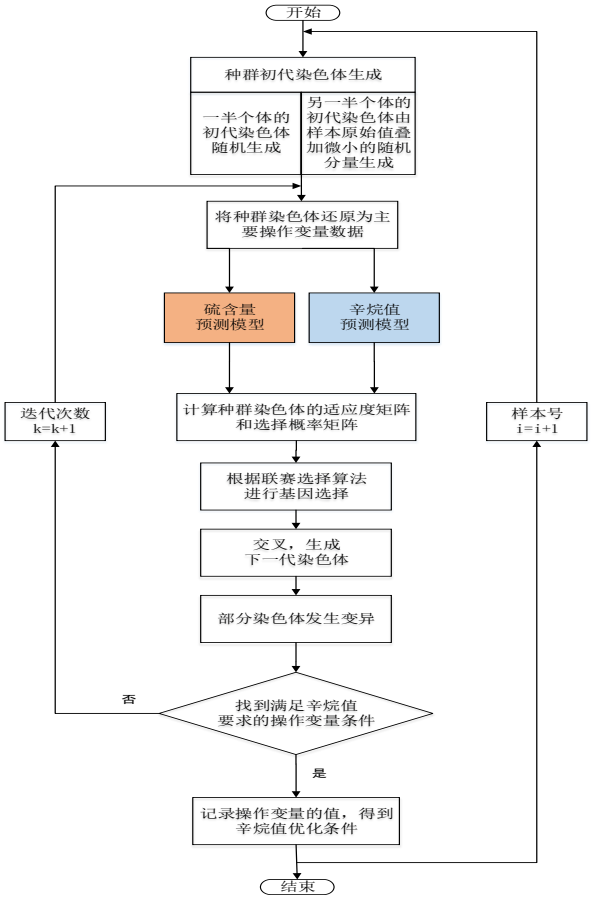
首先需要分析数据驱动技术与 LSTM 关系，LSTM 是深度学习的一种方法，属于数据驱动技术的范畴。然后，对 LSTM 模型进行建模，分析输入输出的维度和训练集、测试集的维数，并作出 LSTM 的基本结构图。同时，为了提高 LSTM 模型的参数适应性，利用 GA 算法优化 LSTM 的参数选择，提高预测的准确度。为通过对比验证 LSTM 预测模型的有效性，提出 RMSE 和 MAPE 两个评价指标。最后，通过 Python 语言求解辛烷值的预测模型，并最终得到辛烷值损失的预测结果与真实值的对比，并进行误差分析。

进一步的，为验证 LSTM 模型在辛烷值预测中的优势，对 RF、SVR 以及 XGBoost 三种预测算法进行简述，并通过 Python 语言实现对辛烷值的预测，再与 LSTM 进行对比，若 LSTM 的 RMSE 和 MAPE 指标更好，则可以从侧面证明 LSTM 模型在辛烷值预测的优势。

2.4 问题四的分析

问题四是建立在问题三辛烷值预测模型的基础上而进一步对辛烷值进行优化。由于在对辛烷值进行优化的过程中硫含量作为约束条件加入，因此也有必要对硫含量进行建模。按照问题二和问题三的方法，对硫含量建立其预测模型，同样是包含有非操作变量和操作变量。并且，脱硫率和硫含量紧密相关，且较多文献证明对脱硫率进行预测精度更高且实用性更好，因此转化为对脱硫率的预测。

在对其建模后，需要利用辛烷值和硫含量的数学模型，选取合适的方法，本文选取遗传算法（GA），粒子群算法（PSO）作为对比。整个流程如下图所示：



2.5 问题五的分析

问题五是问题四的延伸，在问题四中求解出 133 号样本的优化操作变量后，依据操作变量单次最大调整幅值以及工业生产装置平稳运行的需求，本文制定了平均步长的优化策略，并依据辛烷值和硫含量的预测模型绘制了它们随调整次数变化的曲线。

然后为可视化展示模型，采用非线性算法的 t-SNE 降维，相比于线性化降维方法，它在降维后能够有效区分样本点。需要绘制辛烷值和硫含量的二维三维散点变化图（即变化轨迹），并可以判断变化轨迹方向。

三、模型假设

1、假设汽油辛烷值的测量数据准确，不需要对原料辛烷值和产品辛烷值这两个量进行整定。

2、假设汽油辛烷值和硫含量仅可能与 367 个变量有关，不再有其他变量的参与。

3、假设各样本均存在偶然性，也就是不好的样本是可以允许被剔除的。

4、假设附件 2 给定的 5 种整定方法即可完成对数据的良好预处理，不再需要其他整定方法的判定。

5、假设主要变量在 30 个及以下，且非操作变量和操作变量不能全为空。

6、由于部分变量间存在强耦合，假设 Pandas 调包后计算相关度超过 0.9 的视为两个变量高相关，可以剔除一些相关变量。

7、假设相关文献提出的化学工艺影响变量准确，则可以作为筛选结果的对照。

8、假设在预测模型中原料辛烷值是不变的量，也就是可以通过预测产品辛烷值来预测辛烷值损失。

9、假设优化过程中只有操作变量可以调整，且可以调大也可以调小，调整步长依据附件 4 的步长要求。

10、假设优化过程中 325 个样本是独立的，不存在相互影响。

11、硫含量可以用脱硫率来表达，假设脱硫率模型的操作变量与辛烷值的操作变量可以去交集来确定。

12、假设优化得到的辛烷值和硫含量的值均不能超过原料辛烷值和原料硫含量。

13、假设优化时同样也考虑工程实际，辛烷值损失降幅的目标是达到 30%以上，不一定追求全局最优而造成时间过长耽误工艺。

14、假设在形成变化轨迹时操作变量每一步的调整均为定步长。

四、名词解释与符号说明

4.1 补充名词解释

●汽油辛烷值[5]：汽油辛烷值是衡量汽油在气缸内抗爆震燃烧能力的一种数字指标，其值高表示抗爆性好。

●催化裂化：催化裂化是石油炼制过程之一，是在热和催化剂的作用下使重质油发生裂化

反应，转变为裂化气、汽油和柴油等的过程。

●数据挖掘：数据挖掘是指从大量的数据中通过算法搜索隐藏于其中信息的过程。

●降维和特征选择：降维和特征选择都是为了使数据维度减小，但两者依然存在着本质的区别。降维本质上是从一个维度空间映射到另一个维度空间，特征的数目没有改变，而特征值发生改变；特征选择本质上是选择部分特征作为训练集，特征的数目改变了，但特征值没有改变。

●拉依达准则（3σ准则）[6]：指先假设一组检测数据只含有随机误差，对其进行计算处理得到标准偏差，按一定概率确定一个区间，认为凡超过这个区间的误差，就不属于随机误差而是粗大误差，含有该误差的数据应予以剔除。

4.2 符号说明

本文按照各数学符号出现的先后顺序进行注释，并将重要变量的解释统一列于下表。由于本文变量较多，下文中的部分公式会对该公式的变量进行单独说明。

表 1　全文符号说明一览表

序号	符号	说明
1	ΔR	辛烷值损失值
2	R_r	原料辛烷值
3	R_p	产品辛烷值
4	k	数据整定第二阶段的迭代次数
5	J	数据整定第二阶段的总行数
6	I	数据整定第二阶段的总列数
7	$N_{\max}$	数据整定第二阶段的剔除临界值
8	$N_{\min}$	数据整定第二阶段的迭代临界值
9	x	每一列的均值
10	σ	每一列的标准差
11	$A_{m\times n}$	252×229 的筛选后样本矩阵
12	$X_{m\times n}$	$A_{m\times n}$ 标准化矩阵
13	s_j	影响变量的标准差
14	y	每一种方法下归一化后的列向量
15	$y_{\min}$	每一种方法下的列最小值
16	$y_{\max}$	每一种方法下的列最大值
17	y_0	每一种方法下归一化前的列向量

18	RMSE	平方均方误差
19	MAPE	平均相对百分比误差
20	γ	脱硫率
21	S_r	原料硫含量
22	S_p	产品硫含量
23	t	最终需要的总调节次数
24	Δ_i	对应操作变量的单次最大调整量

五、模型的建立与求解

5.1 对问题一的分析与求解

5.1.1 对附件 3 的 285 和 313 号样本进行整定

附件 3 给出了 285 号和 313 号样本的原料性质、产品性质、待生吸附剂性质、再生吸附剂性质共 13 个非操作变量以及 254 个操作变量，且 285 号和 313 号的操作变量按照 3 分钟/次采集数据，分别生成 40 个时间点的 254 个操作位点。其中，附件 3 中的辛烷值损失不作为变量，其数学表达式为：

$$\Delta R = R_r - R_p \qquad (1)$$

由于问题一只涉及 285 和 313 号两个样本的预处理，并不涉及全部 325 个样本，因此在问题一的分析和求解中先对两个样本进行整定，对全部 325 个样本的整定将于问题二中分析和求解。

数据处理方法如下：

（1）对于只含有部分时间点的位点，如果其残缺数据较多，无法补充，将此类位点删除。本文依据 Bijlsma 提出的 80%准则判定数据缺失的多少，即当某一位点的所有时间点非缺失部分低于总量的 80%时，建议删除该位点[7]。换言之，当残缺数据超过 20%时，可以删除该位点。

（2）删除数据全部为空值的位点；

（3）285 和 313 两个样本用其两个小时数据的平均值代替；

（4）根据工艺要求与操作经验，总结出原始数据变量的操作范围，然后采用最大最小的限幅方法剔除一部分不在此范围的样本；

（5）根据拉依达准则（3 σ 准则）去除异常值，设对被测量变量进行等精度测量，得到 x_1，x_2，……，x_n，算出其算术平均值 x 及剩余误差 $v_i=x_i-x$（i=1, 2, . . . , n），并按贝塞尔公式算出标准误差 σ，若某个测量值 x_b 的剩余误差 v_b（1<=b<=n），满足$|v_b|=|x_b-x|>3\sigma$，则认为 x_b 是含有粗大误差值的坏值，应予剔除。贝塞尔公式如下：

$$\sigma=[\frac{1}{n-1}\sum_{i=1}^{n}v_i^2]^{1/2}=\{[\sum_{i=1}^{n}x_i^2-(\sum_{i=1}^{n}x_i)^2/n]/(n-1)\}^{1/2} \qquad (2)$$

5.1.2 问题一的求解结果及分析

按照上述问题一的数据处理方法，得到附件 3 的 285 和 313 号样本的数据处理结果，并与附件 1 给定的数据进行比对，发现只有产品性质中的辛烷值这一变量有所不同，同时导致辛烷值损失与附件 1 也有略微差别，其余变量的数据结果均与附件 1 的完全一致。列

出如下表所示：

表 2　问题一结果与附件 1 的对比结果

	样本号	产品辛烷值	辛烷值损失
附件 3（处理后）	285	**88.1**	**1.2**
	313	**88.9**	**1.4**
附件 1（给定）	285	**88.18**	**1.1**
	313	**89.05**	**1.2**

对 285 号和 313 号两个样本进行拉依达准则的判断，并且只关注非操作变量，验证得到均满足拉依达准则。

根据上表对比结果可以看出，无论是 285 还是 313 号样本，附件 3 处理后求得的产品辛烷值和辛烷值损失都与附件 1 存在略微差别，且偏高。但考虑到化工流程在采样方法的不同，可能会导致这一微小误差的存在，在实际建模时满足工程要求即可，在本文中统一采用附件 1 的数据。

5.2 对问题二的分析与求解

5.2.1 附件 1 全部 325 个样本的数据整定

在对问题一的求解中主要针对 285 和 313 两个样本的整定，验证了附件 1 中两个样本数据的准确性。根据实际情况可以认为辛烷值的测量值是测量时刻前两小时内操作变量的综合效果，因此预处理中取操作变量两小时内的平均值与辛烷值的测量值对应，产生了来自工厂给定的附件 1 数据。

问题二的关键是寻求建模主要变量，本文为了探讨最适合于汽油辛烷值建模的数学模型，对比降维和特征选择的 4 种方法，并考虑工程实际，从代表性和独立性两个角度来对 4 种方法下得到的主要影响变量进行定性和定量分析，最终得到最适合于汽油辛烷值的数据挖掘技术下的数学模型。

然而，在降维和筛选变量之前首先需要对全部 325 个样本数据进行整定。考虑到拉依达准则去除异常值的算法需要不断迭代，区别于空值筛选和最大最小范围筛选，因此在流程设计上采用了两阶段设计。第一阶段包含对空值较多或全为空值的筛选以及全部样本操作变量最大最小范围的筛选，第二阶段为在第一阶段的筛选结果上按照拉依达准则进行筛选，两阶段均采用 Python 语言代码实现。

第一阶段的整定算法流程如下图 1 所示。

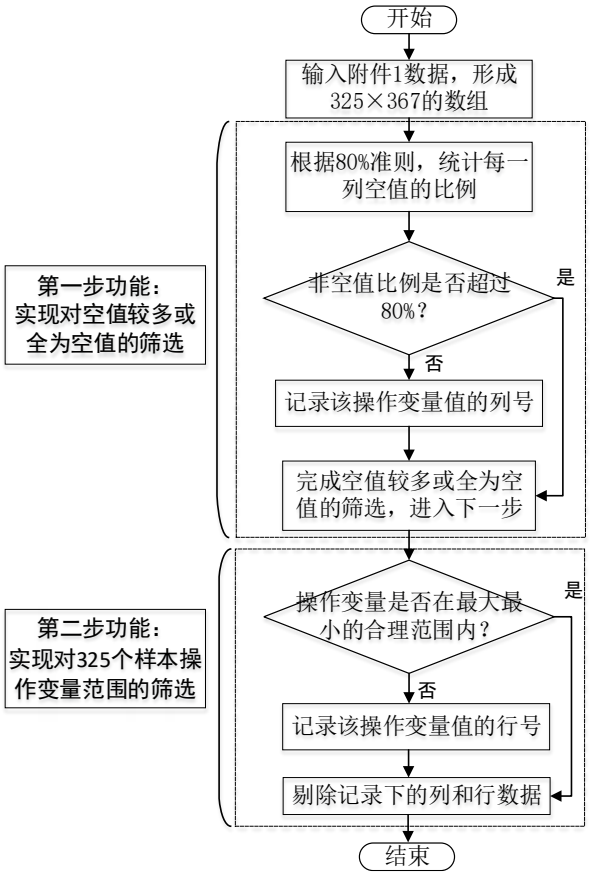


图 1 全部样本的第一阶段整定算法流程

对第一阶段的整定算法进行分析：在空值较多或全为空值的情况下，反映的是操作变量数据的不合理，因此剔除对应列；在操作变量的范围不在最大最小允许范围内的情况下，反映的是样本的不合理，因此剔除对应行。

第二阶段的整定算法流程如下图 2 所示。

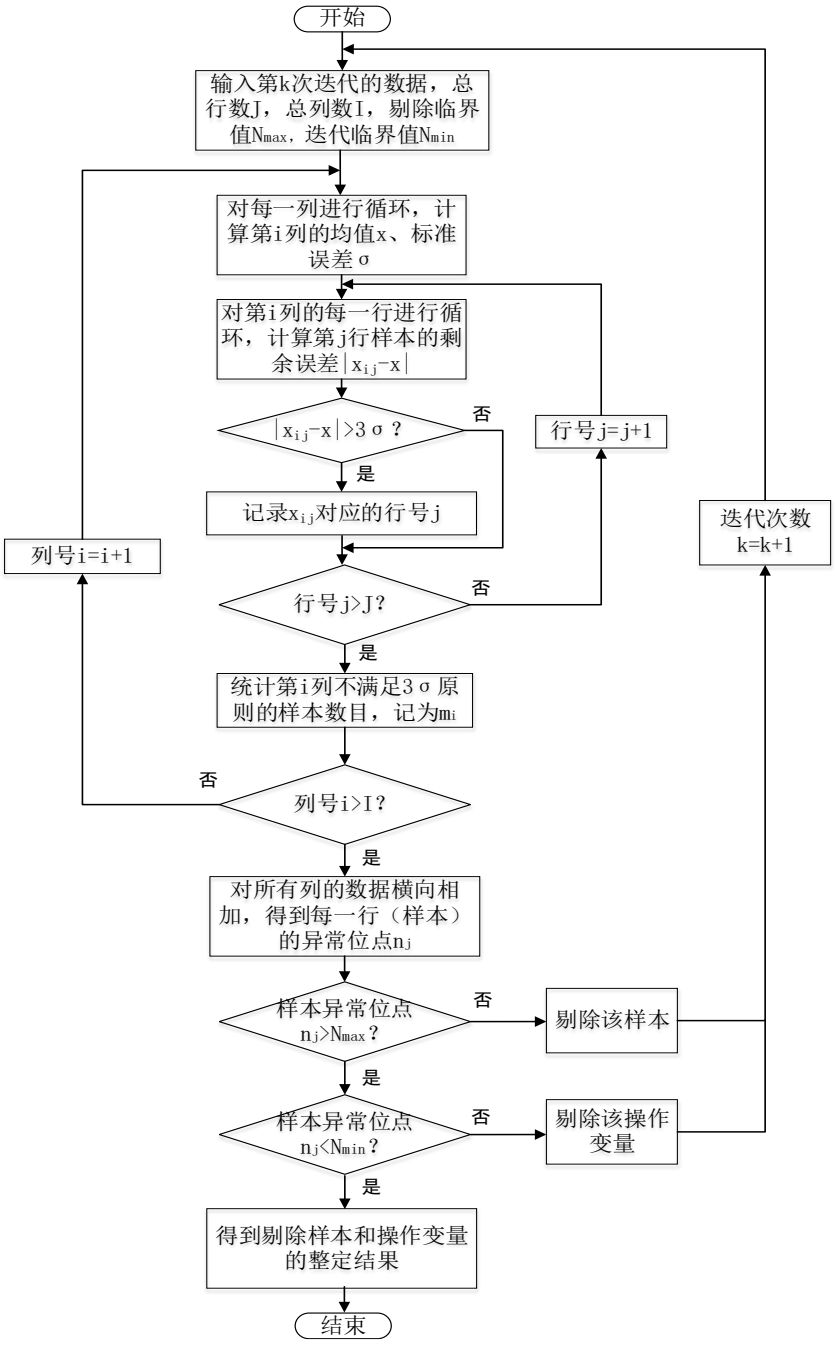


图 2 全部样本的第二阶段整定算法流程

对第二阶段的整定算法进行分析：第二阶段是拉依达准则在辛烷值数据整定的应用，文献[6]和文献[8]均对拉依达准则应用在样本数据的整定中做出了研究，提出了循环剔除异常值的方法。因此，本文将该算法应用在汽油辛烷值的数据整定中。首先，对列进行循环，判断每一样本下的操作变量是否满足 3σ 准则，统计其数目；然后，再对行进行循环，判断每一行出现的不满足 3σ 准则的数目是否超过上限阈值，超过则代表该样本不具有代表性，此时认为影响更大的是样本而不是操作变量，所以对样本进行剔除，反之则不剔除；最后，对循环迭代后依然无法使得每一行满足 3σ 准则的列进行剔除操作，得到了均满足 3σ 准则的预处理数据。

5.2.2 附件 1 的数据整定结果及分析

根据上述数据整定算法可以实现对原始数据效果更佳的筛选，采用 Python 语言编程实现，得到附件 1 的数据整定结果。这里由于篇幅原因，仅给出最终整定筛选后的样本数和操作变量数，不再给出具体剔除的行号和列号，而是放于附件中。

表 3 整定前后的附件 1 数据情况对比

	样本数	操作变量数	迭代次数
整定前	325	367	7
整定后	252	230	

对上述结果进行分析：部分样本由于采样方法或者采样点不适合等因素，已不适合于作为建立辛烷值模型的数据，这里给出循环过程中对于 325 个样本不满足 3σ 准则的统计数据如图 3 所示。

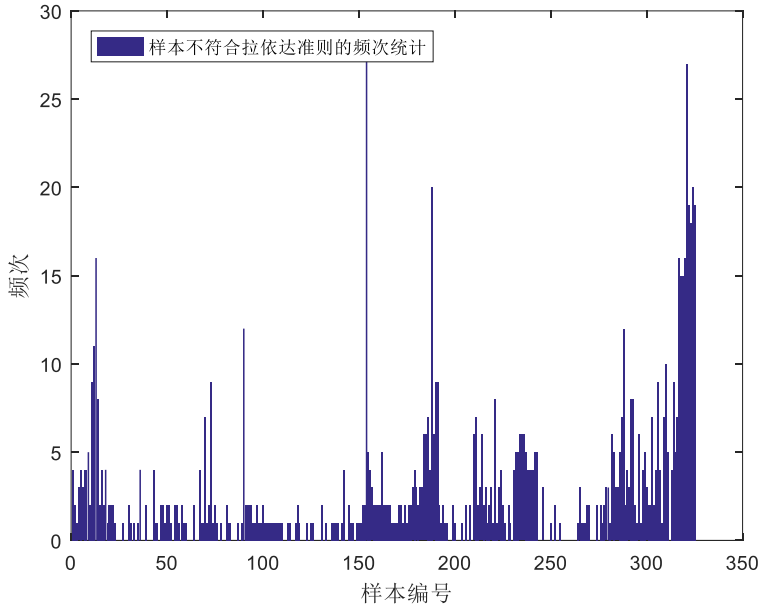


图 3 325 个样本不满足 3σ 准则的统计

如 153 号样本，不满足 3σ 准则的操作变量点高达 28 次，并不能认为是这 28 个操作变量的问题，而是样本统计可能出现了问题或该采样时间不合适，因此此时为保证后续辛烷值建模的准确性以及预测模型的准确性，将该样本剔除。

而对于每一列而言，也就是对于每一个变量而言，依然存在较多不合适的数据，如 96 号新氢进装置流量这一操作变量存在大量为空值的情况，在程序结果中也是由第一阶段整定算法进行了剔除；如 49 号样本存在附件 1 和附件 3 的取值范围不一致的问题，也是由第一阶段整定算法进行了剔除。

综合上述结果及分析，整体样本的变量已知数据均已进行了整定，实现了数据的优化和筛选，为进一步确定主要操作变量打下了基础。

5. 2. 3 主要影响变量的筛选流程

问题二的关键是对 367 个影响变量（不包含辛烷值损失）进行筛选，找到主要影响变量。在上两小节的研究中已经对原始数据进行了整定，优化了数据的可行性和合理性。在对预处理后的数据进行筛选的过程中，可以主要分为两大类方法，包含特征选择和降维。其中，特征选择主要分三大类：过滤式(Filter)、包裹式(Wrapper)、嵌入式(Embedded)；降维

方法常用的比较典型的方法是主成分分析（PCA）。然而，每一种方法都有其优点和缺点，在对复杂的化学工艺流程进行提取主要影响变量的建模中，需要以精确度和合理性为主要依据，相关影响变量可能不仅是线性相关，也可能非线性相关，同时模型的稳定性对提取结果也有较大影响。

基于上述分析，本文实现了四种特征选择和降维方法，并对四种方法进行归一化处理，以代表性和独立性为指标，设计了综合筛选模型，并调用 Pandas 工具包对各变量之间的相关度分析，去除相关度高的部分变量，最终得到符合条件的主要影响变量。整个筛选流程如图 4 所示。

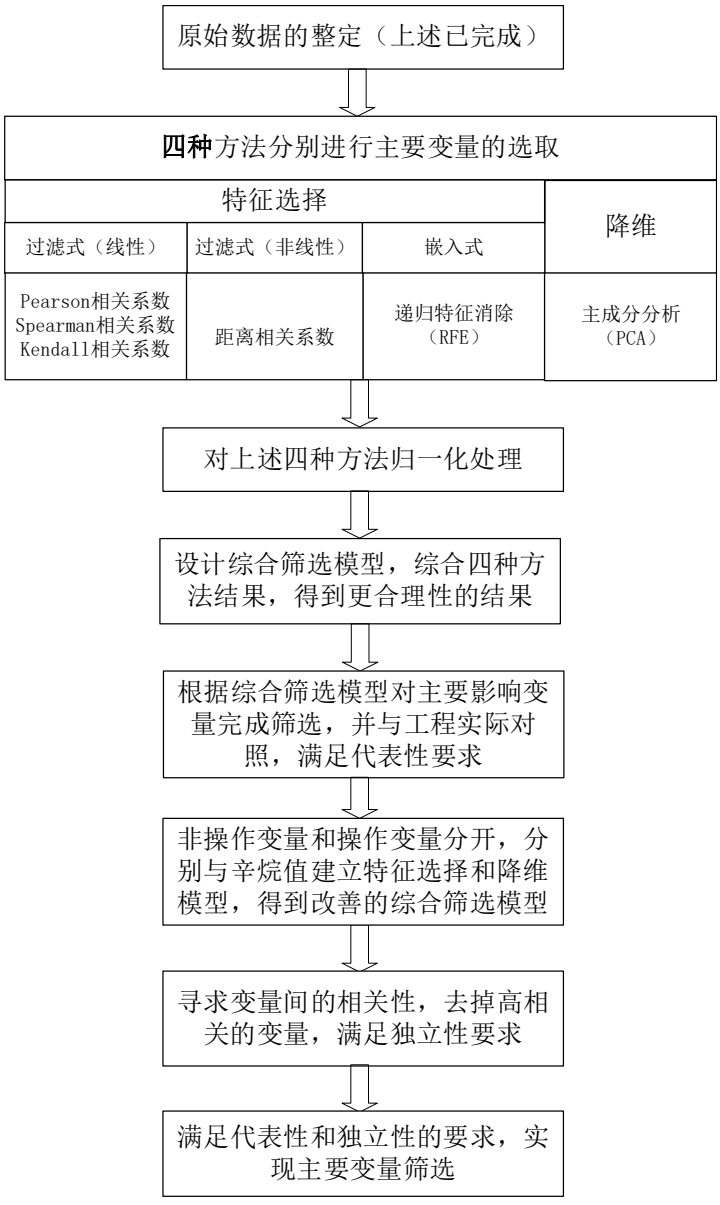


图 4 主要影响变量筛选流程框架

下面，首先对四种方法分别建模，进而设计一套综合筛选模型，并进行代表性和独立性的改善，最终得到主要影响变量。前两小节得到了整定后的 230 个变量，在下面的建模中选择产品的辛烷值 RON 作为因变量，其余 229 个变量作为特征，分别通过相关系数矩阵（Pearson\Spearman\Kendall）、距离相关系数、主成分分析（PCA）、递归特征消除法（RFE）评价各个特征与因变量之间的相关度。

5. 2. 4 相关系数矩阵法

相关系数矩阵法主要包含 Pearson、Spearman、Kendall 三种典型算法，本小节仅建立其主要的数学模型，而不再赘述公式推导。

假设两个随机变量分别为 X、Y，它们的样本个数均为 N，两个随机变量取的第 i 个值分别用 X_i、Y_i 表示（1<=i<=N）。X 与 Y 中的对应元素组成一个元素对集合 XY，其包含的元素为(X_i, Y_i)（1<=i<=N）。当集合 XY 中任意两个元素(X_i, Y_i)与(X_j, Y_j)的排行相同时（也就是说当出现情况 1 或 2 时；情况 1：$X_i>X_j$ 且 $Y_i>Y_j$，情况 2：$X_i<X_j$ 且 $Y_i<Y_j$），这两个元素就被认为是一致的。当出现情况 3 或 4 时（情况 3：$X_i>X_j$ 且 $Y_i<Y_j$，情况 4：$X_i<X_j$ 且 $Y_i>Y_j$），这两个元素被认为是不一致的。当出现情况 5 或 6 时（情况 5：$X_i=X_j$，情况 6：$Y_i=Y_j$），这两个元素既不是一致的也不是不一致的。

（1）Pearson 相关系数

$$r=\frac{\sum_{i=1}^{n}\left(X_i-\bar{X}\right)\left(Y_i-\bar{Y}\right)}{\sqrt{\sum_{i=1}^{n}\left(X_i-\bar{X}\right)^2}\sqrt{\sum_{i=1}^{n}\left(Y_i-\bar{Y}\right)^2}} \quad (3)$$

公式定义为： 两个连续变量(X,Y)的 Pearson 相关性系数 r 等于它们之间的协方差 cov(X,Y)除以它们各自标准差的乘积(σX, σY)。系数的取值总是在-1.0 到 1.0 之间，接近 0 的变量被成为无相关性，接近 1 或者-1 被称为具有强相关性[9]。

（2）Spearman 相关系数

$$\rho=1-\frac{6\sum d_i^2}{n\left(n^2-1\right)} \quad (4)$$

对顺序变量往往使用 Spearman 相关系数，Spearman 相关系数的计算采用的是取值的等级，而不是取值本身。例如，给定三个值：33，21，44，它们的等级就分别是 2，1，3。计算 Spearman 相关系数的公式与计算 Pearson 相关系数的类似，但用等级代替了各自的取值[10]。

（3）Kendall 相关系数[11]

$$\tau_B=\frac{n_c-n_d}{\sqrt{\left(n_0-n_1\right)\left(n_0-n_2\right)}} \quad (5)$$

$$n_0=\frac{n(n-1)}{2},\quad n_1=\frac{\sum_i t_i\left(t_i-1\right)}{2},\quad n_2=\frac{\sum_j u_j\left(u_j-1\right)}{2} \quad (6)$$

其中 n_c 表示 XY 中拥有一致性的元素对数；n_d 表示 XY 中拥有不一致性的元素对数。n_1、n_2 分别是针对集合 X、Y 计算的，其中 n_1 的计算是将 X 中的相同元素组合成小集合，t_i 表示第 i 个小集合所包含的元素个数，n_2 同理。

5. 2. 5 距离相关系数矩阵法

距离相关系数是为了克服 Pearson 相关系数的弱点而生的，它主要解决非线性相关的问题，由于辛烷值的影响变量中可能存在非线性的影响因素，因此需要采用距离相关系数法。

即便 Pearson 相关系数是 0，我们也不能断定这两个变量是独立的（有可能是非线性相关）；但如果距离相关系数是 0，那么我们就可以说这两个变量是独立的。将两个随机变量的距离协方差除以它们的距离标准差的乘积，得到它们的距离相关系数。

$$\mathrm{dCor}(X,Y)=\frac{\mathrm{dCov}(X,Y)}{\sqrt{\mathrm{dVar}(X)\,\mathrm{dVar}(Y)}} \quad (7)$$

5.2.6 递归特征消除法（RFE）

递归特征消除的主要思想是反复的构建模型（如 SVM 或者回归模型），并选出最好的的特征，然后在剩余的特征上重复这个过程，直到所有特征都被遍历。这个过程中特征被消除的次序就是特征的排序。因此，这是一种寻找最优特征子集的贪心算法。

RFE 的稳定性很大程度上取决于在迭代的时候底层用哪种模型。例如，假如 RFE 采用的普通的回归，没有经过正则化的回归是不稳定的，那么 RFE 就是不稳定的；假如采用的是 Lasso，而用 Lasso 正则化的回归是稳定的，那么 RFE 就是稳定的。

因此，在算法上首先对 RFE 进行稳定性选择，再选用 Lasso 正则化的的特征回归，调用 Python 工具包求解 229 得到 229 维的变量顺序。

5.2.7 主成分分析（PCA）

不同于上述三种特征选择方法，主成分分析属于降维方法。特征选择与降维的区别已在名词解释部分进行了阐述，这里不再重复。

主成分分析的目的是通过降维的方法，找到最主要成分中的最高载荷所对应的影响变量。

首先，要进行特征参数标准化处理，下面给出标准化过程。

$$A_{m\times n}=\begin{bmatrix} a_{11} & a_{12}\cdots & a_{1n} \\ a_{21} & a_{22}\cdots & a_{2n} \\ \vdots & \vdots\ \vdots & \vdots \\ a_{m1} & a_{m2}\cdots & a_{mn} \end{bmatrix} \quad (8)$$

式中，a_{ij} (i=1, 2, ……, m ; j−1, 2, ……, n) 是第 i 个样本的第 j 个影响变量。对于本文中定义的特征参数，以辛烷值为因变量，其余 229 个变量为自变量，m 取 252，n 取 229。

对矩阵 $A_{m\times n}$ 进行标准化得到标准化矩阵 $X_{m\times n}$:

$$X_{m\times n}=\begin{bmatrix} x_{11} & x_{12}\cdots & x_{1n} \\ x_{21} & x_{22}\cdots & x_{2n} \\ \vdots & \vdots\ \vdots & \vdots \\ x_{m1} & x_{m2}\cdots & x_{mn} \end{bmatrix} \quad (9)$$

$$x_{ij}=(a_{ij}-\overline{a_j})/s_j \quad (10)$$

$$\overline{a_j}=\frac{1}{m}\sum_{i=1}^{m}a_{ij} \quad (11)$$

$$s_j^2=\frac{1}{m-1}(a_{ij}-\overline{a}_j)^2 \quad (12)$$

根据上述特征值参数值表的数据可见，各特征参数之间并非完全独立，存在一定关系的耦合。除此之外，假如对全部特征参数进行分析，数据量大并且重复性高，对于多维变量的问题常用主成分分析法，能保证求解精度的要求。

因此，为了样本的特征信息不丢失，拟采用主成分分析的方法对采样样本的特征参

数进行降维。

首先，由矩阵$X_{m\times n}$计算协方差矩阵$\sum$：

$$\Sigma = \begin{bmatrix} s_j^2 & cov(1,2) & \dots & cov(1,n) \\ cov(1,2) & s_j^2 & \cdots & cov(2,n) \\ \vdots & \vdots & \vdots & \vdots \\ cov(n,1) & cov(n,2) & \dots & s_n^2 \end{bmatrix} \quad (13)$$

$$s_j^2 = cov(x,x) \quad (14)$$

$$cov(x,y) = cov(y,x) = \frac{1}{m-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \quad (15)$$

进而得相关矩阵 R:

$$\mathrm{R}=\frac{1}{m-1}\begin{bmatrix} r_{11} & r_{12}\dots & r_{1n} \\ r_{21} & r_{22}\cdots & r_{2n} \\ \vdots & \vdots\ \vdots & \vdots \\ r_{m1} & r_{m2}\dots & r_{mn} \end{bmatrix} \quad (16)$$

$$r_{xy} = \frac{cov(x,y)}{s_x s_y} \quad (17)$$

这里用λ_i表示矩阵 R 的特征参数，并且假定$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$，计算出相应的正交化特征参数向量为：

$$[e_1 \quad e_2 \quad \cdots \quad e_n]=\begin{bmatrix} e_{11} & e_{12}\dots & e_{1n} \\ e_{21} & e_{22}\cdots & e_{2n} \\ \vdots & \vdots\ \vdots & \vdots \\ e_{m1} & e_{m2}\dots & e_{mn} \end{bmatrix} \quad (18)$$

设$\lambda_i / \sum_{j=1}^{n}\lambda_j$为第$i$个主成分的贡献率。对于每一个主成分，其贡献率越大，则其所能反映的采样样本特征信息就越多，对辛烷值建模的影响就更大。

5. 2. 8 四种方法的结果及分析

采用 Python 语言对上述四种特征选择和降维方法实现，分别得到如下结果。相关系数矩阵与距离相关系数均采用相关度图的形式加以展现，而 RFE 和主成分分析法采用表格的形式加以展现。

首先，对相关系数矩阵中的 3 种典型方法 Pearson 相关系数、Spearman 相关系数、Kendall 相关系数编程实现，得到如图 5、图 6、图 7 所示的相关度结果图。

现对各图中的序号进行解释。经过整定后的变量总数为 230 个，其中 13 个非操作变量，217 个操作变量。以产品的辛烷值作为因变量，其余 229 个变量作为自变量，并将产品的辛烷值设为 229 号，从 0−229 编号。

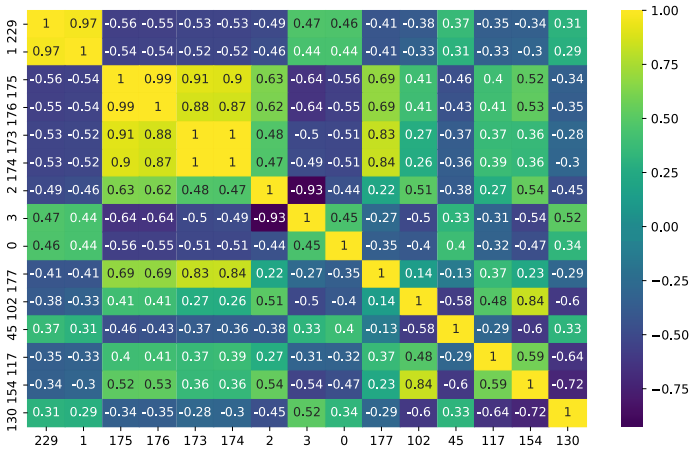


图 5 Pearson 相关系数法结果图

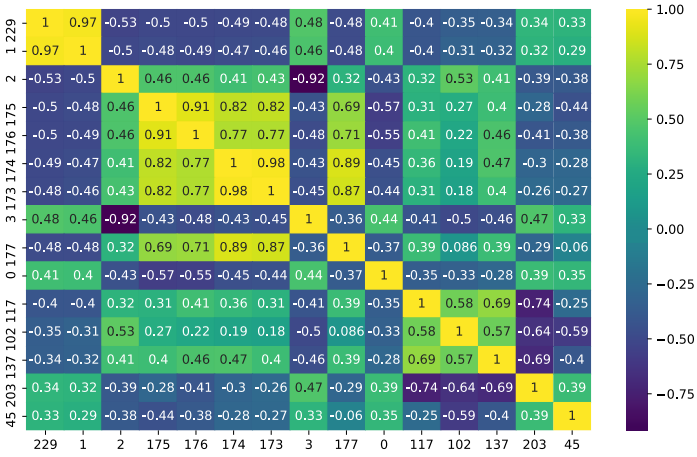


图 6 Spearman 相关系数法结果图

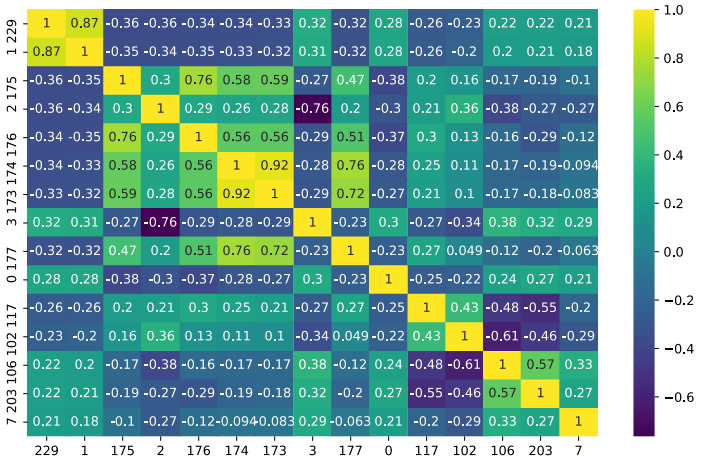


图 7 Kendall 相关系数法结果图

为便于理解，需要对上图的横纵坐标数字对应的变量名称进行解释，在此列出下表，表中的排列顺序按照 Pearson 结果里的横坐标按照从高相关度到低相关度排列。

表 4 相关系数法与距离系数法的横纵坐标解释

编号	对应变量名称	编号	对应变量名称
229	产品辛烷值（因变量）	177	K-101B 进气温度
1	原料辛烷值	45	还原器温度
175	K-101B 排气压力	117	E-206 壳程出口管温度
176	K-101B 进气压力	154	ME-112 过滤器压差
173	K-101B 左排气温度	130	低压热氮气压力
174	K-101B 右排气温度	137	R-102 床层吸附剂料位密度
2	饱和烃	203	S_ZORB AT-0006
3	烯烃	102	A-202A/B 出口总管温度
0	硫含量	106	D-110 顶压力

从结果可以发现，原料辛烷值与产品辛烷值的相关度最高，Pearson 相关系数法、Spearman 相关系数法、Kendall 相关系数法分别为 0. 97、0. 97、0. 87，符合实际生产工艺。除此之外，压力、温度、饱和烃、烯烃、硫含量等因素也具有强相关。

然后，对距离相关系数法进行结果分析，如图 8 所示。

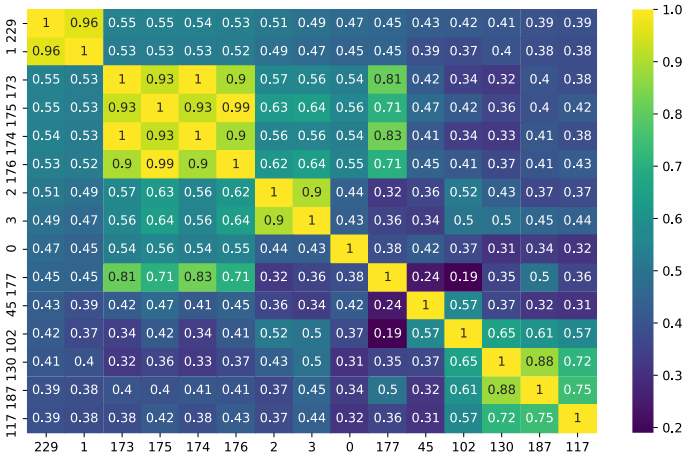


图 8 距离相关系数结果图

将距离相关系数与 Pearson 相关系数、Spearman 相关系数、Kendall 相关系数作对比，可以发现原料辛烷值、压力、温度、饱和烃、烯烃、硫含量等因素也都是高相关，但是顺序发生了略微变化，同时还原器温度、低压热氮气压力等因素也出现在较高相关的结果图中。这是因为距离相关系数表征的是非线性关系的变量，而上述 3 种相关系数矩阵只是表征线性相关的变量，不会完全重合。

然后，对 RFE 法进行结果分析，如表 5 所示（篇幅原因只列出前十个相关度最高的变量信息）。

表 5 RFE 结果表

变量编号	RFE 顺序结果	变量名称
174	1	K-101B 右排气温度
45	2	还原器温度
0	3	硫含量
32	4	非净化风进装置流量
141	5	R-102 底部锥段温度

226	6	8.0MPa 氢气至反吹氢压缩机出口
195	7	E203 重沸器管程出口凝结水流量
30	8	0.3MPa 凝结水出装置流量
86	9	D-123 凝结水入口流量
35	10	混氢点氢气流量

可以发现，RFE 结果与相关系数矩阵法和距离相关法有一定差别，这主要是由于 RFE 的算法与其他两类不一致。RFE 主要面向特征含有权重的预测模型，通过递归减少考察的特征集规模来选择特征。模型在原始特征上训练，每个特征指定一个权重。之后，那些拥有最小绝对值权重的特征被踢出特征集。如此往复递归，直至剩余的特征数量达到所需的特征数量。然而，辛烷值相关的样本中没有对权重有所定义，即使采用稳定性选择的方法也无法很好的达到理想结果，导致会出现不一定符合实际的情况。比如上述结果并没有得到原料辛烷值这一影响，而从实际工艺流程考虑，原料辛烷值应与产品辛烷值具有高相关性，可以判断出此种算法的不稳定。

最后，对第 4 种方法主成分分析（PCA）的结果进行分析，主成分贡献率和部分主成分载荷分别见表 6 和表 7。

表 6 主成分贡献率及累积贡献率

成分	贡献率/%	累积贡献率/%
1	87. 925	87. 925
2	11. 863	99. 788
3	0. 155	99. 943
4	0. 033	99. 976
5	0. 021	99. 997

表 7　部分主成分载荷

特征参数(按编号排序)	M1	M2	M3	M4	M5
0	0	0	0	-0.00001	-0.00001
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
…	…	…	…	…	…
…	…	…	…	…	…
222	0	0.00005	-0.00034	-0.0001	-0.00016
223	0.82319	0.29273	0.28136	0.24031	-0.13205
224	0.02302	-0.02221	0.0049	0.01941	0.14432
225	-0.00178	0.55355	-0.01707	-0.00381	0.74877
226	0	-0.00002	-0.00009	-0.00002	-0.00012
227	0.00067	0.60664	-0.18649	-0.41	-0.53475
228	0	0	0	-0.00001	0

主成分分析结果得到每一个变量对产品辛烷值的载荷值，即 229×1 的列向量结果。上述列表以编号的形式展现是为了便于读者理解程序中如何实现主成分分析的，再与变量名称对照，得到载荷从大到小排列的变量名称。载荷前几名的为加氢裂化轻石脑油进装置累积流量、S-ZORB.FT_9101.TOTAL、S-ZORB.FT_5101.TOTAL、汽油产品去气分累积流量等变量。可以发现，主成分分析的结果没有将原料辛烷值、硫含量、温度、压力等工艺流程会高相关的变量作为主要载荷，效果与实际略有不符。

究其原因，是因为主成分分析并不适用于样本和变量数都很多的场景，以致于结果与工艺常识不接近。

5. 2. 9 建立四种方法的综合筛选模型

经过上述对四种方法所得结果的分析，发现相关系数矩阵法和距离相关系数这两种传统方法更接近实际的工艺流程，而 RFE 和主成分分析得到的结果不一定符合实际。但是，如果要将 RFE 和主成分分析直接剔除，则将会产生只通过过滤法相关系数解决该问题，可能会造成结果不准确，无法适应大量样本大量变量的建模环境，也无法通过降维的方法对变量维数进行优化，而 RFE 和主成分分析则可以补足前两种方法的不足。

因此，本文创新性的提出将这四种方法建立综合筛选模型。

首先，分析四种方法的共性，均是以产品辛烷值为因变量，以 229 个其余的变量为自变量，结果均是以 229×1 的列向量来表达。则可以对四种方法得到的列向量结果分别进行归一化处理，每一种方法下的归一化模型如下：

$$y = \frac{y_0 - y_{\min}}{y_{\max} - y_{\min}} \qquad (19)$$

归一化后，对每种方法按照一定的权重进行加权综合，能够得到综合四种方法优点的新相关度列向量。

$$Y = \sum_{i=1}^{4} \beta_i y_i \qquad (20)$$

式中：Y 为综合四个结果得到的新相关度列向量；β_i 为第 i 个方法下所占的权重；y_i 为第 i 个方法下的旧相关度向量。

权重的选择参考文献[12]的方法，结合上述四种方法结果的对比，相关系数法和距离系数法的权重较高，RFE 和主成分分析的权重较小。

5. 2. 10 求解综合筛选模型得到高相关性变量

求解综合筛选模型，得到的是还未考虑变量之间的独立性和代表性下的过程结果，用 Python 语言求解得到了前 50 个对产品辛烷值有着较高影响的变量。

表 8　综合筛选模型前 50 个结果

前 1-10	前 11-20	前 21-30	前 31-40	前 41-50
原料辛烷值 RON	SZORB.FT_1003.TOTAL	汽油产品去气分累积流量	E203 重沸器管程出口凝结水流量	SZORB.PT_1501.PV
加氢裂化轻石脑油进装置累积流量	K-101B 左排气温度	SZORB.FT_1204.PV	原料进装置流量累计	K-101B 进气压力
K-101B 右排气温度	SZORB.FT_5201.TOTAL	D-110 底压力	D-123 蒸汽出口流量	SZORB.FT_9402.TOTAL
SZORB.FT_9302.T	SZORB.FT_9201.T	原料缓冲罐液	低压热氮气压力	K-101B 排气压力

OTAL	OTAL	位		
硫含量	SZORB.FT_9101.TOTAL	SZORB.FT_5101.TOTAL	SZORB.FT_9001.TOTAL	R-102 床层吸附剂料位密度
还原器温度	精制汽油去进料缓冲罐流量	废氢排放累计流量	火炬气排放累计流量	8.0MPa 氢气至反吹氢压缩机出口.1
饱和烃	SZORB.FT_5104.TOTAL	塔顶回流罐 D201 液位	K-103A 排气压力	反应过滤器压差.1
烯烃	EH101 出口	SZORB.FT_9202.TOTAL	SZORB.FT_3301.TOTAL	8.0MPa 氢气至循环氢压缩机入口.1
SZORB.FT_9401.TOTAL	D-123 凝结水入口流量	SZORB.FT_1002.TOTAL	SZORB.FT_9301.TOTAL	稳定塔顶回流流量
R-101 床层中部温度	0.3MPa 凝结水出装置流量	混氢点氢气流量	原料进装置流量累计	R-102 底部锥段温度

然而，这 50 个变量未进行变量之间的独立性检验，需要进一步剔除高相关性的变量才，满足独立性和代表性的要求。

同时，分析综合筛选模型前 50 个结果可以发现，非操作变量占 4 个，分别为原料辛烷值、硫含量、饱和烃和烯烃，操作变量占 46 个。非操作变量的代表性有待进一步的提高，需要对模型进一步改善。

5. 2. 11 改善综合筛选模型提高代表性

为进一步完善综合筛选模型，提高非操作变量的代表性，同时加强合理性，将模型的非操作变量和操作变量分开，分别单独与产品辛烷值建立特征选择和降维模型，再分别进行独立性的校正剔除互相具有相关性的部分变量，得到最终满足代表性和独立性的主要影响变量。分开后单独建立特征选择和降维模型的方法步骤类似于上文叙述的总体建立，这里不再赘述，仅给出结果与分析。

非操作变量筛选出 12 个，操作变量筛选出 70 个，受篇幅原因，这里只给出非操作变量按相关度高低排列的结果。

表 9　改善综合筛选模型的非操作变量结果

1	硫含量
2	原料辛烷值 RON
3	饱和烃
4	烯烃
5	芳烃
6	溴值, gBr/100g
7	密度(20℃), kg/m³
8	硫含量, μg/g. 1
9	焦炭
10	S
11	焦炭. 1
12	S. 1

该部分结果同样也是过程结果，依然需要进一步的筛选，但是分开非操作变量和操作变量后单独与产品辛烷值建立特征选择和降维模型，使得筛选变量更具代表性。

5. 2. 12 利用改善后的综合筛选模型进行独立性筛选

上述经过改善后的综合筛选模型保证了代表性的要求，但是由于部分变量间存在相互关系，如用某些自变量与某些自变量间具有高相关度，则需要筛选出这部分变量，并选取最合适的变量作为最终的主要影响因素结果。

调用 Python 语言的 Pandas，分别对非操作变量和操作变量进行自变量之间的相关分析，形成 Pandas Profilling Report。截取部分相关变量的结果如图 9 所示：

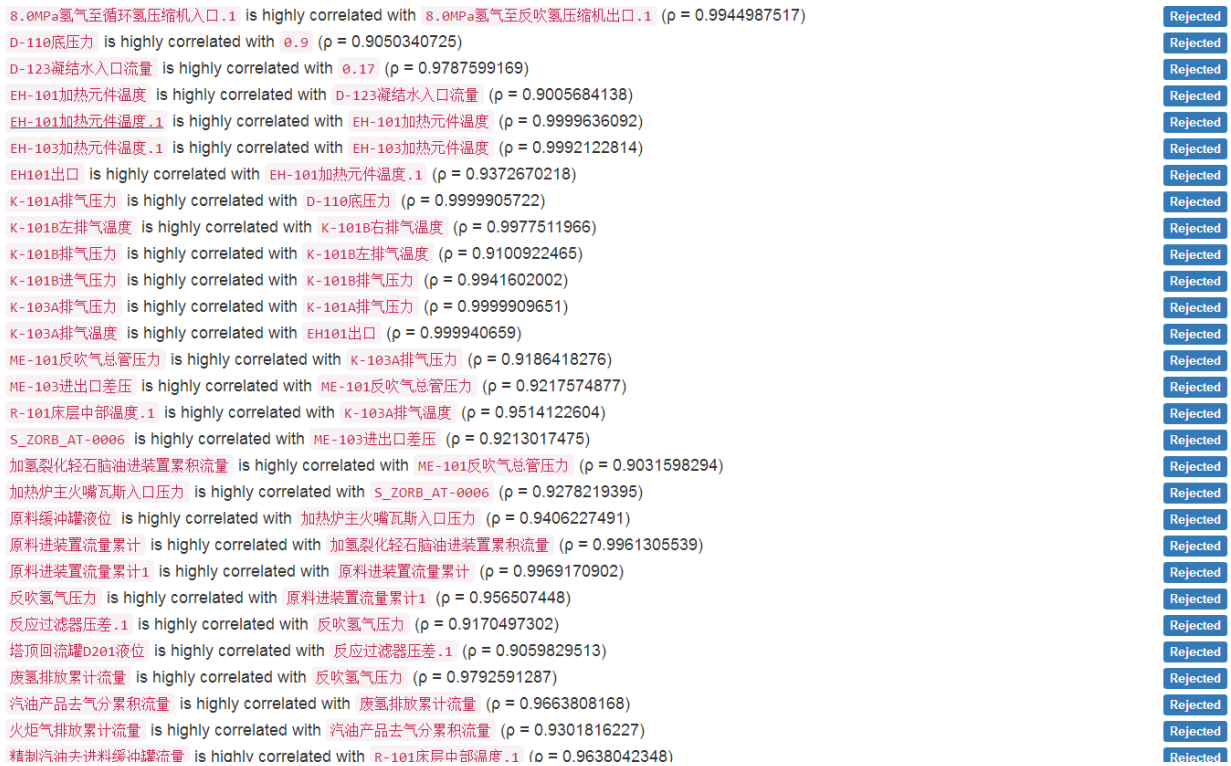


图 9 部分变量的独立性筛选结果

5. 2. 13 形成最终的主要影响变量

在对上一小节寻找到具有高相关性的自变量后，对这些变量进行筛选，参考化工流程，文献[13]和[14]探究了影响产品辛烷值的主要因素，将该模型求解得到的结果与文献[13]和[14]对比，也可以验证模型的准确性。

筛选后得到最终的主要影响变量 30 个，如表 10 所示。

表 10 辛烷值 30 个主要影响变量（最终结果）

前 1-6	前 7-12	前 13-18	前 19-24	前 25-30
硫含量	还原器温度	D-123 蒸汽出口流量	PIC2401B.OP	再生器顶烟气温度
原料辛烷值	SZORB.FT_1003.TOTAL	E203 重沸器管程出口凝结水流量	8.0MPa 氢气至反吹氢压缩机出口.1	R102 转剂线压差
饱和烃	K-101B 进气温度	混氢点氢气流量	EH-103 入口流量	A-202A/B 出口总管温度
烯烃	SZORB.FT_9402.T	稳定塔液位	原料换热器管程	EH-103 加热元件

	OTAL		总管进口温度	温度
S.1	0.3MPa 凝结水出装置流量	稳定塔顶回流流量	D-110 底部温度	R-101 床层下部温度
K-101B 右排气温度	ME-112 过滤器压差	R-101 下部床层压降	空气预热器烟气出口压力	R-102 底部锥段温度

主要影响因素中非操作变量含有 5 个，操作变量含有 25 个，共同组成了 30 个主要影响变量。对比文献[13]和[14]，可以发现原料辛烷值、含硫量、饱和烃。烯烃、排气温度、进出气温度、压力等求解结果中的主要影响与文献一致，也可以验证模型的正确性。

5. 3 对问题三的分析与求解

5. 3. 1 数据挖掘技术与 LSTM

数据挖掘技术应用于辛烷值模型的建立，是通过数据驱动的方式，根据大量的采集数据建立模型，从而实现对基本方程的淡化，应用于传感器越发发达的今天是一种有效的方式。而长短期记忆网络（LSTM）属于深度学习的范畴，也是数据挖掘的技术之一，其关系图解如图 10 所示。

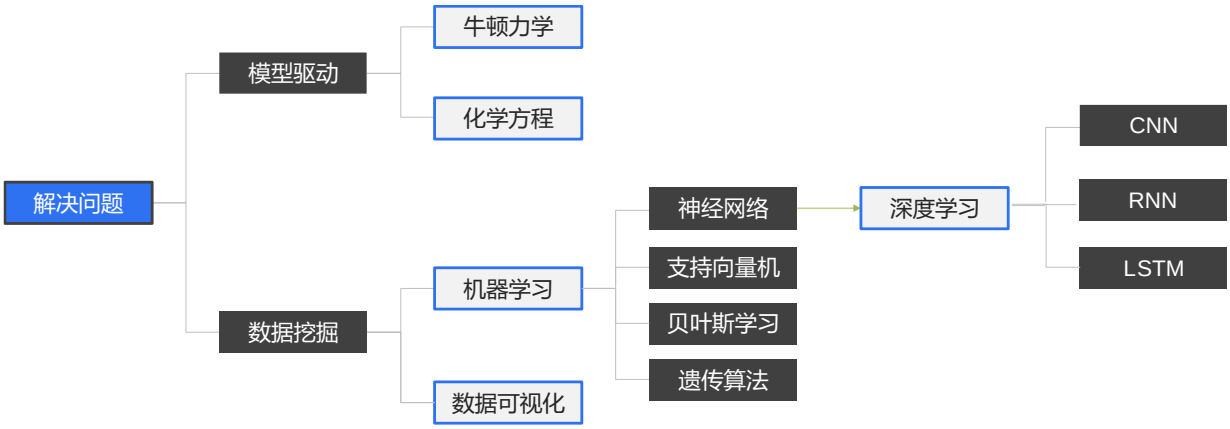


图 10 数据挖掘技术与 LSTM 的关系图

5. 3. 2 长短期记忆网络（LSTM）的建模

长短期记忆网络（LSTM）主要是为了解决训练过程中的梯度消失和梯度爆炸问题。本节将首先介绍 LSTM 模型，然后依据辛烷值预测的要求选定最佳的模型结构和参数。

Hochreiter 等学者提出了 LSTM 以解决此问题[15]。LSTM 是循环神经网络的一种变体，也是链式结构，但是重复模块有所差异，标准循环神经网络的重复模块内只有一个神经网络层，而 LSTM 有四个网络层，它们以一种特殊的形式交互。

LSTM 的核心是引入了一个新的单元状态，在传递线性循环信息的同时输出信息给隐藏状态，如图 11 所示贯穿整个重复结构的水平线就是单元状态。图中 σ 和 tanh 分别代表 sigmoid 和 tanh 激活函数，x_t 为输入量，h_t 、h_{t-1} 为当前和上一时间步的隐藏状态，C_t 、C_{t-1} 为当前和上一时间的单元状态，i_t，f_t，o_t 分别表示当前时刻的输入门、遗忘门和输出门。

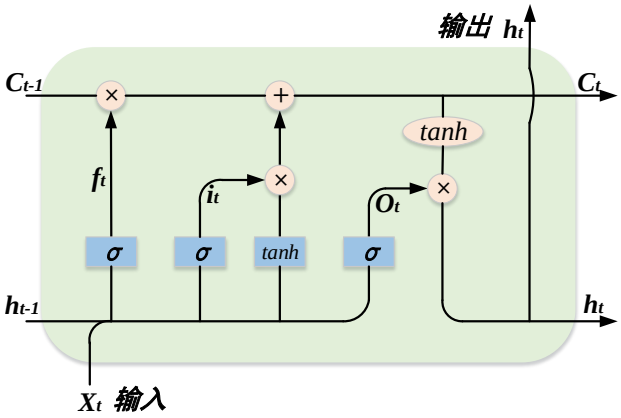


图 11　LSTM 网络结构图

LSTM 通过三个控制门单元（输入门、遗忘门、输出门）选择保留或者忘记信息。长短期记忆网络每一时间步的状态更新公式如下：

式（21）形成了当前时刻的单元状态。

$$C_t = f_t C_{t-1} + i_t \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \quad (21)$$

式（22）形成了当前时刻的隐藏状态。

$$h_t = o_t \tanh\left(C_t\right) \quad (22)$$

输入门式(23)控制当前时刻的输入信息有多少需要保留。

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \quad (23)$$

遗忘门式(24)控制上一时刻的信息需要丢弃多少。

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \quad (24)$$

输出门式(25)控制当前时刻有多少信息需要输出给隐藏状态 ht。

$$o_t = \sigma\left(W_o [h_{t-1}, x_t] + b_o\right) \quad (25)$$

式中：W 为权重矩阵，b 为偏置常数，式中下标 c ，i ， f，o 分别表示单元状态、输入门、遗忘门和输出门。

LSTM 有能力删除或增加神经元状态中的信息，这一机制由被称为门限的结构精心管理。门限是以 *sigmoid* 函数为激励函数的前馈神经网络，本质上是一种让信息选择性通过的方式，通过 *sigmoid* 神经网络层和逐点相乘器形成。使用 *sigmoid* 函数的原因是其输出在[0, 1]区间，可以等效为一组权重值。当输出值为 0，门限将遗忘所有信息；当输出值为 1，门限将保留所有信息。

因此，LSTM 选择性遗忘或保留的能力能够有效解决标准循环神经网络在信息过长时面临的梯度弥散或梯度爆炸问题，有利于信息建模，可以较好的实现在较多影响变量情况下的产品辛烷值预测。

5. 3. 3 基于 LSTM 网络的辛烷值损失预测模型

针对辛烷值预测模型，选取训练集和测试集的比例。在训练负荷预测模型时，为得到最优超参数，首先依照经验试验部分超参数，得到了最优超参数的大致范围；然后依据网格搜索法，搜寻得到适用于辛烷值预测的最优超参数均为：网络层数为 8 层、学习率为

0.006、批处理数为 30。

运用转化的思想，根据公式（1），辛烷值损失与原料辛烷值、产品辛烷值直接相关，而在本文的研究中原料辛烷值作为定值，可以先对产品辛烷值进行预测，再用公式（1）计算辛烷值损失。而且，考虑到预测的准确性，对产品的辛烷值预测比直接对辛烷值损失精度更高。

对辛烷值进行预测的模型建立中，需要明确训练集和测试集的数量以及维度，按照训练集：测试集=80%：20%进行划分，252 个样本中得到 201 个训练集，51 个测试集，其维度信息见表 11。

表 11 训练集、测试集的维度

	X（输入）	Y（输出）
训练集	201×30	201×1
测试集	51×30	51×1

网络的损失函数是均方误差函数式（26）

$$MSE = \left(y_i - y_i'\right)^2 \qquad (26)$$

式中：y_i表示产品辛烷值真实值，y_i'表示产品辛烷值预测值。

在辛烷值预测中，LSTM 网络的输入矩阵维度为（batch_size，input_dim），经过 LSTM 网络层后输出矩阵维度为（batch_size，hidden_size），最后经过回归层得到预测值（batch_size，out_size）。其中 batch_size 为批训练数目，hidden_size 为隐藏层维数，out_size 为输出向量的维度，在预测时为 1。需要特别注意的是，这里的叙述只是为了直观理解采用了二维输入，但在实际编写程序的 LSTM 网络中需要转化为三维输入。

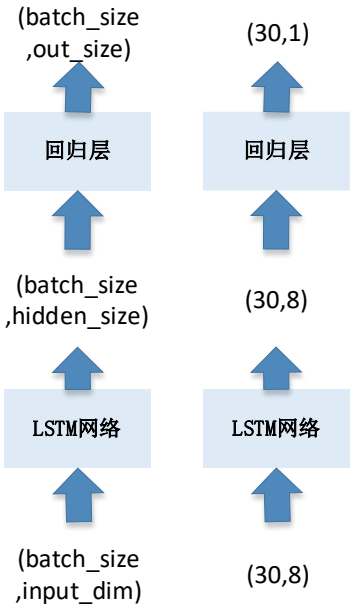


图 12 LSTM 的输入结构图

5.3.4 辛烷值及辛烷值损失预测评价指标

本文的预测评价指标选取为平方均方误差（RMSE）和平均相对百分比误差（MAPE），计算公式分别为式(27)和式(28)。

$$RMSE=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y_i)^2} \tag{27}$$

$$MAPE=\frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i-y_i}{y_i}\right| \tag{28}$$

RMSE 与 MAPE 的值越接近零，预测误差越小，模型的预测效果越好。

5.3.5 GA-LSTM 算法改进参数优化

为了在 LSTM 模型中快速有效的寻找最优参数，可以采用 GA-LSTM 算法，即在 LSTM 网络中加入 GA 算法通过交叉和变异优化参数选择，从而为 LSTM 选择最合适的参数，以实现模型预测精度的提高。其流程如图 13 所示。

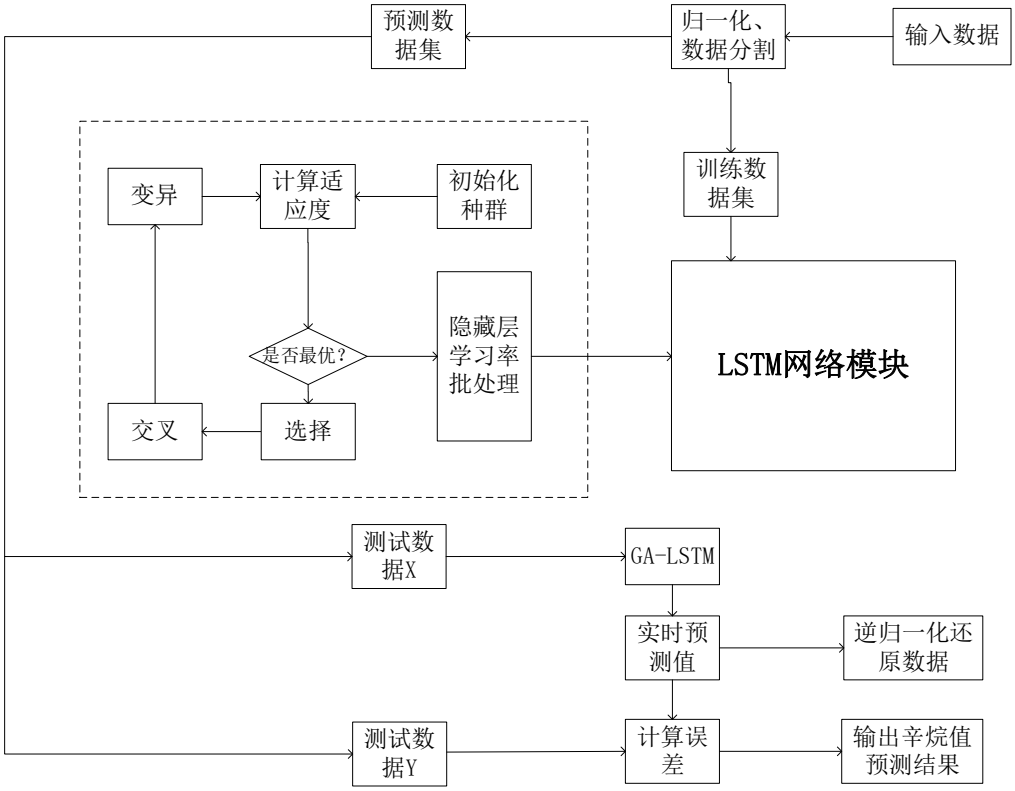


图 13 基于 GA-LSTM 的辛烷值损失预测模型

5.3.6 辛烷值预测结果及分析

根据上述辛烷值预测模型和 GA-LSTM 参数优化，通过 Python 语言编程求解得到 51 个测试集的预测结果，并与真实值对比。辛烷值的预测结果属于直接结果，化成辛烷值损失后为题目的最终预测结果。辛烷值预测值与真实值见图 14，样本编号为 0-50。

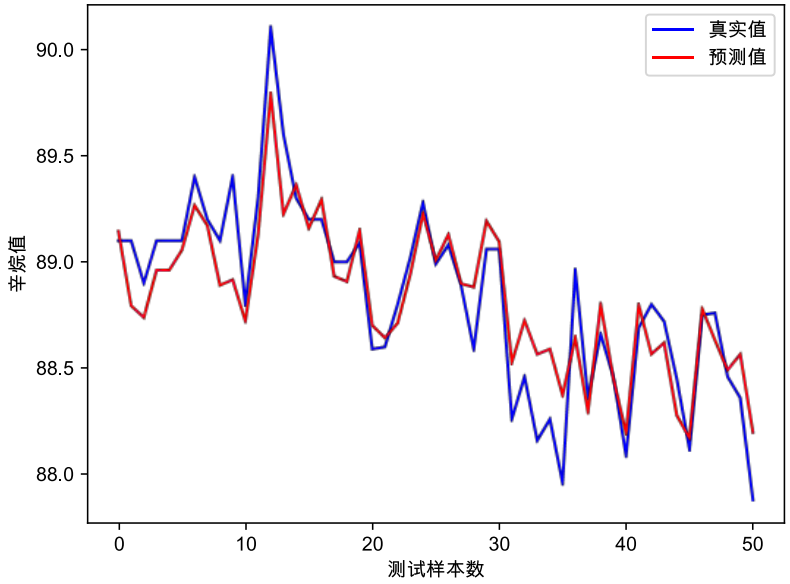


图 14 辛烷值预测结果（直接结果）

由辛烷值预测结果可以看出，51 个测试集样本中，预测值与真实值之间的误差范围为 0. 03-0. 41RON，对每一个样本作其相对百分比误差分析图，如图 15 所示。

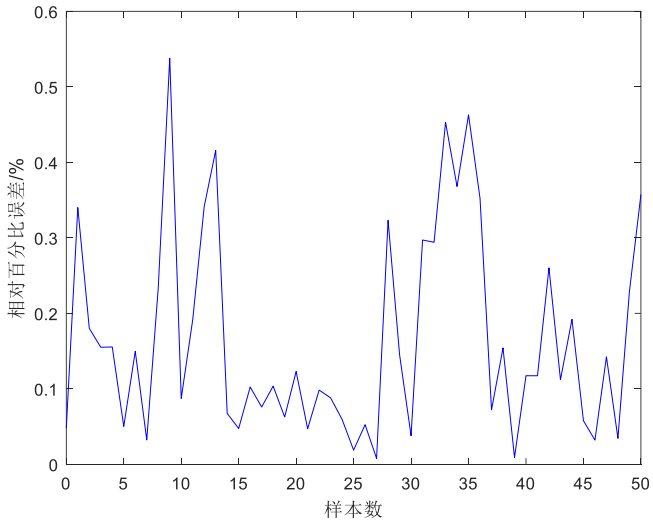


图 15 各样本的相对百分比误差分析图

通过各样本的相对百分比误差分析图可以得出，相对百分比误差最高时所对应的样本为 9 号样本，为 0. 54%，大多数样本集中在 0. 20%以下，辛烷值的预测效果较好。同时，需要对公式（27）和（28）的指标进行计算，以得到对样本整体的 RMSE 和 MAPE，能够更准确的体现预测效果。

LSTM 模型下的 51 个测试集 RMSE=0. 19269，MAPE=0. 49425，均能满足化学工艺流程对于辛烷值预测误差的要求，且效果较好。

5. 3. 7 其他预测方法简介

本小节简单介绍其他几种代表性的预测模型，并用 Python 语言对辛烷值损失数据进行预测，再与 LSTM 的预测结果进行对比。通过这一步可以验证 LSTM 模型预测辛烷值损失

的优势。

随机森林（RF）：RF 在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中引入了随机属性选择。随机森林简单、容易实现、计算机开销小，但是随机森林的起始性能相对较差。

支持向量回归（SVR）：针对线性可分的情况进行分析的。对于线性不可分的情况，通过使用非线性映射算法将低维输入空间线性不可分的样本转化为高维特征空间，使其线性可分，从而使得在高维特征空间中采用线性算法对样本的非线性特征进行线性分析成为可能。SVR 基于结构风险最小化理论，在特征空间中构建最优分类面，使得学习器能够得到全局最优化，并且使整个样本空间的期望风险以某个概率满足一定上界。

极端梯度提升（XGBoost）：以分类回归树为基分类器，采用集成学习中梯度提升的方法进行加法训练，将多个个体分类器组合成一个集成分类器，以提升其预测速度与精度。

XGBoost 模型的基学习器为 CART，对于一棵 CART，其复杂度由结构 q 和叶子节点输出值 w 决定，对于一个确定的输入 x，存在一个 w 与之对应，表征 CART 对当前输入的预测结果。

对于一个有 n 个样本，m 个特征的数据集 $D=\{(x_i,y_i)\}\ (|D|=n,x_i\in\mathbb{R}^m,y_i\in\mathbb{R})$，K 棵 CART 预测最终输出为：

$$\hat{y}_i=\sum_{k=1}^{K}f_k(x_i)\ \ f_k\in F \qquad (29)$$

其中 $F=\{f(x)=\omega_{q(x)}\}\ (q:\mathbb{R}^m\to T,\omega\in\mathbb{R}^T)$ 为 CART 构成的的集合；w 为叶子的权重；T 为叶子节点个数；q 为表示每棵树的结构的向量，由样本指向相应的叶子标签；每个函数 fk 对应一棵独立的树结构 qk 和叶子权重 k。每棵 CART 的每个叶子节点对应一个连续分数值，i 代表第 i 个结点的分数。q(x) 是对样本 x 的打分，即模型预测值。对于每个样本，各个 CART 依据不同分类规则将它分类到叶子节点中，通过累加对应叶子的分数 w 来获得最终的预测结果。

5.3.8 与其他预测方法的对比

为验证 LSTM 对辛烷值预测的合理性和准确性，本文对 5.3.6 节所述 3 种其他方法进行了对比分析，同样在 Python 语言下进行程序编写，这里受篇幅原因不再给出其他 3 个方法的结果，仅将 RMSE 和 MAPE 这两个指标与 LSTM 模型下的指标结果进行对比，如表 12 所示。

表 12　LSTM 与其他 3 个方法的指标结果对比

	LSTM	XGBoost	SVR	RF
RMSE	0.19269	0.21558	0.28495	0.26148
MAPE	0.49425	0.52689	0.59863	0.56217

通过上表可以对比得出辛烷值的预测效果 LSTM>XGBoost>RF>SVR，LSTM 在辛烷值的建模中具有较高的优势，这是由于在大量样本大量数据的情况下，LSTM 能够有效的解决梯度爆炸所带来的训练不准确影响，也能够克服 RF 的起始性能较差、SVR 处理非线性能力不强、XGBoost 对分类规则要求高的缺点。

5.4 对问题四的分析与求解

5.4.1 辛烷值优化流程

辛烷值优化本质是基于 5.3 节中的辛烷值预测模型，在保证产品硫含量不大于 5 μg/g

的前提下，通过改变主要操作变量，对产品的辛烷值进行优化，从而指导汽油生产过程的操作条件。通过改变操作条件从而改变产品的辛烷值，在此过程中产品硫含量也会发生变化，为此需要从操作变量中选择与产品硫含量相关的变量，建立硫含量的预测模型。在此基础上，可以建立辛烷值优化过程中的目标函数与约束条件，从而形成辛烷值优化模型。可以使用智能优化算法对该模型进行求解，如 GA 算法等，其优化流程如下所示。

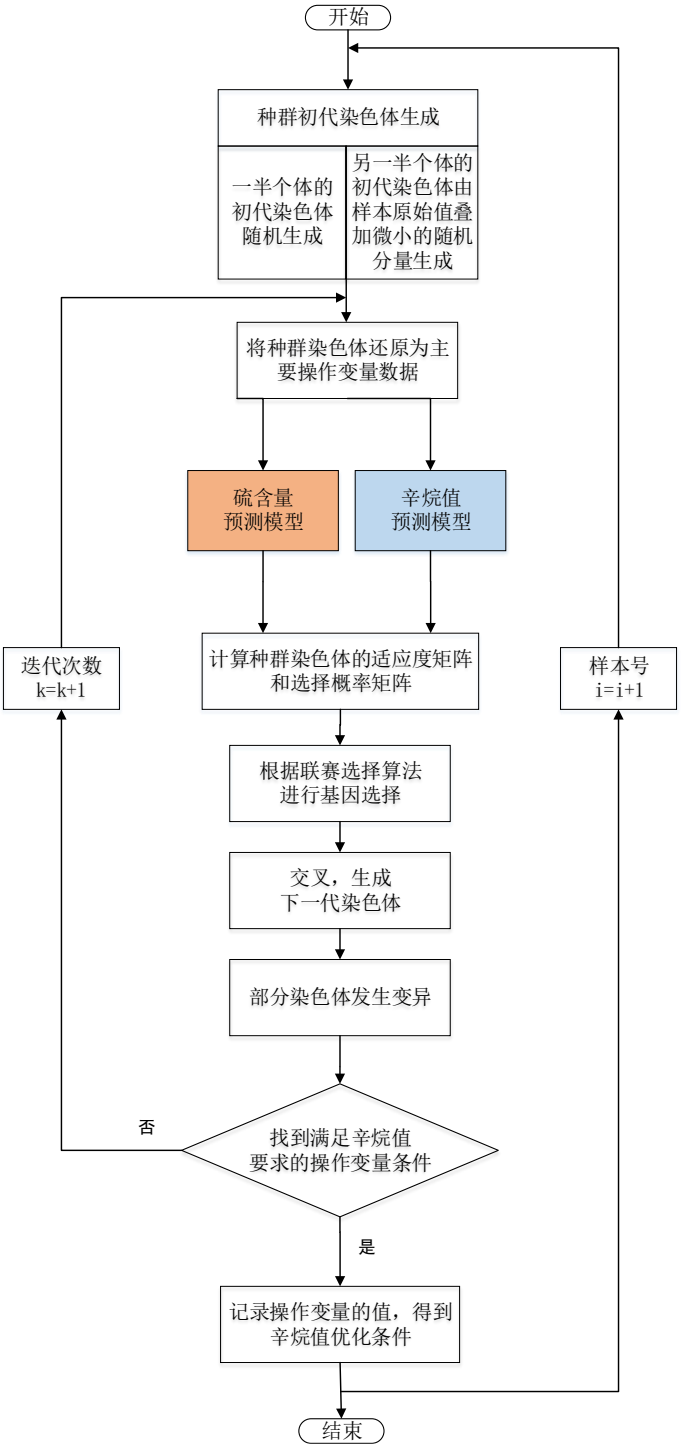


图 16 辛烷值优化流程

另外，对于该辛烷值优化问题，需要考虑到在汽油生产的实际过程中，生产条件等工况的改变是需要成本的，且工况条件变化越剧烈，所付出的代价也越大。相对于搜索全

局最优解，在原工况条件附近搜索在硫含量约束条件范围内，且满足辛烷值优化条件的解更为合理，这不仅大大提高了算法求解的速度，而且更符合工程实际需求，更具指导生产意义。在综合考虑算法求解性能与工程实际意义的基础上，可对 GA 算法的初始种群设计进行优化，让其中一半个体的染色体基因随机生成，另一半个体的染色体基因则在该样本的原始值附近叠加一个轻微的随机分量生成。从而使得该 GA 算法求解辛烷值优化问题时，兼具搜索能力和工程实际意义。

5.4.2 产品硫含量与脱硫率建模

产品含硫量的建模直接影响到辛烷值优化里的含硫量约束，因此在对辛烷值建立优化模型时要采用合适的方法对硫含量精确建模。

再次运用转化的思想，产品硫含量和脱硫率之间具有高相关性，原来硫含量视为定值，则可以建立产品硫含量与脱硫率的函数关系：

$$\gamma = \frac{S_r - S_p}{S_r} \times 100\% \qquad (30)$$

考虑工艺流程，工程中更多考虑的是脱硫率是否达标，同时脱硫率的预测准确率能够得到提升[18-19]。

对脱硫率预测模型的建模过程与上述 5.2 节主要变量筛选和 5.3 节 LSTM 预测模型基本类似，即通过改善后的综合筛选模型并进行独立性改善，得到脱硫率的主要影响变量，通过 LSTM 预测模型对脱硫率精确预测。具体建模过程由于与辛烷值模型类似且篇幅原因不再赘述。

除此之外，附件 1 部分样本的产品含硫量为 3.2μg/g，考虑到实际工艺中低于这个值仪器检测不出来，因此该部分样本不具备代表性，可以去除。经过去除后，样本数从 252 变为 98。

5.4.3 脱硫率预测结果及分析

首先根据问题二的求解方法筛选出与脱硫率高相关的 19 个主要影响变量，列于表 13 所示。

表 13 脱硫率的 19 个主要影响变量

前 1-4	前 5-8	前 9-12	前 13-16	前 17-19
硫含量	S	混氢点氢气流量	原料换热器管程总管进口温度	1.1 步骤 PIC2401B.OP
饱和烃	S.1	稳定塔顶回流流量	SZORB.FT_1003.TOTAL	R-101 床层下部温度
烯烃	8.0MPa 氢气至反吹氢压缩机出口.1	SZORB.FT_9402.TOTAL	ME-112 过滤器压差	还原器温度
溴值	0.3MPa 凝结水出装置流量	R102 转剂线压差	E203 重沸器管程出口凝结水流量	

可以看出，最相关的是产品硫含量，其次是饱和烃、烯烃、溴值、S、S.1 等 19 个变量。其中，前 6 位的都是非操作变量，产品硫含量、饱和烃、烯烃和溴值属于原料性质，S、S.1 分别属于待生吸附剂性质、再生吸附剂性质，表征吸附剂对脱硫率的影响。

然后，根据 5.3 节 LSTM 预测模型方法，对脱硫率根据选出的 19 个变量进行预测，得到如下图 16 所示的预测结果，指标对比等由于在本节中不是重点则未列出。

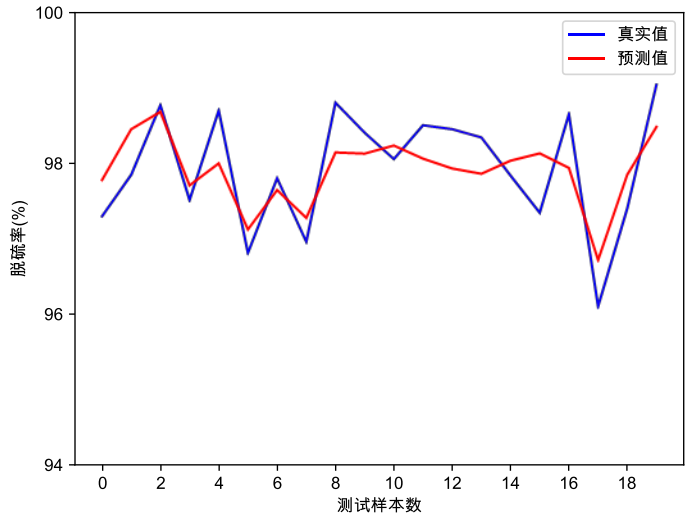


图 17 脱硫率预测结果

各样本的脱硫率预测绝对误差范围在 0.21%-0.67%之间，也是符合工程要求的误差率范围。

5. 4. 4 辛烷值优化问题的建模

在建立产品辛烷值与脱硫率预测模型的基础上，可以在保证产品硫含量低于允许值的前提下，对其生产过程中的主要变量操作方案进行优化，从而提高产品辛烷值。对于汽油辛烷值的优化，可使用智能优化算法，如粒子群优化算法（PSO）、GA 算法等。其中，GA 算法起源于对生物系统所进行的计算机模拟研究，是一种高效的随机搜索和优化算法，具有在搜索过程中获取和积累搜索空间信息，并自适应地控制搜索过程的能力。而 PSO 算法中粒子可依靠自身的经验与记忆，以及其他粒子所提供的经验，来学习并调整自身的移动方向。对于辛烷值优化问题，汽油样本的原始工况可以为我们提供初始值设定的参考，考虑到粒子群算法中初始位置相近的粒子其移动轨迹亦将十分接近，其搜索能力不如 GA 算法，因此本文的辛烷值优化问题选用 GA 算法求解。

辛烷值优化问题以辛烷值预测模型的输出作为目标函数，在主要操作变量的变化范围内进行搜索。具体来说，本问题的目标函数，即为 GA 算法的种群适应度函数。为了控制种群在满足产品硫含量约束条件的前提下，向辛烷值优化的目标进化，可以将辛烷值预测模型的输出作为主要目标函数，并加上带有硫含量预测模型输出的障碍函数，构建 GA 算法的适应度函数。

将 5. 4. 2 节中硫含量的预测模型应用于 325 个样本中，得到各个样本的硫含量预测值，其最大值为 $12\,\mu g/g$，而优化问题中要求产品的硫含量不高于 $5\,\mu g/g$，为此可以构建包含硫含量预测模型输出的障碍函数项 $\log(1-S_p/7)$，其曲线图如下所示。由图可得，该障碍函数在 $S_p<5\mu g/g$ 时是接近 0 的正数，当 S_p 不断增大以至于超过 $5\mu g/g$ 时，该函数将迅速减小以阻碍主要操作变量的变化，从而在迭代求解过程中限制产品硫含量。

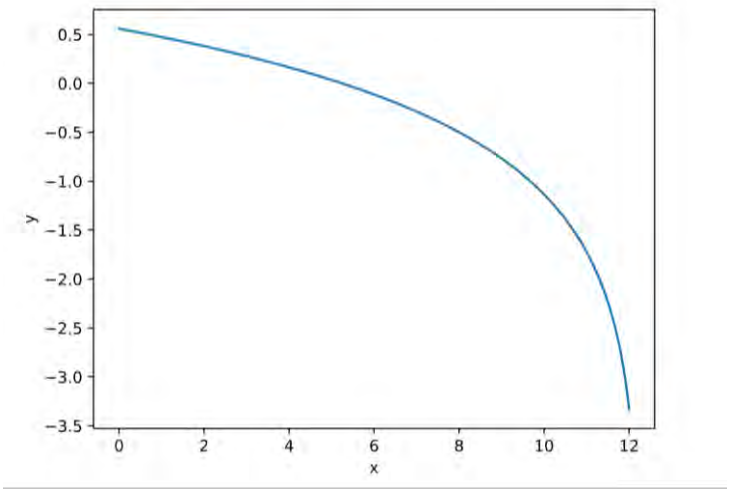


图 18 适应度函数的障碍函数项

综上，可得到 GA 算法的种群适应度函数，也是辛烷值优化问题的目标函数如下，其中 R_p 为辛烷值预测模型的输出值，S_p 为硫含量预测模型的输出值

$$f(X) = R_p + \log(1.75 - \frac{S_p}{7}) \qquad (31)$$

5. 4. 5 GA 算法的初始种群设计

GA 算法常用于全局搜索的优化问题，但对于本题中的辛烷值优化问题，需要考虑到在汽油生产的实际过程中，生产条件等工况的改变是需要成本的，且工况条件变化越剧烈，所付出的代价也越大。相对于搜索全局最优解，在原工况条件附近搜索在硫含量约束条件范围内，且满足辛烷值优化条件的解更为合理，不仅大大提高了算法求解的速度，而且更符合工程实际需求，更具指导生产意义。

在综合考虑算法求解性能与工程实际意义的基础上，GA 算法的初始种群设计为，种群规模 K_p 为 30，其中一半个体的染色体基因根据各主要操作变量在其变化范围内的随机值生成，另一半个体的染色体基因则在该样本的原始工况条件基础上，叠加一个轻微的随机分量生成。如此，GA 算法不但具有良好的搜索能力，而且将尽量在原始工况条件附近搜索满足要求的解，从而更具有实际意义。

5. 4. 6 GA 算法的求解过程

首先对样本操作变量的原始值进行二进制编码，从而形成种群部分个体的初代染色体，剩余部分个体的染色体则在操作变量的取值范围内随机生成。迭代开始后，首先将二进制编码的染色体还原为操作变量的实数值，并将其输入硫含量预测模型与辛烷值预测模型，从而求得种群各个染色体的适应度，并计算相应的选择概率矩阵，适应度较高的染色体在进化时被选择的概率较高。选择时使用联赛选择算法，染色体之间发生交叉、变异等从而形成新的染色体。最后，若找到满足硫含量约束和辛烷值优化条件的操作变量组合，则记录该组合并对下一个样本进行优化，否则继续迭代直至找到优化操作条件。

5. 4. 7 产品辛烷值优化结果及分析

由上述辛烷值 GA 优化模型，代入 Python 环境下求解。考虑到工程实际，实时工况下需要在较短的时间内完成优化，程序按照每一个变量的步长设定最高迭代次数为 4 次，得到了 325 个样本的部分优化结果见表 14。需要说明的是，由于篇幅原因，本文仅展示出开

头和结尾编号的部分样本在辛烷值、辛烷值损失降幅、硫含量、是否达到指标这几个相对更重要的结果。样本编号从 0 到 324，总共 325 个样本，操作条件为辛烷值的 25 个操作变量进行的调整。

表 14 325 个样本中部分样本的辛烷值优化结果

样本编号	辛烷值	辛烷值 损失降幅	硫含量 μg/g	是否达到 指标
0	89.8599777	46.3751973%	3.95730590	是
1	89.6214752	**25.5487474%**	4.51772514	否
2	89.9725799	47.2884026%	4.67362266	是
3	89.6328506	44.4094672%	3.51176353	是
4	89.3786773	82.7091694%	4.39519149	是
…	…	…	…	
…	…	…	…	
…	…	…	…	
319	88.8621215	48.9697266%	4.67482866	是
320	89.1631393	96.7947255%	3.85243770	是
321	88.7009277	60.0742187%	4.47611127	是
322	89.5430679	70.5214874%	4.76654597	是
323	88.5390548	32.7386618%	4.67076858	是
324	89.1155090	37.2407227%	4.71823669	是

统计发现，整个 325 个样本中，有 252 个样本达到了指标，而有 73 个样本未达到指标，达标率为 77.54%。这是由于部分样本迭代次数达到程序设定的上限，若将迭代次数改为 5 次，则达标率为 90.64%，但时间会更长。

基于上述分析，若想满足所有样本都能够在硫含量小于 5μg/g 的前提下，辛烷值损失的降幅大于 30%，则就需要花费更多的时间成本，实际操作时需要取舍达标率和时长的重要性来选择设定迭代次数限制。

5.4.8 优化的操作条件举例分析

程序得到了 325 个样本的操作条件，但由于样本太多，无法一一展示出来，故仅展示编号为 66 号的样本操作条件。

按照表 10 中 6-30 即 25 个操作变量的顺序，可以将下表的数字对照找到对应的变量名称。作出如表 15 的 66 号样本操作条件，其中，1、3、5、7、9 行是变量的编号，2、4、6、8、10 行是操作条件。

表 15　66 号样本的操作条件

6	7	8	9	10
28.18343712	238.1839612	23277775.16	0.612308094	-167452.3175
11	12	13	14	15
5609.820543	-0.128886046	315.8297274	1875.206734	6297.275573
16	17	18	19	20
49.40678116	8951.513022	86.35375385	102.2880991	3721606.529
21	22	23	24	25
290.7859036	57.38840666	158.3413112	-1135.981716	205.137571

26	27	28	29	30
5.995570019	40.07049561	450.0787638	403.5239779	243.0867317

可以发现，所有操作条件均是在变动范围内调整，由于各个变量之间量纲不一致，所以导致数额差别较大，但不影响结果。

为表示 GA 算法的 30 个个体（自设参数）的变化过程，列出了如下辛烷值和硫含量的变化图。

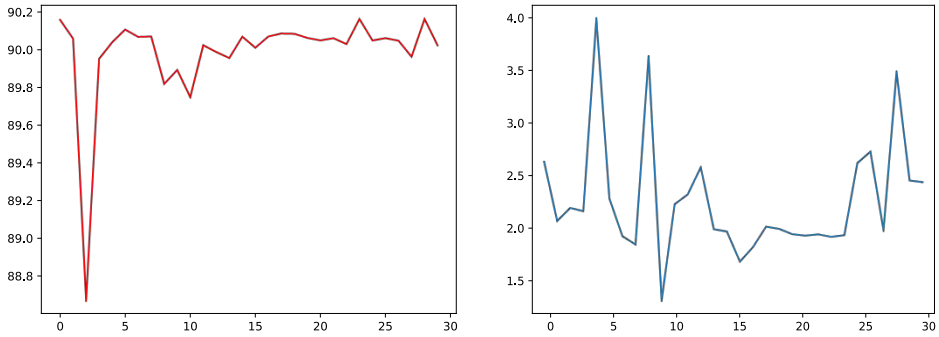


图 19 GA 下 30 个个体的辛烷值和硫含量

5. 4. 9 GA 与 PSO 优化结果对比

为验证优化方法的适用性，与 5.3 节 LSTM 与其他三种预测方法对比类似，需要进行 GA 和 PSO 单独优化加以对比。同样选取 66 号样本作为分析对象，将 GA 与 PSO 的优化结果列于表 16。

表 16 GA 与 PSO 结果对比

	辛烷值	辛烷值 损失降幅	硫含量 μg/g	是否达到 指标
GA	88.4427185	56.4623833%	4.4739167	是
PSO	88.1814452	32.8124526%	4.1527156	是

可以看出，第 66 号样本在 GA 算法和 PSO 算法下都能达到指标，但是显然 GA 下的辛烷值损失降幅大于 PSO 下的辛烷值损失降幅，这是由于 PSO 容易陷入局部最优，造成较快收敛但结果不一定是最好；而 GA 的速度较慢。两种方法各有优劣，在实际应用时需要根据不同的场景判断使用哪种算法。

5. 5 对问题五的分析与求解

5. 5. 1 133 号样本主要操作变量的变化次数

根据上一小节的主要操作变量优化步骤可得到 133 号样本操作变量需要调整到的目标值，其中前八个最重要的操作变量罗列如表 17 所示。

表 17 133 号样本主要操作变量初始值和目标值

前八个重要的操作变量	目标值	初始值
K-101B 右排气温度	3. 160288	49. 1861
还原器温度	209. 8554	261. 8852
SZORB. FT_1003. TOTAL	18066388	327235. 3775
K-101B 进气温度	34. 55355	24. 8762

SZORB. FT_9402. TOTAL	−177143	3314155. 8250
0. 3MPa 凝结水出装置流量	3890. 007	4696. 6507
ME−112 过滤器压差	0. 272678	0. 5380
D−123 蒸汽出口流量	174. 4677	150. 6328

考虑到工业石油裂化装置生产的稳定性，目标操作变量不能够直接根据初始值一步直接调整得到，每个操作变量的单次调整都存在最大值限制，需要根据单次最大调节量进行调整。同时为尽可能地保证装置的稳定性，假设每次调整时各操作变量是等步长调整的。

因此需要根据主要操作变量依照单次最大调节量所需的最多次数确定总调节次数，如式（32）所示

$$t = \max_{i=1}^{25} \left\lceil \left| \frac{x_i - x_i^{'}}{\Delta_i} \right| \right\rceil \qquad (32)$$

其中，t 为最终需要的总调节次数，Δ_i 为对应操作变量的单次最大调整量，x_i 为目标所需的操作变量值初始值，$x_i^{'}$ 为 133 号样本初始操作变量值。

在确定了操作变量调整次数后，依据前述等步长调整的装置稳定性调整假设，可以计算出各步操作变量变化的大小见式（33）。

$$\delta = \left| \frac{x_i - x_i^{'}}{t} \right| \qquad (33)$$

经程序计算得到，133 号样本需要的变化次数为 1519 次，其中每次调节最接近单次最大调整限制的主要操作变量为 SZORB.FT_1003.TOTAL。

5. 5. 2 辛烷值和硫含量随调整步长的变化

根据 5.3 小节和 5.4 节分别针对辛烷值和硫含量的建模过程，我们可以依据 25 个主要操作变量、相应的非操作变量以及训练好的 GA-LSTM 模型参数得到辛烷值和硫含量变化。绘制曲线图分别如图 20 和图 21 所示。

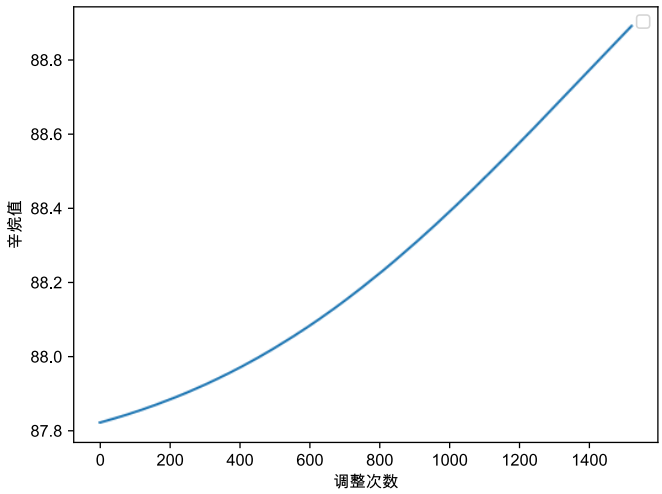


图 20 133 号样本辛烷值随调整次数的变化曲线

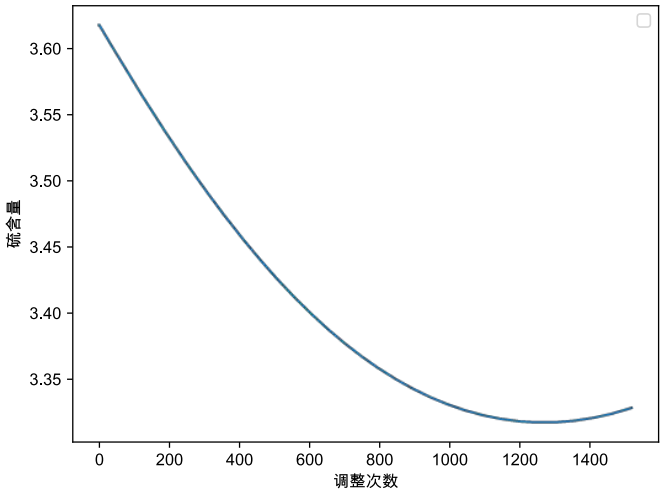


图 21 133 号样本硫含量随调整次数的变化曲线

由图 20 可知，随着调整次数的增加辛烷值逐渐增加，说明操作变量的调整是行之有效的。对比附件一中的 133 号样本原料辛烷值为 89.4，产品辛烷值为 88.09。调整次数的起始点即产品辛烷值的预测值为 87.81，预测误差在 0.2 左右。经过操作变量的调整之后辛烷值通过预测模型可以计算得到约为 88.8。辛烷值损失从起始点的 1.31（实际辛烷值损失）经过操作变量的调整后可以减少到 0.6（预测的辛烷值损失），预计等步长的操作变量调整后，将有效降低 50%的辛烷值损失。该损失值降幅验证了针对问题 4 主要操作变量所提出的优化模型的有效性。

由图 21 可知，随着调整次数的增加硫含量总体呈现略微下降趋势，在主要操作变量调整的最后呈现小幅波动的趋势。对比附件一中的 133 号样本产品实际硫含量为 3.2 μg/g，预测硫含量为 3.6 μg/g（调整次数为零时），且在优化辛烷值损失值过程中也能够一定程度上减少硫含量，也可进一步验证问题 4 模型的有效性。

5.5.3 t 分布随机邻域嵌入降维和 PCA 降维的差异

t 分布随机邻域嵌入（t-distributed Stochastic Neighbor Embedding, t-SNE）是一种非线性学习的算法，它相比于 PCA 和其他降维模型具有更好地可视化效果。

PCA 是一种线性算法，未能对复杂多项式关系进行解释，其降维后的可视化结果常常呈现出一条直线即分量之间存在线性关系，可视化效果较差。而 t-SNE 模型不仅可以关注局部还可以关注全局，通过梯度下降的算法将原空间与嵌入式空间样本分布之间的 KL 散度优化，解决了线性算法在可视化样本分布拥挤、边界不明显的特点，是目前最好的降维可视化手段之一。

因此，针对问题 5 为能够更好地展示 133 号样本主要操作变量调整后的汽油辛烷值和硫含量的变化轨迹，宜采用 t-SNE 降维模型，它所采用的非线性算法将有助于多维变量的降维至二维空间，并以更直观地形式展示。

5.5.4 基于 t-SNE 的汽油辛烷值和硫含量的变化轨迹

编程实现 t-SNE 降维算法后，以汽油辛烷值和硫含量为 z 轴绘制出三维散点图如图 22 和图 23 所示。根据 5.5.2 节汽油辛烷值和硫含量随调整次数的变化曲线可绘制出其变化方向，如图中箭头所示。

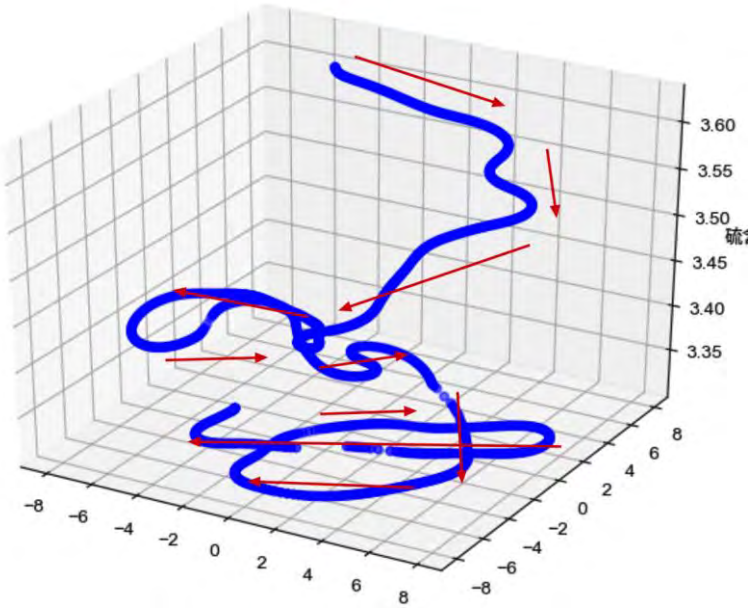


图 22　t-SNE 降维后硫含量三维散点变化轨迹

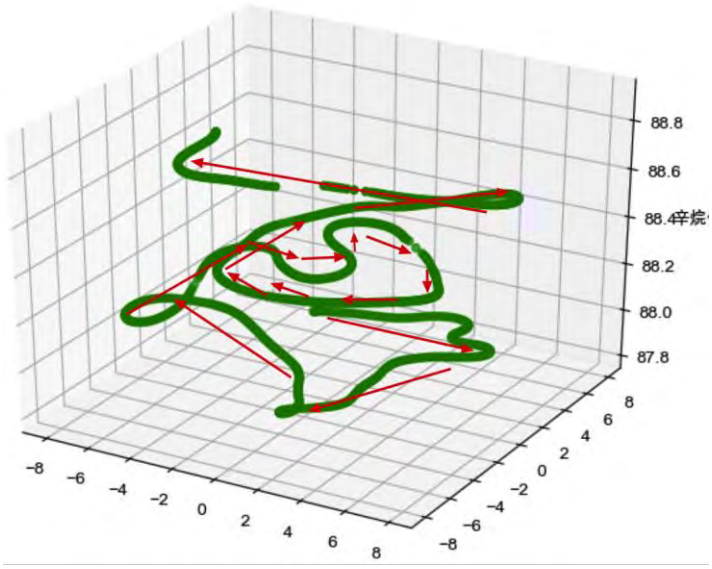


图 23　t-SNE 降维后汽油辛烷值三维散点变化轨迹

以汽油辛烷值和硫含量为变量量绘制出二维散点图如图 24 和图 25 所示。根据 5.5.2 节汽油辛烷值和硫含量随调整次数的变化曲线可绘制出其变化方向，如图中箭头所示。在此需要说明的是，由于汽油辛烷值和硫含量的对应的主要操作变量的降维方法时相同的，所以他们映射到二维平面的曲线形状是相同的，此外将硫含量和辛烷值均作归一化处理后，放大至[0-256]区间并取整依照 RGB 规则取色，所以绘制出颜色不同的二维曲线图。

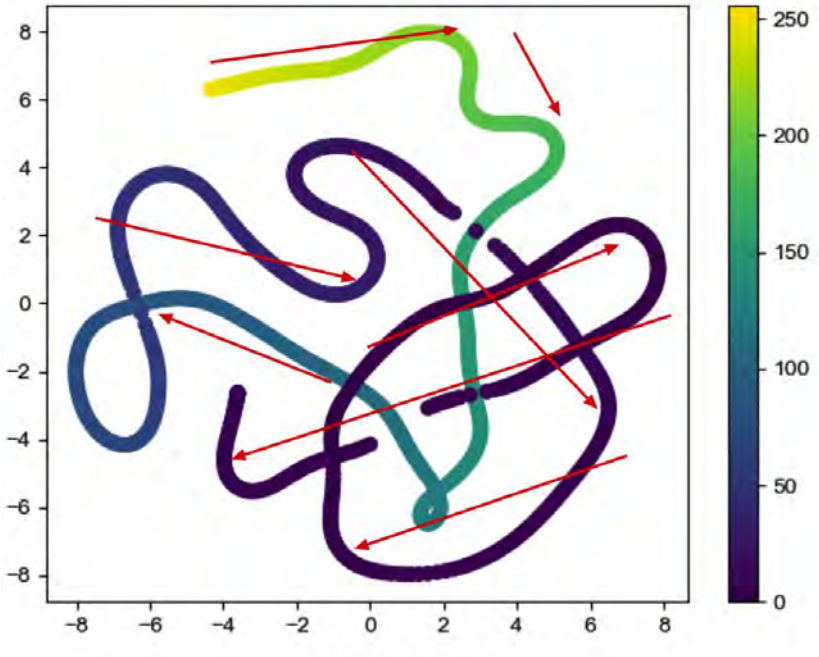


图 24　t-SNE 降维后硫含量二维散点变化轨迹

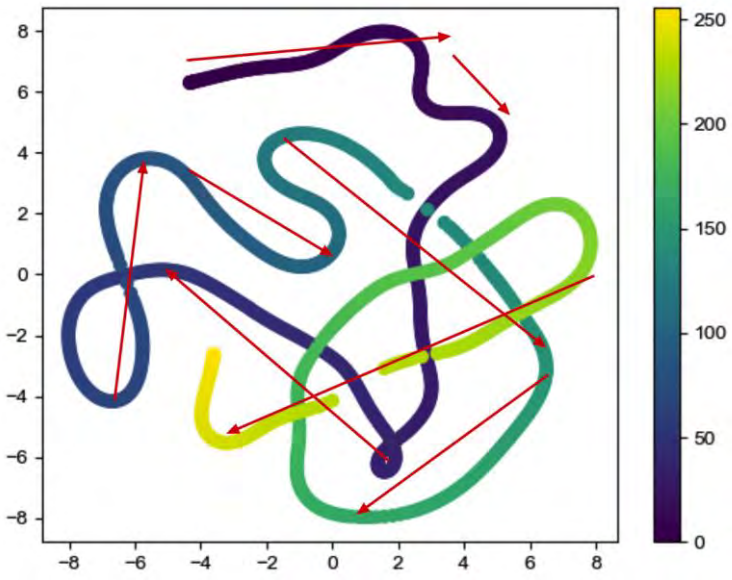


图 25　t-SNE 降维后汽油辛烷值二维散点变化轨迹

六、模型的评价和推广

6.1 模型的评价

6.1.1 模型的优点

经过对比验证、大量的程序分析，本文所构建的模型具有一定的创新性、适用性和正确性，主要体现为如下：

1、站在整体工程问题的角度进行分析，而不是每个问题独立开。文中多处强调了问题一至问题五的内在联系和递进关系，通过数据预处理、主要影响变量筛选、辛烷值损失预测模型、辛烷值优化模型、辛烷值和硫含量变化轨迹这五步能够实现数据挖掘技术对辛烷值的有效建模和优化，对于石化等化工企业有着姣好的借鉴作用。

3、问题一对 285 和 313 号两个样本的数据预处理采用 5 种整定方法，并与附件 1 的结果进行了对比，验证了数据的准确性。

4、问题二在筛选前首先进行了对所有样本的整定和清洗工作，得到更加准确和参考性强的样本、变量数据。

5、问题二设计了一套综合筛选模型，探索了 4 种特征选择和降维方法，并融合各方法的优点，验证了模型的优势。

6、问题二进一步通过将非操作变量和操作变量分开单独建立辛烷值模型，从而分别提取主要影响变量，改善了综合筛选模型，保证了代表性的要求。

7、问题二在改善后的综合筛选模型基础上将自变量间的相关度高的进行剔除，从而确保模型的独立性优势。

8、问题三分析各预测模型的特点，选择能够解决维度爆炸效果较好的 LSTM 模型，利用转化的思想通过对辛烷值进行预测来预测辛烷值损失。

9、问题三通过 GA 算法改进 LSTM 预测模型，能够优化参数选择，提升预测精度。

10、问题三在 LSTM 预测结果分析后，还对比了 LSTM 在辛烷值预测中算法的适用性，通过与 RF、SVR、XGBoost 三种方法对比 RMSE 和 MAPE，验证 LSTM 的模型优势。

11、问题四通过结合辛烷值预测模型和硫含量预测模型，建立优化的目标函数和约束条件。

12、问题四将硫含量的约束条件进行转化，变为障碍函数加入到目标函数中。

13、问题四考虑到工程实际调整的时长限制，在优化时不必趋向全局最优，而是在满足辛烷值损失的降幅大于 30%条件即可停止。

14、整个建模过程不断与实际工艺流程对比，筛选结果、预测结果和优化结果均与实际工艺相符，且与文献中的结果接近，具有工程实用价值。

6.1.2 模型的缺点

1、在综合筛选模型中各方法的权重没有进行细致化探讨，而只是根据各方法对工艺流程的符合度来进行设定，缺乏一般性方法的指导，下一步需要在此处进行进一步的改进。

2、由于样本数据中部分数据需要被删除，造成训练集和测试集的数量减少，可能会影响预测的效果和精度。

3、在将 25 维操作变量经 t-sne 降维后，新生成的两个维度难以有直接的物理变量与其对应，因而生成的可视化结果存在一定物理意义上的欠缺。

6.2 模型的推广

本文立足于整体的角度，提出了一整套采用数据挖掘技术求解辛烷值和优化辛烷值操作的数学方法。从数据预处理开始，到筛选主要影响变量，提出适合于辛烷值的预测模型，建立优化模型，并最终通过可视化图像展示辛烷值和硫含量这两个主要指标的变化轨迹，每一步都很重要，都承接着下一步。

本文模型多处满足工程实际需求，如筛选主要变量时保证代表性和独立性的要求；形成变化轨迹时按照等步长的方法，平稳生产；甚至在建立优化模型的目标函数时就将工艺上对步长梯度的要求考虑了进去，辛烷值损失降幅达到30%以上即可。

综合上述分析，本文不仅具有较高的实际工厂应用价值，还在学术层面具有一定的创新性，可以在实践中丰富理论，也可以在理论中通过实践来检验，具有较高的可推广性。

七、参考文献

[1]李林, 唐铭辰, 胡炎兴. 催化裂化轻汽油醚化催化剂的研究进展[J]. 化学工业与工程技术, 2012, 33(04):44-48.

[2]宋月芹, 徐龙伢, 谢素娟, 吴治华, 王清遐. ZSM-5 分子筛催化剂上液化石油气低温芳构化制取高辛烷值汽油[J]. 催化学报, 2004(03):199-204.

[3]龚剑洪, 毛安国, 刘晓欣, 周庆水. 催化裂化轻循环油加氢-催化裂化组合生产高辛烷值汽油或轻质芳烃(LTAG)技术[J]. 石油炼制与化工, 2016, 47(09):1-5.

[4]熊佐松, 黄辉, 陈春锋, 孙珊珊, 马萌宏, 史彬, 鄢烈祥. 应用智能可视化优化方法确定催化重整的优化操作区域[J]. 西安交通大学学报, 2017, 51(03):136-140.

[5]彭朴, 陆婉珍. 汽油辛烷值和组成的关系[J]. 石油炼制与化工, 1981(06):27-38.

[6]张敏, 袁辉. 拉依达(PauTa)准则与异常值剔除[J]. 郑州工业大学学报, 1997(01):87-91.

[7]Bijlsma S , Bobeldijk I , Verheij E R , et al. Large-scale human metabolomics studies: a strategy for data (pre-) processing and validation.[J]. Analytical Chemistry, 2006, 78(2):567-574.

[8]侍建国, 张亦飞. 拉依达准则在处理区域水文数据异常值中的应用[J]. 海河水利, 2016(05):49-51.

[9]陈功平, 王红. 改进 Pearson 相关系数的个性化推荐算法[J]. 山东农业大学学报(自然科学版), 2016, 47(06):940-944.

[10]胡军, 张超, 陈平雁. 非参数双变量相关分析方法 Spearman 和 Kendall 的 Monte Carlo 模拟比较[J]. 中国卫生统计, 2008, 25(06):590-591.

[11]于福兰. 利用 Kendall 模相关系数法评价地表水水质变化趋势[J]. 水利规划与设计, 2016(03):26-27+73.

[12]刘哲夫. 新型特征选择与机器学习结合方法在化工数据中的应用[D]. 中国石油大学（北京）, 2016.

[13]齐万松, 姬晓军, 侯玉宝, 秦正军. S Zorb 装置降低汽油辛烷值损失的探索与实践[J]. 炼油技术与工程, 2014, 44(11):5-10.

[14]周欢, 齐万松, 李宏勋. S Zorb 装置辛烷值损失大原因的分析与措施[J]. 云南化工, 2019, 46(09):90-91+95.

[15]Hochreiter S，Schmidhuber J. Long short-term memory[J]. Neural Computation，1997，9(8)：1735-1780.

[16]贾华宇, 杜文莉, 范琛, 杨明磊. S-Zorb 反应器的机理建模与基于改进鲸群算法的参数估计[J]. 高校化学工程学报, 2018, 32(06):1395-1402.

[17]温惠英, 张东冉, 陆思园. GA-LSTM 模型在高速公路交通流预测中的应用[J]. 哈尔滨工业大学学报, 2019, 51(09):81-87+95.

[18]徐莉, 邹亢, 徐广通, 毛安国. S Zorb 工业吸附剂结构、组成及再生行为研究[J]. 石油炼制与化工, 2013, 44(06):44-48.

[19]田景芝, 杜晓昕, 郑永杰, 李郁, 荆涛. 基于 PSO-BP 神经网络的加氢脱硫柴油硫含量的预测研究[J]. 石油化工, 2017, 46(01):62-67.

附录

问题一的程序

```
# coding:utf-8
"""
@name:代码模板
@desc:引用、代码注释规范
@author: fengbin
@file:test.py
@time:2020/9/16 9:42  下午
@version:1.0
"""

import numpy as np
import pandas as pd
import pandas_profiling as ppf

def ppff(data, name):
    pfr = ppf.ProfileReport(data)
    pfr.to_file(name)
    return 1


if __name__ == '__main__':
    datapath = "data.xlsx"
    data = pd.read_excel(datapath)


    # 三个数组/列表记录第几列不对  0 过多/siagma 不对
    zero = []
    sigma = []
    total = []
    # 数组/列表记录每个样本出现大于 3sigma 的变量数
    sigmayb = [0]*data.shape[0]

    for i in range(data.shape[1]):
    # 循环  每一列
        lie = data.iloc[:, i].to_numpy()
        mea = np.mean(lie)
        s = np.std(lie, ddof=1)
        # 计算每一列  均值 mea  标准差 s
```

```
        # 计算 0 个数
        zeros = lie.tolist().count(0)
        print(str(i)) # +str(data.shape[i].columns)
        if zeros > 0.2*data.shape[0]:
            zero.append(i)
            total.append(i)
        print("zeros:"+str(zeros/data.shape[0]))
        # 如果大于 20% 列编号记录在 zero 和 total 列表中


        flag = 0
        for t in range(data.shape[0]):
            if (abs(lie[t]-mea) > 3*s):
                print(">3sigma"+str(t))
                sigmayb[t] = sigmayb[t] + 1
                flag = 1
                # 累加样本出现不符合 3sigma 的次数
        if flag == 1:
            sigma.append(i)
            if zeros <= 0.2*data.shape[0]:
                total.append(i)
            # 如果大于 3sigma 列编号记录在 sigma 和 total 列表中

# 开始循环
flag = 0
l0 = 0
l1 = 325
# 循环终止条件
while( abs(l1-l0)>5 ):
        l0 = l1
        sigmaybnp = np.array(sigmayb)
        where = np.where(sigmaybnp > 7)
        # where 输出样本中多于 7 个变量出现大于 3sigma 情况的样本位置

        print("where")
        print(where)
        print(list(where))
        a = np.array(list(where))
        print(a)
        a = a[0]
        print(a)

        print("drop")
```

```
        data = data.drop(labels=a, axis=0)
        data = data.reset_index(drop=True)
        # 删除样本 样本编号更新

        zero = []
        sigma = []
        total = []
        sigmayb = [0]*data.shape[0]
        for i in range(data.shape[1]):
            lie = data.iloc[:, i].to_numpy()
            mea = np.mean(lie)
            s = np.std(lie, ddof=1)

            zeros = lie.tolist().count(0)
            print(str(i)) # +str(data.shape[i].columns)
            if zeros > 0.2*data.shape[0]:
                zero.append(i)
                total.append(i)
            print("zeros:"+str(zeros/data.shape[0]))

            flag = 0
            for t in range(data.shape[0]):
                if (abs(lie[t]-mea) > 3*s):
                    print(">3sigma"+str(t))
                    sigmayb[t] = sigmayb[t] + 1
                    flag = 1
            if flag == 1:
                sigma.append(i)
                if zeros <= 0.2*data.shape[0]:
                    total.append(i)
print(data)
data = data.reset_index(drop=True)
total1 = total[8:]
# total 总计需要删的变量（列） 8 是非操作变量不能删除
data.drop(data.columns[total1], axis=1, inplace=True)
# 删除列
print(data)
data.to_excel("Q111result.xlsx")
print(data.iloc[:, 0])

datarare.drop(datarare.columns[total1], axis=1, inplace=True)
print(datarare)
print(datarare.iloc[:, 0])
de = data.iloc[:, 0] - 1
```

```
    print(de)
    datarare.drop(index=de, inplace=True)
    print(datarare)
    datarare.to_excel("datarare.xlsx")
```

问题二的程序

1. 计算各个评价指标

```
import pandas as pd
import numpy as np
from scipy.spatial.distance import pdist, squareform
from sklearn.decomposition import PCA
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import SVR


# 加载石油化工生产数据
def load_data():
    data = pd.read_excel('data_delete_zero.xlsx')
    X = []
    names = []
    for i in data:
        # RON 损失列为因变量
        if i == '辛烷值 RON.1':
            Y = np.array(data[i])
        # index 列，直接跳过
        elif i == 'index' or i == 'RON 损失':
            continue
        # 其余变量均作为特征进行筛选
        else:
            X.append(np.array(data[i]))
            names.append(i)
    X = np.array(X)
    print('X.shape', X.shape)
    print('Y.shape', Y.shape)
    return X, Y, names


# 将数据写入 excel 文件
def data_write_excel(data, filename, writer=None):
    data = pd.DataFrame(data)
```

```
    if not writer:
        writer = pd.ExcelWriter(filename + '1.xlsx')
        data.to_excel(writer, float_format='%.5f', sheet_name=filename)
        writer.save()
        writer.close()
    else:
        data.to_excel(writer, float_format='%.5f', sheet_name=filename)


# 计算皮尔森相关系数矩阵，保存至 pearson.xlsx
def cal_Pearson(X, Y):
    # Y 变量在矩阵最后一行(列)
    pearson = np.corrcoef(X, Y)
    data_write_excel(pearson, 'pearson')


# 计算 Spearman 相关系数矩阵，保存至 spearman.xlsx
def cal_Spearman(X, Y):
    # Y 变量在矩阵最后一行(列)
    X = X.T
    Y = Y.reshape(X.shape[0], 1)
    X = np.append(X, Y, axis=1)
    spearman = pd.DataFrame(X)
    spearman = spearman.corr('spearman')
    data_write_excel(spearman, 'spearman')


# 计算 kendall 相关系数矩阵，保存至 kendall.xlsx
def cal_Kendall(X, Y):
    # Y 变量在矩阵最后一行(列)
    X = X.T
    Y = Y.reshape(X.shape[0], 1)
    X = np.append(X, Y, axis=1)
    spearman = pd.DataFrame(X)
    spearman = spearman.corr('kendall')
    data_write_excel(spearman, 'kendall')


# 距离相关系数，来自
def distcorr(X, Y):
    X = np.atleast_1d(X)
    Y = np.atleast_1d(Y)
    if np.prod(X.shape) == len(X):
        X = X[:, None]
```

```
    if np.prod(Y.shape) == len(Y):
        Y = Y[:, None]
    X = np.atleast_2d(X)
    Y = np.atleast_2d(Y)
    n = X.shape[0]
    if Y.shape[0] != X.shape[0]:
        raise ValueError('Number of samples must match')
    a = squareform(pdist(X))
    b = squareform(pdist(Y))
    A = a - a.mean(axis=0)[None, :] - a.mean(axis=1)[:, None] + a.mean()
    B = b - b.mean(axis=0)[None, :] - b.mean(axis=1)[:, None] + b.mean()

    dcov2_xy = (A * B).sum() / float(n * n)
    dcov2_xx = (A * A).sum() / float(n * n)
    dcov2_yy = (B * B).sum() / float(n * n)
    dcor = np.sqrt(dcov2_xy) / np.sqrt(np.sqrt(dcov2_xx) * np.sqrt(dcov2_yy))
    return dcor


# 计算距离相关系数并保存在 dcorr.xlsx
def cal_dcorr(X, Y, names):
    dcorr = []
    for i in range(X.shape[0]):
        dcorr.append([distcorr(X[i], Y), names[i]])
    data_write_excel(dcorr, 'dcorr')


# 计算主成分
def cal_PCA(X):
    X = X.T
    # 五个主成分
    pca = PCA(n_components=5)
    # 降维后的数据
    data_low = pca.fit_transform(X)
    # 降维后近似恢复的数据
    data_restore = pca.inverse_transform(data_low)
    writer = pd.ExcelWriter('pca1.xlsx')
    data_write_excel(data_low, 'low', writer=writer)
    data_write_excel(data_restore, 'restore', writer=writer)
    # 主成分贡献率
    data_write_excel(pca.explained_variance_ratio_, 'ratio', writer=writer)
    # 主成分方差
    data_write_excel(pca.explained_variance_, 'variance', writer=writer)
    # 主成分在各个变量的负载
```

```
    data_write_excel(pca.components_.T, 'component', writer=writer)
    # 主成分个数
    print(pca.n_components_, ' n_components')
    writer.save()
    writer.close()


# 使用 RFE 递归特征消除法
def cal_RFE(X, Y, names):
    X = X.T
    # use linear regression as the model
    lr = LinearRegression()
    # rank all features, i.e continue the elimination until the last one
    rfe = RFECV(lr, step=1, cv=5)
    rfe.fit(X, Y)
    res = list(zip(map(lambda x: round(x, 4), rfe.ranking_), names))
    print("Features sorted by their rank:")
    print(res)
    data_write_excel(res, 'RFElr')

    # use lasso as the model
    lr = Lasso()
    # # rank all features, i.e continue the elimination until the last one
    rfe = RFECV(lr, step=1, cv=5)

    # svc = SVR(kernel="linear", C=1)
    # rfe = RFECV(svc, step=1, cv=5)
    rfe.fit(X, Y)
    res = list(zip(map(lambda x: round(x, 4), rfe.ranking_), names))
    print("Features sorted by their rank:")
    print(res)
    data_write_excel(res, 'RFElasso')

    # X = X.T
    # # use linear regression as the model
    # # lr = LinearRegression()
    # # lr = Lasso()
    # # # rank all features, i.e continue the elimination until the last one
    # # rfe = RFECV(lr, step=1, cv=5)
    #
    # svc = SVR(kernel="linear", C=1)
    # rfe = RFECV(svc, step=1, cv=5)
    # rfe.fit(X, Y)
    # res = list(zip(map(lambda x: round(x, 4), rfe.ranking_), names))
```

```
    # print("Features sorted by their rank:")
    # print(res)
    # data_write_excel(res, 'RFEsvr')


if __name__ == '__main__':
    X, Y, names = load_data()
    ## 计算皮尔森相关系数矩阵
    # cal_Pearson(X, Y)
    ## 计算斯皮尔曼相关系数矩阵
    # cal_Spearman(X, Y)
    ## 计算肯达尔相关系数矩阵
    # cal_Kendall(X, Y)
    # 计算距离相关系数
    cal_dcorr(X, Y, names)
    ## 计算主成分
    # cal_PCA(X)
    ## 使用 RFE 计算特征顺序
    # cal_RFE(X, Y, names)
```

2. 根据综合评价体系筛选特征

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from VariableFilter_Operation import load_data
from test import ppff


def scale(data):
    minmax = MinMaxScaler()
    data = minmax.fit_transform(data)
    # print(input_reframed)
    print(minmax.scale_)
    print(minmax.min_)
    return data, minmax.scale_[-1], minmax.min_[-1]


if __name__ == '__main__':
    num_variable = 217
    num_sample = 252
    suffix = '_Operation'                    # 文件名后缀
```

```
rfela = pd.read_excel("RFElasso" + suffix + ".xlsx")
rfela1 = ((num_variable - rfela.iloc[:, 1]) / num_variable).to_numpy().reshape(-1, 1)

rfelr = pd.read_excel("RFElr" + suffix + ".xlsx")
rfelr1 = ((num_variable - rfelr.iloc[:, 1]) / num_variable).to_numpy().reshape(-1, 1)

dco = pd.read_excel("dcorr" + suffix + ".xlsx")
dco1, inscale_, inmin_ = scale(dco.iloc[:, 1].to_numpy().reshape(-1, 1))

pers = pd.read_excel("pearson" + suffix + ".xlsx")
pers1, inscale_, inmin_ = scale(np.abs(pers.iloc[:-1, num_variable + 1].to_numpy().reshape(-1, 1)))

comp = pd.read_excel("pca" + suffix + ".xlsx", sheet_name="component")
comp1, inscale_, inmin_ = scale(comp.abs())
ratio = pd.read_excel("pca" + suffix + ".xlsx", sheet_name="ratio").iloc[:, 1].to_numpy()
# print(ratio)
# t = comp1[...,0]
# t1 = comp1[...,0]*ratio[0]
# print(comp1[...,0]*ratio[0])
# print(comp1[...,4]*ratio[4])

pca1 = np.abs(comp1[..., 1]) * ratio[0] + \
       np.abs(comp1[..., 2]) * ratio[1] + \
       np.abs(comp1[..., 3]) * ratio[2] + \
       np.abs(comp1[..., 4]) * ratio[3] + \
       np.abs(comp1[..., 5]) * ratio[4]

pca11, inscale_, inmin_ = scale(pca1.reshape(-1, 1))

total = rfela1 + dco1 + pers1 + pca11
total = total.squeeze()
top_k = 50
top_k_idx = total.argsort()[::-1][0:top_k]
print(top_k_idx)

X, Y, names = load_data()

name50 = np.array(names)[top_k_idx].reshape(-1, 1)

tttemp = top_k_idx.reshape(-1, 1)
result50 = np.hstack((tttemp, name50))

result50 = pd.DataFrame(result50)
result50.to_excel("result40" + suffix + ".xlsx")
```

```
    print(np.array(names)[top_k_idx])

    print(X[top_k_idx, :])
    X1 = X[top_k_idx, :]

    X1 = X1.T
    Y = Y.reshape(num_sample, 1)
    XY = np.append(X1, Y, axis=1)
    XY = pd.DataFrame(XY)
    tem = np.array(names)[top_k_idx].tolist()
    print("tem")
    print(tem)

    tem.append('辛烷值')
    print("tem1")
    print(tem)
    XY.columns = tem

    ppff(XY, "top40+1" + suffix + ".html")

    # target = 229
    # k = 15
    # cols = total.nlargest(k, target)[target].index

    # dfname.columns.values.tolist()  # 列名称
```

3. 并查集对高度相关变量进行划分

```
import pandas as pd
# datapath = 'result40_Operation.xlsx'
datapath = 'result40_S.xlsx'
data = pd.read_excel(datapath, sheet_name='Sheet2')
corr = data['corr']
parent = {}
cnt = {}
res = {}


def find(v):
    if parent[v] != v:
        parent[v] = find(parent[v])
```

```
        return parent[v]


names = []
for i in corr:
        words = i.split(' ')
        print(words)
        idx2 = 5
        if words[1] != 'is':
                words[0] += ' ' + words[1]
                idx2 += 1
        if idx2 + 1 < len(words) and words[idx2 + 1] != '（':
                words[idx2] += ' ' + words[idx2 + 1]
        parent[words[0]] = words[0]
        parent[words[idx2]] = words[idx2]
        names.append(words[0])
        names.append(words[idx2])
        cnt[words[0]] = cnt[words[0]] + 1 if words[0] in cnt else 1
        cnt[words[idx2]] = cnt[words[idx2]] + 1 if words[idx2] in cnt else 1

for i in corr:
        words = i.split(' ')
        idx2 = 5
        if words[1] != 'is':
                words[0] += ' ' + words[1]
                idx2 += 1
        if idx2 + 1 < len(words) and words[idx2 + 1] != '（':
                words[idx2] += ' ' + words[idx2 + 1]
        p0 = find(words[0])
        p2 = find(words[idx2])
        if cnt[p0] < cnt[p2] or words[0] > words[idx2]:
                p0, p2 = p2, p0
        parent[p2] = p0


for i in parent:
        parent[i] = find(i)
        if parent[i] in res:
                res[parent[i]].add(i)
        else:
                res[parent[i]] = {i, parent[i]}
print(res)
result = []
for i in res:
```

```
        tmp = list(res[i])
        result.append(tmp)
result = pd.DataFrame(result)
writer = pd.ExcelWriter('result40_S1.xlsx')
result.to_excel(writer, sheet_name='Sheet1')
writer.save()
writer.close()
```

问题三的程序

1. 预测主程序

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import xgboost as xgb
import torch
from torch import nn
from torch.autograd import Variable
import torch.utils.data as Data
from predict.lstm import lstm_reg
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sko.GA import GA
from sko.PSO import PSO
from sko.SA import SA
from sko.AFSA import AFSA
from sko.DE import DE

from sklearn.ensemble import RandomForestRegressor

def scale(data):
    minmax = MinMaxScaler()
    data = minmax.fit_transform(data)
    # print(input_reframed)
    np.savetxt('Smin_', minmax.min_)
    np.savetxt('Sscale_', minmax.scale_)
    # print(minmax.scale_)
```

```
    # print(minmax.min_)
    return data, minmax.scale_[-1], minmax.min_[-1]

def mape(y_true, y_pred):
    return np.mean(np.abs((y_pred - y_true) / y_true)) * 100


def train_test(input, output, split=0.8):

    useful_data = pd.DataFrame(input)
    useful_data = useful_data.values
    # print(useful_data)

    n_features = useful_data.shape[1]
    # input_reframed = series_to_supervised(useful_data, last, 1)
    # print(input_reframed)

    num = int(useful_data.shape[0]*split)

    # print(n_hours*n_features)

    # input_reframed = input_reframed.values

    # train_x, train_y, test_x, test_y = train_test_split(input, output, test_size=0.2, random_state=42)

    # temp = last * n_features
    train_x = useful_data[:num, :]
    test_x = useful_data[num:, :]
    # print(n_hours)
    # print(n_features)
    train_y = output[:num]
    test_y = output[num:]

    # reshape 将输入的 x 数据 变为三维
    train_x = train_x.reshape((train_x.shape[0], 1, n_features))
    train_y = train_y.reshape(-1, 1, 1)
    test_x = test_x.reshape((test_x.shape[0], 1,    n_features))

    # print(train_x.shape)
    # print(train_y.shape)
    # print(test_x.shape)
    # print(test_y.shape)

    train_x = torch.from_numpy(train_x).float()
```

```
    test_x = torch.from_numpy(test_x).float()
    train_y = torch.from_numpy(train_y).float()

    # print("test_y"+str(np.size(test_y)))
    # print("train_x"+str(np.size(train_x)))

    return train_x, train_y, test_x, test_y


def trainmain(inputdata, outputdata, input_size, classname, batch_size = 32, epoch = 1000, learning_rate = 0.0015, hidden_size = 8):
    """
    训练的主程序:
    :param inputdata:
    :param output:
    :param input_size:
    :param split:
    :param classname:
    :return:
    """
    minloss = 100000000   # 便于比较设置初始值为 100

    # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    #
    # if torch.cuda.is_available():
    #         print("cuda")

    # from nnprocess import train_test, scale

    input, inscale_, inmin_ = scale(inputdata)
    # outputdata = outputdata.reshape(-1, 1)

    outputdata = outputdata.to_numpy().reshape(-1, 1)
    output, outscale_, outmin_ = scale(outputdata)
    # output = output.squeeze()
    # input_size = 1
    train_x, train_y, test_x, test_y = train_test(input, output, split=0.8)
    # train_x, train_y, test_x, test_y = train_test_split(input, output, test_size = 0.2, random_state = 42)

    # train_x = train_x.to(device)
    # train_y = train_y.to(device)
    # test_x = test_x.to(device)

    net = lstm_reg(input_size, hidden_size)
```

```
# (input_size, hidden_size)
# input_size need to be the n_features; hidden_size should be adjustable

# net.to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

# 先转换成 torch 能识别的 Dataset
torch_dataset = Data.TensorDataset(train_x, train_y)
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,    # torch TensorDataset format
    batch_size=batch_size,    # mini batch size
    shuffle=False,   # 要不要打乱数据
    # num_workers=2,   # 多线程来读数据
)

var_data = Variable(test_x)
var_datatrain = Variable(train_x)

net = net.train() # 转换成测试模式

# 开始训练
for i in range(epoch):
    epoch_loss = 0
    sum = 0

    for step, (batch_x, batch_y) in enumerate(loader):
        sum = sum + 1
        var_x = Variable(batch_x)
        var_y = Variable(batch_y)
        # 前向传播
        out = net(var_x)
        # print(out[:, -1])
        loss = criterion(out, var_y)
        # print(loss.data.numpy())
        # print("w")
        # 反向传播
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # if torch.cuda.is_available():
        #         loss = loss.cpu()
        epoch_loss = epoch_loss+loss.data.numpy()
```

```
                # if (i + 1) % 5 == 0: # 每 10 次输出结果
                    # if torch.cuda.is_available():
                    #     loss = loss.cpu()
                    # if step % 20 == 0:
                    #     print('Epoch: {}, step:{}, batchLoss: {:.5f}'.format(i + 1, step, loss.data.numpy() ))

            if (i + 1) % 20 == 0: # 每 10 次输出结果
                # print("------------")
                # print("epoch_train_loss"+str(epoch_loss/sum))
                pred_test = net(var_data)  # 测试集的预测结果
                # if torch.cuda.is_available():
                #     pred_test = pred_test.cpu()

                # 改变输出的格式
                pred_test = pred_test.data[:, -1].numpy().squeeze(1)
                # print(test_loss)
                tloss = metrics.mean_squared_error(pred_test, test_y)
                print("test_loss:" + str(tloss))

                if minloss > tloss:
                    torch.save(net, classname+'net.pkl')  # 保存整个网络
                    minloss = tloss
        torch.save(net, classname+'net'+str(minloss)+'.pkl')
        return minloss


def restore_net(inputdata, outputdata, split, classname, batch_size=16):
        """
        重新加载保存的最优网络 并 进行训练
        :return: int，加和结果
        """
        # restore entire net
        net = torch.load(classname+"net.pkl")
        net = net.eval() # 转换成测试模式


        # device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        #
        # if torch.cuda.is_available():
        #     print("cuda")
        #
        # input, inscale_, inmin_ = scale(inputdata)
        # output, outscale_, outmin_ = scale(outputdata)
        # train_x, train_y, test_x, test_y = train_test(input, output, split, last)
```

```
# device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#
# if torch.cuda.is_available():
#         print("cuda")

# from nnprocess import train_test, scale

input, inscale_, inmin_ = scale(inputdata)
# outputdata = outputdata.reshape(-1, 1)

outputdata = outputdata.to_numpy().reshape(-1, 1)
output, outscale_, outmin_ = scale(outputdata)
# output = output.squeeze()
# input_size = 1
train_x, train_y, test_x, test_y = train_test(input, output, split)
# train_x, train_y, test_x, test_y = train_test_split(input, output, test_size = 0.2, random_state = 42)

# train_x = train_x.to(device)
# train_y = train_y.to(device)
# test_x = test_x.to(device)

# 先转换成 torch 能识别的 Dataset
torch_dataset = Data.TensorDataset(train_x, train_y)
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
        dataset=torch_dataset,    # torch TensorDataset format
        batch_size=batch_size,    # mini batch size
        shuffle=False,    # 要不要打乱数据
        # num_workers=2,    # 多线程来读数据
)

var_data = Variable(test_x)
var_datatrain = Variable(train_x)

pred_test = net(var_data) # 测试集的预测结果

# if torch.cuda.is_available():
#         pred_test = pred_test.cpu()

# 改变输出的格式
pred_test = pred_test.data[:, -1].numpy().squeeze(1)

test_y = (test_y-outmin_)/outscale_
```

```
pred_test = (pred_test-outmin_)/outscale_

plt.plot(test_y, 'b', label='真实值')
plt.plot(pred_test, 'r-', label='预测值')
plt.xlabel('测试样本数')
if classname == "229test":
    plt.ylabel('辛烷值')
    plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
    plt.legend()
    plt.savefig("辛烷值预测.svg", format="svg")

if classname == "S229test":
    plt.ylabel('脱硫率(%)')
    plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
    plt.legend()
    xxx = range(0, 20, 2)
    yyy = range(94, 101, 2)
    plt.xticks(xxx)
    plt.yticks(yyy)
    plt.savefig("脱硫率预测.svg", format="svg")

plt.show()

plt.close()

# 导出辛烷值预测以及真实值到 excel 中
# out_pred_test = pred_test.reshape(-1, 1)
#
# dfout_pred_test = pd.DataFrame(out_pred_test)
# dftest_y = pd.DataFrame(test_y)
# dfwucha = dfout_pred_test - dftest_y
# df_out = pd.concat([dfout_pred_test, dftest_y, dfwucha], axis=1)
# df_out.columns=['pred', 'true', 'pred-true']
# df_out.to_excel("xinwanpred.xlsx")


# plot(test_y, pred_test, classname)

rmse = np.sqrt(metrics.mean_squared_error(test_y, pred_test))
print(classname+"rmse"+str(rmse))
# maezhi = metrics.mean_absolute_error(test_y, pred_test)
# print(classname+"mae"+str(maezhi))
mmape = mape(test_y, pred_test)
print(classname+"mape"+str(mmape))
```

```
    # np.save("predshangwang.npy", pred_test)

    return pred_test


def traintest(name, trainflag, batch_size, epoch, learning_rate, hidden_size):
    savename = str(name)+"test"
    minloss = 0
    if name == 350:
        col = pd.read_excel("./367-17result/result40_1revised.xlsx")
        colu = col.iloc[:, 1]
        colu = colu + 3
        # print(colu)

        data = pd.read_excel("./367-17result/data_delete_zero.xlsx")
        X = data.iloc[:, colu]
        Y = data.iloc[:, 2]
        if trainflag == True:
            trainmain(X, Y, X.shape[1], savename, batch_size = 32, epoch = 1000, learning_rate = 0.005,
hidden_size = 8)
        # X 输入变化  Y 不动

    if name == 229 :
        col = pd.read_excel("result40revised.xlsx")
        colu = col.iloc[:, 1]
        colu = colu + 3
        # print(colu)

        data = pd.read_excel("Q1result.xlsx")
        # data = pd.read_excel("datarare.xlsx") # 70+样本验证
        X = data.iloc[:, colu]
        Y = data.iloc[:, 2]
        if trainflag == True:
            trainmain(X, Y, X.shape[1], savename, batch_size = 64, epoch = 2000, learning_rate = 0.005,
hidden_size = 8)
        # X 输入变化  Y 不动
        #
        # X 输入变化  Y 不动

        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)

        # model = xgb.XGBRegressor(max_depth=5, learning_rate=0.1, n_estimators=160, silent=False,
objective='reg:gamma')
        model = RandomForestRegressor()
```

```
            model.fit(X_train, y_train)
            #  对测试集进行预测
            ans = model.predict(X_test)

            rmse = np.sqrt(metrics.mean_squared_error(ans, y_test))
            print( "rmse" + str(rmse))
            # maezhi = metrics.mean_absolute_error(test_y, pred_test)
            # print(classname+"mae"+str(maezhi))
            mmape = mape(ans, y_test)
            print("mape" + str(mmape))
            print("0")

        if name == "S229":
            col = pd.read_excel("Srevised.xlsx")
            colu = col.iloc[:, 1]
            colu = colu + 4
            # print(colu)

            data = pd.read_excel("./deleteS/Sdata.xlsx")
            X1 = data.iloc[:, colu]
            Y1 = data.iloc[:, 11]
            randsta = 8
            X = shuffle(X1, random_state=randsta)
            Y = shuffle(Y1, random_state=randsta)
            if trainflag == True:
                # minloss = trainmain(X, Y, X.shape[1], savename, batch_size=8, epoch=8000,
learning_rate=0.0059, hidden_size=3)
                minloss = trainmain(X, Y, X.shape[1], savename, batch_size, epoch, learning_rate, hidden_size)



        pred_test = restore_net(X, Y, 0.8, savename)
        # pred_test  是 ndarray 输出

        # print("minloss"+str(minloss))

        return minloss

def getPredictVal(X0):
        # X 为操作变量
        # X0 为非操作变量
        # index 为非操作变量的位置
        data = pd.read_excel("./predict/Q1result.xlsx")
        # print(X.shape)
```

```
    # print(X0.shape)
    Y = data.iloc[:, 2]
    # X = list(X)
    # X0 = list(X0)
    # for i in range(len(X0)):
    #         X0[i] = np.append(X0[i], X[i])
    X0 = pd.DataFrame(X0)
    # X0 = X0.T
    # print(X0.shape)
    pred_test = restore_net(X0, Y, 0, "test")
    return pred_test

def ValS(X):
    print("batch_size:"+str(X[0]))
    print("learning_rate:"+str(X[1]))
    print("hidden_size:"+str(X[2]))
    batch_size = int(X[0])
    learning_rate = X[1]
    hidden_size = int(X[2])
    epoch = 1500
    # minloss = traintest("S229", True, batch_size, epoch, learning_rate, hidden_size)

    minloss = traintest(229, True, batch_size, epoch, learning_rate, hidden_size)
    print("minloss:"+str(minloss))

    return minloss

def schaffer(X):
    return ValS(X)

# 遗传算法
def cal_ga( dim, lbb, ubb):
    print('GA')
    # n_dim 为变量维数,
    # lb 和 ub 为变量的上下限，维数应等于 n_dim
    # size_pop 为种群规模
    # precision 为变量的变化精度，维数应等于 n_dim
    ga = GA(func=schaffer, n_dim=dim, size_pop=50, max_iter=100, lb=lbb, ub=ubb, precision=[1, 1e-4, 1])
    # ga.Chrom = [X1 for i in range(dim)]
    best_x, best_y = ga.run()
    print('best_x:', best_x, '\nbest_y:', best_y)
```

```
if __name__ == '__main__':
    # traintest(350, True)

    # traintest(229, False, batch_size=64, epoch=1500, learning_rate=0.005, hidden_size=8)
    # minloss = traintest(229, True, batch_size, epoch, learning_rate, hidden_size)

    # traintest("S229", False, batch_size=32, epoch=1000, learning_rate=0.003, hidden_size=6)

    # # batch_size = X[0]
    # # learning_rate = X[1]    0.005
    # # hidden_size = X[2]


    # #    64 1500 0.005 8
    dim = 3
    lbb = [32, 0.001, 4]
    ubb = [64, 0.01, 10]
    cal_ga(dim, lbb, ubb)
    #
    # # dim = 3
    # lbb = [2, 0.001, 3]
    # ubb = [32, 0.01, 6]
    # cal_ga(dim, lbb, ubb)
    #
    # cal_PSO(dim, lbb, ubb)
```

2. LSTM 模型程序

```
import torch
from torch import nn
from torch.autograd import Variable
# GPU 使用情况
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class lstm_reg(nn.Module):
    """
    定义长短期记忆网络模型
    :return:  返回网络
```

```
"""
def __init__(self, input_size, hidden_size, output_size=1, num_layers=1):
    super(lstm_reg, self).__init__()

    self.rnn = nn.LSTM(input_size, hidden_size, num_layers)    # rnn
    self.reg = nn.Linear(hidden_size, output_size)    #  回归

def forward(self, x):
    # print("input")
    # print(x.size())
    x, _ = self.rnn(x)    # (seq, batch, hidden)
    # print("rnnout")
    # print(x.size())

    s, b, h = x.shape
    x = x.view(s * b, h)    #  转换成线性层的输入格式
    # print("xview")
    # print(x.size())

    x = self.reg(x)
    # print("reg")
    # print(x.size())
    x = x.view(s, b, -1)
    # x = self.lu(x)
    # print("out")
    # print(x.size())
    return x
```

问题四的程序

1. 智能搜索算法

```
from sko.GA import GA
from sko.PSO import PSO
from sko.SA import SA
from sko.AFSA import AFSA
from sko.DE import DE
from predict.predmain import getPredictVal
from predict.predmain import getInitialPredictVal
from predict_S.predmain import getPredictVal_S
from predict_S.predmain import getInitialPredictVal_S
```

```
import numpy as np
import pandas as pd
import time
import os


# 从 boundary.xlsx 中加载上限、下限、delta 数据
def load_data():
    data = pd.read_excel('boundary.xlsx')
    low_bound = data['下限']
    high_bound = data['上限']
    return low_bound, high_bound


def schaffer(X_RON_Operation):
    X_S_Operation = np.array([X_RON_Operation[i] for i in S_index])          # S 的操作变量
    # RON 为辛烷值
    RON = getPredictVal(X_RON_Operation, X_RON_NoOpearation)
    print('RONVal', RON)
    # 脱硫率
    S_de_rate = getPredictVal_S(X_S_Operation, X_S_NoOpearation) / 100
    print('S_de_rate', S_de_rate)
    # S_product 为产品硫含量
    S_product = S_original * (1 - S_de_rate)
    print('S_product', S_product)
    # punish 为惩罚函数项
    punish = np.log(1.75 - S_product / 7)
    print('punish', punish)
    return RON, punish, S_product


# 遗传算法
def cal_ga(target1, target2):
    print('GA')
    # n_dim 为变量维数,
    # lb 和 ub 为变量的上下限，维数应等于 n_dim
    # size_pop 为种群规模
    # precision 为变量的变化精度，维数应等于 n_dim
    start = time.time()
    ga = GA(func=schaffer, n_dim=dim, target1=target1, target2=target2, size_pop=30, max_iter=100,
lb=list(low_bound)[:dim],
            ub=list(high_bound)[:dim], precision=1e-6, prob_mut=0.01, X0=X_RON_Operation)
    end = time.time()
    time_delta = end - start
```

```
    best_x, best_y, S_product, satisfied, iterNum = ga.run()
    print('best_x:', best_x, '\nbest_y:', best_y)
    print('S_product:', S_product)
    return best_x, best_y, S_product, satisfied, iterNum, time_delta


# 粒子群
def cal_PSO():
    print('PSO')
    # c1 为个人学习因子
    # c2 为全局学习因子
    pso = PSO(func=schaffer, dim=dim, pop=50, max_iter=600, lb=list(low_bound)[:dim],
              ub=list(high_bound)[:dim], w=0.8, c1=0.5, c2=0.5, X0=X_RON_Operation)
    pso.run()
    print('best_x：', pso.gbest_x, '\nbest_y', pso.gbest_y)


# # 模拟退火
# def cal_SA():
#     print('SA')
#     sa = SA(func=schaffer, x0=X1, T_max=1, T_min=1e-9, L=300, max_stay_counter=150)
#     best_x, best_y = sa.run()
#     print('best_x:', best_x, '\nbest_y', best_y)
#     return best_x, best_y


# # 人工鱼群算法
# def cal_AFSA():
#     afsa = AFSA(schaffer, n_dim=dim, size_pop=50, max_iter=300,
#                 max_try_num=100, step=0.5, visual=0.3,
#                 q=0.98, delta=0.5)
#     best_x, best_y = afsa.run()
#     print('best_x:', best_x, '\nbest_y', best_y)


# # 差分进化算法
# def cal_DE():
#     print('DE')
#     constraint_eq = [
#         lambda x: 1 - x[1] - x[2]
#     ]
#     constraint_ueq = [
#         lambda x: 1 - x[0] * x[1],
#         lambda x: x[0] * x[1] - 5
```

```
#         ]
#             de = DE(func=schaffer, n_dim=dim, size_pop=50, max_iter=800, lb=list(low_bound)[:dim],
ub=list(high_bound)[:dim],
#                     constraint_eq=constraint_eq, constraint_ueq=constraint_ueq)
#
#         best_x, best_y = de.run()
#         print('best_x:', best_x, '\n', 'best_y:', best_y)


if __name__ == '__main__':
    S_index = [14, 5, 9, 11, 4, 20, 16, 2, 6, 8, 13, 23, 1]   # S 模型的操作变量在 RON 模型操作变量中的位置
    name = ['best_X', 'best_Y', 'S_product', 'satisfied', 'iter_num', 'cal_time']
    low_bound, high_bound = load_data()                                  # 加载上下限数据
    data = pd.read_excel('data_325.xlsx')                                          # 读取样本数据

    idx = 0
    while os.path.exists('Best' + str(idx) + '.xlsx'):
        idx += 1
    file_name = 'Best' + str(idx) + '.xlsx'
    cal_data = [pd.DataFrame([]) for i in range(6)]
    writer = pd.ExcelWriter(file_name)
    # cal_data 分别记录优化结果的 X 值、优化结果的 Y 值、优化结果是否满足、迭代计算时间与迭代计算次数

    """
    # 测试将 252 个样本直接输入 S 预测模型的结果
    X_S_NoOpearation = data.iloc[:, list(range(3, 9))].values              # S 的非操作变量
    X_RON_Operation = data.iloc[:, list(range(9, 34))].values               # RON 的操作便量
    X_S_Operation = np.array([X_RON_Operation[:, i] for i in S_index]).T        # S 的操作变量
    X_S = np.append(X_S_NoOpearation, X_S_Operation, axis=1)
    de_S_rate = getInitialPredictVal_S(X_S)
    """

    S = data['硫含量,μg/g'].values                                                  # 读取样本原料硫含量数据


    for i in range(66, 67):
        sample = data[i:i+1]                                                    # 读取第 i+1 个样本
        RON_loss = sample.iloc[:, list(range(1, 2))].values[0][0]               # 获取其 RON 损失
        RON_after = sample.iloc[:, list(range(2, 3))].values[0][0]              # 获取成本 RON 值
        RON_original = sample.iloc[:, list(range(4, 5))].values[0][0]           # 获取原料 RON 值
```

```
            S_original = S[i]
            print(i, '的原料硫含量为', S_original)

            target1 = RON_original - RON_loss * (1 - 0.3)                    # 获取优化目标值
            target2 = RON_original - 0.7                                     # 获取优化目标值上限(一般工况条件下，辛烷值损失最低为 0.7)
            print('optimization target is ', target1)

            initVal = sample.iloc[:, [i for i in range(3, 34) if i != 8]].values          # 获取其变量初始值
            predictVal = getInitialPredictVal(initVal)                       # 变量初始值的预测值
            print('initPredictVal is ', predictVal)

            X_RON_NoOpearation = sample.iloc[:, [3, 4, 5, 6, 8]].values      # RON 的非操作变量
            X_RON_Operation = sample.iloc[:, list(range(9, 34))].values      # RON 的操作便量
            X_S_NoOpearation = sample.iloc[:, list(range(3, 9))].values      # S 的非操作变量
            print(X_RON_Operation.shape, X_RON_NoOpearation.shape, X_S_NoOpearation.shape)

            # print(X0.shape, X1.shape, 'XO and X1 shape')
            # # print(X0, X1)
            dim = 25

            best = cal_ga(target1, target2)                                  # 返回各个样本的 best_X, best_Y, 是否满足, 迭代次数, 迭代时间
            print(best)
            best = list(best)
            for j in range(len(cal_data)):
                if type(best[j]) != np.ndarray:
                    best[j] = [best[j]]
                cal_data[j] = pd.concat([cal_data[j], pd.DataFrame(best[j])], axis=1)

    for i in range(len(cal_data)):
        cal_data[i].to_excel(writer, sheet_name=name[i])
    writer.save()
    writer.close()
    # cal_DE()
    # cal_PSO()
    # cal_SA()
    # cal_AFSA()
```

2. 遗传算法中二进制编码的染色体与汽油生产的操作变量值的转换

```
def x2chrom(self, X0):
    # real 为 X0 上各个变量初始值在其区间内的相对大小，如为上限值则为 1，下限值则为 0
    if self.int_mode:
        real = (X0 - self.lb) / (self.ub_extend - self.lb)
    else:
        real = (X0 - self.lb) / (self.ub - self.lb)

    # cumsum_len_segment 为各个变量所占的遗传信息数量累计值数组，如第一个变量占有 2 条遗传信息，第二个变量有 5 个遗传信息，则数组为[2, 7, ...]
    cumsum_len_segment = self.Lind.cumsum()
    # Chrom 为种群初始染色体信息
    Chrom = []
    # 外循环对个体循环，计算每个个体初始值对应的遗传信息，并添加至种群染色体库 Chrom
    for j in range(self.size_pop - self.size_pop // 2):
        # 内循环对变量循环，对于每个变量计算其在对应长度空间内的 gray code，并添加至个体染色体 Chrom_person
        real_random = []
        # 在 real 的基础上叠加一个随机分量，得到种群各个个体的 real_random
        for i in range(real.shape[1]):
            tmp = real[0][i] + np.random.randint(-1, 1) / 100
            tmp = max(0, min(1, tmp))
            real_random.append(tmp)

        # Chrom_person 最终是一个 1 * cumsum_len_segment[-1]的数组，代表一个个体的染色体
        Chrom_person = np.array([], dtype=int)
        for i in range(len(cumsum_len_segment)):
            if i == 0:
                len_segment = cumsum_len_segment[0]
            else:
                len_segment =cumsum_len_segment[i] - cumsum_len_segment[i-1]
            Chrom_person = np.append(Chrom_person, self.rv2gray(real_random[i], len_segment))
        Chrom.append(Chrom_person.copy())
    Chrom = np.array(Chrom)
    return Chrom

# rv2gray 将小数转换为长度为 len_segment 的 gray_code
def rv2gray(self, real, len_segment):
    chromosome = []
    base = 1 - 0.5 ** len_segment
    real *= base
    for i in range(len_segment):
```

```
            chromosome.append(int(real // 0.5 ** (i + 1)))
            real %= 0.5 ** (i + 1)
        #  二进制转换为格雷码
        res = []
        for i in range(len_segment):
            res.append(int(chromosome[i] ^ chromosome[i-1]))
        return res

    def gray2rv(self, gray_code):
        # Gray Code to real value: one piece of a whole chromosome
        # input is a 2-dimensional numpy array of 0 and 1.
        # output is a 1-dimensional numpy array which convert every row of input into a real number.
        _, len_gray_code = gray_code.shape
        b = gray_code.cumsum(axis=1) % 2
        mask = np.logspace(start=1, stop=len_gray_code, base=0.5, num=len_gray_code)
        return (b * mask).sum(axis=1) / mask.sum()

    def chrom2x(self, Chrom):
        cumsum_len_segment = self.Lind.cumsum()
        # print(cumsum_len_segment)
        X = np.zeros(shape=(self.size_pop, self.n_dim))
        for i, j in enumerate(cumsum_len_segment):
            if i == 0:
                Chrom_temp = Chrom[:, :cumsum_len_segment[0]]
            else:
                Chrom_temp = Chrom[:, cumsum_len_segment[i - 1]:cumsum_len_segment[i]]
            X[:, i] = self.gray2rv(Chrom_temp)

        if self.int_mode:
            X = self.lb + (self.ub_extend - self.lb) * X
            X = np.where(X > self.ub, self.ub, X)
            # the ub may not obey precision, which is ok.
            # for example, if precision=2, lb=0, ub=5, then x can be 5
        else:
            X = self.lb + (self.ub - self.lb) * X
        return X
```

3. 遗传算法主循环

```
    def run(self, max_iter=None):
        satisfied = False
        iter = self.max_iter                    # iter 记录迭代次数，若无 break 则必为 self.max_iter
```

```
self.max_iter = max_iter or self.max_iter
for i in range(self.max_iter):
    self.X = self.chrom2x(self.Chrom)
    self.Y = self.x2y()
    self.ranking()
    self.selection()
    self.crossover()
    self.mutation()

    # record the best ones
    generation_best_index = self.FitV.argmax()
    self.generation_best_X.append(self.X[generation_best_index, :])
    self.generation_best_Y.append(self.Y[generation_best_index])
    self.all_history_Y.append(self.Y)
    self.all_history_FitV.append(self.FitV)
    print(self.Y, 'Y')
    print(self.RON, 'RON')
    # 绘制 Y 值、辛烷值与产品硫含量的图(1 行 3 列)
    draw_all(self.Y, self.RON, self.S_product, self.size_pop)

    # 记录最好的辛烷值数据，存储在 bestRONdata，同时记录满足 30%要求的辛烷值数据
    if self.target1:
        best_RON = 0 if not self.generation_best_RON else self.generation_best_RON[-1][1]
        best_RON_data = None
        for j in range(len(self.RON)):
            if self.S_product[j] <= 5:
                if self.RON[j] > self.target1:
                    self.generation_satisfied.append([self.X[j, :], self.RON[j], self.S_product[j]])
                if self.RON[j] > best_RON:
                    best_RON = self.RON[j]
                    best_RON_data = [self.X[j, :], self.RON[j], self.S_product[j]]
        if best_RON_data:
            self.generation_best_RON.append(best_RON_data)
        if len(self.generation_satisfied) >= 5:
            iter = i + 1
            satisfied = True
            print('Satisfied', self.generation_satisfied)
            # break

    if self.generation_best_RON and self.generation_best_RON[-1][1] >= self.target2:
        iter = i + 1
        satisfied = True
        # break
```

```
            # global_best_index = np.array(self.generation_best_Y).argmin()
            # self.best_x = self.generation_best_X[global_best_index]
            # self.best_y = self.func(np.array([self.best_x]))

            #  若记录了 RON 最好值则更新 best_x, best_y 和 S_product
            if self.generation_best_RON:
                self.best_x, self.best_y, S_product = self.generation_best_RON[-1]
                if self.best_y >= self.target1:
                    self.satisfied = True
            #  否则(迭代次数达到最大值仍未找到符合的值)，返回空字符
            else:
                self.best_x, self.best_y, S_product = '', '', ''
            return self.best_x, self.best_y, S_product, satisfied, iter
```

问题五的程序

```
import numpy as np
import pandas as pd
from predict.predmain import getPredictVal
from predict_S.predmain import getPredictVal_S
from sklearn.preprocessing import MinMaxScaler
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def scale(data):
    minmax = MinMaxScaler()
    data = minmax.fit_transform(data)
    # print(input_reframed)
    np.savetxt('Smin_', minmax.min_)
    np.savetxt('Sscale_', minmax.scale_)
    # print(minmax.scale_)
    # print(minmax.min_)
    return data, minmax.scale_[-1], minmax.min_[-1]

def ope():
    Best = pd.read_excel("Best1.xlsx", sheet_name="best_X").iloc[:, 1].to_numpy()
```

```
    bound = pd.read_excel("boundary.xlsx")
    lb = bound.iloc[:, 2].to_numpy()
    ub = bound.iloc[:, 3].to_numpy()
    delta = bound.iloc[:, 4].to_numpy()
    index = bound.iloc[:, 0].to_numpy()
    # 目前是 numpy   series to numpy
    # 25 个操作变量

    sss = pd.read_excel("./predict/Q1result.xlsx")

    index = index + 3
    # 133 号变量
    start = sss.iloc[118, index]

    X_NO = sss.iloc[118, [3,4,5,6,14]].to_numpy()
    SX_NO = sss.iloc[118, [2,5,6,8,12,14]].to_numpy()

    end = Best
    gap = end - start
    steps = gap/delta

    print(steps)
    print(steps.max())
    print(int(steps.max())+1)
    stepend= int(steps.max())+1

    state = np.zeros([stepend+2, 25])
    gapeach = gap/stepend
    state[0] = start
    for i in range(stepend+1):
        state[i+1] = start + i*gapeach

    return state, stepend, X_NO, SX_NO

def reverse(X_Opeartion, X_No0peration, SX_Opeartion, SX_No0peration):
    xinwan = getPredictVal(X_Opeartion.reshape(1,-1), X_No0peration.reshape(1,-1))
    S = getPredictVal_S(SX_Opeartion.reshape(1,-1), SX_No0peration.reshape(1,-1))
    return xinwan, S


def visual(X, Y):
    Y1X = Y[..., 0]
    Y2S = Y[..., 1]
    #print(iris.data)
```

```
#print(iris.target)
#exit()
X_tsne = TSNE(n_components=2,learning_rate=0.1).fit_transform(X)
print("finishe!")

X1 = X_tsne[:, 0]
X2 = X_tsne[:, 1]
# tt = (Y1X-80)
# tt2 = tt*2
# cc = np.rint(tt2)

temp, a, b = scale(Y1X.reshape(-1,1))
temp1, a, b = scale(Y2S.reshape(-1,1))
temp = temp.squeeze()
temp1 = temp1.squeeze()

plt.scatter(X1, X2, c=np.rint(temp*256))
plt.colorbar()
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
plt.savefig("scaxinwan.svg", format="svg")
plt.savefig("scaxinwan.png", format="png")
plt.show()
plt.close()

plt.scatter(X1, X2, c=np.rint(temp1*256))
plt.colorbar()
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
plt.savefig("scaS.svg", format="svg")
plt.savefig("scaS.png", format="png")
plt.show()
plt.close()

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X1, X2, Y1X, c='g')
ax.set_zlabel('辛烷值')  # 坐标轴
# ax.set_ylabel('Y')
# ax.set_xlabel('X')
plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
plt.savefig("3Dxinwan.svg", format="svg")
plt.savefig("3Dxinwan.png", format="png")
plt.show()
plt.close()
```

```
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.scatter(X1, X2, Y2S, c='b')
    ax.set_zlabel('硫含量')  # 坐标轴
    # ax.set_ylabel('Y')
    # ax.set_xlabel('X')
    plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
    plt.savefig("3DS.svg", format="svg")
    plt.savefig("3DS.png", format="png")
    plt.show()
    plt.close()
    # X_pca = PCA().fit_transform(X)
    # plt.scatter(X_pca[:, 0], X_pca[:, 1], c=Y)
    # plt.colorbar()
    # plt.show()
    # plt.close()

    plt.plot(Y1X)
    plt.xlabel("调整次数")
    plt.ylabel("辛烷值")
    plt.legend()
    plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
    plt.savefig("XINWAN.svg", format="svg")
    plt.show()
    plt.close()

    plt.plot(Y2S)
    plt.xlabel("调整次数")
    plt.ylabel("硫含量")
    plt.legend()
    plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
    plt.savefig("SSS.svg", format="svg")
    plt.show()
    plt.close()

    # C = plt.contour(X, Y, height(X, Y), 8, colors="black", linewidth = .5)
    # adding label
    # plt.clabel(C, inline=True, fontsize=10)
    # clabel:cycle 的 label，inline=True 表示 label 在 line 内，fontsize 表示 label 的字体大小

    # plt.show()


if __name__ == '__main__':
```

```
    state, stepend, X_NO, SX_NO = ope()
    # S_index = [14, 5, 9, 11, 4, 20, 16, 2, 6, 8, 13, 23, 1]   # S 模型的操作变量在 RON 模型操作变量中的位置
    #
    # result = np.zeros([stepend+2, 2])
    # for i in range(stepend+2):
    #         result[i, 0], result[i, 1] = reverse(state[i], X_NO, state[i][S_index], SX_NO)
    #
    # np.save("filename.npy", result)
    result = np.load("filename.npy")

    result[..., 1] = (1-result[..., 1]/100)*248 #  还原至硫含量
    visual(state, result)

    print(result)
```