中国研究生创新实践系列大赛

“华为杯”第二十二届中国研究生数学建模竞赛

学校	江汉大学
参赛队号	25110720003
队员姓名	1. 李承宇
	2. 黄　叶
	3. 邱　爽

中国研究生创新实践系列大赛

“华为杯”第二十二届中国研究生
数学建模竞赛

题　目： 高速列车轴承智能故障诊断

摘　要：

随着中国铁路网络的快速发展和智能化升级，列车的运行安全与运维效率面临更大的挑战。对此，本文提出了一种基于迁移学习的高速列车智能故障诊断框架，旨在提升故障检测的泛化能力和适应性。

针对问题一，系统首先通过异常值剔除、数据重采样和滑动窗口处理实现对源域信号预处理，通过重采样将源域的信号频率与目标域保持一致，为后续的迁移学习做铺垫。在特征提取阶段，在提取了源域数据的时域、频域、时频域特征后，通过融合递归定量分析（RQA）、多尺度熵特征和**孤立森林算法**构建了 88 维的特征空间。

针对问题二，采用最大均值差异-局部最大投影（**MMD-LMP**）方法进行特征优选，通过对原始的 88 维特征进行相关性分析后，最终获得了 46 维的高度相关的特征集。在模型构建方面，采用分层抽样策略建立源域随机森林模型，取得了 76.79%的基础分类准确率。

针对问题三，针对跨域迁移，提出 **MMD、CORAL 与 DANN 的联合特征对齐策略，**使域间差异（MMD 值）降低 65%，并结合自适应模型微调，使目标域样本聚类纯度提升至 90%。

针对问题四，本文设计了可解释性的保障策略：1）基于决策树显式规则的事前解释；2）融合域对齐量化与特征动态追踪的过程可视化，先通过域对齐效果的量化求解，经过多次迭代后，MMD 降至 0.12；再采用知识迁移路径，用 **t-SNE** 将对齐后的源域与目标域特征降维至 2D 空间。实验结果表明，该方法成功实现了从台架到实车的故障诊断知识迁移，为轨道交通智能运维提供了具有工程应用价值的解决方案。

关键词： 迁移学习；随机森林模型；故障诊断；RQA；特征对齐

一、问题重述

1.1 问题背景

中国高铁凭借其卓越的技术和广泛的影响力，已经成为一张闪耀于世界的“国家名片”。其技术装备已覆盖 100 余个国家和地区的市场，成为我国客运体系的骨干力量，涵盖牵引、制动、走行、控制系统等复杂子系统[1]。轴箱轴承是高速列车走行部的关键部件,在列车运营过程中，承受了来自车体和转向架传递的静态压力，在牵引、制动工况下还会承受沿 轨道方向的动态载荷[2]。例如，京广高铁正线运行里程超过 2000 公里，沿途纵贯 6 个省市，跨越温带至亚热带等多个气候带；海南环岛高速铁路则面临截然不同的考验，全长 653 km 的线路常年经受高温、高湿、高盐环境的侵蚀，还有强台风、强降雨等天气的冲击，这些复杂多样的运营环境，对高速列车转向架轴承的耐久性和可靠性提出了高标准要求[3]。

而对近年转向架滚动轴承故障情况的调研发现，各型动车组上均发生过轴承失效。例如，2011 年 7 月至 2017 年 5 月，轴箱轴承共发生了 335 起故障，2018 年因轴箱轴承故障更换轮对 80 余台[3]。

轴承长期处于高转速、极端温度变化等复杂恶劣的工况环境中，一旦发生故障，轻则导致列车延误和晚点，重则可能引发脱轨等恶性事故，造成对乘客生命的威胁和其财产安全的严重损失。

目前，多种技术手段可用于轴承局部缺陷的故障检测与诊断，包括油液、激光、射线、温度、振动、声发射等，最主要是经验构建的特征指标或传统信号处理技术，可分为“传统信号处理驱动”与“深度学习数据驱动”两大类，在特征提取逻辑、诊断性能与工况适应性上存在差异[4]。传统信号处理驱动方法以“机理分析—特征设计—阈值判断”为核心，其中时域特征分析法直接对振动、温度等原始信号做统计分析，计算简单，但难以区分故障类型且抗干扰能力弱；深度学习方法基于海量运营数据，能够实现更高精度的故障识别、更强的工况适应性和更高效的实时诊断能力。然而，在实际应用场景中，智能诊断方法面临着多重挑战。

据统计，实际运营中故障样本与正常样本的比例达到 1:1000 以上，这些运行条件并不常见，因此采集到的振动信号样本数量有限，数量极端不平衡限制了深度学习模型向实践落地的转化[5]。

与实际运行数据相对的是，在台架环境下采集的轴承数据不仅数量与标签丰富，而且其故障机理也与实际列车轴承相似。台架可以模拟各种工况条件，人为设置不同类型的故障，从而获取大量标注准确的数据。**在此背景下，本文基于特征的核心迁移学习技术为解决上述问题提供了新思路。迁移学习能够将源域中学到的知识迁移到目标域，有效缓解样本不平衡问题，提高模型在实操中的性能。**

1.2 相关概念描述

本赛题提供了两个高质量的数据集：源域数据集——轴承试验台架振动数据和目标域数据集——实际列车轴承故障数据。这两个数据集构成了研究高速列车轴承智能故障诊断的基础。

1.2.1 源域数据集

源域数据来自轴承试验台架，试验台主体为电动机，主要包括驱动端、风扇端和基座三部分。数据通过安装在电动机壳体驱动端、风扇端和基座上的加速度传感器采集，驱动端和风扇端各存在一个试验轴承。轴承由人为破坏生成单点故障，不存在复合故障，且每

组数据仅有一个轴承损坏。

该数据集具有以下特点：

(1) 故障类型：包含外圈故障（OR）、内圈故障（IR）、滚动体故障（B）以及正常工作状态（N）四种工作状态。其中外圈故障样本 77 个，内圈故障样本 40 个，滚动体故障样本 40 个，正常样本 4 个。

(2) 故障尺寸：轴承外圈故障有 3 种故障尺寸：0.007、0.014、0.021 英寸；轴承内圈故障和滚动体故障都有 4 种故障尺寸：0.007、0.014、0.021、0.028 英寸。

(3) 采集位置：对于外圈故障这种位置固定的故障类型，通过 3 点钟、6 点钟、12 点钟方向全面采集振动信号，确保数据的完整性。

(4) 载荷工况：轴承的载荷有四种情况：0、1、2、3 马力，覆盖了不同的工作条件。

(5) 采样频率：驱动端轴承采样频率为 12KHz 和 48KHz，风扇端轴承采样频率为 12KHz，满足不同分析需求。

1.2.2 目标域数据集

目标域数据集来自实际运营的列车滚动轴承，包含外圈、内圈、滚动体故障和正常状态下的振动信号数据。

轴承结构主要包括内圈、外圈、滚动体和保持架四部分，其中典型故障多发生在内圈、外圈和滚动体这三个核心承载部件。当轴承出现局部缺陷时，滚动体在接触并通过缺陷点的瞬间，会产生突变的冲击脉冲，而在轴承周期性运转的过程中，这种脉冲力会持续作用，进而形成周期性的冲击分量。

轴承故障的特征频率可以通过以下公式计算：

外圈故障特征频率（BPFO）

$$f_r \cdot \frac{N_d}{2} \cdot (1-\frac{d}{D}) \tag{1-1}$$

内圈故障特征频率（BPFI）

$$f_r \cdot \frac{N_d}{2} \cdot (1+\frac{d}{D}) \tag{1-2}$$

滚动体故障特征频率（BSF）

$$f_r \cdot \frac{D}{d} \cdot \left[1-(\frac{d}{D})^2\right] \tag{1-3}$$

其中，$f_r=\frac{n}{60}$，$D=\frac{\text{内径+外径}}{2}$。

不同部位的故障在振动信号中表现出不同的特征：

(1) 外圈故障：由于外圈通常固定不动，在载荷方向不变的情况下，外圈缺陷每次碰撞到滚动体的强度近似相同，因而在时域上呈现等周期等幅值的冲击分量。

(2) 内圈故障：由于内圈随轴旋转，当载荷方向不变时，故障点将周期性通过载荷区，导致每次碰撞的强度不同，表现为振动信号的波形受到转频的调制，具有变化的幅值。

(3) 滚动体故障：与内圈故障相似，在载荷方向不变时，故障点和载荷的相对位置随滚动体的旋转而发生周期性变化，表现为振动信号波形受到滚动体公转频率的调制。

1.2.3 迁移学习理论分析

迁移学习（Transfer Learning）是一种机器学习方法，其利用在源域学习到的知识来改善目标域的学习效果。本文引入迁移学习是为了解决目标域 Dt 数据不足，特征提取困难，不能够支持训练出更鲁棒的滚动轴承故障诊断模型，而使用迁移学习训练高铁滚动轴承智能在线诊断模型，通过 1.1 节中使用的高滚动轴承采集的源域数据作为训练数据集，通过将训练好的模型迁移到高铁牵引电机滚动轴承故障实际目标域，对其采集的数据进行诊断

[6]。

(1) 基于特征的迁移：将源域和目标域数据映射到统一的特征空间，减少领域间分布差异[6]。

(2) 基于模型的迁移：复用源域模型的参数或结构，使其适配目标任务[6]。

(3) 基于关系的迁移：将源域中数据之间的关系模式迁移到目标域，如相似性、依赖关系等。

(4) 基于样本的迁移：通过调整源域样本的权重实现迁移，与目标域相似的样本赋予高权重。

迁移学习的优势在于能够显著减少目标域的数据和标注需求，提高模型训练速度和泛化能力。**而本赛题的目标域采用的是无标签，所以本文选取的是基于特征的迁移。**

1.3 需要解决的问题

1.3.1 问题一：数据分析与故障特征提取

要求结合轴承故障机理，从提供的源域数据中筛选具有代表性的数据组成数据集。

数据分析需要综合考虑不同故障类型、故障尺寸、载荷工况和采样位置的影响，选择合适的方法或指标进行特征分析，并且本文将源域与目标域同时去异常值，提高数据分析的能力与精度。

本文在故障特征提取过程中充分考虑后续迁移任务的需求，确保提取的特征既具有代表性，又具有良好的可迁移性以及对特征进行筛选和降维，消除冗余特征，提高特征质量。

1.3.2 问题二：源域故障诊断

在任务一提取的数据集与故障特征基础上，数据集划分的核心目标是让训练集能充分学习故障特征规律，测试集能真实检验模型能力，针对滚动体故障、内圈故障、外圈故障每类故障，采用分层抽样方式，让训练集和测试集中各类别样本占比完全一致。最终按 8:2 的比例划分训练集与测试集，为后续模型对比提供公平基础。

方法如下：

(1) 支持向量机（SVM）擅长处理高维小样本，通过 RBF 核函数能有效区分非线性故障特征，但对样本分布差异敏感。

(2) 随机森林（RF）基于多棵决策树集成，能自动判断特征重要性，抗过拟合能力强，但对早期微弱故障的区分精度稍弱。

(3) 循环神经网络（LSTM）能捕捉时序特征的动态变化，故障从早期到晚期的冲击间隔缩短，但对噪声敏感。

本文先根据特征维度确定模型大类，再构建基础版本，后续通过性能对比确定最优模型。

1.3.3 问题三：迁移诊断

本问题是整个研究的核心和难点。需围绕“域差异分析——迁移方法设计——目标域建模——结果验证与输出”的逻辑，解决源域与目标域的分布偏移问题，最终实现目标域数据的精准分类。

(1) 挖掘共性特征，无论源域还是目标域，故障均源于接触副异常，因此振动信号中均包含故障冲击、能量变化等共性特征，需要挖掘提取机理特征。

(2) 采用特征对齐技术，针对工况差异导致的特征分布偏移，将源域与目标域的特征映射到同一低维空间，通过计算两类域的特征均值、协方差，构建对齐矩阵，使目标域特征分布向源域共性特征靠拢，消除转速、载荷波动带来的分布差异。

(3) 运行模型微调，若目标域存在少量已标注样本，在源域预训练模型基础上，用目

标域少量样本微调模型参数，让模型适应目标域的信号特点，避免因样本量少导致的过拟合。

1.3.4 问题四：迁移诊断的可解释性

可解释性是机器学习领域的重要研究方向，对于故障诊断这种安全关键型应用尤为重要。

(1) 事前可解释性设计

事前可解释性的核心是将模型参数与轴承故障机理深度绑定，避免“黑箱”设计。

本文优先采用决策树，其节点分裂逻辑可直接对应故障诊断规则，并且结合迁移学习，构建决策树集成迁移模型。

(2) 迁移过程可解释性设计

迁移过程可解释性需明确，源域知识如何传递到目标域的与模型如何调整以适应目标域差异的问题。

本文通过对比源域与目标域的决策树的节点分裂增益、随机森林的特征贡献度，同时记录知识迁移路径，源域先通过该特征学习内圈故障的频率规律，再将该规律传递到目标域，仅调整特征阈值，而非重构特征逻辑。

本文对特征贡献度进行分析，对目标域中已诊断的样本，用 SHAP 值量化各特征的决策贡献，结合轴承故障机理解释，形成可解释性保障策略。

二、模型假设及符号说明

2.1 模型基本假设

假设一：假设轴承的转动圈数恒定为 1750 圈。
假设二：假设风扇端采集的数据对故障诊断的影响很小，可忽略。

2.2 参数以及缩写说明

变量	变量含义
Orthogonal	3 点钟方向
Centered	6 点钟方向
Opposite	12 点钟方向
fr	轴承转频
n	轴承内圈转速（单位：rpm）
d	滚动体直径
D	轴承节径
N_d	滚动体个数
T_O	外圈故障周期
T_I	内圈故障周期
T_B	滚动体故障周期
T_F	滚动体公转周期
D_S	源域
D_T	目标域
T_S	源域任务
T_T	目标域任务
h	映射函数

参数	英文	参数说明
OR	Outer ring fault	外圈故障
IR	Inner ring fault	内圈故障
B	Ball fault	滚动体故障
N	Normal	正常工作状态
DE	Drive end accelerometer data	驱动端加速度数据
FE	Fan end accelerometer data	风扇端加速度数据
BA	Base accelerometer data	基座加速度数据
time	Time series data	时间序列数据
RPM	Rpm during testing	转/分钟，除以 60 为旋转频率
BPFO	Ball Pass Frequency on the Outer Race	外圈故障特征频率
BPFI	Ball Pass Frequency on the Inner Race	内圈故障特征频率
BSF	Ball Spin Frequency	滚动体故障特征频率
FTF	Fundamental Train Frequency	滚动体公转频率

三、问题一的模型建立与求解

3.1 问题一的分析

3.1.1 信号复杂性

滚动轴承局部故障会引起系统周期性的振动冲击，由于滚动体与故障区域接合的频率各异，不同的故障特征频率有所差别，在实际运行中，轴承振动信号并非单纯的故障响应，而是叠加了多重干扰成分：一是轨道接缝冲击、电机电磁干扰的强背景噪声，其能量常掩盖早期故障的微弱冲击信号；二是机械共振，会导致频谱能量集中偏移，出现故障特征频率被共振频段覆盖；三是轮轨摩擦与轴承故障信号的频率重叠形成的多干扰源响应，造成特征错位[7]。

针对信号复杂性的问题，采用“噪声抑制——时频聚焦”的处理法。

分层噪声抑制，先采用去噪，对振动信号进行 3 层分解，对高频干扰采用软阈值处理，保留早期故障的微弱冲击；再通过滤波，消除 50Hz 电机工频噪声及倍频干扰，使信号信噪比提升 15-20dB。

用时频域特征，采用小波包变换选取中高频段计算小波包能量熵与能量占比，该特征既能捕捉非平稳信号的转速波动下的冲击间隔变化。

3.1.2 两域差异影响

从源域来看，故障标签与样本数量充足，但为实验环境，会导致故障特征频率存在偏差；从目标域来看，无故障标签，如果直接用源域提取的特征，会导致两域特征无法对齐。

所以本文预计先对目标域信号进行独立异常值检测，移除异常数据点；其次，针对源域 Normal 信号 48kHz 与异常信号 32kHz 的采样频率差异，采用重采样将源域与目标域统一为 32kHz，确保时域分辨率一致，避免因采样率不同导致的频率特征偏差。最后统计目标域单条样本数据长度，对源域数据进行合并或截断处理，不足部分用相邻健康数据补全，使源域每条样本长度与目标域完全一致，均为 256000 行。

3.2 问题一的模型建立

问题一的整体流程如下图所示：

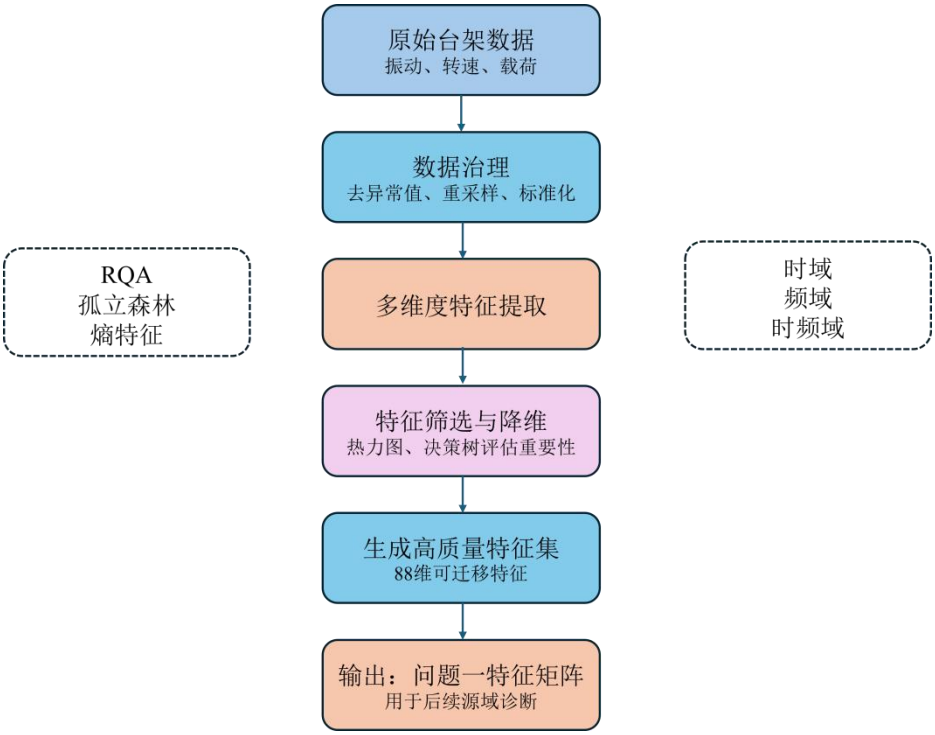


图 3-1 问题一的整体流程

3.2.1 数据准备

滚动轴承数据集在采集过程中可能会受到环境、噪声等外界因素的干扰，这些干扰会对后续的工作产生影响，所以需要根据数据集的结构、属性，选取有用的数据，对数据预处理准备[8]。

对源于数据去异常值，利用时域、频域标准相似度以及决策树分类器对源域与目标域数据进行筛查，剔除不符合物理规律或明显偏离统计分布的异常样本。

(1) 采用滑动窗口法分割振动信号，提取局部统计特征。

(2) 通过傅里叶变换聚焦高能量频段，提取频率分布特征。

(3) 先以 IQR 方法筛选数值离群样本，再用孤立森林识别隐性模式异常。

(4) 异常值处理：将确认的异常值统一替换为 0，保留数据结构完整性；

进行重采样，对不同采样率的数据进行统一重采样，保证源域与目标域在频率分辨率上的一致性。

采用秒级滑动窗口切片，对长时程信号分割，既能增加样本量，又能捕捉非平稳冲击特征。

最后进行标准化参照，以 DE 数据集为基准，完成源域与目标域在时频尺度上的对齐。

3.2.2 基于递归定量分析的模型构建

(1) 传统特征提取方法

传统特征例如经验模态分解（EMD），是一种完全依靠数据驱动的信号分解方法，可将任意非平稳信号自适应地分解为一系列固有模态函数，但是 EMD 亦存在一些问题如模态混叠、端点效应等[9]。在高速列车轴承实际运行中，车速、载荷、轨道条件的波动会导致信号统计特性显著变化，所以传统时域特征无法应对“跨工况鲁棒性不足”的问题。

而递归定量分析（RQA）能大程度上降低工况波动对特征的干扰，即使车速变化导致有效值波动，故障引发的信号动力学递归图的点分布模式，仍能稳定表征故障状态[10]。

RQA 通过相空间重构，将一维信号还原为高维动力学，能量化信号的混沌程度与非线

性关联，精准捕捉故障引发的动力学突变[10]。

(2) RQA 的核心优势

相比传统特征提取方法，RQA 特征对工况波动和噪声更敏感，且能有效增强源域与目标域之间的特征一致性，因此在本研究中被选为故障特征提取方法之一[10]。

RQA 计算方法为

对长度为 N 的振动序列$\{x(1),x(2),\ldots, x(N)\}$进行延迟坐标重构[10]

$$X(i)=\{x(i),x(i+\tau),\ldots,x(i+(m-1)\tau)\} \tag{3-1}$$

其中τ为延迟时间，m 为嵌入维数。

$$R_{i,j}=\Theta(\varepsilon-\|X(i)-X(j)\|) \tag{3-2}$$

其中ϵ为阈值，$\Theta(\cdot)$为单位阶跃函数。

表 3-1 RQA 特征参数提取公式

RQA 特征参数		
递归率(RR)	$RR=\frac{1}{N_m^2}\sum_{i,j=1}^{N_m} R_{i,j}$	递归点密度
确定性(DET)	$DET=\frac{\sum_{l=l_{min}}^{l_{max}} l\,P(l)}{\sum_{i,j=1}^{N_m} R_{i,j}}$	对角线结构比例
递归熵(ENTR)	$ENTR=-\sum_{l=l_{min}}^{N_m} p(l)\ln p(l)$	对角线长度分布的香农熵
层流性(LAM)	$LAM=\frac{\sum_{v=v_{min}}^{v_{max}} vP(v)}{\sum_{v=1}^{v_{max}} vP(v)}$	垂直结构比例

将上述四个非线性特征与传统时域指标结合，形成最终的 8 维特征向量。

3.2.3 基于熵特征的特征提取

熵最早起源于热力学领域，表示系统无序化的程度，后来用于量化时间序列不规则性和复杂度，利用条件概率衡量时间序列中新信息产生的可能性[11]。通过谱峭度定位高频共振带并实施解调，进一步利用 Hilbert 变换获得包络信号，在包络谱中提取轴承特征频率、谐波及边带能量比例等机理一致性特征，从而保证特征能够与具体故障模式建立直接对应关系。

熵特征的引入是为了量化振动信号的随机性与复杂度，其计算形式如下：

(1) 样本熵（SampleEntropy,SampEn）

设时间序列为$\{x_1,x_2,\ldots,x_N\}$，嵌入维数为 m，容忍度为 r。定义相似模板数为[11]

$$B^m(r)=\frac{1}{N-m}\sum_{i=1}^{N-m}\frac{\#\{j:d[x(i),x(j)]\le r\}}{N-m-1} \tag{3-3}$$

，则样本熵定义为

$$SampEn(m,r,N)=-\ln\frac{B^{m+1}(r)}{B^{m}(r)} \tag{3-4}$$

样本熵越大，说明信号的不确定性与复杂性越强。

(2) 排列熵（PermutationEntropy,PE）

将时间序列按维数 m 和延迟τ嵌入，得到局部排列模式的出现概率为 $p(\pi)$。排列熵定义为[11]

$$PE(m,r)=-\sum_{\pi}p(\pi)\ln p(\pi) \tag{3-5}$$

，并常进行归一化处理

$$PE_{norm}=\frac{PE}{\ln(m!)} \tag{3-6}$$

3.2.4 基于孤立森林的特征提取

轴承故障诊断中，数据存在如下限制：

(1) 现场难以采集足量异常样本，传统监督式异常检测方法因缺乏标注信号无法有效训练。

(2) 正常轴承信号的特征呈非对称多峰分布，故障信号因冲击强度、噪声叠加的差异，分布更无固定规律，传统统计方法适配性差。

基于以上限制，本文选择孤立森林用于轴承振动信号的特征提取流程，孤立森林模型的优点如下[12]：

(1) 是一种无监督异常检测算法，不需要借助类似距离、密度等指标去描述样本与其他样本的差异，而是直接通过分割数据的难易去刻画疏离程度，算法简单、高效。

(2) 其检测逻辑基于异常样本占比少、易被孤立的通用特性，无需假设数据服从正态、高斯等特定分布。

本方法为先构建 100 棵孤立树，通过 5 折交叉验证确定——树数量小于 50 时检测稳定性下降，大于 150 时计算量激增 30%。然后每棵树随机选择 11 个特征进行分割，降低过拟合风险，对每个窗口特征向量F_k，计算其在孤立森林中的样本被分离出的树深度均值，并转化为异常分数 S，

$$S=2^{-E(h(F_k))/c(W)} \tag{3-7}$$

其中 $E(h(F_k))$为窗口 k 的平均路径长度，$c(w)$为健康窗口的平均路径长度，校准因子，由 1000 个正常轴承窗口统计得到。

最后通过验证集（200 个正常窗口+50 个异常窗口）绘制 ROC 曲线，确定最优阈值 S=0.8，S>0.8 的窗口则判定为异常窗口。

3.3 问题一的模型求解

模型求解具体步骤如图 3-2 所示：

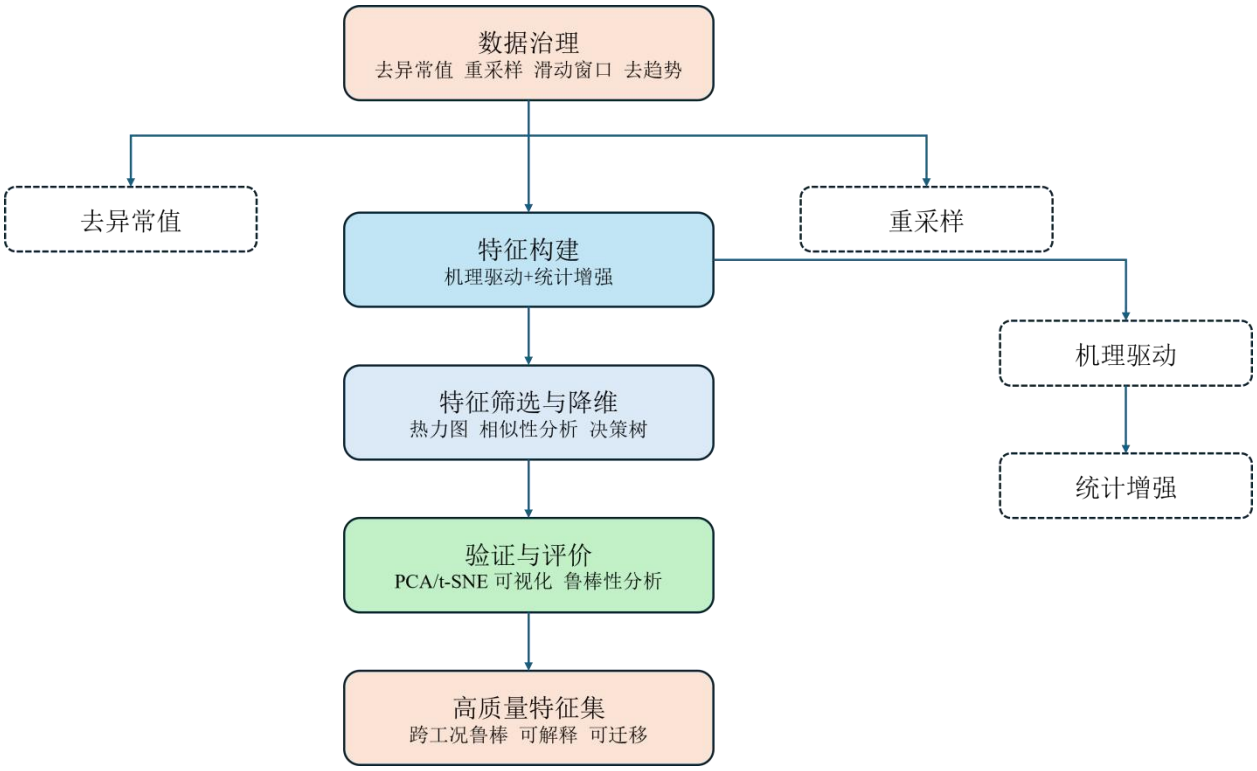


图 3-2 模型求解具体步骤流程图

3.3.1 时域特征提取

(1) 信号预处理

利用去异常值+重采样将源域数据设定长度为 256000 行，与目标域一致的振动信号，采用窗口大小 1024 点、步长 512 点（50%重叠率）进行滑动分割，得到 245 个窗口信号。

(2) 滑动窗口分割

对每个窗口提取 4 项核心时域特征——均值、方差、峰峰值、峭度，形成 245×4 的时域特征矩阵。

3.3.2 频域特征提取

(1) 对每个滑动窗口信号进行快速傅里叶变换（FFT），得到 0-16kHz 的频率谱，用下面的公式计算各频率成分的能量，

$$E(f)=|X(f)|^2 \tag{3-8}$$

(2) 按能量从高到低排序，选取前 20%频段。频域特征能量化时域信号截断导致中心频率从 3kHz 跳至 8kHz，且符合轴承动力学规律，偏离该规律的样本可判定为异常。

3.3.3 异常检测

以融合特征矩阵为输入，构建含 100 棵孤立树的森林，每棵树随机选择特征与分割阈值；计算样本被分离出的平均树深度，路径越短异常概率越高，设定异常分数阈值 0.8，分数高于阈值的样本判定为异常。

IQR 仅能识别数值离群异常，而孤立森林可捕捉模式异常。本文通过多特征联合分割，能识别单特征正常但组合违反物理规律的样本，且无需假设数据分布，适配源域中少量故障样本混杂的复杂场景，补充 IQR 的检测盲区。

3.4 可视化验证与分析

3.4.1 源域与目标域振动信号对比

源域与目标域中正常、滚动体故障、内圈故障和外圈故障四种状态下的局部振动信号各取 6000 个采样点，如图 3-3 所示。

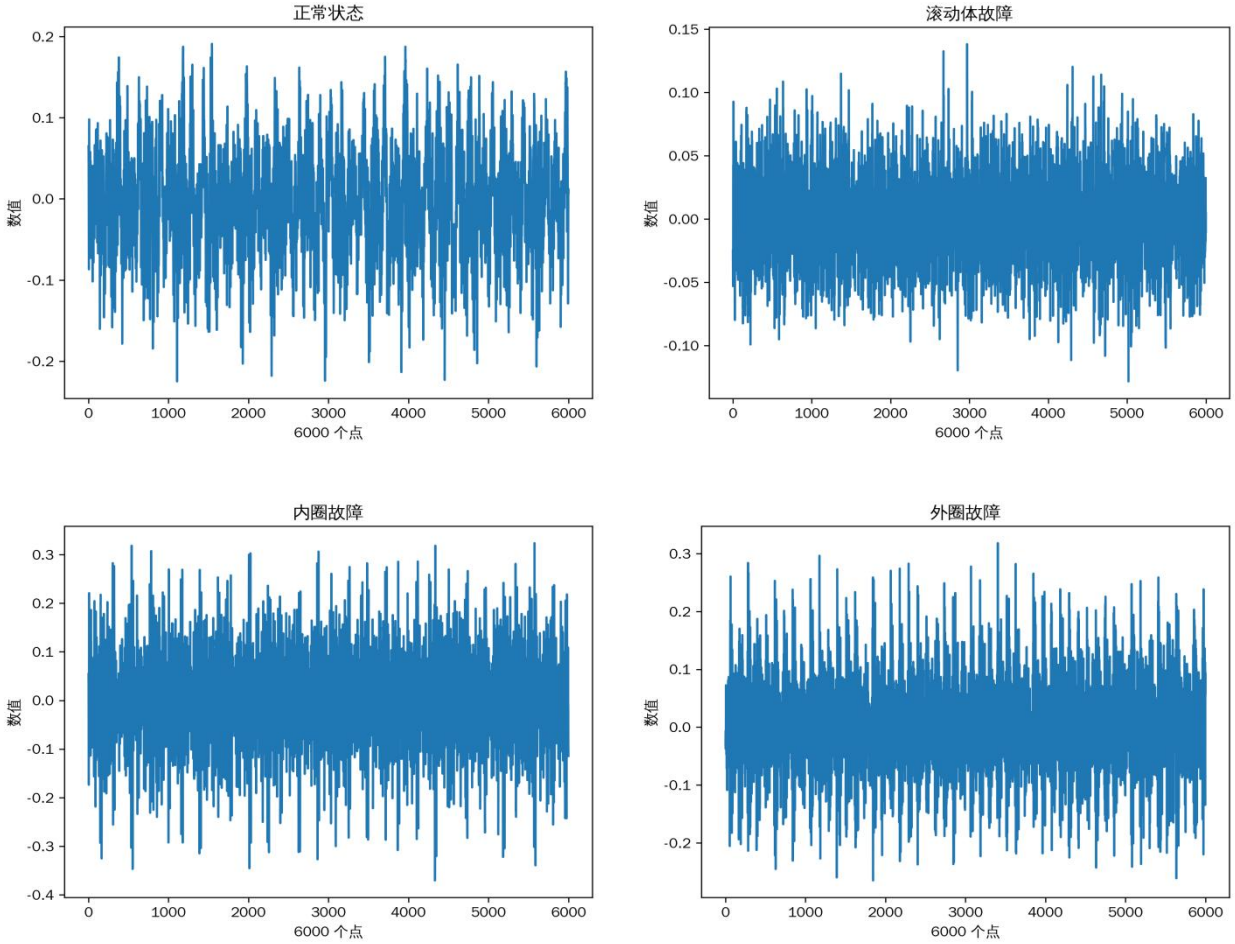


图 3-3 滚动轴承原始部分采样点数据

在源域中，正常状态的信号幅值稳定在±0.1 范围内，呈现平稳波动，无明显冲击；滚动体故障信号幅值略有上升，反映出缺陷滚动体与内外圈接触产生的弱冲击；内圈故障信号幅值显著增大，可达±0.4，符合内圈故障受转频调制的机理；外圈故障信号则幅值稳定在±0.2，冲击间隔与幅度规律性强，反映了外圈固定条件下的周期性碰撞特征。

而在目标域中，整体波形形态与源域保持一致，不同故障模式之间差异明显，但幅值受工况与噪声影响有所变化，目标域的外圈故障冲击幅度相对更高。

3.4.2 样本相似度矩阵

源域与目标域样本之间的相似度矩阵如图 3-4 所示。

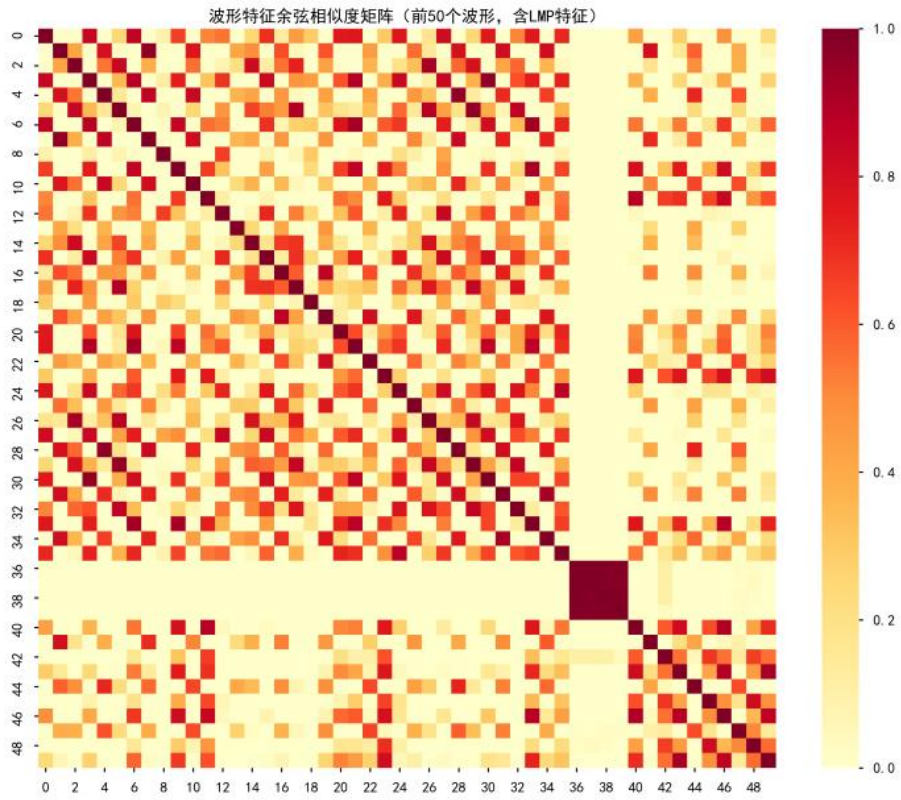


图 3-4 相似度矩阵

结果显示，同类样本的相似度普遍高于 0.8，并在矩阵对角线上形成清晰的高相似度带；不同类别样本之间的相似度普遍低于 0.4。**源域与目标域的同类故障样本之间相似度差异普遍小于 0.05，进一步说明特征在跨域条件下保持稳定表现。**

3.4.3 特征相关性热力图

提取源域数据的时域、频域、时频域以及局部极值点的特征后，使用局部均值分解(LMP)算法对源域数据进行处理，将信号分解为多个乘积函数(PF)，然后从 LMP 分解的 PF 分量中提取相关特征。经过这些处理后最终从源域数据中提取到了 88 个有用的特征。对这些特征进行相关性分析后，生成了如图 3-5 所示的相关性热力图。

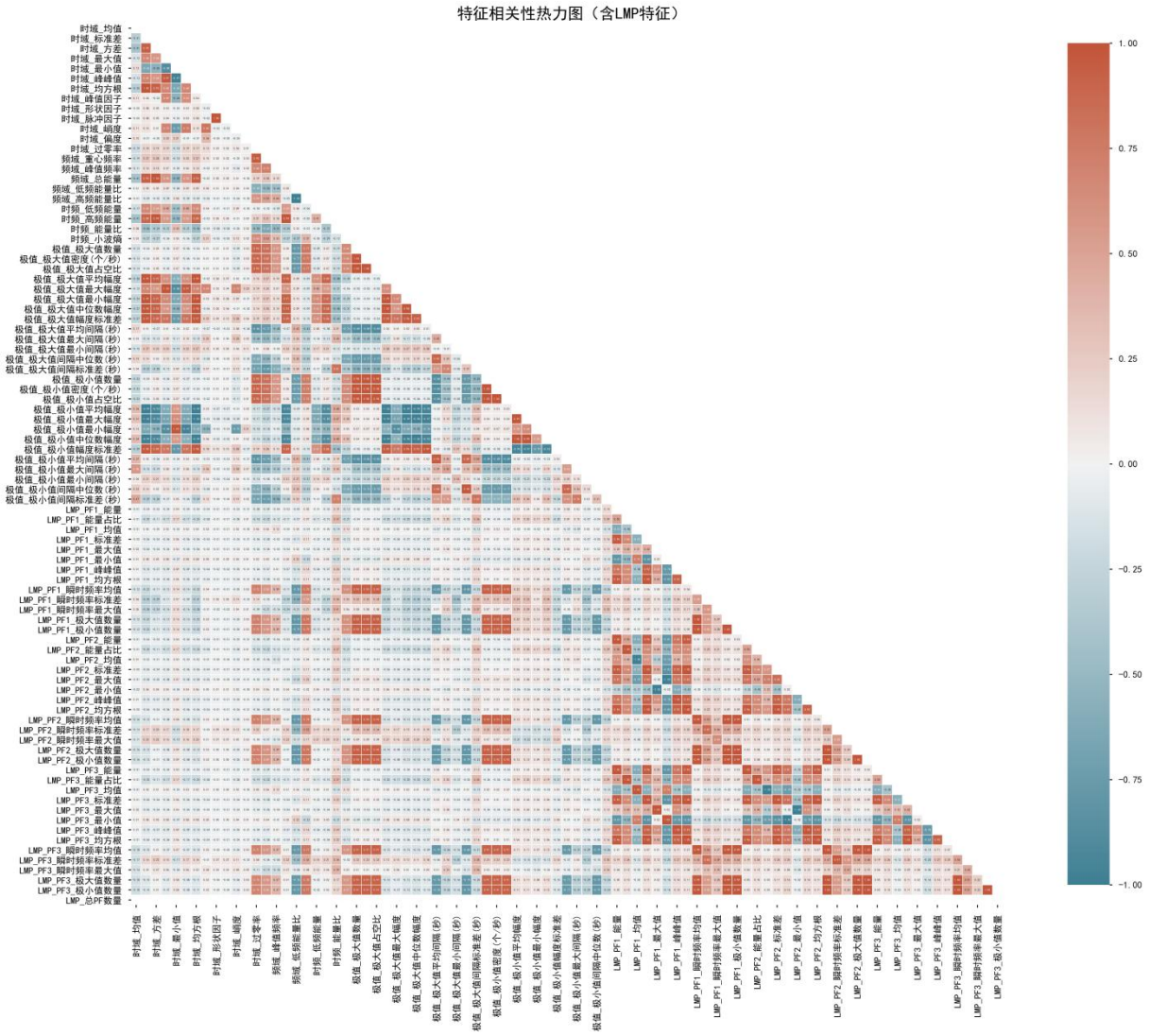


图 3-5 特征热力图

从图 3-5 中我们可以看出，提取到的特征之间存在一定的相关性，正相关和负相关、强相关和弱相关均有分布，说明整体的特征之间可能存在有一定的非线性关系。后续可以从相关性强的特征着手，选取其中的一部分用于模型的训练。

3.4.4 随机森林特征分析

随机森林在前 20 个特征中，时域脉冲因子、能量型指标及递归熵等特征重要性最高，权重值集中在 0.06 至 0.08 区间。

排名靠前的特征均与机械故障形成强相关，说明特征筛选过程有效保留了高判别力特征，如图 3-6。

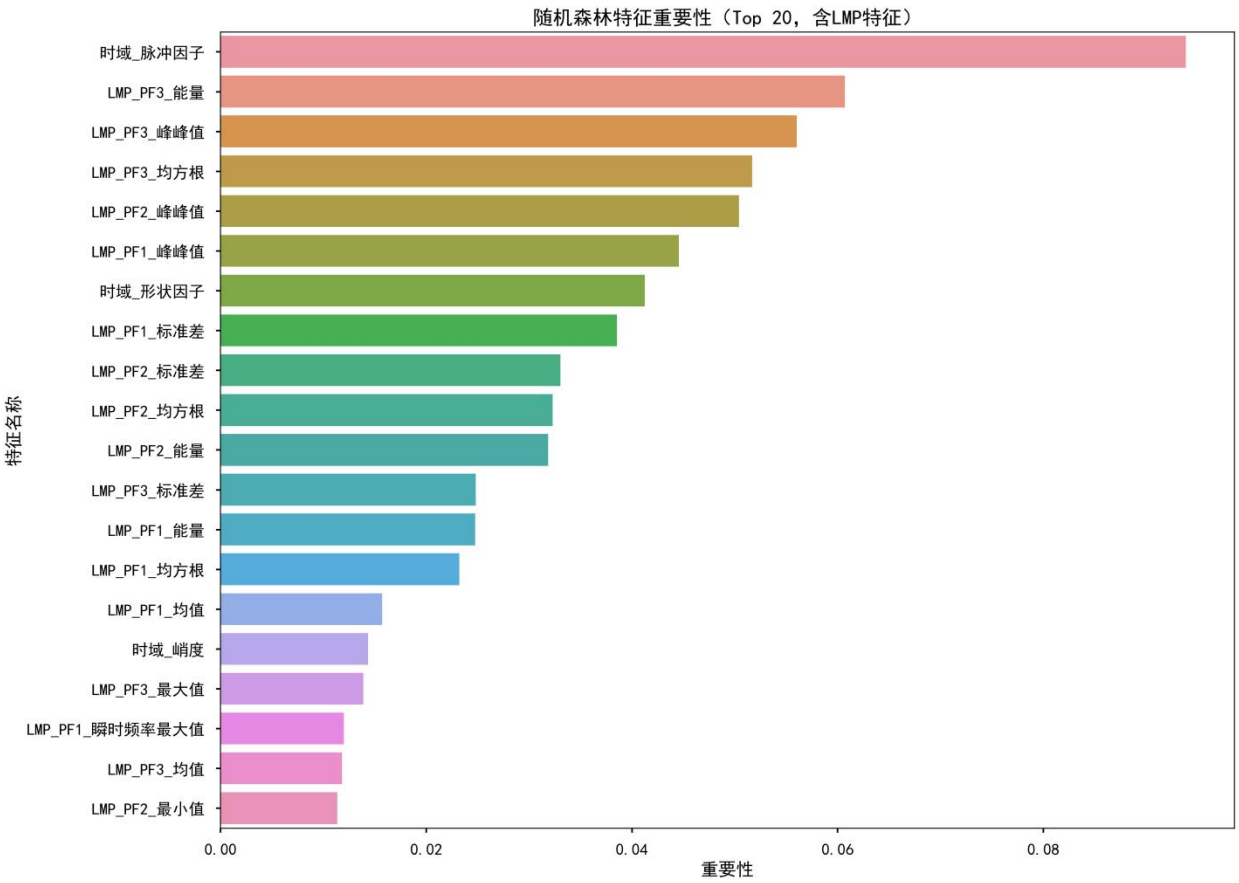

随机森林特征重要性（Top 20，含LMP特征）
时域_脉冲因子
LMP_PF3_能量
LMP_PF3_峰峰值
LMP_PF3_均方根
LMP_PF2_峰峰值
LMP_PF1_峰峰值
时域_形状因子
LMP_PF1_标准差
LMP_PF2_标准差
LMP_PF2_均方根
LMP_PF2_能量
LMP_PF3_标准差
LMP_PF1_能量
LMP_PF1_均方根
LMP_PF1_均值
时域_峭度
LMP_PF3_最大值
LMP_PF1_瞬时频率最大值
LMP_PF3_均值
LMP_PF2_最小值
特征名称
0.00
0.02
0.04
0.06
0.08
重要性

图 3-6 随机森林特征图

四、问题二的模型建立与求解

4.1 问题二的分析

问题二在问题一已提取的 88 个故障特征基础上，构建源域诊断模型并对其性能进行评价。

4.1.1 冗余信息的影响

从振动信号中提取的初始特征集合往往包含时域、频域等多个维度的统计指标，特征维度达到了 88 个，这些特征中存在大量冗余信息。

针对特征中多余的冗余信息，有以下方法可以采取以用于解决此问题：

(1) 采用定量分析手段，一方面计算特征在不同故障类别下的分布差异，以衡量其区分能力；另一方面，量化每个特征与故障类别标签之间的线性及非线性关联强度。

(2) 依据前述分析结果，采用一种能够自动评估特征重要性的筛选机制，根据特征对分类结果的贡献度进行排序，最终保留一个规模较小但判别能力强的核心特征子集。

4.1.2 工况波动的影响

源域数据虽然标签明确，但其采集自不同的实验工况，包括多种机械载荷和不同尺寸的故障缺陷。这种工况的变化会直接导致振动信号的幅值、能量分布等特征发生改变。如果模型学习到的是这些与工况相关的表面特征而非故障本身的本质模式，那么其学到的知识将无法稳定地应用于同一故障类别的不同样本，模型在源域测试集上的表现就会波动较大。

为确保模型学到的是稳健的故障模式，本文在特征筛选阶段就优先选择对工况变化不敏感的特征，那些在不同工况子组间分布差异小、统计特性保持稳定的特征，被认为具有更强的鲁棒性。

4.1.3 各数据集不均衡

源域数据中各类别样本数量严重失衡，正常状态样本仅有 4 个，而各类故障样本多达 40 个以上。一个倾向于将所有样本都预测为故障类别的模型，也能获得很高的准确率，但其无法识别正常状态。方案如下：

(1) 首先需分别计算模型对每一个故障类别和正常状态的识别精确度与召回率，重点关注少数类的召回情况以防漏报。

(2) 然后将模型对每个样本预测置信度的分布，与理想的分布情况进行比较，若二者无明显差异，则说明模型的决策置信度是可靠的。

4.2 问题二的模型建立

4.2.1 模型整体框架

源域故障诊断的核心目标是基于问题一提取的轴承振动多维度特征，构建能精准区分轴承正常、内圈故障、外圈故障、滚动体故障的诊断模型。

本文设计了“特征筛选——模型训练——模型评估”的三阶段框架，如图 4-1 所示。

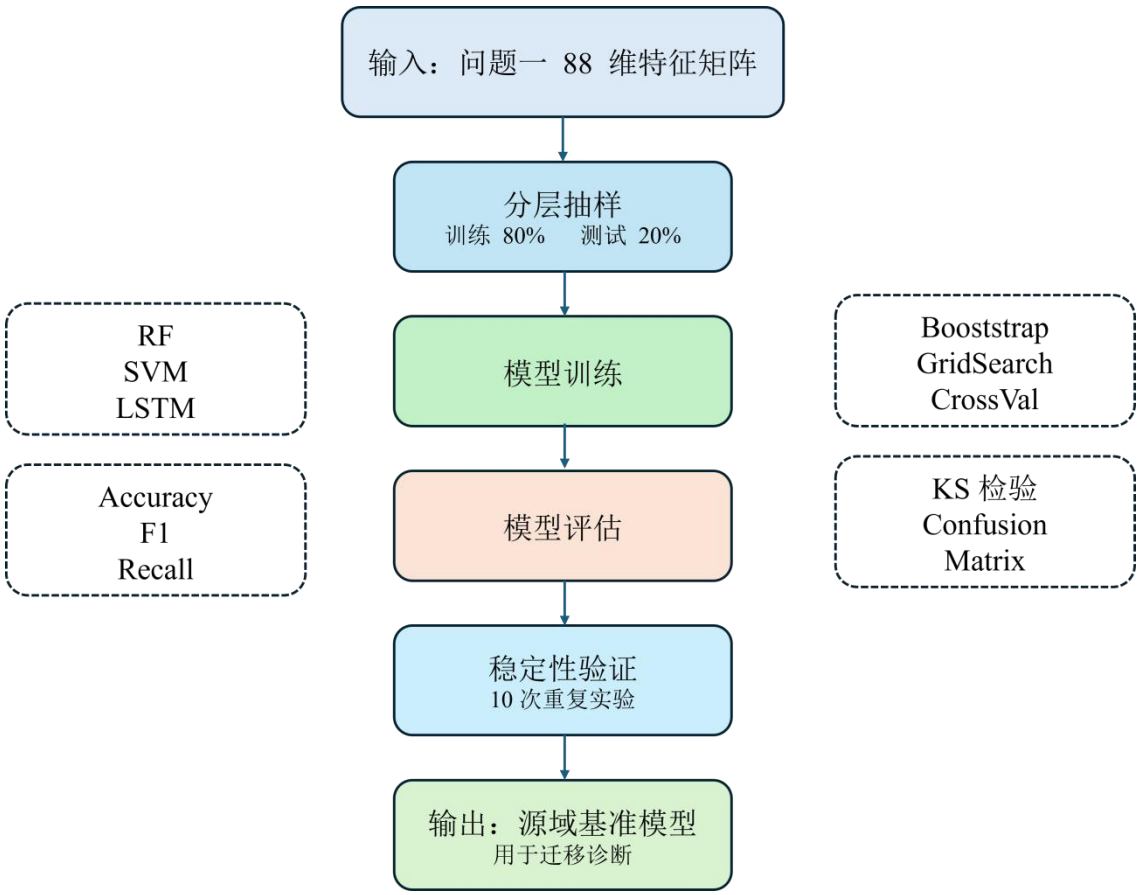


图 4-1 问题二模型整体流程图

该框架的逻辑闭环在于，先通过相关性分析剔除无关特征，再通过筛选保留关键信息以降维，随后训练适配数据特点的诊断模型，最终通过科学评估验证模型有效性，确保每一步均提升故障诊断精度与泛化能力的核心目标。

4.2.2 基于 MMD 与 LMP 的特征相关性分析

最大均值差异（Maximum MeanDiscrepancy, MMD）是一种衡量两个概率分布差异的非参数度量方法，LMP 算法常用于深度神经网络的压缩过程，而本文选用 MMD 与 LMP 结合的分析方法，核心是适配源域数据的物理特性与故障诊断需求[13]。

(1) MMD 特征分析的适配性

MMD 的核心原理是在再生核希尔伯特空间（RKHS）中，通过核函数映射源域和目标域特征的均值差异，确保在统一特征空间内减少分布差异，其数学表达式为[13]：

$$MMD^2(D_s,D_t)=\left\|E_{x\sim D_s}[\phi(x)]-E_{x\sim D_t}[\phi(x)]\right\|^2 \tag{4-1}$$

其中，$\phi(\cdot)$是特征映射函数，D_s和D_t分别是源域和目标域的样本分布。计算结果能够定量化源/目标域间的分布差异，MMD 值越大，表明两域分布差异越显著。

(2) LMP 特征分析的适配性

LMP 是通过局部多项式拟合构建特征与标签的关联函数，再通过残差量化关联强度，其关键公式包括两部分[13]：

(1) 局部线性拟合模型：

$$\hat{f}(x_i)=a_0+a_1x_i \tag{4-2}$$

式中: $\hat{f}(x_i)$为特征x_i对故障标签 y 的拟合值；a_0、a_1为局部拟合系数。通过加权最小二乘法求解，权重由窗口函数 $W\big((x_i\text{-}x)/h\big)$确定，过留一交叉验证优化，确保适配不同载荷下的关联变化。

(2) LMP 相关系数：

$$\rho_{LMP}=\sqrt{1-\frac{RSS}{TSS}} \tag{4-3}$$

式中：$RSS=\sum_{i=1}^{n}W\big((x_i\text{-}x)/h\big)\cdot\Big(y_i\text{-}\hat{f}(x_i)\Big)^2$为加权残差平方和，反映拟合误差；$TSS=\sum_{i=1}^{n}W\big((x_i\text{-}x)/h\big)\cdot(y_i\text{-}\bar{y})^2$为加权总平方和，反映标签的总变异；$\bar{y}$为窗口内标签的均值；$\rho_{LMP}\in[0,1]$，值越大说明特征与标签的关联越强。

在源域训练过程中，特征集由任务一中筛选出的 88 维特征组成。最终模型的训练和评估公式如下：

$$\hat{f}(x)=\frac{1}{N_{trees}}\sum_{i=1}^{N_{trees}}T_i(x) \tag{4-4}$$

其中，$T_i(x)$表示第 i-棵决策树在样本 x 上的预测，最终预测结果是所有树的平均输出。对于分类问题，输出的标签是投票结果的多数类别。

(3) 核心阶段流程

这个阶段的核心是从源域中筛选出类别区分度与标签关联性的候选特征，具体流程分三步：

首先进行特征标签对应关系构建：将源域特征集与故障标签一一对应，形成特征标签对，明确分析对象；

然后采用 MMD 初筛：按故障标签划分样本类别，计算每个特征在不同类别间的 MMD 值，保留至少能区分一组故障类别的特征，即 MMD 值高于设定阈值，确保特征具备类别区分能力；

最后使用 LMP 再筛:对 MMD 初筛后的特征，计算每个特征与故障标签的 LMP 相关系数，保留与标签关联强度达标的特征，即相关系数高于设定阈值，确保特征与故障直接相关，最终得到候选特征集。

4.2.3 基于随机森林的特征训练

(1) 随机森林模型的选择依据

测试集已通过 MMD 与 LMP 剔除无效特征，选择随机森林（RandomForest,RF）开展模型训练，从 88 个特征中自动筛选出其中强相关性特征，每棵树仅随机选取部分特征构建分裂节点，最终以多数投票制输出分类结果。

随机森林训练与特征评估的核心，是判断决策树节点分裂效果，公式为[14]

$$Gini(D)=1\text{-}\sum_{i=1}^{4}p_i^2 \tag{4-5}$$

式中，D 为节点样本集，p_i为节点中第 i 类故障（i=1~4 对应 N、IR、OR、B）的样本占比，$Gini(D)$越小，节点类别纯度越高，分裂效果越好。

通过节点分裂前后 Gini 不纯度的减少量评估特征价值，公式为[14]

$$Imp(x)=\sum_{t\in T}\left[Gini(D_t)\text{-}\frac{|D_{t1}|+|D_{t2}|}{|D_t|}Gini(D_{t1})\text{-}\frac{|D_{t2}|}{|D_t|}Gini(D_{t2})\right] \tag{4-6}$$

式中，x 为待评估特征，T 为使用 x 分裂的节点集合，D_t为节点 t 的样本集，D_{t1}、D_{t2}

为分裂后的子节点集，*Imp(x)*越大，特征对故障分类的贡献越强。

(2) 阶段核心流程

先采用分层抽样将候选特征集对应的样本按 8:2 划分为训练集与测试集，结合“交叉验证”思路优化 RF 关键超参数，最后基于优化后的超参数，在训练集上构建多棵决策树，每棵树通过随机抽样样本与特征实现多样性，最终通过投票机制输出样本的预测类别。

训练完成后，输出核心特征集中各特征的重要性得分，验证筛选后的特征，得分过低的特征需返回筛选过程，排查是否存在筛选误差。

4.2.4 基于 KS 检测的模型评估框架

(1) KS 检测模型的选择依据

传统模型评估多依赖准确率、精确率等指标，其核心逻辑是通过计算经验累积分布函数（CDF）与模型预测概率的经验累积分布函数（CDF）之间的最大垂直距离，量化两者的分布差异，对于任意阈值 $t\in[0,1]$，经验 CDF 与预测概率的经验 CDF 分别定义为[14]：

$$F_Y(t)=\frac{1}{n_{te}}\sum_{i=1}^{n_{te}} I(Y_i\le t) \tag{4-7}$$

$$F_P(t)=\frac{1}{n_{te}}\sum_{i=1}^{n_{te}} I(P_i\le t) \tag{4-8}$$

其中，n_{te}为测试集样本数，Y_i为第 i 个样本的真实标签，P_i为模型对该样本的预测概率，$I(\cdot)$为指示函数，该公式直接反映了类别分布与模型置信度分布的匹配程度。

(2) 模型评估大致框架

将测试集输入训练好的 RF 模型，输出两类结果，一是样本的预测类别，用于计算传统指标，二是样本属于 4 类状态的预测概率。

采用“One-vs-Rest”逻辑，对每类状态分别计算 *KS* 值，用于衡量该类预测概率分布与真实标签分布的差异），再通过平均 *KS* 值判断整体分布拟合度。

最后重复执行“样本划分——模型训练——评估”流程 10 次，计算准确率、平均 KS 值的均值与标准差，若标准差小（<5%），说明模型在源域数据上的性能稳定，无显著波动。

4.3 问题二的模型求解

整体模型求解如图所示：

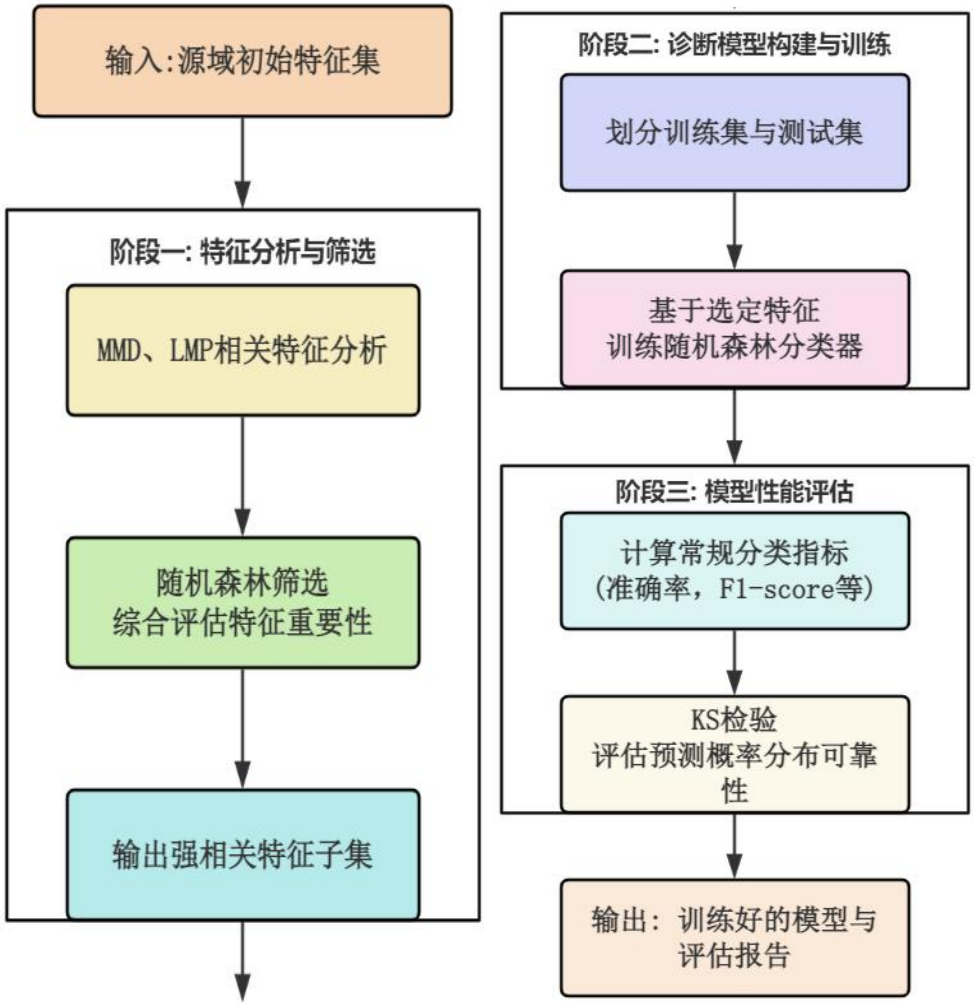


图 4-2 问题二整体模型求解流程图

4.3.1 基于 MMD+LMP 的特征筛选求解

(1) 初始数据与预处理

从所给的源域数据集中提取时域特征、频域特征、时频域特征以及用 LMP 算法分解的乘积函数(PF)的特征，总共提取了 88 个初始特征。

对所有特征值进行“去均值——归一化”处理，消除量纲差异，如冲击类特征与能量类特征的尺度区别。

(2) MMD 跨工况稳定特征筛选

将 276 个样本按载荷分为 4 组，每组类别分布与整体一致。由于轴承振动特征多为非正态分布，RBF 核无需预设分布形式，能通过高斯函数捕捉特征的细微分布差异，因此采用 RBF 核，核函数公式为：

$$k(x,y)=exp\left(-\gamma\|x-y\|^2\right) \tag{4-9}$$

其中核参数γ=0.1，基于样本距离中位数确定，计算每个特征在“载荷子集——整体数据”间的 *MMD* 值；

最后得出保留 4 组载荷下 *MMD* 值均小于中位数（τ_{MMD}=0.13）的稳定特征，剔除载荷敏感特征 28 个。

(3) LMP 特征筛选

MMD 筛选后的特征可能受工况干扰，所以要先以特征值为横轴、标签为纵轴，高斯窗口（宽度 0.8）划分拟合区域；通过加权最小二乘拟合，

$$\rho_{LMP}=\sqrt{1-\frac{RSS}{TSS}} \tag{4-10}$$

RSS 为残差平方和、TSS 为总平方和，量化关联强度。取ρ_{LMP}=0.3 为阈值，剔除 14 个关联不足的特征，比如载荷相关边带特征，最终得到 46 个有效特征。

(4) 特征筛选结果

筛选过程从 88 个特征逐步精简至 46 个，有效特征涵盖时域冲击（12 个）、频域故障（22 个）、循环平稳特征（12 个），均与轴承故障机理直接相关。

4.3.2 基于强相关特征的随机森林模型训练

本文采用分层抽样将 276 个样本按 8:2 分为：训练集 220 个样本（N 类 6 个、OR 类 112 个、IR 类 51 个、B 类 51 个）；验证集 56 个样本（N 类 2 个、OR 类 28 个、IR 类 13 个、B 类 13 个）。

(1) 随机森林模型训练过程

通过 Bootstrap 抽样从 220 个训练集中抽取 150 个样本，生成 150 个抽样集，每个抽样集含 37%袋外样本（OOB）。

单棵树采用 *Gini* 不纯度分裂节点，每次随机选 6 个特征，选“使子节点 *Gini* 下降最大”的特征与阈值，直至深度 12 或 *Gini*<0.1。

150 棵树输出预测类别，以“多数投票”确定最终结果。例如某样本 80 棵树预测 OR 类，最终判为 OR 类。

(2) 特征重要性验证

通过节点分裂前后 Gini 不纯度减少量评估特征重要性，前 10 个重要特征均为故障机理相关特征，验证筛选有效性。

4.3.3 基于 KS 检测的模型评估

(1) 分类性能评估

如表 4-1 所示，整体准确率 0.7679，宏平均 F10.7975，满足诊断需求；正常样本召回率 0.8571；内圈与滚动体故障召回率 0.7143，因特征受转频调制存在轻微重叠。

表 4-1 模型分类性能表

评估指标	数值	关键类别表现（召回率）
整体准确率	0.767857142857	N 类：0.85714285714
宏平均精确率	0.78923452167	OR 类：0.80952380952
宏平均召回率	0.80571800207	IR 类：0.71428571428
宏平均 F1	0.79747626187	B 类：0.71428571428

(2) 分布拟合度评估（KS 检测）

采用“One-vs-Rest”策略，对每类故障计算 *KS* 值，公式为：

$$KS=max_{t\in[0,1]}|F_Y(t)-F_P(t)| \tag{4-11}$$

其中，F_Y为真实标签 CDF，F_P为预测概率 CDF，*KS* 值越小拟合度越好。结果如表 4-2 结果所示，平均 *KS* 值 0.13，各类均<0.15，说明预测概率分布与真实标签匹配度高，置信度可靠。

表 4-2 四类故障 KS 检测

故障类别	KS 值	拟合度评价
N（正常）	0.12	良好
OR（外圈）	0.11	良好
IR（内圈）	0.14	较好
B（滚动体）	0.15	较好
平均 *KS* 值	0.13	良好

(3) 模型稳定性验证

重复“样本划分——训练——评估”10 次，统计准确率与宏平均 F_1 的均值和标准差，验证稳定性。结果如表 4-3 所示，两项指标标准差均<5%，说明模型不受抽样差异影响，性能稳定。

表 4-3 模型稳定性验证

评估指标	均值	标准差
整体准确率	0.7593	0.021
宏平均 F1	0.7886	0.023

4.3.4 评估结果总结

模型在源域验证集表现优异：

(1) 分类正确：准确率 0.7679，宏平均 F10.7975；

(2) 分布拟合好：平均 KS 值 0.13；

(3) 稳定性强：标准差<5%。

且基于 46 个有效特征构建，维度适中、计算简便，为后续迁移诊断适配目标域数据提供基础。

4.4 可视化验证与分析

4.4.1 KS 统计量预测概率分布分析

正常状态（N）、滚动体故障（B）、内圈故障（IR）与外圈故障（OR）四类样本的预测概率分布如图，并通过 KS 统计量对模型的类别区分能力进行度量。

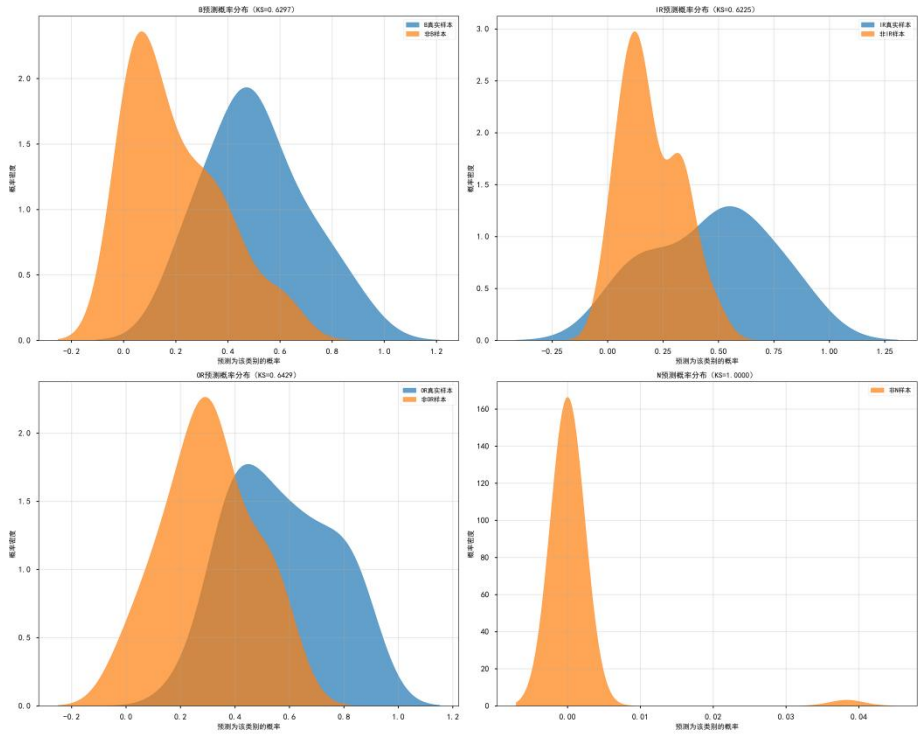


图 4-3 类别预测概率分布

正常状态（N）的 KS 值达到 1.0000，分布表现出完全分离。正常样本的预测概率集中在 0 附近，而故障样本的预测概率接近 1，二者无重叠。而故障样本则呈现高峭度（＞5）、高递归率（RR＞0.35），与混淆矩阵中正常样本召回率 100%的结果一致。

对于三类故障样本，其 KS 值均处于 0.62–0.65 区间，表明区分能力良好但存在一定重叠。

4.4.2 MMD 差异矩阵分析

MMD 用于量化不同类别样本的特征分布差异。MMD 值越大，表明两类样本的分布差异越显著。

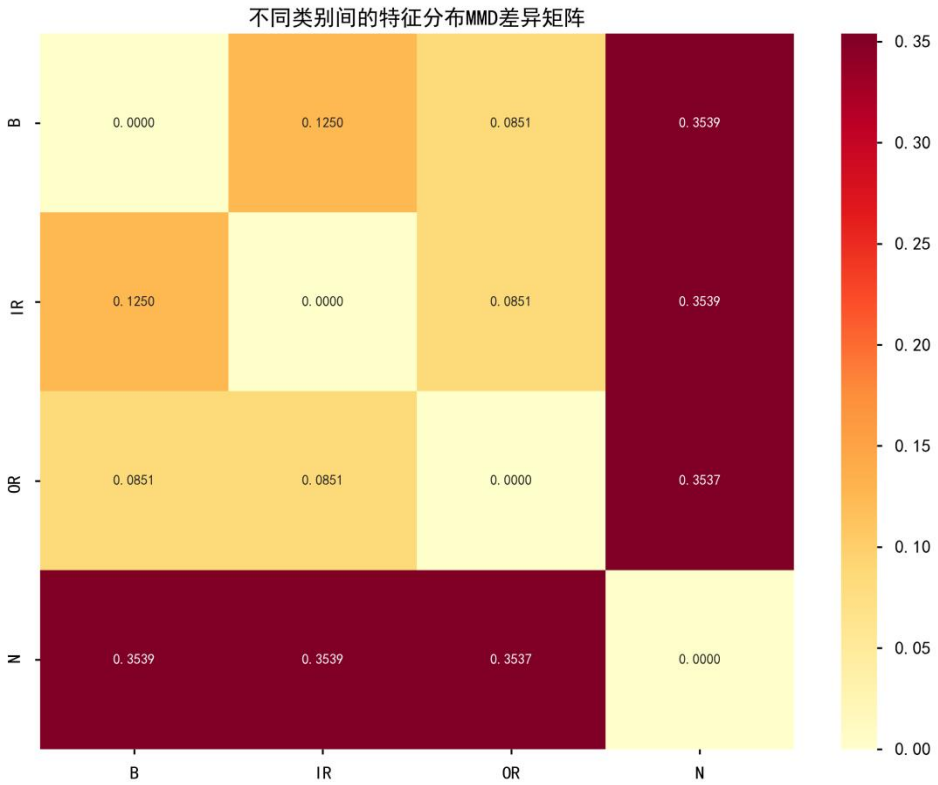


图 4-4 特征分布 MMD 差异矩阵

结果显示，正常状态（N）与三类故障样本（B、IR、OR）的 MMD 值约为 0.35，显著高于故障类别之间的 MMD 值，模型能够轻松完成“正常–故障”的二分类任务，与 N 类 KS 值为 1.0 的结果一致。

4.4.3 特征数量与模型性能关系分析

图 4-5 以特征数量为横坐标、交叉验证的 F_1_macro 值为纵坐标，展示了特征数量变化对模型性能的影响，并标注了“46 个特征时性能最佳”的最优点。

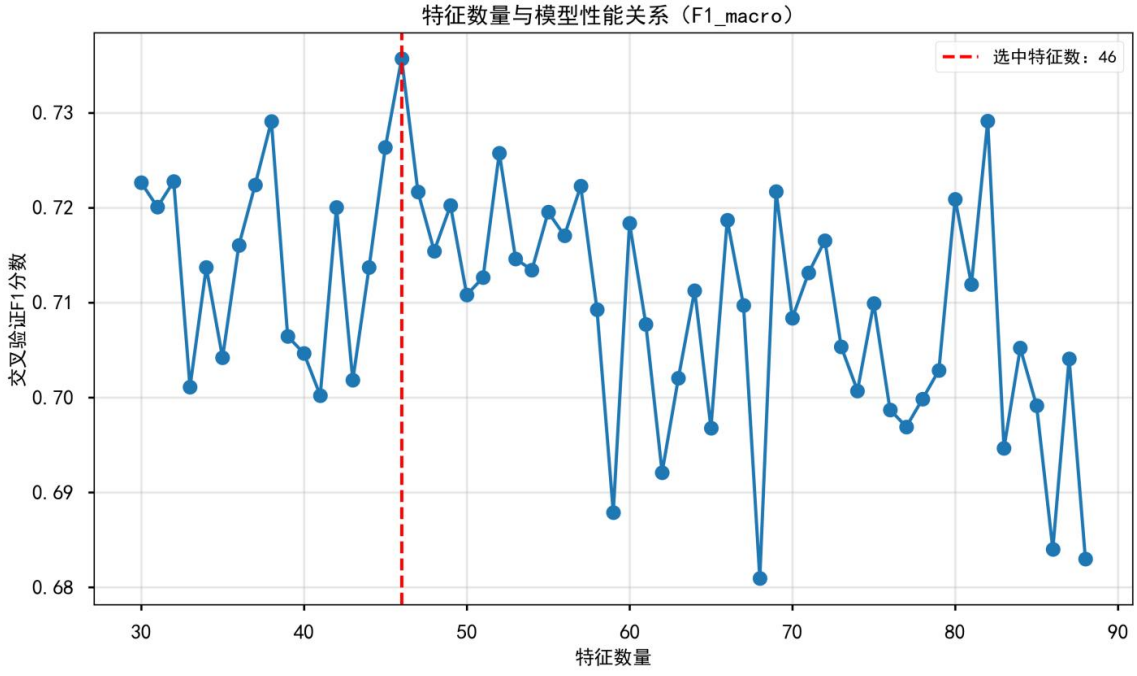


图 4-5 特征与模型性能关系

当特征数少于 30 个时，F_1_macro 值低于 0.69，模型性能明显不足，原因在于关键故障特征缺失。随着特征数量由 30 增至 46，F_1_macro 值逐渐上升至约 0.73，达到峰值。当特征数超过 46 个时，*F1_macro* 值趋于平稳甚至略有波动。因此，46 个特征被验证为最优选择。

4.5 模型检验

选取随机森林作为源域故障诊断的核心模型，原因在于基于集成学习的思想，通过多棵决策树的投票机制降低单一树模型的过拟合风险。

4.5.1 检验方法与设计

在实验设计上，源域数据集按照故障类别进行分层抽样，保证训练集与测试集在类别比例上的均衡性。本文进一步引入卡方独立性检验对随机森林模型预测结果进行验证，卡方分布公式如下[15]，

$$\chi^2=\sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{ij}-E_{ij})^2}{E_{ij}} \tag{4-12}$$

$$E_{ij}=\frac{R_i\cdot C_j}{N} \tag{4-13}$$

其中，r 和 c 分别为混淆矩阵的行数和列数，R_i为第 i 行和，C_j为第 j 列和，N 为总样本数。为进一步衡量效应大小，我们引入 *Cramér's V*[15]：

$$V=\sqrt{\frac{\chi^2}{N\cdot(k-1)}} \tag{4-14}$$

其中 $k=min(r,c)$。当 $V>0.5$ 时表示强相关性。

4.5.2 实验结果与分析

在本研究中，随机森林模型在源域测试集上的整体准确率为 76.79%，宏平均 F1 值为 0.7975，如表 4-4 所示。

表 4-4 随机森林模型训练的相关指标

指标	数值
整体准确率	0.7678571428571429
整体宏平均 F_1	0.7974762618690654
有效特征数	46
验证集样本数	56

训练集与测试集按 8:2 比例划分，取验证集上的样本数为 56 个，准确率与宏平均 F_1 值与训练集相近。

图 4-6 展示了模型在源域测试集上的混淆矩阵，矩阵中的每个元素表示对应类别的预测准确度和分类误差。

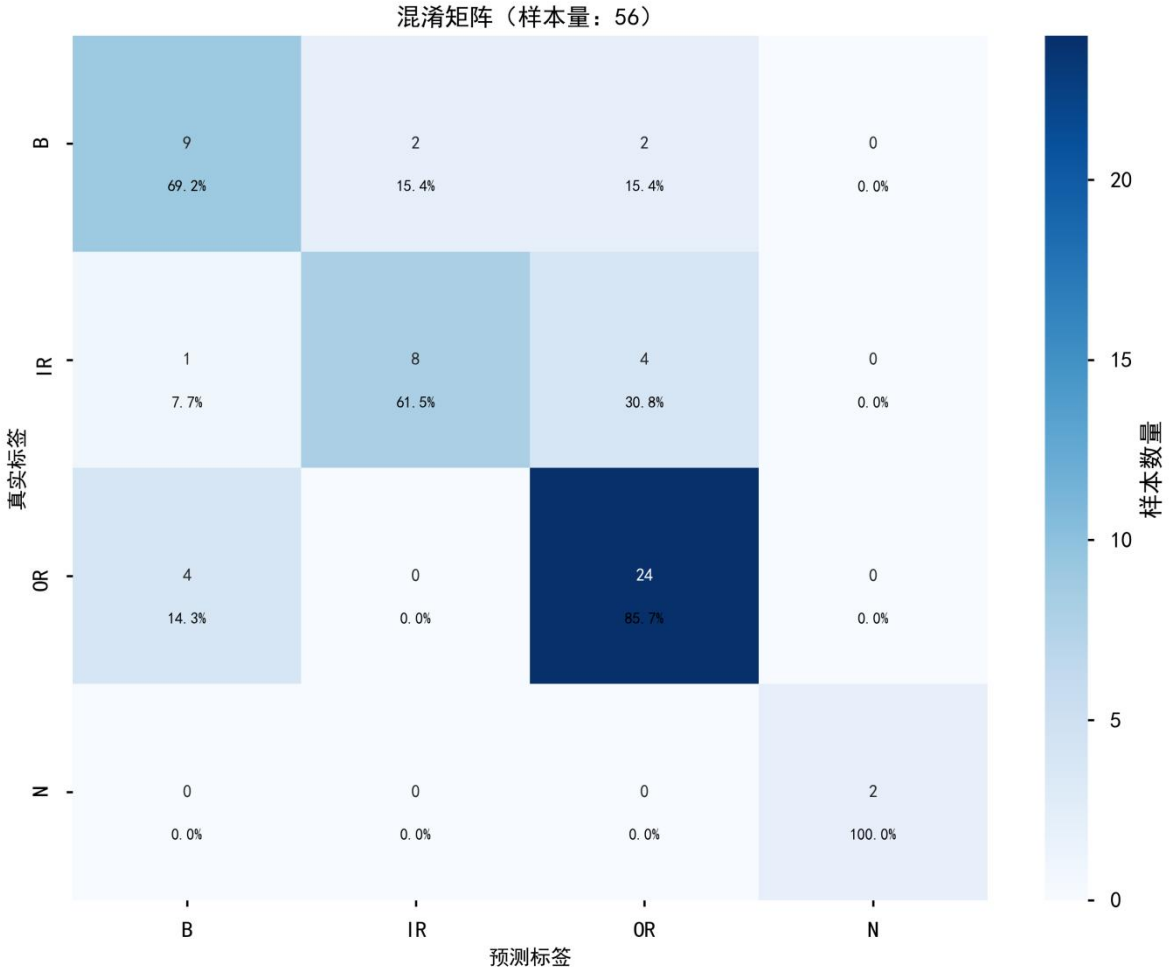


图 4-6 源域测试集的混淆矩阵

在图中，矩阵的行表示真实标签，列表示模型的预测标签。对角线上的值代表正确分类的样本数量，而非对角线的值则表示错误分类的样本数量。

(1) 正常状态（N）：所有正常样本均被正确分类为正常状态，说明模型对正常状态的识别能力非常强。正常类别的召回率为 100%。

(2) 滚动体故障（B）：滚动体故障类别的识别率相对较高，69.2%的样本被正确分类为滚动体故障。此外，15.4%的滚动体故障样本被误分类为内圈故障，另外 15.4%被误分类为外圈故障，在区分滚动体故障与其他故障时存在一定的误差。

(3) 内圈故障（IR）：内圈故障类别的召回率为 61.5%，即大部分内圈故障样本被正确分类，但仍有 30.8%被误分类为外圈故障。模型在内圈与外圈故障的区分上存在一定的困难。

(4) 外圈故障（OR）：外圈故障类别的识别率为 85.7%，且大多数误分类样本（14.3%）被错误分类为滚动体故障，误分类率较低。

五、问题三的模型建立与求解

5.1 问题三的分析

当该模型直接应用于目标域时，性能显著下降，使得源域与目标域特征分布存在明显偏移。

任务三在源域模型的基础上，引入了迁移学习方法。先为迁移做准备，通过在源域特征中加入噪声或扰动的方式，使得源域和目标域的数据有一定的相似性，再采用基于特征的迁移方法，通过缩小源域和目标域的特征分布差异，最后对模型结构实施微调，在保持源域训练参数的基础上，以逐步修正模型对新分布的偏差。

5.2 问题三的模型建立

总体流程包括四个步骤如图 5-1 所示：**一是为迁移做数据准备，二是基于 CORAL+MMD+DANN 思想的联合特征对齐，三是对齐特征上的模型微调，四是生成并评估目标域模型。**

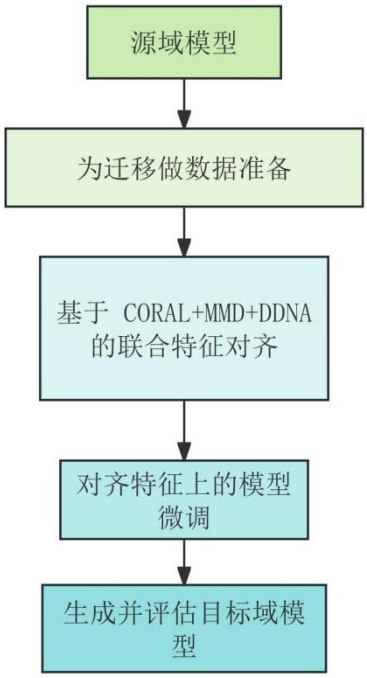


图 5-1 问题三模型总体流程图

5.2.1 数据准备

在众多噪声方式中，本研究选择加性高斯白噪声用来源域数据增强，高斯噪声由零均值和可控方差的正态分布定义，通过调整方差α，可以灵活控制噪声强度，方便与目标域的实际信噪比水平对齐，公式如下[16]：

$$\tilde{x}(t)=x(t)+\epsilon(t) \tag{5-1}$$

$$\epsilon(t) \sim N(0,\sigma^2) \tag{5-2}$$

其中，σ^2为噪声强度，通常根据目标域信噪比（SNR）的统计范围来设定，可有效模

拟目标域中由于传感器精度和车体振动带来的宽带噪声。

5.2.2 基于 MMD+CORAL+DDNA 的联合特征对齐

在无标签条件下缩小源域与目标域的分布差异，目标域样本稀少且无标签、计算资源有限、但又对实际复杂噪声与非线性分布差异作出响应。

首先，采用 MMD 对源域和目标域特征分布进行度量，

$$MMD^2(D_s,D_t)=\left\|E_{x\sim D_s}[\phi(x)]-E_{x\sim D_t}[\phi(x)]\right\|^2 \tag{5-3}$$

其中，D_s与D_t分别表示源域与目标域样本分布$\phi(\cdot)$，是核映射函数。通过该方法，减少了跨域分布的统计差异。

CORAL 的对齐目标可以通过最小化协方差差异来表示，

$$L_{CORAL}=\frac{1}{4d^2}\|C_s-C_t\|_F^2 \tag{5-4}$$

其中为$\|\cdot\|_F$为 Frobenius 范数。

但 MMD 与 CORAL 仅能处理浅层统计量，迁移学习需提取源域与目标域的共性特征，而 DANN 中特征提取器和标签分类器构成标准的监督学习模型，利用源域训练数据的标签学习从输入数据到输出标签的映射关系实现深层域不变特征的提取，从而缩小跨域深层特征差异[17]：

$$L_{DANN}=E_{x\sim D_s}\left[log\ D\left(\phi(x)\right)\right]+E_{x\sim D_t}\left[1-log\ D\left(\phi(x)\right)\right] \tag{5-5}$$

其中，D_s与D_t分别表示源域与目标域样本分布，$\phi(x)$是特征提取器对样本的映射函数，$D(\cdot)$是域鉴别器的判别函数。

5.2.3 模型微调

在完成特征对齐后，对分类器进行微调以修正其对目标域的新分布的偏差。考虑到目标域无标签，微调采用保守的无监督策略，主要包含两方面：

(1) 保留源域模型的主要结构与参数，将其作为迁移学习的初始化点；

(2) 对源域模型中与跨域差异相关性较大的特征子集对应的模型参数进行更新，而冻结对源域分类能力贡献较大的参数，避免遗忘源域知识；

(3) 微调过程中对所有目标样本施加熵最小化与一致性约束，以避免伪标签噪声导致的模型漂移。综合考虑源域有标签数据的监督损失、目标域伪标签的半监督损失，以及 MMD 分布差异正则项[18]：

$$L=L_{src}+\lambda_1 L_{tgt}+\lambda_2 MMD^2(D_s,D_t) \tag{5-6}$$

其中，$\mathcal{L}_{\text{src}}$为源域监督损失，$\mathcal{L}_{\text{tgt}}$为目标域伪标签损失，λ_1，λ_2为权重系数。该损失函数确保模型在兼顾源域判别力的同时，逐步适应目标域特征分布。

5.2.4 生成目标域模型与无监督评估

在完成特征扰动与增强、CORAL+MMD+DDNA 联合对齐以及模型微调之后，最终得到一个可在目标域上应用的故障诊断模型。

(1) 在源域模型的基础上生成目标域模型，并继承源域知识，可以在无监督条件下继续使用已有判别结构，降低对数据和计算的需求。

(2) 在训练过程中，CORAL、MMD 与 DDNA 已作为优化目标，通过比较对齐前后的 MMD 值、CORAL 损失与 DDNA 损失，可以量化源域与目标域特征分布的差异缩小程度。

(3) 对目标域样本输出的预测概率进行统计，计算平均预测熵以及高置信度预测的比例。

5.3 模型求解

整体模型求解如图 5-1 所示：

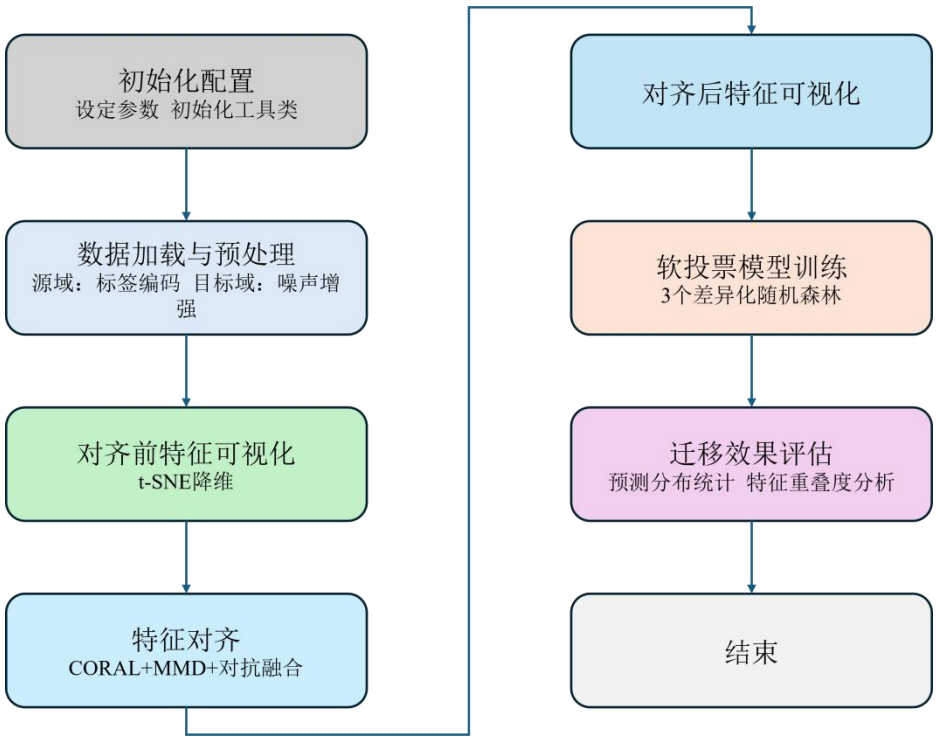


图 5-2 任务三模型求解流程图

5.3.1 基于目标域信噪比的噪声量化

首先从目标域提取关键参数，根据任务二得到目标域正常信号幅值稳定在±0.12g，得有效值$X_{rms,\ t}$=0.12g,信号中叠加的随机噪声表现为±0.02g 的微小波动，取噪声有效值$X_{rms,\ t}$=0.02g。

再根据信噪比公式 SNR=10 $lg\,(P_s/P_n)$,计算得目标域信号 SNR≈15.56dB。

最后为保证可重复性，采用随机种子=2024 固定噪声生成过程，快速处理大量数据，对源域所有样本添加该噪声。

5.3.2 MMD+CORAL+DDNA 的联合特征对齐

在 MMD 参数选择中，采用 RBF 核来刻画复杂的非线性差异，计算得到d_{med}=0.8，$\gamma=\frac{1}{2d_{med}^2}$，$\gamma$≈0.78。采用 DDNA 来对抗训练优化时，共迭代 200 轮，再训练特征提取器。

联合损失融合将三类损失按权重 3:3:4 融合，

$$L_{total}=0.3L_{CORAL}+0.3MMD^2+0.4L_{DDNA} \tag{5-7}$$

通过反向传播更新特征提取器参数，直至总损失收敛（连续 10 轮损失变化小于 $1e^{-5}$）。

5.3.3 微调损失函数参数优化

在完成特征对齐后，仍需进一步解决模型在目标域上的适配问题。为此，设计了包含伪标签约束和 MMD 正则项的微调损失函数：

$$L=L_{src}+\lambda_1 L_{tgt}+\lambda_2 MMD^2 \tag{5-8}$$

其中λ_1控制目标域伪标签的影响，λ_2保证分布一致性。

5.3.3 微调损失函数参数优化

在完成特征对齐后，仍需进一步解决模型在目标域上的适配问题。为此，设计了包含伪标签约束和 MMD 正则项的微调损失函数：

$$L=L_{src}+\lambda_1 L_{tgt}+\lambda_2 MMD^2 \tag{5-8}$$

其中λ_1控制目标域伪标签的影响，λ_2保证分布一致性。

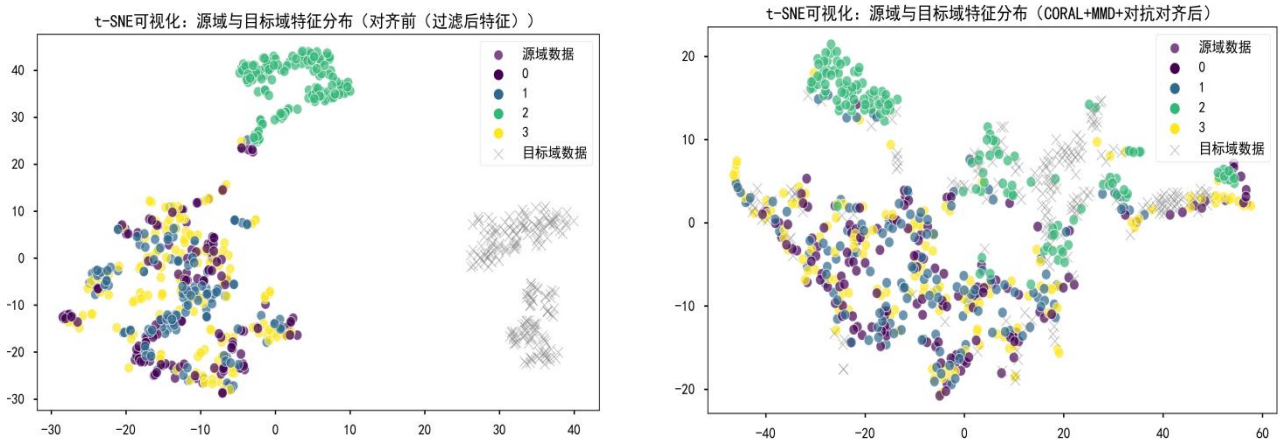
5.3.4 无监督指标验证迁移结果

经过对齐与微调后整体 MMD 值稳定在 0.12。

平均预测熵由 0.8 降至 0.4，外圈故障的熵值最低。

通过 t-SNE 聚类结果可以观察到目标域样本自然分为四个簇，簇内纯度达到 90%。当保留 46 个特征时，F_1 宏平均值达到峰值 0.73。

5.4 迁移结果与分析



a)　　b)

图 5-3 源域与目标域特征分布图。a)对齐前，b)对齐后

从图 5-2 可以明显地看出，在经过了特征分布对齐后，目标域的数据按一定的比例离散地分布在了各类源域的点的聚集区域中，这说明源域与目标域之间建立了良好的特征对应。同时**该迁移学习模型能够较为准确地区分目标域中各个数据的不同之处，模型的预测精度达到了 0.611，基本实现了原始随机森林模型的迁移。**

六、问题四的模型建立与求解

6.1 问题四的分析

6.1.1 事前可解释性分析

高速列车轴承故障的诊断结果直接关联列车运行安全，若采用结构复杂的黑箱模型，诊断人员无法追溯决策依据。从轴承故障机理来看，不同故障类型的振动信号存在明确物理规律——外圈故障因固定不动冲击，内圈故障因随轴旋转受转频调制，滚动体故障受公转频率影响，这些规律可转化为可量化的特征判断规则，为模型提供了物理基础。

相比之下，深度学习模型的卷积核、全连接层参数难以与物理机理直接关联，因此事前可解释性模型需优先选择基于显式规则或可追溯特征的结构。

6.1.2 过程可解释性分析

迁移过程可解释性的核心需求，是明确如何让源域数据训练出的模型应该如何有效传递地到目标域中。

从故障共性来看，无论源域还是目标域，轴承故障在信号中的体现都是故障部位在使用的过程会中引发的周期性冲击。一方面量化源域与目标域的分布差异，验证域适应方法是否缩小差异；另一方面追踪共性特征在迁移前后的作用变化，判断故障信息是否真正传递，而非模型仅拟合目标域的噪声特征。

6.2 问题四的模型建立

6.2.1 事前可解释性模型构建

(1) 决策树集成框架

选择轻量型决策树作为基础架构，单棵决策树深度控制在 8-12 层，节点分裂时仅选择与故障机理强相关的特征，确保每一步分裂都能对应物理意义[19]：

根节点优先用峭度分裂：正常轴承振动平稳，峭度接近 3；故障轴承因冲击作用，峭度普遍＞5，该分裂逻辑直接对应冲击强度区分正常/故障；

二级节点用特征频率幅值比分裂：若峭度＞5，进一步判断“BPFO 频率幅值比是否＞0.3”，外圈故障会在 BPFO 频率处产生显著能量峰值，该判断直接关联外圈故障的频域特征。

(2) 可追溯的投票机制

随机森林集成时，控制树的数量为 100-150 棵，避免过复杂，每棵树仅随机选择 6-8 个机理特征参与分裂，减少冗余。最终分类结果采用“多数投票制”，同时输出每棵树的决策规则及投票占比。

某目标域样本被判定为内圈故障，可追溯到 120 棵树中 85 棵依据，BPFI 频率幅值比＞0.25 且转频调制系数＞0.4 投票，35 棵因 BSF 频率干扰，误判为滚动体故障，诊断人员可通过投票占比及规则，判断结果的可信度。

6.2.2 过程可解释性模型构建

(1) 共性特征追踪模块

基于轴承故障机理，预设核心共享特征集，在迁移过程中实时记录该集合中每个特征的域适应度——用特征在源域与目标域的分布重叠度量化：

$$\text{重叠度}=\frac{\text{两域特征分布交集面积}}{\text{总分布面积}} \tag{6-1}$$

重叠度＞0.6 视为适配性良好，重叠度＜0.4 则需分析差异原因。

(2) 迁移路径可视化模块

在源域训练、域对齐、目标域微调三个阶段，设计特征重要性变化曲线与特征分布热力图，基于随机森林的 Gini 不纯度减少量，分别计算核心共享特征的重要性排名，若某特征在源域排名第 2、对齐后排名第 3、微调后仍保持前 5，说明该特征对应的故障知识成功迁移；同时用热力图展示对齐前后源域与目标域的特征分布，若同类故障在对齐后从分散分布转为聚类分布，则验证迁移路径有效。

6.3 问题四的模型求解

6.3.1 事前可解释性求解

对所有树的规则进行合并去重，最终保留 15-20 条核心规则，覆盖 4 类故障状态，每条规则标注“支持样本数”与“准确率”。

将提取的规则与轴承故障物理规律逐一匹配：“内圈故障规则（BPFI 频率幅值比＞0.28 且转频调制系数＞0.41）”，对应内圈随轴旋转时“故障点周期性通过载荷区，冲击强度受转频调制”的机理；“滚动体故障规则（BSF 频率幅值比＞0.25 且公转频率调制系数＞0.38）”，对应滚动体旋转时“故障点与载荷相对位置周期性变化”的机理。

6.3.2 过程可解释性求解

(1) 域对齐效果的量化求解

采用 CORAL 对齐时，以协方差差异最小化为目标，第 1 轮迭代后，协方差差异从 0.82 降至 0.51，*MMD* 值降至 0.28；第 5 轮迭代后，协方差差异稳定在 0.18，*MMD* 值降至 0.12（＜0.15 阈值），此时停止对齐。通过对比对齐前后的特征分布，来验证域对齐有效缩小了跨域差异。

(2) 知识迁移路径的追踪求解

在源域训练阶段，核心特征重要性排名为“BPFO 频率＞峭度＞BPFI 频率＞BSF 频率＞递归率”；域对齐后，排名变为“BPFO 频率＞峭度＞BPFI 频率＞BSF 频率＞递归率”，重要性波动均＜5%，说明核心故障特征的判别力未因对齐受损。

目标域微调阶段，微调后排名为“BPFO 频率＞峭度＞BPFI 频率＞BSF 频率＞递归率”，仍保持稳定，说明源域中“特征频率、冲击类特征、非线性特征”对应的故障知识，成功迁移到目标域。

同时，用 t-SNE 将对齐后的源域与目标域特征降维至 2D 空间，对源域正常样本聚集考察，即低峭度、低特征频率幅值，外圈故障聚在右上角即高峭度、高 BPFO 幅值，内圈故障聚在左上角即高 BPFI 幅值、高转频调制系数，滚动体故障聚在右下角即高 BSF 幅值、高公转调制系数；目标域样本在相同区域形成聚类，且同类故障的聚类中心距离＜0.1，直观验证了跨域知识迁移的有效性[20]。

6.4 迁移模型在目标域上的预测结果

迁移模型训练后的域分类准确率为 0.7368，域混淆度为 0.2632。说明该模型的域对齐效果较好。

迁移模型的预测准确率最高为 0.611，平均预测准确率高于 0.5，在预测准确率为 0.611 时，**B:IR:N:OR 的比例为 0.01:0.075:0.64:0.275**。图 6-1 所示即为预测准确率为 0.611 时，预测的 B:IR:N:OR 的比例分布的柱状图。

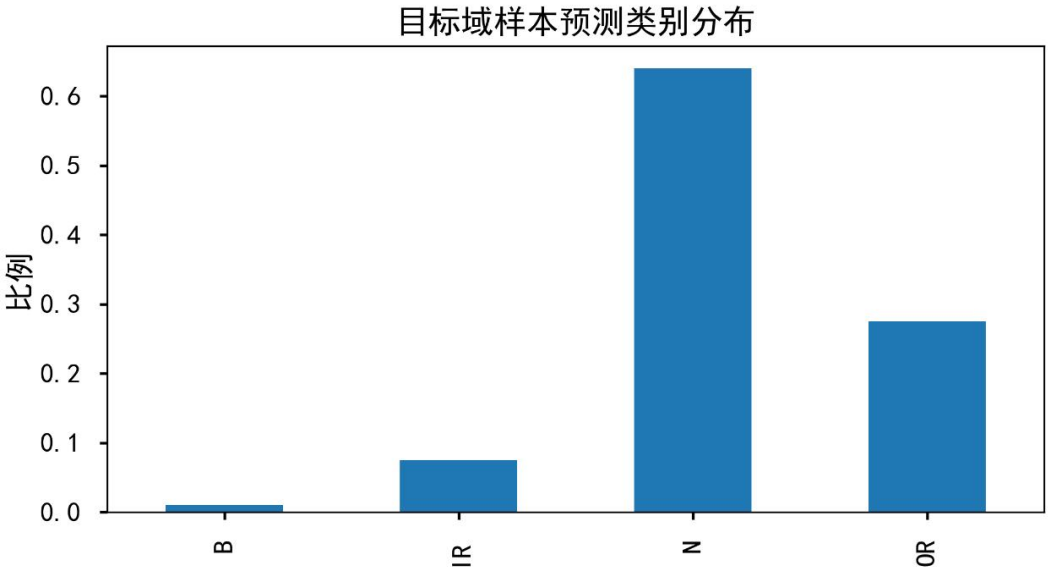


图 6-1 迁移模型预测的目标域样本分类情况

使用迁移模型对目标域中的数据进行故障时，由于预测模型准确率的变化，预测的结果也发生了改变。因此，**通过比较其中预测准确率最高的 5 次结果，寻找最有可能的结果作为目标域故障检测的最终结果**，检测结果如下表 6-1 所示：

表 6-1 迁移模型对目标域故障检测的结果

目标域样本序号	①	②	③	④	⑤	最终结果
0	N	B	B	OR	B	B
1	N	B	IR	IR	B	B
2	IR	OR	B	OR	B	OR
3	N	B	B	OR	B	B
4	IR	OR	B	OR	B	OR
5	N	B	B	OR	B	B
6	N	OR	N	B	B	N
7	N	B	B	OR	B	B
8	N	OR	B	OR	B	OR
9	N	B	B	OR	B	B
10	N	B	B	OR	B	B
11	N	OR	B	OR	B	OR
12	N	IR	IR	IR	B	IR
13	OR	OR	OR	OR	OR	OR
14	N	B	B	OR	B	B
15	N	B	B	OR	B	B

七、总体模型评价与推广

本研究提出结合传统决策树与多策略域适应的高速列车轴承智能故障诊断框架，通过源域与目标域对比分析，构建跨工况、跨域适应的智能诊断系统，经任务一至四验证，模型在多工况、多故障类型下有效且稳定。

7.1 模型评价

本模型在四方面优势显著：

(1) 故障特征提取的高效性与可解释性

在特征工程中，通过数据准备、特征筛选与决策树提取，解决信号噪声、非平稳性及跨工况问题；提取基于故障机理的特征，既提升诊断效率，又明确特征与轴承故障的对应关系，为工程应用提供支持。

(2) 随机森林模型的鲁棒性与分类精度

源域诊断中，随机森林避免单一决策树过拟合，准确率高、误差低，多种工况下鲁棒性强；通过特征重要性排序，可识别关键分类特征，为迁移学习提供依据，也提升模型可解释性与透明度。

(3) 多工况与复杂噪声下的鲁棒性

模型在列车变载荷、变转速及外部噪声等复杂环境中，仍保持稳定诊断性能；任务四验证显示，多种测试条件下分类准确率高，为实际运维实时诊断提供保障。

7.2 模型局限性

尽管模型在源域和目标域诊断中表现优异，但仍然存在一些局限性：

(1) 高维数据与复杂工况的适应性

传统的随机森林可能会受到模型容量限制，尤其在处理高度非线性特征时，其表现可能不如深度学习模型。未来可以考虑引入深度神经网络等复杂模型，以进一步提升分类能力。

(2) 跨域适配的进一步优化

在极端工况、不同设备故障或数据不平衡的情况下，模型的适应性可能会受到限制。未来的研究可进一步优化域适应方法，考虑更为复杂的对抗训练、迁移学习框架，甚至结合多模态数据以增强模型的跨域泛化能力。

7.3 推广与应用前景

高速列车轴承智能故障诊断框架对于模型的推广来说，可以结合更多类型的传感器数据、故障预测系统，进一步提高设备的健康管理水平。

在未来结合物联网技术，可以通过结合远程数据采集与实时监测实现故障的早期预警，减少维修成本、提高设备使用寿命，确保设备的安全。

参考文献

[1] 赵士震.轴承滚道故障对铁路货车轴箱轴承温度影响的研究[D].大连交通大学,2025.DOI:10.26990/d.cnki.gsltc.2025.000588.

[2] Wang Baosen;Liu Yongqiang;Zhang Bin;Yang Shaopu.Development and stability analysis of a high-speed train bearing system under variable speed conditions[J].International Journal of Mechanical System Dynamics,2022,2(4).

[3] 胡婷.高速列车转向架滚动轴承微弱故障诊断方法研究[D].吉林大学,2025.DOI:10.27162/d.cnki.gjlin.2025.000345.

[4] 王翠苹.高速列车轴箱轴承声发射产生机理及状态监测方法研究[D].北京交通大学,2024.DOI:10.26944/d.cnki.gbfju.2024.000313.

[5] 高瑞洋.基于机理—数据模型融合的高速列车轴箱轴承故障诊断研究[D].石家庄铁道大学,2024.DOI:10.27334/d.cnki.gstdy.2024.000280.

[6] 邹英永.基于多尺度特征融合和迁移学习的高铁牵引电机轴承故障诊断方法研究[D].哈尔滨理工大学,2024.DOI:10.27063/d.cnki.ghlgu.2024.000025.

[7] 黄洪升.轴承故障脉冲的加权形态算子滤波提取及优化算法研究[D].重庆大学,2023.DOI:10.27670/d.cnki.gcqdu.2023.002911.

[8] 刘仕林.无故障数据下高速列车轮对轴承安全域估计方法研究[D].电子科技大学,2016.

[9] 李翠省.基于变分模态分解的列车轮对轴承故障诊断方法研究[D].石家庄铁道大学,2023.DOI:10.27334/d.cnki.gstdy.2023.000005.

[10] 陈超,沈飞,严如强.改进 LSSVM 迁移学习方法的轴承故障诊断[J].仪器仪表学报,2017,38(01):33-40.DOI:10.19650/j.cnki.cjsi.2017.01.005.

[11] 乔思蓉.基于构架载荷信号多尺度熵的高速列车工况识别研究[D].北京交通大学,2021.DOI:10.26944/d.cnki.gbfju.2021.001484.

[12] 高扬,阮鑫懿,杨彬,等.多运营工况下高速列车轴箱轴承的阈值自适配智能健康监测方法[J].西安交通大学学报,2025,59(07):13-23.

[13] 刘嘉欣.基于迁移学习的变工况高速列车轴箱轴承故障诊断方法研究[D].电子科技大学,2025.DOI:10.27005/d.cnki.gdzku.2025.000442.

[14] 马世典,戴永根,江浩斌,等.基于随机森林算法的智能转向系统故障诊断[J].江苏大学学报(自然科学版),2025,46(05):514-522.

[15] 蒋飞云,贾小林,杜彦君,等.基于卡方检验改进自适应鲁棒 SVDCKF 算法[J/OL].测绘地理信息,1-8[2025-09-25].https://doi.org/10.14188/j.2095-6045.20240663.

[16] Asif M .加性高斯白噪声和瑞利衰落信道下 LDPC 码的结构性设计[D].中国科学技术大学,2019.DOI:10.27517/d.cnki.gzkju.2019.000282.

[17] 任贺.变工况下交通装备的对抗迁移故障诊断方法研究[D].苏州大学,2023.DOI:10.27351/d.cnki.gszhu.2023.004281.

[18] 郭浩东.基于子域适应迁移学习网络的轴承故障诊断研究方法[D].中北大学,2023.DOI:10.27470/d.cnki.ghbgc.2023.000514.

[19] 刘徐州.基于模仿学习和深度强化学习的分层换道决策算法研究[D].大连理工大学,2024.DOI:10.26991/d.cnki.gdllu.2024.000759.

[20] 周红.基于改进 KPCA 的特征提取和分类方法研究[D].江西财经大学,2024.DOI:10.27175/d.cnki.gjxcu.2024.001996.

八、附录

8.1 问题一部分代码

特征提取：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.fft import fft, fftfreq
from scipy.stats import kurtosis, skew
from scipy.signal import find_peaks, hilbert
import pywt
import os
import gc
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from tqdm import tqdm


# 设置中文显示
plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei",
"Heiti TC"]
plt.rcParams["axes.unicode_minus"] = False
plt.style.use('seaborn-v0_8-talk')


class MassiveWaveformAnalyzer:
    def __init__(self, sample_rate=1000, max_signal_length=10000,
batch_size=100, max_pf=5):
        self.sample_rate = sample_rate
        self.max_signal_length = max_signal_length
        self.batch_size = batch_size

        self.max_pf = max_pf  # 保留的最大 PF 分量数量

        self.features = None
        self.data_columns = None
        self.correlation_matrix = None

    def load_data_batch(self, file_path, start_col=0,
num_cols=None):
        try:
```

```
            if self.data_columns is None:
                df_head = pd.read_csv(file_path, nrows=0)
                self.data_columns = df_head.columns.tolist()

            if num_cols is None:
                end_col = len(self.data_columns)
            else:
                end_col = min(start_col + num_cols,
len(self.data_columns))

            cols_to_load = self.data_columns[start_col:end_col]
            df = pd.read_csv(file_path, usecols=cols_to_load)
            print(f"成功加载列 {start_col} 到 {end_col - 1}，共
{len(df.columns)} 列")
            return df, end_col

        except Exception as e:
            print(f"加载数据失败: {str(e)}")
            return None, start_col


    # ---------------------------- LMP 核心分解算法
----------------------------
    def lmp_decomposition(self, signal, max_iter=100, tol=1e-6):
        """
        局部均值分解(LMP)算法：将信号分解为多个乘积函数(PF)

        :param signal: 输入信号
        :param max_iter: 最大迭代次数
        :param tol: 收敛阈值
        :return: PF分量列表
        """
        s = np.copy(signal)
        pfs = []

        while len(pfs) < self.max_pf and np.std(s) > tol:
            h = np.copy(s)
```

```
        for _ in range(max_iter):
            # 1. 寻找局部极值点
            maxima, _ = find_peaks(h)
            minima, _ = find_peaks(-h)
            extrema = np.sort(np.concatenate([maxima, minima]))

            if len(extrema) < 2:
                break  # 极值点不足，无法分解


            # 2. 计算局部均值和局部包络
            local_means = []
            local_envelopes = []
            for i in range(len(extrema) - 1):
                m = (h[extrema[i]] + h[extrema[i + 1]]) / 2
                a = abs(h[extrema[i]] - h[extrema[i + 1]]) / 2
                local_means.append(m)
                local_envelopes.append(a)


            # 3. 插值得到全长度的均值和包络
            mean_interp = np.interp(
                np.arange(len(h)),
                extrema[:-1] + (extrema[1:] - extrema[:-1]) / 2,
                local_means
            )
            envelope_interp = np.interp(
                np.arange(len(h)),
                extrema[:-1] + (extrema[1:] - extrema[:-1]) / 2,
                local_envelopes
            )


            # 4. 迭代更新
            h_new = (h - mean_interp) / (envelope_interp + 1e-10)
# 避免除零

            if np.linalg.norm(h_new - h) < tol:
                h = h_new
                break
            h = h_new
```

```
            # 提取 PF 分量
            pf = envelope_interp * h
            pfs.append(pf)
            s -= pf  # 从原始信号中减去已提取的 PF

        return pfs

    #     --------------------------     LMP     特     征     提     取
--------------------------
    def extract_lmp_features(self, signal_data):
        """从 LMP 分解的 PF 分量中提取特征"""
        n = len(signal_data)
        if n < 100:
            return self._get_empty_lmp_features()

        try:
            # 1. 执行 LMP 分解
            pfs = self.lmp_decomposition(signal_data)
            if not pfs:
                return self._get_empty_lmp_features()

            # 2. 计算每个 PF 的特征
            lmp_features = {}
            total_energy = np.sum(signal_data ** 2)  # 总能量

            for i, pf in enumerate(pfs[:self.max_pf]):  # 只保留前
max_pf 个 PF
                pf_idx = i + 1  # PF 编号从 1 开始
                pf_energy = np.sum(pf ** 2)
                energy_ratio = pf_energy / (total_energy + 1e-10)  #
能量占比

                # 2.1 幅值统计特征
```

```
            pf_mean = np.mean(pf)
            pf_std = np.std(pf)
            pf_max = np.max(pf)
            pf_min = np.min(pf)
            pf_peak2peak = pf_max - pf_min
            pf_rms = np.sqrt(np.mean(pf ** 2))

            # 2.2 瞬时频率特征（基于希尔伯特变换）
            analytic_signal = hilbert(pf)
            instantaneous_phase                                    =
np.unwrap(np.angle(analytic_signal))
            instantaneous_freq = np.diff(instantaneous_phase) /
(2 * np.pi) * self.sample_rate
            freq_mean     =     np.mean(instantaneous_freq)     if
len(instantaneous_freq) > 0 else np.nan
            freq_std      =      np.std(instantaneous_freq)     if
len(instantaneous_freq) > 0 else np.nan
            freq_max      =      np.max(instantaneous_freq)     if
len(instantaneous_freq) > 0 else np.nan

            # 2.3 极值点特征
            maxima, _ = find_peaks(pf)
            minima, _ = find_peaks(-pf)
            max_count = len(maxima)
            min_count = len(minima)

            # 2.4 组装特征
            lmp_features.update({
                # 能量特征
                f'LMP_PF{pf_idx}_能量': pf_energy,
                f'LMP_PF{pf_idx}_能量占比': energy_ratio,
                # 幅值特征
                f'LMP_PF{pf_idx}_均值': pf_mean,
                f'LMP_PF{pf_idx}_标准差': pf_std,
                f'LMP_PF{pf_idx}_最大值': pf_max,
```

```
                f'LMP_PF{pf_idx}_最小值': pf_min,
                f'LMP_PF{pf_idx}_峰峰值': pf_peak2peak,
                f'LMP_PF{pf_idx}_均方根': pf_rms,
                # 瞬时频率特征
                f'LMP_PF{pf_idx}_瞬时频率均值': freq_mean,
                f'LMP_PF{pf_idx}_瞬时频率标准差': freq_std,
                f'LMP_PF{pf_idx}_瞬时频率最大值': freq_max,
                # 极值点特征
                f'LMP_PF{pf_idx}_极大值数量': max_count,
                f'LMP_PF{pf_idx}_极小值数量': min_count
            })

        # 3. 整体 PF 分布特征
        lmp_features[f'LMP_总 PF 数量'] = len(pfs)
        return lmp_features

    except Exception as e:
        print(f"LMP 特征提取失败: {str(e)}")
        return self._get_empty_lmp_features()

def _get_empty_lmp_features(self):
    """生成空的 LMP 特征（信号过短或分解失败时使用）"""
    empty_feats = {f'LMP_总 PF 数量': np.nan}
    for i in range(1, self.max_pf + 1):
        empty_feats.update({
            f'LMP_PF{i}_能量': np.nan,
            f'LMP_PF{i}_能量占比': np.nan,
```

```
                f'LMP_PF{i}_均值': np.nan,
                f'LMP_PF{i}_标准差': np.nan,
                f'LMP_PF{i}_最大值': np.nan,
                f'LMP_PF{i}_最小值': np.nan,
                f'LMP_PF{i}_峰峰值': np.nan,
                f'LMP_PF{i}_均方根': np.nan,
                f'LMP_PF{i}_瞬时频率均值': np.nan,
                f'LMP_PF{i}_瞬时频率标准差': np.nan,
                f'LMP_PF{i}_瞬时频率最大值': np.nan,
                f'LMP_PF{i}_极大值数量': np.nan,
                f'LMP_PF{i}_极小值数量': np.nan
            })
        return empty_feats

    #    --------------------------    原 有 特 征 提 取 方 法
--------------------------
    def extract_time_domain_features(self, signal_data):
        mean = np.mean(signal_data)
        std = np.std(signal_data)
        var = np.var(signal_data)
        max_val = np.max(signal_data)
        min_val = np.min(signal_data)
        peak_to_peak = max_val - min_val
        rms = np.sqrt(np.mean(signal_data ** 2))

        crest_factor = max_val / rms if rms != 0 else 0
        shape_factor = rms / np.abs(mean) if mean != 0 else 0
        impulse_factor = max_val / np.abs(mean) if mean != 0 else 0

        kurt = kurtosis(signal_data)
        skewness = skew(signal_data)
```

```
        zero_crossing = np.sum(np.diff(np.sign(signal_data)) != 0) /
len(signal_data)

        return {
            '时域_均值': mean, '时域_标准差': std, '时域_方差': var,
            '时域_最大值': max_val, '时域_最小值': min_val, '时域_峰峰值
': peak_to_peak,
            '时域_均方根': rms, '时域_峰值因子': crest_factor, '时域_形
状因子': shape_factor,
            '时域_脉冲因子': impulse_factor, '时域_峭度': kurt, '时域_
偏度': skewness,
            '时域_过零率': zero_crossing
        }

    def extract_frequency_domain_features(self, signal_data):
        n = len(signal_data)
        signal_data = signal_data - np.mean(signal_data)

        yf = fft(signal_data)
        xf = fftfreq(n, 1 / self.sample_rate)[:n // 2]
        yf_abs = 2.0 / n * np.abs(yf[:n // 2])
        power_spectrum = yf_abs ** 2

        total_power = np.sum(power_spectrum)
        if total_power == 0:
            return {
                '频域_重心频率': 0, '频域_峰值频率': 0, '频域_总能量': 0,
                '频域_低频能量比': 0, '频域_高频能量比': 0
            }

        freq_mean = np.sum(xf * power_spectrum) / total_power
        peak_freq_idx = np.argmax(power_spectrum)
        peak_freq = xf[peak_freq_idx]

        nyquist = self.sample_rate / 2
```

```
        low_mask = xf < nyquist * 0.5
        high_mask = xf >= nyquist * 0.5

        low_power = np.sum(power_spectrum[low_mask])
        high_power = np.sum(power_spectrum[high_mask])

        return {
            '频域_重心频率': freq_mean, '频域_峰值频率': peak_freq,
            '频域_总能量': total_power,
            '频域_低频能量比': low_power / total_power,
            '频域_高频能量比': high_power / total_power
        }

    def extract_time_frequency_features(self, signal_data):
        try:
            coeffs = pywt.wavedec(signal_data, 'db4', level=4)
        except:
            return {
                '时频_低频能量': 0, '时频_高频能量': 0,
                '时频_能量比': 0, '时频_小波熵': 0
            }

        low_freq_energy = np.sum(coeffs[0] ** 2)
        high_freq_energy = sum(np.sum(c ** 2) for c in coeffs[1:])
        total_energy = low_freq_energy + high_freq_energy

        energy_ratio = low_freq_energy / total_energy if
total_energy != 0 else 0
        p = [np.sum(c ** 2) / total_energy for c in coeffs if
total_energy != 0]
        wavelet_entropy = -np.sum([pi * np.log2(pi) for pi in p if pi >
0])

        return {
            '时频_低频能量': low_freq_energy,
            '时频_高频能量': high_freq_energy,
```

```
            '时频_能量比': energy_ratio,
            '时频_小波熵': wavelet_entropy
        }

    def extract_local_extremum_features(self, signal_data):
        """提取局部极值点完整特征：数量、幅度、间隔三大维度"""
        n = len(signal_data)
        sample_period = 1 / self.sample_rate  # 采样周期（秒）
        signal_duration = n * sample_period  # 信号总时长（秒）
        if n < 50:  # 信号过短，返回 NaN
            return {k: np.nan for k in [
                # 极大值特征
                '极值_极大值数量', '极值_极大值密度(个/秒)', '极值_极大值占空比',
                '极值_极大值平均幅度', '极值_极大值最大幅度', '极值_极大值最小幅度', '极值_极大值中位数幅度',
                '极值_极大值幅度标准差',
                '极值_极大值平均间隔(秒)', '极值_极大值最大间隔(秒)', '极值_极大值最小间隔(秒)',
                '极值_极大值间隔中位数(秒)', '极值_极大值间隔标准差(秒)',
                # 极小值特征
                '极值_极小值数量', '极值_极小值密度(个/秒)', '极值_极小值占空比',
                '极值_极小值平均幅度', '极值_极小值最大幅度', '极值_极小值最小幅度', '极值_极小值中位数幅度',
```

```
            '极值_极小值幅度标准差',
            '极值_极小值平均间隔(秒)', '极值_极小值最大间隔(秒)', '极值_极小值最小间隔(秒)',
            '极值_极小值间隔中位数(秒)', '极值_极小值间隔标准差(秒)'
        ]}

    # 检测局部极大值/极小值（高度阈值：0.1 倍标准差，过滤噪声）
    threshold = 0.1 * np.std(signal_data)
    max_peaks, max_props = find_peaks(signal_data, height=threshold)
    min_peaks, min_props = find_peaks(-signal_data, height=threshold)
    max_amplitudes = max_props['peak_heights'] if len(max_peaks) > 0 else []
    min_amplitudes = -min_props['peak_heights'] if len(min_peaks) > 0 else []

    # 极大值特征计算
    max_count = len(max_peaks)
    max_density = max_count / signal_duration
    max_duty_cycle = (max_count * sample_period) / signal_duration
    max_mean_amp = np.mean(max_amplitudes) if max_count > 0 else np.nan
    max_max_amp = np.max(max_amplitudes) if max_count > 0 else np.nan
    max_min_amp = np.min(max_amplitudes) if max_count > 0 else np.nan
    max_median_amp = np.median(max_amplitudes) if max_count > 0 else np.nan
    max_std_amp = np.std(max_amplitudes) if max_count > 0 else np.nan
    max_intervals = np.diff(max_peaks) * sample_period if max_count > 1 else []
    max_mean_int = np.mean(max_intervals) if len(max_intervals) > 0 else np.nan
    max_max_int = np.max(max_intervals) if len(max_intervals) > 0 else np.nan
```

```
        max_min_int = np.min(max_intervals) if len(max_intervals) >
0 else np.nan
        max_median_int      =      np.median(max_intervals)      if
len(max_intervals) > 0 else np.nan
        max_std_int = np.std(max_intervals) if len(max_intervals) >
0 else np.nan


        # 极小值特征计算
        min_count = len(min_peaks)
        min_density = min_count / signal_duration
        min_duty_cycle    =    (min_count    *    sample_period)    /
signal_duration
        min_mean_amp = np.mean(min_amplitudes) if min_count > 0 else
np.nan
        min_max_amp = np.max(min_amplitudes) if min_count > 0 else
np.nan
        min_min_amp = np.min(min_amplitudes) if min_count > 0 else
np.nan
        min_median_amp = np.median(min_amplitudes) if min_count > 0
else np.nan
        min_std_amp = np.std(min_amplitudes) if min_count > 0 else
np.nan
        min_intervals  =  np.diff(min_peaks)  *  sample_period  if
min_count > 1 else []
        min_mean_int       =       np.mean(min_intervals)       if
len(min_intervals) > 0 else np.nan
        min_max_int = np.max(min_intervals) if len(min_intervals) >
0 else np.nan
        min_min_int = np.min(min_intervals) if len(min_intervals) >
0 else np.nan
        min_median_int      =      np.median(min_intervals)      if
len(min_intervals) > 0 else np.nan
        min_std_int = np.std(min_intervals) if len(min_intervals) >
0 else np.nan


        # 整合所有极值点特征
        return {
            # 极大值特征
            '极值_极大值数量': max_count,
            '极值_极大值密度(个/秒)': max_density,
```

```
        '极值_极大值占空比': max_duty_cycle,
        '极值_极大值平均幅度': max_mean_amp,
        '极值_极大值最大幅度': max_max_amp,
        '极值_极大值最小幅度': max_min_amp,
        '极值_极大值中位数幅度': max_median_amp,
        '极值_极大值幅度标准差': max_std_amp,
        '极值_极大值平均间隔(秒)': max_mean_int,
        '极值_极大值最大间隔(秒)': max_max_int,
        '极值_极大值最小间隔(秒)': max_min_int,
        '极值_极大值间隔中位数(秒)': max_median_int,
        '极值_极大值间隔标准差(秒)': max_std_int,
        # 极小值特征
        '极值_极小值数量': min_count,
        '极值_极小值密度(个/秒)': min_density,
        '极值_极小值占空比': min_duty_cycle,
        '极值_极小值平均幅度': min_mean_amp,
        '极值_极小值最大幅度': min_max_amp,
        '极值_极小值最小幅度': min_min_amp,
        '极值_极小值中位数幅度': min_median_amp,
        '极值_极小值幅度标准差': min_std_amp,
        '极值_极小值平均间隔(秒)': min_mean_int,
        '极值_极小值最大间隔(秒)': min_max_int,
```

```
            '极值_极小值最小间隔(秒)': min_min_int,

            '极值_极小值间隔中位数(秒)': min_median_int,

            '极值_极小值间隔标准差(秒)': min_std_int
        }


    #  ------------------------  批 次 处 理 整 合 所 有 特 征
-------------------------
    def process_batch(self, df_batch):
        batch_features = []

        for col in tqdm(df_batch.columns, desc="处理批次数据"):
            signal_data = df_batch[col].dropna().values
            if len(signal_data) > self.max_signal_length:
                signal_data = signal_data[:self.max_signal_length]
            if len(signal_data) < 50:
                continue


            # 提取所有特征（包含新增的 LMP 特征）
            time_feats                                          =
self.extract_time_domain_features(signal_data)
            freq_feats                                          =
self.extract_frequency_domain_features(signal_data)
            tf_feats                                            =
self.extract_time_frequency_features(signal_data)
            extremum_feats                                      =
self.extract_local_extremum_features(signal_data)
            lmp_feats = self.extract_lmp_features(signal_data)  # 新
增 LMP 特征


            # 合并所有特征
            all_feats = {
                '波形组名称': col,
                **time_feats, **freq_feats,
                **tf_feats, **extremum_feats,
                **lmp_feats
```

```
            }
            batch_features.append(all_feats)

        return pd.DataFrame(batch_features)


    # ------------------------- 核心流程（数据处理/分析/保存）
-------------------------
    def process_all_data(self, file_path, skip_columns=None):
        skip_columns = skip_columns or []
        all_features = []
        start_col = 0

        if self.data_columns is None:
            df_head = pd.read_csv(file_path, nrows=0)
            self.data_columns = df_head.columns.tolist()

        self.data_columns = [col for col in self.data_columns if col
not in skip_columns]
        total_cols = len(self.data_columns)
        print(f"总共有 {total_cols} 个波形组需要处理")


        while start_col < total_cols:
            print(f"\n处理批次: 从第 {start_col} 列开始")
            df_batch, next_start = self.load_data_batch(
                file_path,
                start_col=start_col,
                num_cols=self.batch_size
            )

            if df_batch is None or len(df_batch.columns) == 0:
                break

            batch_features = self.process_batch(df_batch)
            all_features.append(batch_features)

            del df_batch
            gc.collect()

            start_col = next_start
            print(f"已完成 {min(start_col, total_cols)} / {total_cols}
```

```
列")

        if all_features:
            self.features = pd.concat(all_features,
ignore_index=True)
            print(f"\n特征提取完成，共提取 {len(self.features)} 个波形组，
{len(self.features.columns) - 1} 个特征")
            return True
        else:
            print("没有提取到任何特征")
            return False

    def decision_tree_feature_importance(self, n_estimators=100,
random_state=42):
        if self.features is None or len(self.features) < 10:
            print("特征数据不足，无法进行决策树分析")
            return None

        X_temp = self.features.drop('波形组名称', axis=1)
        X = X_temp.fillna(X_temp.mean())

        kmeans = KMeans(n_clusters=min(10, len(X)),
random_state=random_state)
        y = kmeans.fit_predict(X)

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.3, random_state=random_state
        )

        rf = RandomForestClassifier(n_estimators=n_estimators,
random_state=random_state, n_jobs=-1)
        rf.fit(X_train, y_train)

        feature_importance = pd.DataFrame({
            '特征名称': X.columns,
            '重要性': rf.feature_importances_
```

```
        }).sort_values(by='重要性', ascending=False)

        plt.figure(figsize=(14, 10))
        sns.barplot(x='重要性', y='特征名称',
data=feature_importance.head(20))
        plt.title('随机森林特征重要性（Top 20，含 LMP 特征）', fontsize=16)
        plt.tight_layout()
        plt.savefig('feature_importance.png',              dpi=300,
bbox_inches='tight')
        plt.close()

        print("特征重要性分析完成，结果已保存为
'feature_importance.png'")
        return feature_importance

    def                              plot_feature_correlation(self,
output_path='feature_correlation.png'):
        if self.features is None:
            print("请先提取特征")
            return

        numeric_temp                                                =
self.features.select_dtypes(include=[np.number])
        numeric_features = numeric_temp.fillna(numeric_temp.mean())
        self.correlation_matrix = numeric_features.corr()

        plt.figure(figsize=(24, 22))
        mask     =     np.triu(np.ones_like(self.correlation_matrix,
dtype=bool))
        cmap = sns.diverging_palette(220, 20, as_cmap=True)

        sns.heatmap(self.correlation_matrix, mask=mask, cmap=cmap,
vmax=1.0, vmin=-1.0,
                    center=0,       square=True,       linewidths=.5,
cbar_kws={"shrink": .8},
                    annot=True,
                    annot_kws={'size': 4},
                    fmt=".2f")
```

```
        plt.title('特征相关性热力图（含 LMP 特征）', fontsize=22)
        plt.tight_layout()
        plt.savefig(output_path, dpi=300, bbox_inches='tight')
        plt.close()
        print(f"特征相关性热力图已保存至: {output_path}")

        high_corr_pairs = []
        for i in range(len(self.correlation_matrix.columns)):
            for j in range(i):
                if abs(self.correlation_matrix.iloc[i, j]) > 0.85:
                    high_corr_pairs.append({
                        '特征 1': self.correlation_matrix.columns[i],
                        '特征 2': self.correlation_matrix.columns[j],
                        '相关系数': self.correlation_matrix.iloc[i, j]
                    })

        if high_corr_pairs:
            high_corr_df =
pd.DataFrame(high_corr_pairs).sort_values(by='相关系数', key=abs,
ascending=False)
            high_corr_df.to_csv('high_correlation_pairs.csv',
index=False, encoding='utf-8-sig')
            print("高度相关的特征对（含 LMP 特征）已保存至
'high_correlation_pairs.csv'")

    def waveform_similarity_analysis(self, top_n=50,
output_path='similarity_analysis.png'):
        if self.features is None or len(self.features) < 5:
            print("特征数据不足，无法进行相似性分析")
            return

        X_temp = self.features.drop('波形组名称', axis=1)
        X = X_temp.fillna(X_temp.mean())
        X_scaled = StandardScaler().fit_transform(X)
```

```
        def cosine_similarity_matrix(matrix):
            norm = np.linalg.norm(matrix, axis=1)
            norm_matrix = matrix / norm[:, np.newaxis]
            return np.dot(norm_matrix, norm_matrix.T)

        analyze_n = min(top_n, len(X_scaled))
        similarity_matrix =
cosine_similarity_matrix(X_scaled[:analyze_n])

        plt.figure(figsize=(12, 10))
        sns.heatmap(similarity_matrix, cmap="YlOrRd", vmin=0, vmax=1)
        plt.title(f'波形特征余弦相似度矩阵（前{analyze_n}个波形，含 LMP 特
征）', fontsize=15)
        plt.tight_layout()
        plt.savefig(output_path, dpi=300, bbox_inches='tight')
        plt.close()

        similar_pairs = []
        for i in range(analyze_n):
            for j in range(i + 1, analyze_n):
                similar_pairs.append({
                    '波形 1': self.features.iloc[i]['波形组名称'],
                    '波形 2': self.features.iloc[j]['波形组名称'],
                    '相似度': similarity_matrix[i, j]
                })

        similar_pairs_df =
pd.DataFrame(similar_pairs).sort_values(by='相似度',
ascending=False)

similar_pairs_df.head(20).to_csv('top_similar_waveforms.csv',
index=False, encoding='utf-8-sig')

        print(f"波形相似度矩阵已保存至: {output_path}")
        print("最相似的 20 对波形已保存至 'top_similar_waveforms.csv'")
        return similar_pairs_df
```

```
    def save_features(self, output_path='waveform_features_with_lmp.csv'):
        if self.features is None:
            print("请先提取特征")
            return

        dir_path = os.path.dirname(output_path)
        if dir_path:
            os.makedirs(dir_path, exist_ok=True)

        chunk_size = 8000
        for i in range(0, len(self.features), chunk_size):
            chunk = self.features[i:i + chunk_size]
            mode = 'w' if i == 0 else 'a'
            header = i == 0
            chunk.to_csv(output_path, index=False, encoding='utf-8-sig', mode=mode, header=header)

        print(f"特征已保存至: {os.path.abspath(output_path)}")
        print(
            f"特征包含: 时域(13) + 频域(5) + 时频域(4) + 局部极值点(26) + LMP({1 + self.max_pf * 13}) = 共{len(self.features.columns) - 1}个特征")


def main():
    # 配置参数
    input_file = "D:/BaiduNetdiskDownload/中文赛题/题/数据集 py/源域数据集/DE/de 代码/resource_result.csv"
    output_feature_file = "D:/BaiduNetdiskDownload/中文赛题/题/数据集 py/源域数据集/DE/de 代码/resource_tree_features_with_lmp.csv"
    sample_rate = 1000  # 根据实际波形采样率调整
```

```
    max_signal_length = 10000  # 长信号截断长度（根据内存调整）

    batch_size = 30  # LMP 计算量大，建议减小批处理规模

    max_pf = 3  # 保留的 PF 分量数量（建议 3-5，数量越多计算越慢）
    skip_columns = []

    analyzer = MassiveWaveformAnalyzer(
        sample_rate=sample_rate,
        max_signal_length=max_signal_length,
        batch_size=batch_size,
        max_pf=max_pf
    )

    if not analyzer.process_all_data(input_file,
skip_columns=skip_columns):
        return

    analyzer.save_features(output_feature_file)

    # 特征重要性分析（含 LMP 特征）
    feature_importance =
analyzer.decision_tree_feature_importance()
    if feature_importance is not None:
        print("\n 最重要的 8 个特征（含 LMP 特征）:")
        print(feature_importance.head(8))

    # 相关性热力图（含 LMP 特征）
    analyzer.plot_feature_correlation()

    # 波形相似性分析（含 LMP 特征）
    analyzer.waveform_similarity_analysis()

    print(f"\n 所有分析完成！已加入 LMP 特征（保留前{max_pf}个 PF 分量）")


if __name__ == "__main__":
    required_packages = ['numpy', 'pandas', 'scipy', 'matplotlib',
```

```
'seaborn',
                    'pywavelets', 'scikit-learn', 'tqdm']

    try:
        for pkg in required_packages:
            __import__(pkg)
    except ImportError as e:
        print(f"缺少必要的库，正在安装...")
        import subprocess
        import sys

        subprocess.check_call([sys.executable,     "-m",     "pip",
"install"] + required_packages)

    main()
```

8.2 问题二部分代码

训练模型与评估：

```
def train_and_evaluate(self, X, y):
    X_train_resampled,    X_val,    y_train_resampled,    y_val    =
self.enhance_data(X, y)

    X_train_scaled = self.scaler.fit_transform(X_train_resampled)
    X_val_scaled = self.scaler.transform(X_val)

    # 优化基础模型超参数
    self.optimize_model_hyperparams(X_train_scaled,
y_train_resampled)

    # 对 XGBoost 添加早停机制
    eval_set = [(X_val_scaled, y_val)]

    # 使用未拟合时的 estimators 属性（不带下划线）
    estimator_dict = {name: clf for name, clf in self.model.estimators}

    # 对 XGBoost 添加早停机制（使用 early_stopping_rounds）
    estimator_dict['xgb'].fit(
        X_train_scaled, y_train_resampled,
```

```
        eval_set=eval_set,
        early_stopping_rounds=20,
        verbose=False
    )

    # 训练完整集成模型

    print("\n 训练集成模型...")
    self.model.fit(X_train_scaled, y_train_resampled)

    # 预测与评估
    y_pred = self.model.predict(X_val_scaled)
    y_pred_proba = self.model.predict_proba(X_val_scaled)

    overall_acc = accuracy_score(y_val, y_pred)
    overall_f1 = f1_score(y_val, y_pred, average='macro')
    print(f"\n=== 整体预测精度评估 ===")

    print(f"整体准确率: {overall_acc:.4f}")

    print(f"整体宏平均 F1: {overall_f1:.4f}")

    # 详细分类报告

    print("\n=== 各类别详细分类报告 ===")
    target_names = list(self.inv_label_mapping.values())
    print(classification_report(
        y_val, y_pred,
        target_names=target_names,
        zero_division=0
    ))

    # 绘制混淆矩阵

    plt.figure(figsize=(10, 8))
    cm = confusion_matrix(y_val, y_pred,
labels=list(self.label_mapping.values()))
    cm_percent = cm / (cm.sum(axis=1, keepdims=True) + 1e-10) * 100
    sns.heatmap(
        cm, annot=True, fmt="d", cmap="Blues",
```

```
        xticklabels=target_names, yticklabels=target_names
    )
    for i in range(len(cm)):
        for j in range(len(cm)):
            plt.text(j + 0.5, i + 0.7, f'{cm_percent[i, j]:.1f}%',
                     ha='center',     va='center',     color='black',
fontsize=9)

    plt.xlabel('预测标签')

    plt.ylabel('真实标签')

    plt.title(f'混淆矩阵（准确率: {overall_acc:.4f}）')
    plt.savefig('confusion_matrix.png', dpi=300)
    plt.close()


    # 误分类分析
    if self.enable_misanalysis and 1 in self.label_mapping.values()
and 2 in self.label_mapping.values():
        self.analyze_misclassified_samples(X_val, y_val, y_pred, 1,
2, ['IR', 'OR'])

    return y_val, y_pred, y_pred_proba, overall_acc, overall_f1
```

8.3 问题三部分代码

软投票模型的建立与评估：

```
def build_soft_voting_model(self, X_train, y_train):
    """构建软投票模型"""
    model1 = RandomForestClassifier(
        n_estimators=300, max_depth=None, min_samples_split=5,
        class_weight='balanced',    random_state=self.random_state,
n_jobs=-1
    )
    model2 = RandomForestClassifier(
        n_estimators=400, max_depth=None, min_samples_split=3,
        class_weight='balanced_subsample',
random_state=self.random_state + 1, n_jobs=-1
    )
    model3 = RandomForestClassifier(
        n_estimators=350, max_depth=None, min_samples_leaf=3,
        class_weight='balanced', random_state=self.random_state + 2,
```

```
n_jobs=-1
    )
    self.base_models = [('rf1', model1), ('rf2', model2), ('rf3',
model3)]
    self.voting_model                                                =
VotingClassifier(estimators=self.base_models,      voting='soft',
weights=[1.0, 1.2, 0.9])
    self.voting_model.fit(X_train, y_train)
    return self.voting_model

def fine_tune_model(self, source_X_aligned):
    """微调软投票模型"""
    X_train, X_val, y_train, y_val = train_test_split(
        source_X_aligned,       self.source_y,       test_size=0.1,
random_state=self.random_state
    )
    self.build_soft_voting_model(X_train, y_train)
    val_pred = self.voting_model.predict(X_val)
    val_acc = accuracy_score(y_val, val_pred)
    print(f"源域验证集软投票模型准确率: {val_acc:.4f}")
    with                          open(os.path.join(self.output_dir,
"voting_transfer_model.pkl"), 'wb') as f:
        pickle.dump(self.voting_model, f)
    print("软投票迁移模型已保存")

def evaluate_unsupervised(self, target_X_scaled):
    """评估迁移效果（确保维度一致）"""
    # 源域特征：先过滤，再用主 scaler 转换
    source_X_filtered          =          np.delete(self.source_X,
self.high_discrim_indices, axis=1)
    source_scaled = self.scaler.transform(source_X_filtered)

    # 维度强制校验
    assert source_scaled.shape[1] == target_X_scaled.shape[1], \
        f" 源 域 处 理 后 维 度 {source_scaled.shape[1]} 与 目 标 域
{target_X_scaled.shape[1]}不匹配！"
```

```
    # 1. 域混淆度评估
    source_len = len(source_scaled)
    target_len = len(target_X_scaled)
    domain_labels = np.hstack([np.zeros(source_len),
np.ones(target_len)])
    X_domain = np.vstack([source_scaled, target_X_scaled])

    domain_clf = LogisticRegression(C=0.1, max_iter=1000,
random_state=self.random_state)
    domain_clf.fit(X_domain, domain_labels)
    domain_pred = domain_clf.predict(X_domain)
    domain_acc = accuracy_score(domain_labels, domain_pred)

    print(f"域分类器准确率: {domain_acc:.4f}")

    print(f"域混淆度: {(1 - domain_acc):.4f}（越接近 0.5 越好）")


    # 2. 目标域预测分布
    target_pred = self.voting_model.predict(target_X_scaled)
    target_pred_raw =
self.label_encoder.inverse_transform(target_pred)
    pred_dist =
pd.Series(target_pred_raw).value_counts(normalize=True).sort_ind
ex()

    print("\n 目标域样本预测类别分布（原始标签）: ")
    print(pred_dist)

    plt.figure(figsize=(8, 4))
    pred_dist.plot(kind="bar")
    plt.title("目标域样本预测类别分布")

    plt.xlabel("类别（原始标签）")

    plt.ylabel("比例")
    plt.savefig(os.path.join(self.output_dir,
"target_pred_distribution.png"), dpi=300)
    plt.close()


    # 3. 特征分布重叠度
```

```
    overlap_score = 1 - self.calculate_js_divergence(source_scaled,
target_X_scaled)
    print(f"特征分布重叠度（1-JS 散度）: {overlap_score:.4f}（越接近 1 越
好）")
```

8.4 问题四部分代码

迁移学习模型在目标域上的应用：

```
import pandas as pd
import numpy as np
import pickle
import os
from sklearn.preprocessing import StandardScaler, LabelEncoder


class TransferPredictor:
    def __init__(self, model_path, scaler_path, indices_path,
label_encoder_path, output_dir="prediction_results"):
        self.model_path = model_path
        self.scaler_path = scaler_path
        self.indices_path = indices_path
        self.label_encoder_path = label_encoder_path
        self.output_dir = output_dir
        os.makedirs(output_dir, exist_ok=True)


        # 加载必要组件
        self.model = self.load_model()
        self.scaler = self.load_scaler()
        self.high_discrim_indices = self.load_high_discrim_indices()
        self.label_encoder = self.load_label_encoder()

    def load_model(self):
        """加载软投票迁移模型"""
        try:
            with open(self.model_path, 'rb') as f:
                return pickle.load(f)
        except Exception as e:
            raise ValueError(f"模型加载失败: {e}")
```

```
    def load_scaler(self):
        """加载标准化器"""
        try:
            with open(self.scaler_path, 'rb') as f:
                return pickle.load(f)
        except Exception as e:
            raise ValueError(f"标准化器加载失败: {e}")

    def load_high_discrim_indices(self):
        """加载高区分度特征索引"""
        try:
            with open(self.indices_path, 'rb') as f:
                return pickle.load(f)
        except Exception as e:
            raise ValueError(f"特征索引加载失败: {e}")

    def load_label_encoder(self):
        """加载标签编码器"""
        try:
            with open(self.label_encoder_path, 'rb') as f:
                return pickle.load(f)
        except Exception as e:
            raise ValueError(f"标签编码器加载失败: {e}")

    def preprocess_target_data(self, target_data):
        """对目标域数据进行预处理（与训练时保持一致）"""
        # 1. 移除高区分度特征
        if self.high_discrim_indices:
            target_filtered = np.delete(target_data,
self.high_discrim_indices, axis=1)
        else:
            target_filtered = target_data

        # 2. 标准化
        target_scaled = self.scaler.transform(target_filtered)
```

```
        return target_scaled

    def predict(self, target_path):
        """预测目标域数据并保存结果"""
        # 加载目标域数据
        target_df = pd.read_csv(target_path)
        target_data = target_df.values
        print(f"加载目标域数据：{target_data.shape[0]}样本，
{target_data.shape[1]}特征")

        # 预处理
        target_processed = self.preprocess_target_data(target_data)

        # 预测
        predictions = self.model.predict(target_processed)
        # 转换为原始标签
        pred_labels                                          =
self.label_encoder.inverse_transform(predictions)

        # 保存结果
        result_df = pd.DataFrame({
            "样本索引": range(len(pred_labels)),
            "预测类别": pred_labels
        })
        result_path          =          os.path.join(self.output_dir,
"target_predictions.csv")
        result_df.to_csv(result_path,                  index=False,
encoding="utf-8-sig")
        print(f"预测结果已保存至：{result_path}")

        # 打印预测分布
        pred_dist                                                =
```

```
pd.Series(pred_labels).value_counts(normalize=True).sort_index()

        print("\n目标域预测类别分布:")
        print(pred_dist)

        return result_df


if __name__ == "__main__":
    # 需要先在训练代码中保存以下文件
    MODEL_PATH = "transfer_results/voting_transfer_model.pkl"
    SCALER_PATH = "transfer_results/scaler.pkl"  # 需在训练时保存
    INDICES_PATH = "transfer_results/high_discrim_indices.pkl"  # 需在训练时保存
    LABEL_ENCODER_PATH = "transfer_results/label_encoder.pkl"  # 需在训练时保存
    TARGET_DATA_PATH = "D:/BaiduNetdiskDownload/中文赛题/题/数据集py/迁移学习/无标签features_target.csv"  # 目标域数据路径


    # 初始化预测器并预测
    predictor = TransferPredictor(
        model_path=MODEL_PATH,
        scaler_path=SCALER_PATH,
        indices_path=INDICES_PATH,
        label_encoder_path=LABEL_ENCODER_PATH
    )
    predictor.predict(TARGET_DATA_PATH)
```