宜昌地区就业状态分析与预测及人岗匹配模型研究

摘　要

本文聚焦宜昌地区就业状态，基于 5000 名被调查者数据及宏观外部信息，通过构建多个模型分析就业现状、预测趋势并实现人岗匹配，为区域就业政策提供支撑。

针对问题 1，清洗数据，剔除无关冗余特征、处理缺失值并筛选低方差变量。基于就业时间等的时序逻辑关系，初步得出部分标记样本。针对未标记就业状态样本，采用**标签传播算法**利用已知样本特征模式预测，最终确定就业 **4341** 人、失业 **639** 人。接着，通过**卡方检验**和**描述性统计分析**各特征与就业状态的关系。结果显示，户口所在地、所学专业、毕业日期等与失业显著相关，农村户口、特定行业及年轻群体失业风险较高。

针对问题 2，由于就业状态为不平衡数据，故本文在传统 SVM 模型的基础上引入 LINEX 损失函数，构建了 **LINEX-SVM** 模型，同时对比 SVM 模型及重采样模型进行加权对数损失以及准确率、精确率、召回率、F1 值和 AUC 值等指标的评估对比，由表 5-5 结果显示所提出模型性能最佳，并将其应用于“预测集”的就业状态预测。此外，基于 **XGBoost** 可解释模型 **SHAP** 进行特征重要性排序与可视化分析，并综合 F 分数和 SHAP 值结果，**毕业日期、户口所在地**为核心影响特征。

针对问题 3，对收集到的外部数据集 1 利用线性插值法填补缺失值，并通过**指数加权移动平均（EWMA）**构建动态特征，并将其与个体数据进行整合。其次，在自步学习（SPL）模型的基础上引入一正则化项得到**多模块自步学习（MSPL）**，最终将各模块分类器的结果基于侧重少数类标签（0）集成。由表 5-8 中的 MSPL 以及各模块的预测对比结果显示，MSPL 对失业标签识别准确率达 **0.75**，综合准确率 **0.9323**。最后，对“预测集”就业状态进行预测。

针对问题 4，对收集到的外部数据集 2 进行异常值和缺失值的预处理，接着将其与已有数据进行特征对齐，提取出**年龄、教育程度、毕业年份、期望工作、期望薪资**五个输入特征，**实际工作和实际薪资**两个输出特征，并将其序列化处理以构建 **Transformer 端到端**的人岗匹配模型。结果显示，BLEU 评分为 0.636，人岗匹配度为 0.86，并对原始数据中的失业人员进行岗位匹配。

关键词：标签传播算法；卡方检验；LINEX-SVM；XGBoost；SHAP；多模块自步学习（MSPL）；Transformer

目 录

一、问题重述

1.1 研究背景

就业是最基本的民生，关乎人民群众切身利益。我国就业形势总体保持稳定，城镇调查失业率基本可控。但同时也面临着结构性矛盾、摩擦性失业等挑战。党的十八大以来，以习近平同志为核心的党中央在实践中不断深化对新时代就业工作规律的认识，促进高质量充分就业，是新时代新征程就业工作的新定位、新使命[1]。

宜昌作为湖北省的重要城市，统筹发展所需、产业所急、群众所盼，积极构建覆盖全民、贯穿全程、辐射全域、便捷高效的全方位公共就业服务体系[2]。但宜昌市就业同样面临挑战，传统产业在转型升级过程中，部分企业面临环保、技术改造等压力，可能会出现人员调整。新兴产业虽然发展迅速，但在人才吸引和培养方面还需要进一步加强，以满足产业发展对专业人才的需求。

1.2 研究问题

基于赛题提供的宜昌地区 5000 名被调查者的脱敏数据，需建立数学模型解决以下问题：

问题一：依据被调查者当前的就业状态，分析宜昌地区当前就业整体情况，给出就业状态数量统计表，呈现不同就业状态的具体人数分布。同时，按照年龄、性别、学历、专业、行业等多维度特征对人员进行分类，深入分析各层面因素对就业状态的影响，并运用图表进行可视化展示。

问题二：首先，基于问题一的分析结果，挑选与就业状态相关的特征，构建就业状态预测模型，选取准确率、查准率、召回率、F1 等关键指标对模型进行全面评估，给出模型的评估指标结果表，并利用构建的模型对附件 1 中的“预测集”进行就业状态预测，给出预测结果表。其次，还需建立专门的模型进行各特征的重要性排序，绘制特征重要性排序条形图。

问题三：收集宏观经济、政策、劳动力市场、宜昌市居民消费价格指数、招聘信息等外部数据，提取对就业状态有显著影响的因素，同时结合问题一数据，进一步完善就业状态预测模型。利用该模型对附件 1 中的“预测集”进行预测，并使用准确率、查准率、召回率、F1 等指标评估模型，给出评估指标结果表。

问题四：将赛题提供的数据与收集到的招聘数据、社交媒体数据、薪资水平等外部数据相结合，建立人岗匹配模型，旨在寻找求职者和岗位之间的匹配关系，并据此为赛题数据中的失业人员进行工作推荐。

二、问题分析

基于重述及附件数据，本文对问题进一步分析如下：

2.1 问题 1 分析

对整体数据进行分析可知原始数据存在不规范性，因此首先需要进行数据清洗，包括移除无关和冗余特征，对特殊特征值进行编码，检验并处理缺失值和异常值，以及筛选并剔除低方差特征。随后，基于就业时间、合同终止日期和失业信息登记表登记日期之间的逻辑关系，统计附件 1 的数据集中就业、失业和未标

记就业状态的样本数量。针对未标记就业状态的样本，采用标签传播算法，利用已知就业（1）和失业（0）记录的特征模式进行预测，并汇总生成就业状态数量统计表。进一步，为了探究各类特征对就业状态的影响，采用卡方检验和描述性统计分析方法，深入解读性别、年龄、教育程度、专业和行业代码等特征，分析各变量与就业状态之间的相关关系和影响程度。问题 1 的思路流程图如下。

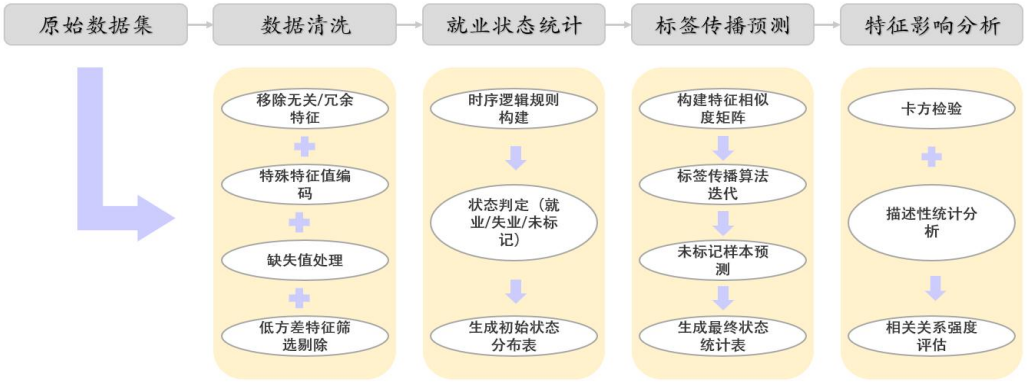

图 2-1 问题 1 的思路流程图

2.2 问题 2 分析

基于问题 1 的分析结果，筛选出与就业状态具有相关性的数据，为后续模型构建奠定基础。针对数据集中就业状态数据的不平衡性，在传统 SVM 模型的基础上引入 LINEX 损失函数，构建 LINEX-SVM 模型，旨在更好地处理非对称误分类惩罚问题。具体而言，将附件 1 数据集按照 4:1 的比例划分为训练集和测试集，并分别构建原始 SVM 模型、SVM_{SMOTE} 模型、SVM_{RUS} 模型、SVM_{ROS} 模型、SVM_{DEC} 模型以及 LINEX-SVM 模型。采用网格搜索法进行五折交叉验证，寻找各模型的最优参数，并在此基础上进行模型拟合。为全面评估模型性能，本文采用加权对数损失以及基于混淆矩阵的准确率、精确率、召回率、F1 值和 AUC 值等指标。通过评估指标结果表，验证 LINEX-SVM 模型在处理不平衡数据时的优越性，并将其应用于数据清洗后的附件 1“预测集”的就业状态预测，给出预测结果表。进一步，利用 XGBoost 模型进行特征重要性排序，并通过 SHAP 值绘制特征 SHAP 值分布图、SHAP 特征交互热力图和 SHAP 特征重要性排序图，深入分析各特征对就业状态的影响。问题 2 的思路流程图如下。

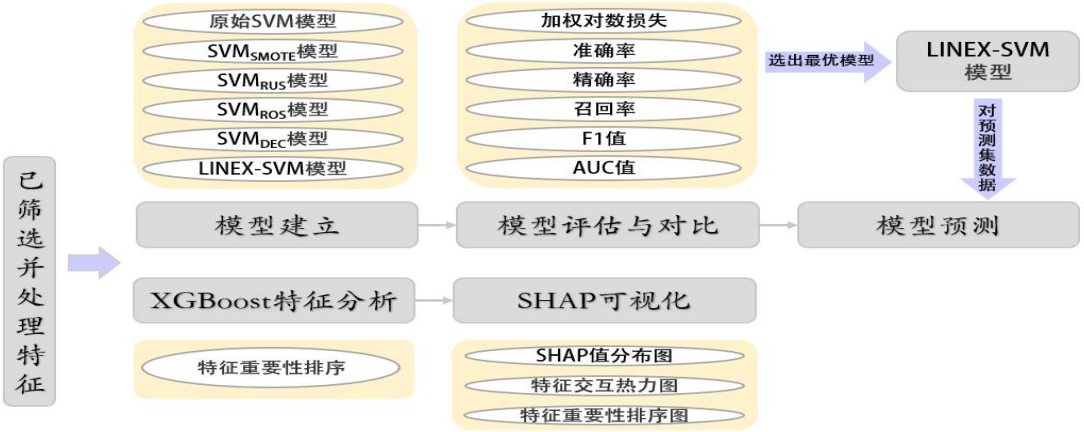

图 2-2 问题 2 的思路流程图

2.3 问题 3 分析

通过《中国城市统计年鉴》和宜昌市统计局官网，收集到经济指标、劳动力市场、薪资情况、消费水平和政府支出五大类共计 24 个指标。针对缺失值，采用线性插值法进行填补，并使用指数加权移动平均法处理上述特征。基于“毕业日期”的年份，将这 24 个特征与问题一的数据进行整合。为解决问题，考虑采用多模块融合分类学习方法。首先，对传统的自步学习（SPL）模型进行改进，提出 MSPL 算法，用于为每个模块选择最具代表性的样本。随后，每个模块基于问题二中提出的 LINEX-SVM 模型进行独立训练。最终，将各模块的分类器集成，得到总分类器。为验证 MSPL 算法的有效性，利用评估指标结果表展示 MSPL 以及各模块的预测结果，并据此预测附件 1“预测集”的就业状态，预测结果以预测结果表的形式呈现。问题 3 的思路流程图如下。

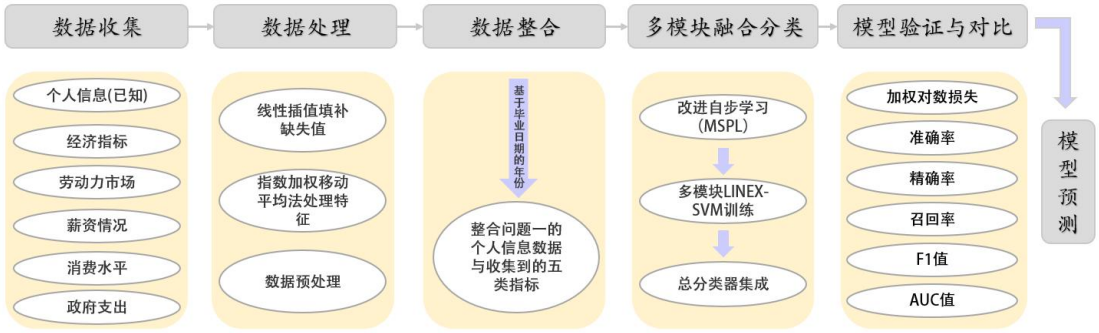

图 2-3 问题 3 的思路流程图

2.4 问题 4 分析

为了解决人岗匹配问题，首先对收集到的招聘数据、社交媒体数据、薪资水平、宏观经济数据等外部数据进行预处理，包括清洗异常值、填补缺失值。随后，将预处理后的外部数据与赛题提供的数据进行特征对齐，提取出年龄、教育程度、毕业年份、期望工作、期望薪资五个关键输入特征，以及实际工作和实际薪资两个目标输出特征。基于这些特征，将求职者信息转化为序列化数据，并构建 Transformer 模型，旨在学习特征之间的复杂关联。最终，通过 BLEU 评分和人岗匹配度等指标，全面评估模型的性能表现，然后对赛题数据中的失业人员进行工作推荐。问题 4 的思路流程图如下。

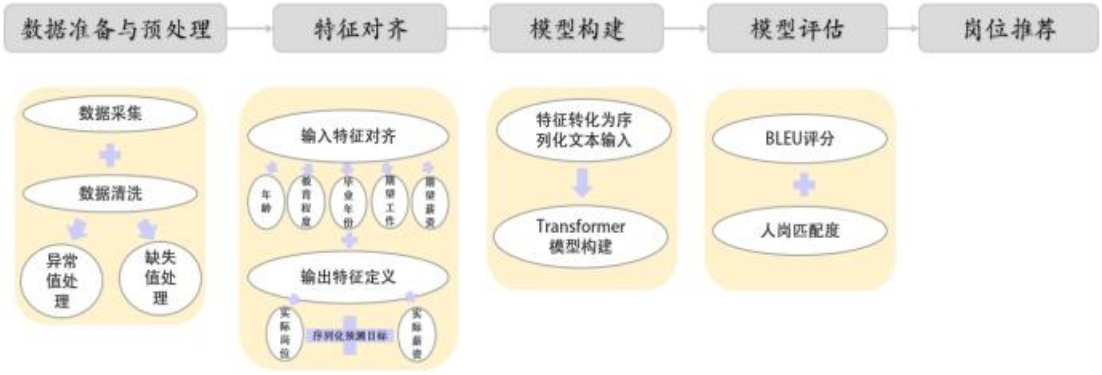

图 2-4 问题 4 的思路流程图

三、模型假设

针对以上问题，本文假设如下：

1.假设影响就业状态的因素仅包含题目所给定的特征信息以及收集到的部分特征信息；

2.低方差特征对分类任务信息贡献度可忽略；

3.假设数据清洗后的数据有效可信，能够反映就业状况与各特征的关联性。

四、符号说明

表 4-1 当前就业状态

符号	含义
x_i	样本特征向量
y_i	样本标签
W	相似度矩阵
P	概率转移矩阵
O_i	观测频数
E_i	期望频数
χ^2	卡方统计量
w	超平面的法向量
b	超平面的偏置型
ξ_i	松弛变量
$L(\theta)$	总体损失函数
n	样本数量
k	树的数量
λ	自步学习超参数
f	模型估计值
T	最大训练轮数

五、模型的建立与求解

5.1 问题 1 模型求解与建立

5.1.1 数据清洗

首先导入题目所给的数据，通过 python 读取附件 1 数据中工作名为数据集的原始数据，经统计，该数据集包含 54 个特征和 4980 条样本。

1.无关特征与冗余特征剔除

对于附件一中数据，id、人员编号、姓名、就业登记编号、录用单位编号和失业登记编号是唯一的，不具备分析价值，故将这些特征全部剔除；户籍信息和原单位名称与就业状态无显著关联，因此将这些变量删除；教育程度已涵盖文化程度，故剔除后者；生日和年龄为重复特征，户口所在地区（代码）和户口所在地区（名称）为重复特征，所学专业代码和所学专业名称是重复特征，就业时间和合同起始时间是重复特征，失业审核日期和失业信息登记表登记日期数据二者

时间非常接近，分别保留其中一个字段即可，选择年龄、户口所在地区（代码）、所学专业代码、就业时间和失业信息登记表登记日期这四个变量。

2.分类变量编码

通过附件中的字段说明以及查看字相应段的值，发现剔除后的剩余数据绝大多数均为分类数据，本文采用自定义编码的方式将其表示，替换为编码 1，2，3，4 等。具体操作为将毕业学校按照层次划分为 985、211 以及双非三种类别，毕业学校为 985 的编码为 1，毕业学校为 211 的编码为 2，毕业学校为双非的编码为 3,；录用单位根据单位性质划分为政府机关、央国企已经其他企业（私企外企），并分别编码为 1,2,3；行业代码也按照表格中所给的标准进行编码，A0000，B0000，C0000 等分别编码为 1,2,3 等以此类推，共标记为 30 个类别。

对于年龄这一连续型变量，本文通过分析得出该变量不存在异常数据，被调查者全部年龄均处于 18 岁至 65 岁之间，故本文按照 18-24 岁为大学生，25-29 岁为职场新人，30-39 岁为职场骨干，40-49 岁为中年群体，50-59 岁为准退休和 60 岁+为退休的分类方式将全部年龄进行连续变量离散化的分类处理，并分别编码为 1-6 六个数字。

对于除失业注销时间外的毕业日期、就业时间、合同终止日期等日期型数据，把它们按照年月格式进行编码划分，例如将 2008-06-3000:00:00 编码为 200806。失业注销时间按照 2023 年之后注销失业的标记为就业（1），其余均标记为未知的方式进行划分，在下一步中对这些未知数据进行编码。

3.缺失值处理

下面对所保留数据进行缺失值可视化，得到图 5-1，

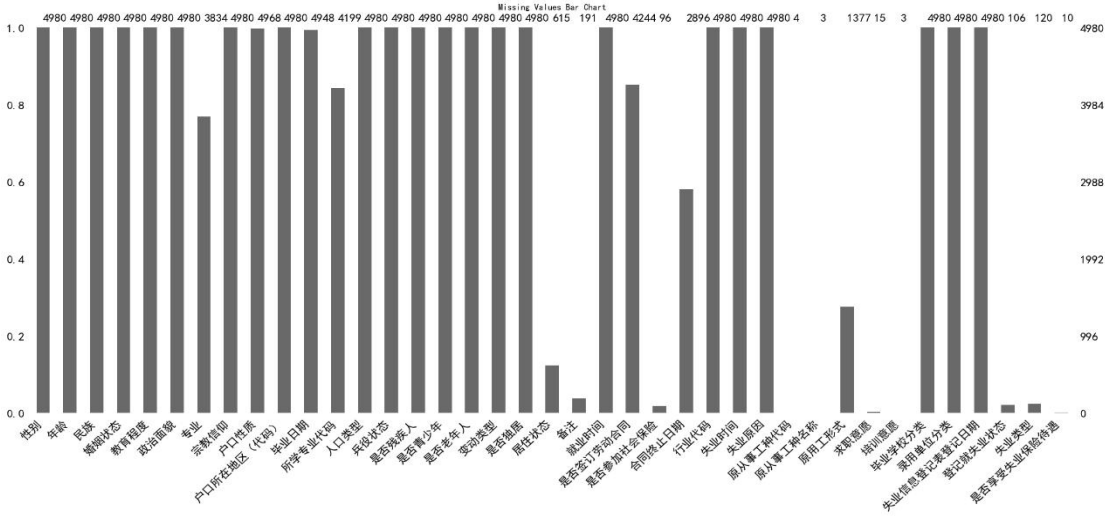

图 5-1 缺失值处理后的数据

由图 5-1 可知，部分数据缺失比例过大，这对后续结果精度有不利影响，故本文剔除掉缺失比例大于 50%的数据，即剔除掉了居住状态、备注、是否参加社会保险、原从事工种代码、原从事工种名称、原用工形式、求职意愿和培训意愿这几个特征。同时对其余缺失比例不大的分类型数据利用众数填充法进行填充。

4.低方差特征筛选与剔除

本文对缺失值处理后的数据进行了分类变量统计分析，结果显示兵役状态、是否残疾人、是否青少年、是否老年人、变动类型及是否独居这六个特征存在显著的类别不平衡问题，各类别占比超过 99%，类别可视化图如下，

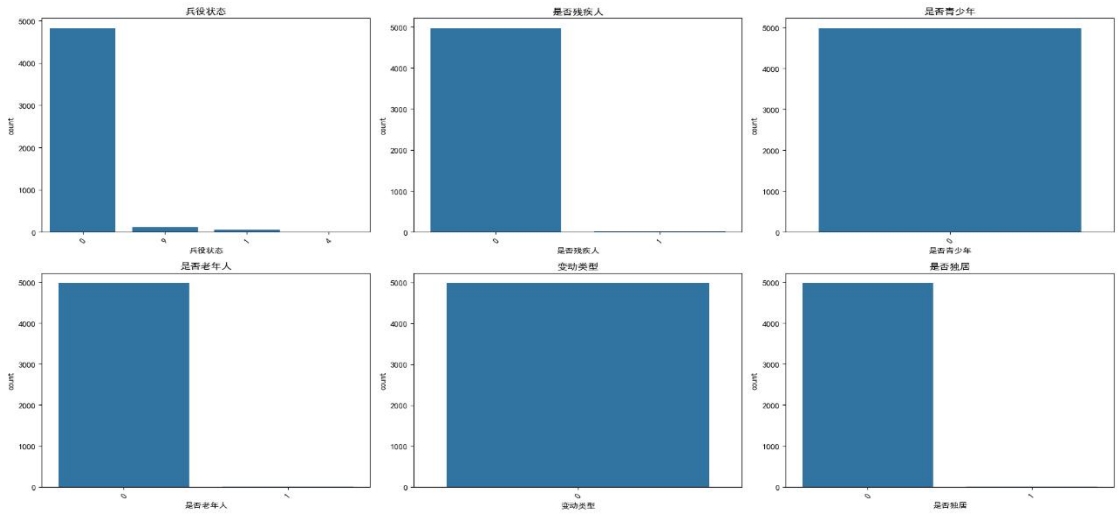

图 5-2 低方差特征

鉴于此类低方差特征对模型训练不仅缺乏区分能力，反而可能引入噪声并降低模型泛化性能，因此本研究决定在后续分析中剔除上述六个特征变量。通过上述步骤，有效提升了数据质量，为后续建模分析奠定基础。

5.1.2 标签传播算法

1.人为规则设计板块

关于就业失业状态的说明，用序号表示，就业为 1，失业为 0。通过查阅相关资料，失业注销时间：失业登记信息已经注销，意味着该失业登记已经被取消或终止，不再有效。这可能发生在失业人员已经找到工作、不再符合失业登记的条件，或者由于其他原因需要注销失业登记。故第一步当样本失业注销无缺失值时，标记其为就业状态，由图 5-3 结合表中数据可知，即共有 4036 个样本为失业注销，但考虑到数据时效性，可能有些人在 21 年找到工作注销了失业，在之后几年又面临失业，故只将 23 年注销失业的样本认为目前状态为就业（1），共计 820 个，还剩余 4160 条为未标记。

第一若就业时间等于失业信息登记表登记日期，说明在就业后没有去消除失业信息，但实际为未就业状态故标记为就业（1）；其次若合同终止日期在 2024 之后，说明数据是就业状态故标记其为就业（1）；最后若失业时间晚于合同终止日期的未标记样本，说明未在合同到期后找到工作，故可将其标记为失业（0）。基于这三条递进规则，就业样本数量:1880；失业样本数量:478；仍未标记样本数量:2622。

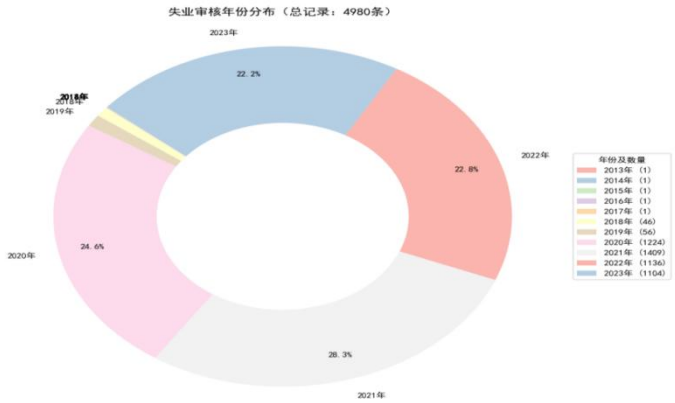

图 5-3 失业审核时间分布

2.标签传播板块

标签传播算法(Label Propagation Algorithm,LPA)是一种半监督学习方法，它利用已标记的数据和未标记的数据之间的相似性，将标签信息从已标记样本传播到未标记样本。该算法基于这样一个假设：相似的样本应该具有相似的标签[3]。

在就业状态预测项目中，我们使用标签传播算法来预测那些未标记的样本（即失业状态未知的记录）。通过已知的就业(1)和失业(0)记录的特征模式，算法可以学习出未标记样本的标签。

（1）算法步骤

①构建相似度图，这里我们利用 KNN 近似，仅保留与样本最近的 K 个邻居的连接，其他权重为 0，会得到相似度矩阵 W[13]；

②构建概率转移矩阵，将相似度矩阵 W 转换为概率转移矩阵 P，表示样本 i 跳转到 j 的概率；

$$P_{i,j} = \frac{W_{ij}}{\sum_{k=1}^{n} W_{ik}} \tag{5.1}$$

③标签矩阵初试化

$$Y = \begin{bmatrix} 1 & \cdots & 0(\text{已标记就业}) \\ \vdots & \ddots & \vdots \\ 0.5 & \cdots & 0(\text{为标记，初试概率均匀}) \end{bmatrix} \tag{5.2}$$

（2）算法结果

标签传播算法可以被视为一个随机游走过程。假设在图中随机游走，每次从当前节点随机选择一个边连接的节点跳转，跳转概率由边的权重决定。这个过程中，已标记节点的标签是固定的，而未标记节点的标签会不断更新，直到达到稳态[15]。

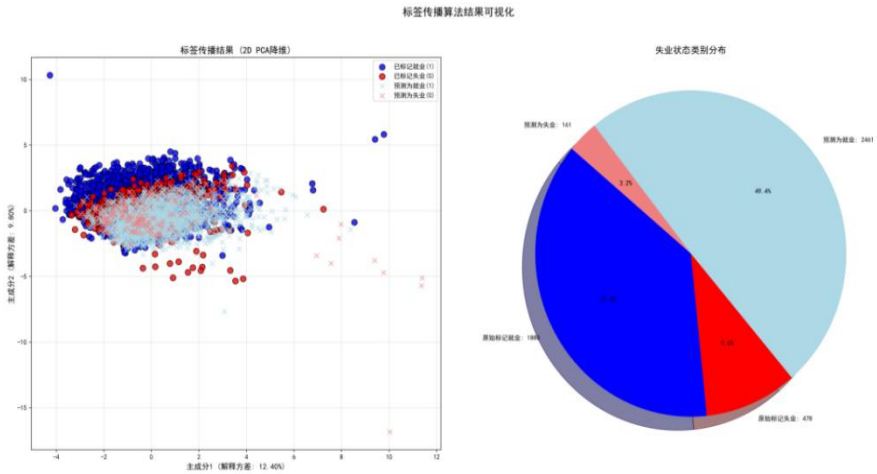

图 5-4 标签传播结果

图 5-4 展示了标签传播算法在失业状态预测中的可视化结果。左侧的散点图使用 PCA 降维将多维特征压缩为二维空间展示，其中蓝色圆点表示原始已标记为就业(1)的样本，红色圆点表示已标记为失业(0)的样本，浅蓝色叉号表示被算法预测为就业的未标记样本，浅红色叉号表示被预测为失业的未标记样本。从散点分布可以看出，就业与失业状态在特征空间中有一定的聚集性，但也存在重叠区域，说明分类任务具有一定挑战性。右侧的饼图展示了各类别的分布比例：原

始标记的就业样本 1889 个(37.9%)，原始标记的失业样本 478 个(9.6%)，通过标签传播预测为就业的样本 2461 个(49.4%)，预测为失业的样本 161 个(3.2%)。最终标签结果由表 5-1 所示

表 5-1 当前就业状态

就业失业状态	就业	失业
数量（人）	4341	639

5.1.3 卡方检验

为探究各类特征其对就业状态的影响，我们采用常用的统计分析方法——卡方检验（Chi-squaretest），并结合数据分析结果对性别、年龄、教育程度、专业、行业代码等特征进行解读。

1.卡方检验

卡方检验用于评估两个分类变量之间是否存在统计显著的关联关系。通过构建列联表（交叉频数表），计算实际频数与理论期望频数的偏离程度[4]。若偏离显著（$p<0.05$），则说明变量间存在显著关联。在本文中，性别、教育程度、专业、行业代码均为离散型分类变量，因此适合采用卡方检验方法，判断这些变量与失业状态（分类变量）之间是否存在统计显著关系。

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{5.3}$$

其中，O_i为观测频数，E_i为期望频数，χ^2为卡方统计量。卡方值越大，表示观测频数与期望频数之间的偏差越大，变量间的关联越显著。

2.结果汇总与解释

为探究各变量对失业状态的独立影响，我们对全部特征变量进行了卡方检验。检验显著性水平设定为 0.05，若 p 值小于 0.05 则认为变量与“失业状态”存在统计显著的关联。下列变量与失业状态具有显著的统计关联，说明它们在不同失业状态下的分布差异明显：

人口与社会属性类：性别、年龄分组、婚姻状态、教育程度、户口性质、户口所在地、宗教信仰。这些变量在样本群体中的分布对失业状态具有显著影响。例如，性别差异可能反映在就业岗位类型的性别偏好；婚姻状态与家庭责任相关，可能影响求职行为和单位选择；教育程度、户口属性和宗教背景则体现出社会身份对就业机会的影响。

时间事件类：毕业日期。毕业时间与失业状态的关联显著，可能与疫情发生时间、招聘周期等因素有关，提示个体处于不同时间节点所面临的就业环境存在差异。

职业背景类：行业代码、所学专业代码、失业原因、录用单位分类。不同行业和专业对应的失业风险不同，失业原因的差异也提示群体间的结构性就业风险差异。录用单位分类则反映了不同性质单位对用工稳定性的影响。

不显著变量包括：专业、民族、政治面貌、人口类型、毕业学校分类。这些变量与失业状态的统计关系不显著，可能由于样本中分布均衡、类别划分过细或信息含量不足等原因造成。

卡方统计量最高的特征依次为：

户口所在地（χ^2=294.39,p<0.001）：说明失业状态在不同地区间分布显著不同，可能与地区间就业机会、产业结构和治理能力差异有关；

所学专业代码（χ^2=292.61,p<0.001）：不同专业毕业生的失业概率显著不同，可能受到供需匹配程度和行业趋势影响；

毕业日期（χ^2=262.51,p<0.001）：表明毕业时间对就业状态影响显著，可能由于某些届次毕业生恰逢疫情或招聘淡季；

行业代码（χ^2=96.90,p<0.001）：不同行业在疫情期间的抗风险能力和复工节奏不同，形成结构性失业差异；

年龄分组（χ^2=31.66,p<0.01）：这一结果说明，不同年龄阶段的群体在失业状态上的分布差异显著，个体所处的年龄段在很大程度上影响其就业稳定性和失业风险。

表 5-2 各特征卡方检验

Feature	Chi2_Statistic	P_value	Degrees_of_Freedom
户口所在地	294.3886096	2.2295E-06	191
所学专业代码	292.6142149	0.000398083	216
毕业日期	262.5102031	2.61597E-06	166
行业代码	96.90292526	3.0435E-09	29
年龄分组	66.68682215	5.00503E-13	5
失业原因	31.66447244	0.002693037	13
户口性质	30.36696365	3.34708E-05	6
录用单位分类	12.57183626	0.001862346	2
婚姻状态	10.52594343	0.014585937	3
宗教信仰	8.243237769	0.041243303	3
教育程度	7.949487525	0.018784114	2
专业	7.365542883	0.497755019	8
性别	7.100915463	0.007704459	1
民族	6.589566456	0.581490916	8
政治面貌	5.850127943	0.210623874	4
人口类型	4.533518903	0.103647511	2
毕业学校分类	1.242107618	0.537377845	2

图 5-5 是分类变量与“失业状态”关系的可视化条形图，图中每个子图展示了一个变量在不同失业状态（“0”表示在职，"1"表示失业）下的频次分布情况，比通过较各个变量的不同类别在失业与非失业群体中的分布差异，可以初步判断变量与“失业状态”的相关性。

例如，从性别条形图可以看出，男女的失业人数差异不大，但从数值看，女性失业者略高；性别对就业的影响可能因职业分布、行业结构差异体现出间接效应。婚姻状态条形图显示未婚人群失业者最多，尤其在未婚和已婚之间形成明显差异；可能反映年轻群体就业不稳定或婚育阶段对就业的干扰作用。教育程度条形图显示教育层次越高失业人数越多，尤其在中间学历（如大专/本科）处人数最多；可能说明学历通胀、行业匹配不均衡等问题存在。户口性质/户口所在地则可以看出，农村户口群体失业人数更高，且不同地区（如部分代码区域）失业者差异显著；反映城乡就业资源不均、迁移障碍等问题。行业代码条形图说明某些行业（如代码为 9、21）失业人数明显更多；表明特定行业就业稳定性较差，应在模型中加入行业类别作为强预测因子。

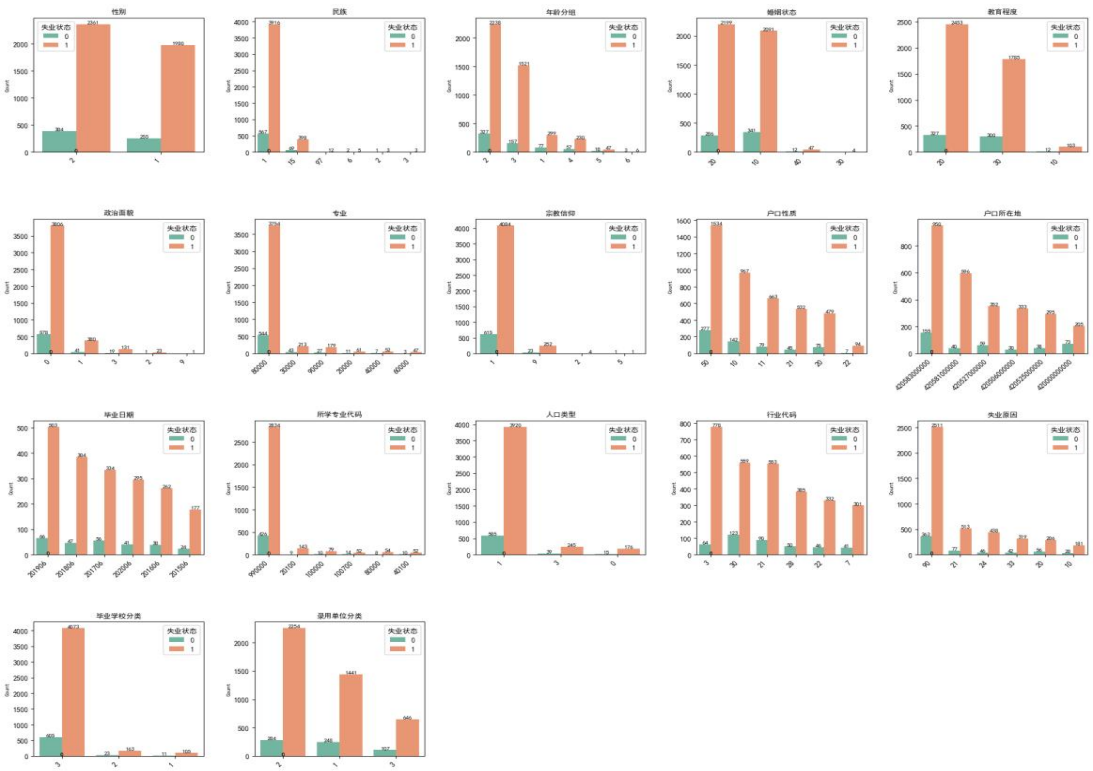

图 5-5 分类变量与“失业状态”关系的可视化条形图

5.2 问题 2 模型求解与建立

5.2.1 特征筛选

观察测试集中存在有姓名、性别、生日等共计 28 个特征，存在大量的冗余特征、相关特征，如果直接投入训练不宜模型的泛化能力，故需特征工程去找出与就业状态显著相关的特征。

Step1:基于问题一的特征工程，剔除了人员编号、姓名、生日、户籍地址、户口所在地区（名称）、文化程度、所学专业名称、兵役状态、是否残疾人、是否青少年、是否老年人、变动类型、是否独居、居住状态；

Step2：基于问题一的卡方检验，将所有不显著的特征剔除；

Step3:留下最终的特征共计 9 个，性别、年龄、婚姻状态、教育程度、宗教信仰、户口性质、户口所在地区（代码）、毕业日期、所学专业。

5.2.2 基于“LINEX-SVM”预测

1.模型理论

由于该二分类数据为不平衡数据集，故本文选取 SVM 模型进行拟合，并根据 Ma et al.(2019)的研究，在 SVM 模型基础上通过修改损失函数的方式改进 SVM 模型，建立 LINEX-SVM 模型[14]。下面是模型的理论部分。

（1）SVM 模型

支持向量机（SupportVectorMachine，SVM）是一种经典的监督学习算法，广泛应用于分类和回归任务。SVM 的核心思想是通过寻找一个最优的超平面，将不同类别的数据点尽可能分开，并最大化类别之间的间隔。根据数据的线性可分性，SVM 可以分为线性可分和线性不可分两种情况[7]。

①线性可分情况

在二维空间中，SVM 的目标是找到一条直线，将两类数据点完全分开；在

多维空间中，SVM 的目标是找到一个超平面，将两类数据点完全分开。SVM 不仅要求分类正确，还要求分类边界与最近的数据点（支持向量）之间的间隔最大化。这个间隔被称为“最大间隔”。位于间隔边界上的数据点被称为支持向量，它们决定了分类边界的位置[18]。

对于二分类问题，线性可分 SVM 的目标函数可以表示为：

$$\min_{w,b} \frac{1}{2}\|w\|^2 \tag{5.4}$$

约束条件为：

$$y_i(w \cdot x_i + b) \geq 1, i = 1, 2, \cdots, n$$

其中 w 是超平面的法向量，b 是偏置型，y_i 是样本标签，x_i 是样本特征向量。

②线性不可分情况

当数据无法通过一条直线完全分开时，SVM 通过引入“松弛变量”（SlackVariables）来允许部分数据点位于分类边界的错误一侧，从而实现软间隔分类。软间隔分类的目标是在最大化间隔的同时，最小化分类错误。

引入松弛变量 ξ_i 后，目标函数变为：

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \tag{5.5}$$

约束条件为：

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0$$

其中 C 是正则化参数，用于控制间隔和分类错误之间的权衡；ξ_i 是松弛变量，表示允许的分类误差。

（2）LINEX 损失模型

在 SVM 的优化过程中，损失函数的选择对模型的性能和优化目标有重要影响。LINEX（LinearExponential）损失是一种非对称损失函数，适用于对误分类的惩罚具有不对称性的场景[8]。LINEX 损失的定义为：

$$L_{LINEX}(y, f(x)) = e^{\alpha(y \cdot f(x) - 1)} - \alpha(y \cdot f(x) - 1) - 1 \tag{5.6}$$

其中 α 是控制损失函数非对称性的参数，当 $\alpha \succ 0$ 时，损失函数对正误差 $y \cdot f(x) \succ 1$ 的惩罚更重；当 $\alpha \prec 0$ 时，损失函数对负误差 $y \cdot f(x) \prec 1$ 的惩罚更重。LINEX 损失具有以下特点：

①非对称性：LINEX 损失对误分类的惩罚具有非对称性，适用于对某些类型的误差更敏感的场景。例如，在医疗诊断中，将患者误判为健康的代价远高于将健康人误判为患者。

②连续可微：LINEX 损失是连续可微的，便于优化。传统的 SVM 使用合页损失（HingeLoss）作为损失函数，用于衡量分类错误的程度，与 Hinge 损失相比，LINEX 损失的梯度在所有点都是连续的，这使得优化算法更容易收敛。

③灵活性：通过调整参数 α 可以灵活控制损失函数的形状。当 α 取不同值时，LINEX 损失可以逼近其他常见的损失函数，例如：当 $\alpha \to 0$ 时，LINEX 损失逼

近平方损失（SquaredLoss）；当$\alpha \to \infty$，时，LINEX 损失逼近指数损失（ExponentialLoss）。

（3）LINEX-SVM 模型

参照 Maetal.(2019)的研究，LINEX-SVM（Linear Exponential Support Vector Machine）是一种结合了 LINEX 损失函数的支持向量机模型。与传统的 SVM 使用合页损失（HingeLoss）不同，LINEX-SVM 通过引入 LINEX 损失函数，能够更好地处理误分类的非对称惩罚问题。通过调整参数α和C可以灵活控制模型的性能。参数α控制损失函数的非对称性，参数C控制间隔和分类错误之间的权衡。其目标函数可以表示为：

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} L_{LINEX}(y_i, f(x_i)) \tag{5.7}$$

其中$L_{LINEX}(y_i, f(x_i))$是 LINEX 损失函数。LINEX-SVM 的约束条件与传统 SVM 类似，通常为：

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0 \tag{5.8}$$

将 LINEX 损失函数代入 SVM 的目标函数中，构建新的目标函数：

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} (e^{\alpha(y_i f(x_i)-1)} - \alpha(y_i f(x_i) - 1) - 1) \tag{5.9}$$

计算目标函数对参数w和b的梯度：

$$\frac{\delta L}{\delta w} = w + C\sum_{i=1}^{n} (e^{\alpha(y_i f(x_i)-1)} - \alpha(y_i f(x_i) - 1) - 1) y_i x_i \tag{5.10}$$

$$\frac{\delta L}{\delta b} = C\sum_{i=1}^{n} (e^{\alpha(y_i f(x_i)-1)} - \alpha(y_i f(x_i) - 1) - 1) y_i \tag{5.11}$$

可以使用梯度下降、随机梯度下降（SGD）或拟牛顿法等优化算法求解目标函数的最优解。

2.实证分析

基于问题一的分析，筛选出性别、年龄分组、教育程度、失业原因和录用单位分类等与就业状态具有相关性的数据，接着对比附件 1 工作表名称为数据集和预测集中的特征，发现失业原因和录用单位分类等部分特征仅存在于数据集，而预测集中没有，故在进行后续建模前将这些特征剔除，不让其进入模型中。

通过对数据集中就业状态的数据进行可视化，如图 5-6 所示，可以看出当前就业状态数据为不平衡数据，故本文引入 LINEX 损失函数，构建了 LINEX-SVM 模型，以更好地处理误分类的非对称惩罚问题。为了验证 LINEX-SVM 模型的优越性，本文将数据按照 4:1 的比例划分为训练集和测试集，并分别构建了原始 SVM 模型、SVM_{SMOTE}模型、SVM_{RUS}模型、SVM_{ROS}模型、SVM_{DEC}模型以及 LINEX-SVM 模型。通过网格搜索法进行五折交叉验证调参，找到每个模型的最优参数，并在此基础上拟合模型。

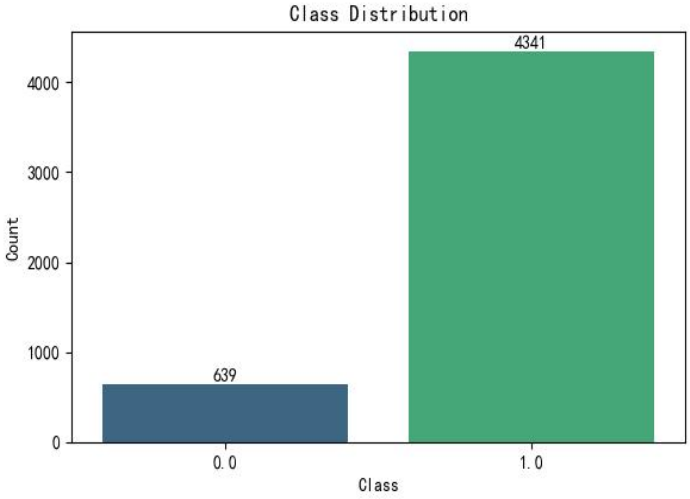

图 5-6 就业状态分布

（1）模型建立

为了体现本文建立的模型是否优于已有模型，泛化能力是否更好，故建立了原始 SVM 模型以及其余处理不平衡数据集的方法来进行拟合，共包括以下几种模型及方法：

①SVM 模型：一种基于超平面分类的支持向量机模型，适用于二分类问题，通过最大化分类边界来实现高泛化能力。

②SMOTE 方法：一种解决类别不平衡问题的过采样技术。通过在少数类样本之间插值生成新的合成样本，从而增加少数类样本的数量。

③ROS 方法：一种简单的过采样技术。通过随机复制少数类样本来平衡数据分布。

④RUS 方法：一种欠采样技术。通过随机删除多数类样本来平衡数据分布。

⑤DEC 方法：一种基于密度的聚类算法。通过分析样本的密度分布，将高密度区域划分为簇，低密度区域作为噪声或边界。

⑦LINEX-SVM 模型：基于 LINEX 损失函数的 SVM 模型，通过调整损失函数对误分类样本的惩罚，适用于对误分类成本敏感的场景。

（2）模型调参

本文通过利用网格搜索（GridSearchCV）进行模型最优参数的确定，其中 C 控制模型的正则化程度，影响分类边界的宽窄；kernel 定义核函数类型，决定模型的线性或非线性分类能力；gamma 控制核函数的形状，影响分类边界的复杂度。结果如表 5-3 所示，

表 5-3 SVM 模型及基于重采样方法的 SVM 模型最优参数的确定

参数	SVM	SVM_{SMOTE}	SVM_{ROS}	SVM_{RUS}	SVM_{DEC}
C	10	20	5	10	20
gamma	scale	scale	scale	scale	scale
kernel	rbf	rbf	rbf	rbf	rbf

除此之外，LINEX-SVM 的最优参数为学习率为 0.0001,正则化参数λ=0.001,核函数的系数为 1。

（3）模型评估

本文在对就业状态进行预测时，侧重就业状态被准确预测的比例。本文将使用加权对数损失以及基于混淆矩阵的准确率、精确率、召回率、F1 值和 AUC 值

组成的评价项来衡量分类性能。选择这些性能指标是因为这些的模型性能和它们被广泛用于衡量信用评分模型的性能，指标涵盖了几乎所有方面。

本文将就业记为 1，失业记为 0，通过混淆矩阵可以统计各个类别被错误分成其他类别的情况。如表 5-4 所示：

表 5-4 混淆矩阵

	预测类别 0	预测类别 1
真实类别 0	TP	FN
真实类别 1	FP	TN

其中 TP 表示失业被正确预测的数量，TN 表示就业被正确预测的数量，FN 表示失业被误判为就业的数量，FP 表示就业被误判为失业的数量。各评价指标公式如下：

$$Accuracy=\frac{TP+TN}{TP+TN+FN+FP} \tag{5.12}$$

$$\text{Precision}=\frac{TP}{TP+FP} \tag{5.13}$$

$$\text{Re}call=\frac{TP}{TP+FN} \tag{5.14}$$

$$F1score=\frac{2\cdot\text{Precision}\cdot\text{Re}call}{\text{Precision}+\text{Re}call} \tag{5.15}$$

准确率就是所有预测正确的类别数目占总数目的比例；精确率表示正确预测为正类的占全部预测为正类的比例；召回率表示正确预测为正类的占全部实际为正类的比例；F1 值是精确率与召回率的调和平均；而 AUC 则是衡量模型在各种分类阈值下分类能力的指标，通过计算 ROC 曲线下面积，AUC 值越接近 1，说明模型的区分能力越强。

加权对数损失（WeightedLog-Loss）是一种用于分类问题的损失函数，通常用于处理类别不平衡问题。它通过对不同类别的样本赋予不同的权重，来调整模型对不同类别样本的惩罚程度。加权对数损失的公式如下：

$$Logloss=-\frac{\sum_{i=1}^{N}w_i(y_i\log(p_i)+(1-y_i)\log(1-p_i))}{\sum_{i=1}^{N}w_i} \tag{5.16}$$

其中 N 为样本总数；y_i 为第 i 个样本的真实标签；p_i 是预测第 i 个测试样本为 1 的概率；w_i 是权重。如果 $y_i=1$，那么 $w_i=2$；如果 $y_i=0$，那么 $w_i=1$。换句话说，阳性样本在评价中权重更大。

（4）模型对比

本文通过 python 软件对处理过的数据建立了 SVM 模型，SVM_{SMOTE} 模型，SVM_{ROS} 模型，SVM_{RUS} 模型，SVM_{DEC} 以及 LINEX-SVM 模型，利用这六种机器学习模型来预测是否就业，分别计算各个模型的加权对数损失以及基于混淆矩阵的准确率、精确率、召回率、F1 值和 AUC 值，将其作为模型评估指标。各模型的混淆矩阵如图 5-7 所示，

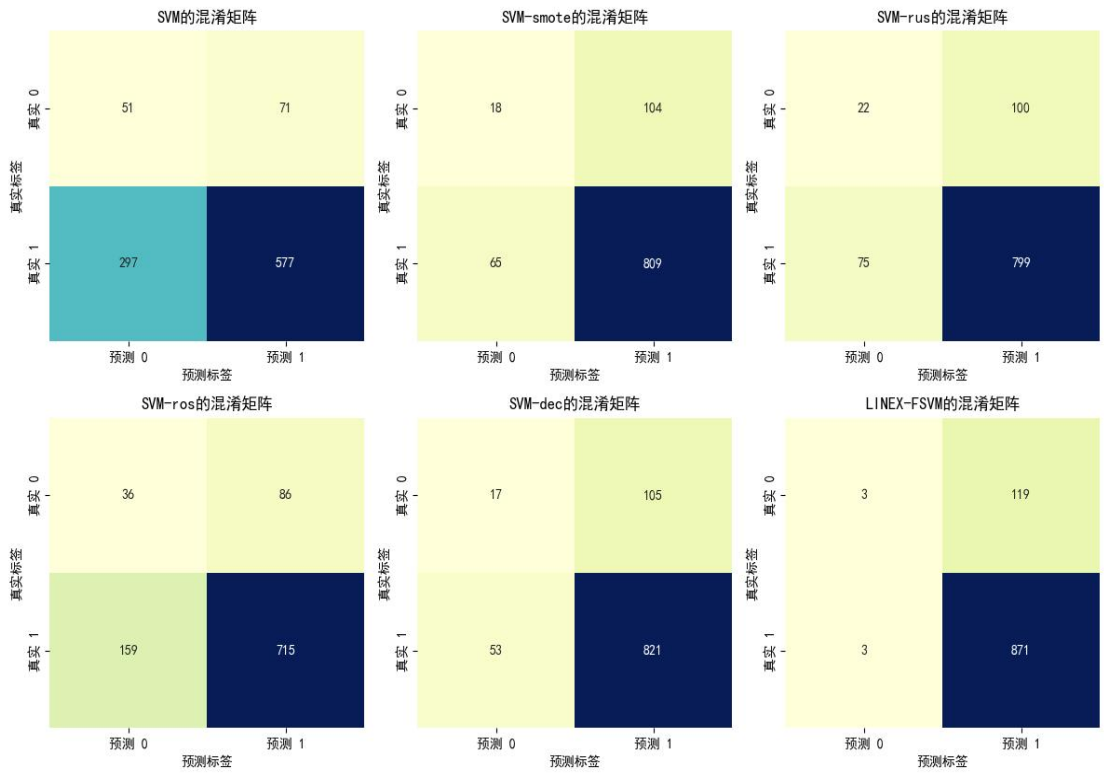

图 5-7 各模型的混淆矩阵

根据混淆矩阵计算的准确率、精确率、召回率、F1 值和 AUC 值以及各个模型的加权对数损失结果如表 5-5，

表 5-5 模型评估结果

模型	Accuracy	Precision	Recall	F1 Score	AUC	Logloss
SVM	0.6305	0.8904	0.6602	0.7582	0.5974	0.6347
SVM_{SMOTE}	0.8303	0.8861	0.9256	0.9054	0.6359	0.4912
SVM_{RUS}	0.8243	0.8888	0.9142	0.9013	0.5993	0.5242
SVM_{ROS}	0.7540	0.8926	0.8181	0.8537	0.6257	0.4562
SVM_{DEC}	0.8414	0.8798	0.9394	0.9122	0.6471	0.3318
LINEX-SVM	0.8775	0.8866	0.9966	0.9345	0.6696	0.2611

通过性能指标对比，结果显示 LINEX-SVM 模型在所有关键指标上都表现出色，尤其是在准确率（0.8775）、F1 值（0.9345）、AUC（0.6696）以及加权对数损失（0.2611）方面均达到最佳，展示出其在模型性能和预测能力上的明显优势。其 0.9966 的召回率也表明几乎能够识别所有正样本，有效减少漏检。相较之下，采用重采样技术后得到的模型也表现突出，提升了正样本的检测能力，但整体仍略逊于 LINEX-SVM。原始的 SVM 模型在准确率和 AUC 方面表现较差，显示其在处理不平衡数据集时存在局限，特别是在召回率和 F1 值方面也低于采样后的模型。其他模型如 ROS 和 DEC 则表现中等，但整体不及 LINEX-SVM 和经过重采样的模型。综上所述，LINEX-SVM 在性能和综合表现方面都优于其他模型，适合用于后续是否就业的预测，性能良好。

对预测集数据进行预测，得到表 5-6 所示结果，

表 5-6 模型预测结果

预测	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	就业数量小计	失业数量小计
就业状态	1	1	1	1	1	0	1	1	1	1		
预测	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	18	2
就业状态	1	1	1	1	1	1	0	1	1	1		

5.2.3 特征排序

1.XGBoost 模型

XGBoost（ExtremeGradientBoosting）是一种高效的梯度提升树（GradientBoostingTree）算法：XGBoost 引入了 L1（Lasso）和 L2（Ridge）正则化，帮助减少过拟合[5]。

XGBoost 通过构建决策树来进行预测,它的目标是最小化损失函数，公式如下：

$$L(\theta)=\sum_{i=1}^{n} L=l(y_i,\hat{y}_i) \tag{5.17}$$

其中 L（θ）：总体损失函数；n：样本数量；K：树的数量。

在每一轮迭代中，XGBoost 使用泰勒展开来近似损失函数。XGBoost 核心思想：通过反复添加和按照决策树的方法生成树，每次这样操作一次，就是学习一个新的 f(x),去拟合上次预测的残差。当我们训练完会得到 K 个树，对于分类任务，采用多数投票法[16]' 另外，XGboost 一个显著的功能是可以进行特征值排序。

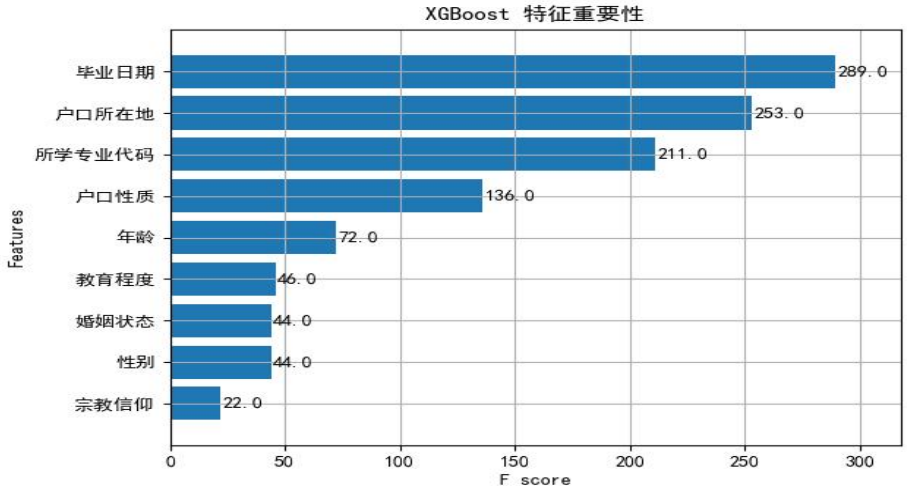

图 5-8 XGboost 特征重要性排序

图 5-8 显示毕业日期是预测失业状态的最关键特征，F 分数高达 289.0，表明其对模型预测结果具有最显著影响。户口所在地次之，F 分数为 253.0，同样具有很强的预测能力。所学专业代码(211.0)和户口性质(136.0)也有较高的重要性，而年龄、教育程度、婚姻状态和性别的影响相对较小。宗教信仰(22.0)对预测结果的影响最小。这表明时间因素(毕业日期)和地理因素(户口所在地)是影响失业

状态的主导因素。

2.SHAP 可解释模型

SHAP 模型，是比较全能的模型可解释性的方法，既可作用于之前的全局解释，也可以局部解释，即单个样本来看，模型给出的预测值和某些特征可能的关系，这就可以用到 SHAP[6]。SHAP 属于模型事后解释的方法，它的核心思想是计算特征对模型输出的边际贡献，再从全局和局部两个层面对“黑盒模型”进行解释。SHAP 构建一个可解释模型，所有的特征都视为“贡献者”[17]。而 XGboost 模型本身就作为一个黑箱模型，故选取 SHAP 对其进行分析。

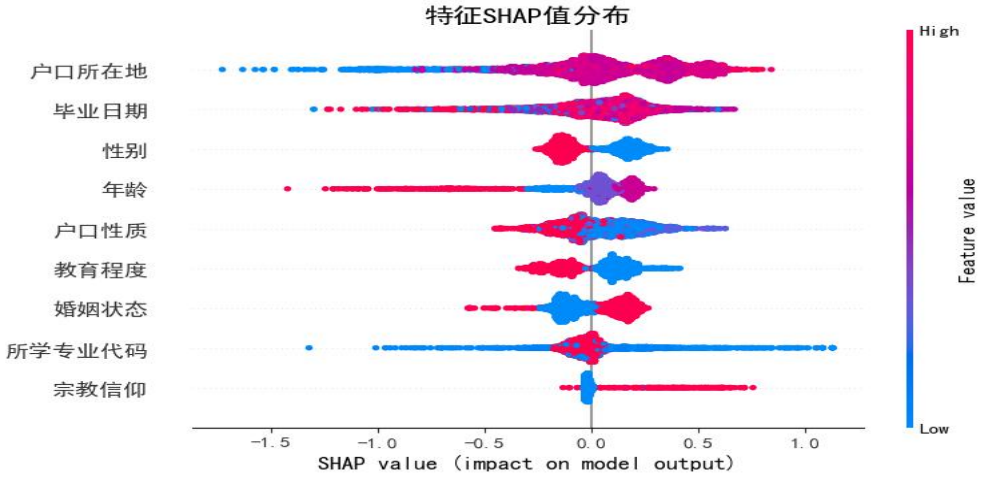

图 5-9 SHAP 特征分布

图 5-9 显示户口所在地特征具有最广泛的分布范围，其正负 SHAP 值均较大，表明不同地区对失业风险有明显不同影响。毕业日期的 SHAP 值分布呈现出两个峰值，说明某些时间点可能是就业的转折期。还可以分析出特征对模型的影响方向。由图 5-9 所示，红色代表特征值变大，蓝色意味着特征变小，分析和 SHAP 变化的方向是否一致就可得出特征对失业状态是正向影响还是负向影响。可以发现，户口所在地、毕业日期、婚姻状态和宗教信仰对目标变量呈现正向影响，其余特征的影响方向是相反的。

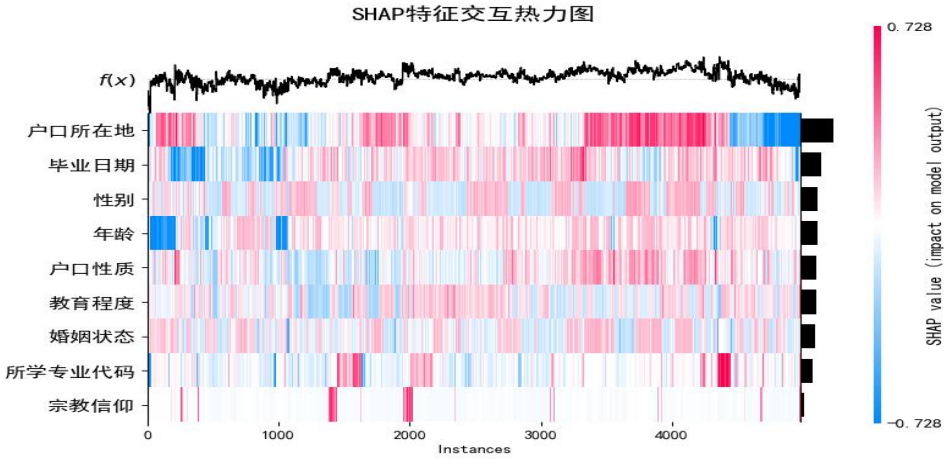

图 5-10 交互热力图

图 5-10 进一步揭示了特征间的复杂交互关系，红色区域表示正向影响(增加失业可能性)，蓝色区域表示负向影响。这些分析表明失业预测模型中，地理因

素和时间因素不仅单独重要，它们的交互效应也显著影响预测结果。图 4 量化了特征的平均绝对影响，户口所在地(0.28)影响最大，约占毕业日期(0.18)的 1.5 倍，依次的特征排序如图 5-11 所示。

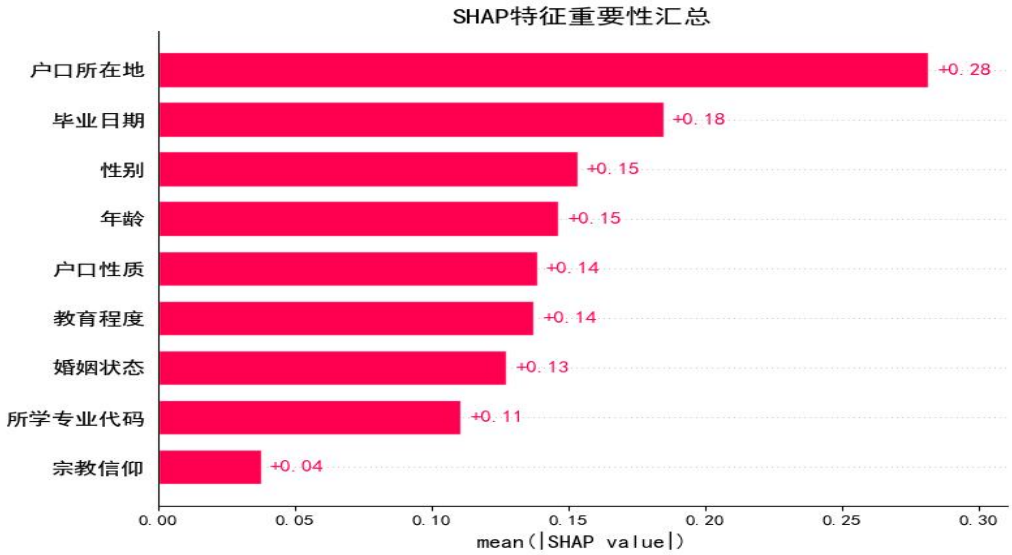

图 5-11 SHAP 特征重要性排序

5.3 问题 3 模型求解与建立

5.3.1 特征融合

为有效融合宏观因素对个体就业状态的潜在影响，我们基于“毕业日期”这一变量，使用指数加权移动平均（Exponential Weighted Moving Average,EWMA）的构造特征。该方法的思想是：个体在毕业后所面对的宏观就业环境比在校期间对其就业状态影响更为直接，因此我们利用毕业年份之后的相关宏观数据构建其宏观特征，并通过 EWMA 方法实现历史信息的动态加权整合[9]。

1.特征来源与选择

引入外部数据集 1，所用数据来源于《中国城市统计年鉴》与宜昌市统计局官网，包括以下五大类共计 24 个指标，对于缺失值采用了线性插值进行填补：

表 5-7 宏观指标

类别	特征
经济指标	地区生产总值(万元)
	第一产业增加值(万元)
	第二产业增加值(万元)
	第三产业增加值(万元)
	第一产业增加值占 GDP 比重(%)
	第二产业增加值占 GDP 比重(%)
	第三产业增加值占 GDP 比重(%)
	人均地区生产总值(元)
劳动力市场	城镇非私营单位从业人员数(万人)
	第一产业从业人员数(万人)
	第二产业从业人员数(万人)
	第三产业从业人员数(万人)

	第二产业从业人员比重(%) 第三产业从业人员比重(%) 城镇非私营单位在岗职工平均人数(万人)
薪资情况	城镇非私营单位在岗职工工资总额(万元) 城镇非私营单位在岗职工平均工资(元)
消费水平	社会消费品零售总额(万元) 限额以上批发零售业商品销售总额(万元)
政府支出	当年新签项目(合同)个数(个) 当年实际使用外资金额(万美元) 地方财政一般预算内收入(万元) 地方财政一般预算内支出(万元)

2.EWMA 方法原理与公式

EWMA 是一种将历史数据进行加权平均的平滑方法，其特征是对近期数据赋予更高权重，从而更敏感地反映趋势变化[19]。

设某一宏观变量在毕业年份 y 之后的时间序列为：

$$x_{y+1}, x_{y+2}, ..., x_T$$

则该变量的指数加权移动平均值定义为：

$$EWMA_y = \sum_{t=y+1}^{T} \alpha(1-\alpha)^{t-(y+1)} x_t \tag{5.18}$$

其中，$\alpha \in (0,1)$为平滑参数，控制近期数据权重（本研究中设定为$\alpha=0.3$）。

该方法的优点在于，能更好地捕捉毕业后宏观环境变化对个体的动态影响；有效保留长期趋势信息，降低短期异常波动干扰。

3.特征融合

对于每一位调查对象，我们按照其毕业年份提取其之后年份（如 2009 至 2023 年）的宏观数据，通过 EWMA 方法分别计算各宏观指标的加权值，并作为新的特征列添加至个体样本中，从而构建个性化的宏观影响特征。

例如，若某位个体的毕业时间为 2008 年 6 月，则其“地区生产总值”特征对应的值为：

$$GDP_{EWMA} = EWMA(GDP_{2009}, GDP_{2010}, ..., GDP_{2023})$$

最终，我们获得了增强数据集，每位个体样本都包含了 24 个基于其毕业时间推算的宏观特征。

5.3.2 多模块自步学习

首先在本文中有个人信息、经济指标、劳动力市场、薪资情况、消费水平、政府支出六大模块，这里考虑多模块融合分类学习。具体来说，首先基于我们提出的 MSPL 算法去为每个模块选择样本，由于我们相较于传统的自步学习（SPL）模型加了一正则化项，不仅可以将所有视角统一在一个框架下去选择课程，还在一定程度上保证了每个模块都会被平缓地计入课程学习中，避免出现模型偏袒某个模块而导致不断恶化对其他模块的学习[20]。再选择样本后，每个模块基于问题二我们提出的分类模型 LINEX-SVM 分别训练得到分类器$f_1, f_2, ..., f_6$,最终集成得到分类器 $F = \sum_{i=1}^{6} w_i f_i$,其中 6，在问题二中已经分析得到，本文数据存在极大的

类别不均衡，故可以通过每个模块预测少数类样本的准确率分配权重，从而更加关注对少数类样本的学习。这是整个我们提出的 MSPL 模型框架，下面对这些细节进行补充。

1.MSPL

首先介绍下自步学习（Self-PacedLearning,SPL），其将课程设计嵌入模型优化过程中。核心思想如下：假设训练集为 $D=\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$，其中 $x\in R^d$ 为样本，y 为对应标签。SPL 的目标是通过交替优化模型参数 w 和变量 $v=[v_1,v_2,...,v_n]$，最小化以下目标函数[10]：

$$\min_{w,y}\sum_{i=1}^{n}v_iL(y_i,f(x_i,w))-\lambda\sum_{i=n}^{n}v_i \tag{5.19}$$

其中 L 为损失函数，f 为模型估计值。λ 为自步学习超参数。固定上式 v 时，目标函数退化为标准分类任务。固定 w 时，隐变量 v 的解取决于 SPL 的惩罚项 $\lambda\sum_{i=1}^{n}v_i$ 的设计。最优解为：

$$v_i=\begin{cases}1,.....if\ L<\lambda\\0,.....else\end{cases} \tag{5.20}$$

随着 λ 逐步增大，更多样本会被纳入训练，直到覆盖大部分数据。

（1）选择样本

在我们固定每个模块或群分类器参数时，有

$$\min_{v}\sum_{i=1}^{n}v_iL_i-\lambda\sum_{i=1}^{n}v_i-\gamma\sum_{j=1}^{b}\|v^{(j)}\|_2\ \ s.t.\ v\in[0,1]^n \tag{5.21}$$

注意到，式子（5.21）是一个非凸优化问题，为简化问题，对每个群进行单独求解，故式子（5.21）可变为：

$$\min_{v}\sum_{i=1}^{n}v_i^{(j)}L_i^{(j)}-\lambda\sum_{i=1}^{n_j}v_i^{(j)}-\gamma\|v^{(j)}\|_2\ \ s.t.\ v^{(j)}\in[0,1]^n \tag{5.22}$$

令 $\|v^{(j)}\|_1=k$,则式子（5.22）可等价为：

$$\min_{k}E^k=\sum_{i=1}^{k}L_i^{(j)}-\lambda k-\gamma\sqrt{k} \tag{5.23}$$

其中第 j 个群中有 k 个样本被选择，且若对第 j 个群所有样本按损失从小到大排序设为 $\left(L_1^{(j)},L_2^{(j)},...,L_{n_j}^{(j)}\right)$，显然这 k 个样本为前 k 个损失最小的样本。

$$diff=E^{k+1}-E^k \tag{5.24}$$

$$L_{k+1}^{(j)}-(\lambda+\gamma\frac{1}{\sqrt{k+1}+\sqrt{k}}) \tag{5.25}$$

$L_{k+1}^{(j)}>0$，且随着 k 单调递增；$\lambda+\gamma\frac{1}{\sqrt{k+1}+\sqrt{k}}$大于 0，且随着 k 单调递减。

当 diff 小于等于 0 时，说明第 k+1 个最小损失样本可以被模型选择，因为此时的E^{k+1}是变小的，符合最优化趋势。当 diff 大于 0，说明E^{k}即为最小目标函数，得到模型选择的前 k 个最小损失样本。

因此，对第 j 个群的所有样本按损失从小到大排序设为（$L_1^{(j)}$，$L_2^{(j)}$，…，$L_{n_j}^{(j)}$），式子（5.22）所求的 v 可被求解为：

$$v_i^{(j)} = \begin{cases} 1, \ldots\ldots L_i^{(j)} < \lambda + \gamma \dfrac{1}{\sqrt{i-1}+\sqrt{i}} \\ 0, \ldots\ldots else \end{cases} \tag{5.26}$$

注意到，若我们将每个群看成模块的话，就得到了我们 MSPL 选择样本的准则，对于第 j 个视角的损失排名为第 i 个的样本，基于式子（5.26），当损失小于阈值为 1，否则则为 0。另外，为了考虑到每个模块损失之间可能存在量级差异，我们将每个模块的损失都进行了标准化以避免模型会过多关注量级较小的模块，得到式子：

$$v_i^{(j)} = \begin{cases} 1, \ldots\ldots \overline{L}_i^{(j)} < \lambda + \gamma \dfrac{1}{\sqrt{i-1}+\sqrt{i}} \\ 0, \ldots\ldots else \end{cases} \tag{5.27}$$

（2）更新分类器参数，模块权重 w

分类器参数更新可见问题二中有详细介绍；

模块权重更新基于以下规则：

$$W_i = \frac{P_i}{\sum_{j=1}^{6} P_j} \tag{5.28}$$

其中W_i为第 i 个模块的权重，P_i为第 i 个模块预测少数类标签（0）的准确率。不难发现，基于此原则，模型最后的集成结果会更加关注少数类标签，增加对失业样本的准确，这将有利于处理类别不均衡数据。

2.实证分析

（1）数据预处理

Step1:将含有缺失值数据的样本剔除；

Step2:对所有特征进行归一化处理；

Step3:将数据基于标签分层抽样得到训练集和测试集，训练集和测试集比例为 8：2；

Step4:为了解决类别不均衡问题，基于 GAN 去生成样本。

图 5-12 显示的周期性误差峰值（高达约 0.14）反映了 GAN 训练中生成器和判别器之间的"博弈"过程。每个峰值可能代表判别器性能临时提升，识别出生成样本的能力增强。峰值后误差迅速下降表明生成器快速调整并改进其生成能力，产生更难以被判别器区分的样本。

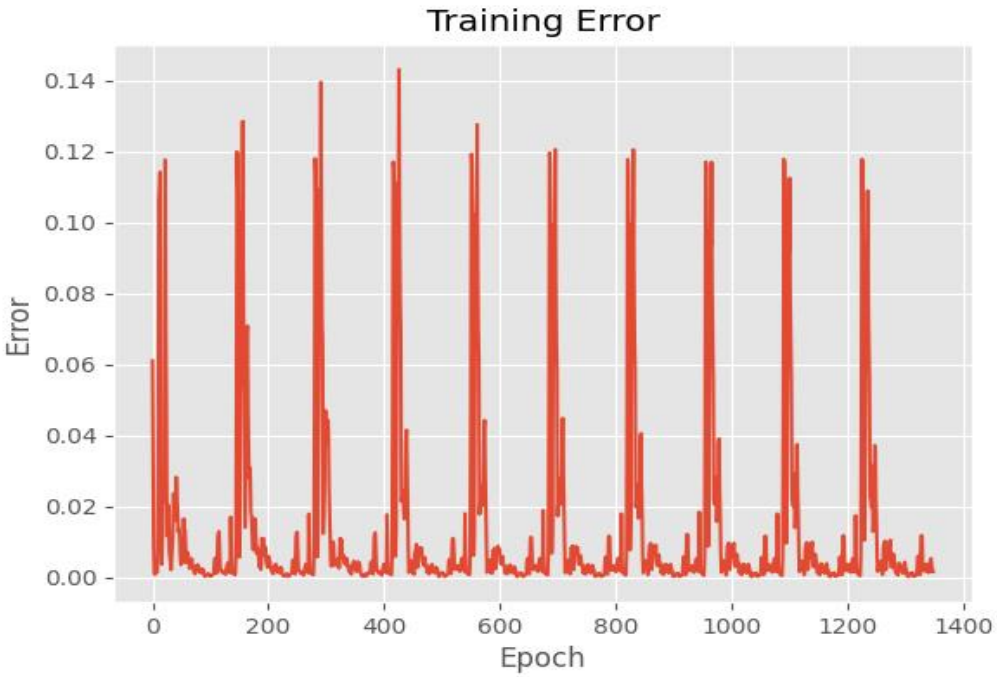

图 5-12 GAN 损失过程

（2）课程学习进程

超参数设置：分别初始化 λ 和 γ，步长设为 1 分别逐步增加。

终止条件：1.达到最大训练轮数 T（设置的为 20）；2.所有样本数据均投入各个模块训练；

（3）最终集成分类结果收敛。达到任意一条件即停止迭代。

课程迭代过程：

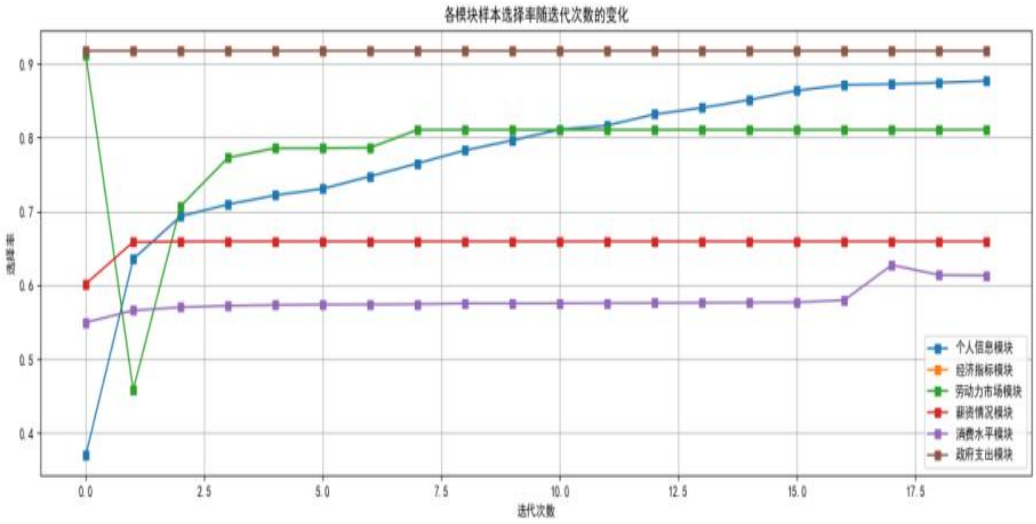

图 5-13 课程迭代样本选择

由图 5-13 可知，所有模块的样本选择均在 0.5 以上，且选择较为平缓，这得益于在 SPLD 算法 a 中我们最后将损失标准化，消除了不同模块间量级的差异，不会存在某个模型的选择水平极低。

模块权重学习结果：

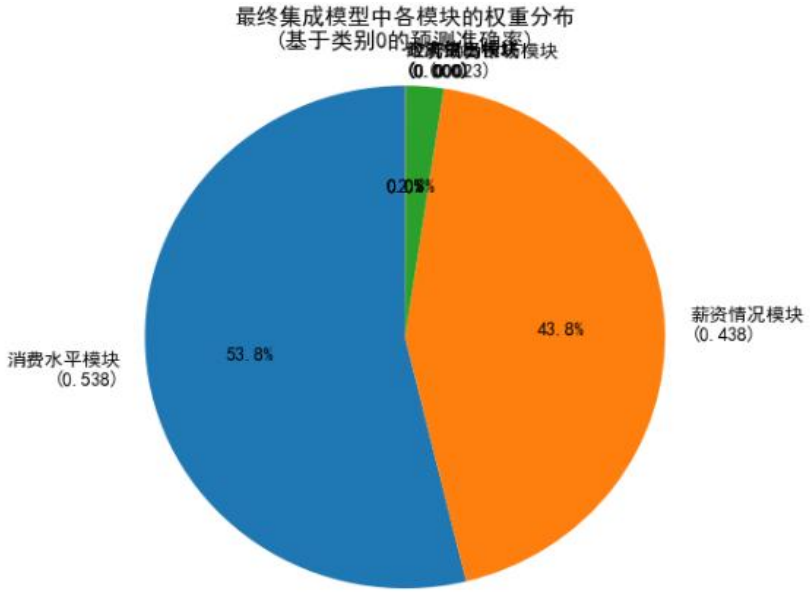

图 5-14 模块权重

由图 5-14 可知，各个模块权重的模块分配中消费水平和薪资情况模块权重最大，这说明了失业状态影响主要受消费水平和薪资情况影响较大。

对比实验：

表 5-8 模型评价指标

模型	Accuracy	Precision	Recall	F1	AUC	Logloss	class_0_accuracy
MSPL	0.9323	0.9827	0.9444	0.9632	0.852	0.4958	0.75
model-1	0.8823	0.9613	0.9111	0.9355	0.678	0.7633	0.45
model-2	0.8688	0.9574	0.9	0.9278	0.6558	0.8083	0.4
model-3	0.8552	0.9535	0.8889	0.9201	0.6105	0.8754	0.35
model-4	0.8417	0.9495	0.8778	0.9122	0.6145	0.9123	0.3
model-5	0.8281	0.9455	0.8667	0.9043	0.5576	0.9944	0.25
model-6	0.8177	0.9448	0.8556	0.898	0.5409	1.0048	0.25
model-all	0.8073	0.9441	0.8444	0.8915	0.5537	0.9989	0.25

表 5-8 中 MSPL 为我们提出的模型，model-6 分别对应着个人信息、经济指标、劳动力市场、薪资情况、消费水平、政府支出六大模块的单独输出，model-all 指的是考虑六种模块集成输出，另外 class_0_accuracy 代表对标签为失业的预测。可以发现，表 5-8 中的所有评价指标表现均最好，尤其在对少数类标签预测上，性能远远超越其他模型。

由图 5-15 的混淆矩阵中可以发现，所有模型在预测标签为 1 的准确率都达到了很高的水平，这是由于原始数据集中标签为 1 的数据占绝大多数，模型难免会更加侧重对标签为 1 的样本学习，但是就会存在一个问题，若模型在对标签输出为 1 的准确率很高的情况下，有一种极端的例子照应就是模型把所有预测的样本均标记为 1，这样也会有很高的准确率，然而，这是模型并未从模型中学到很多信息，这是由于类别不均衡主要造成的。

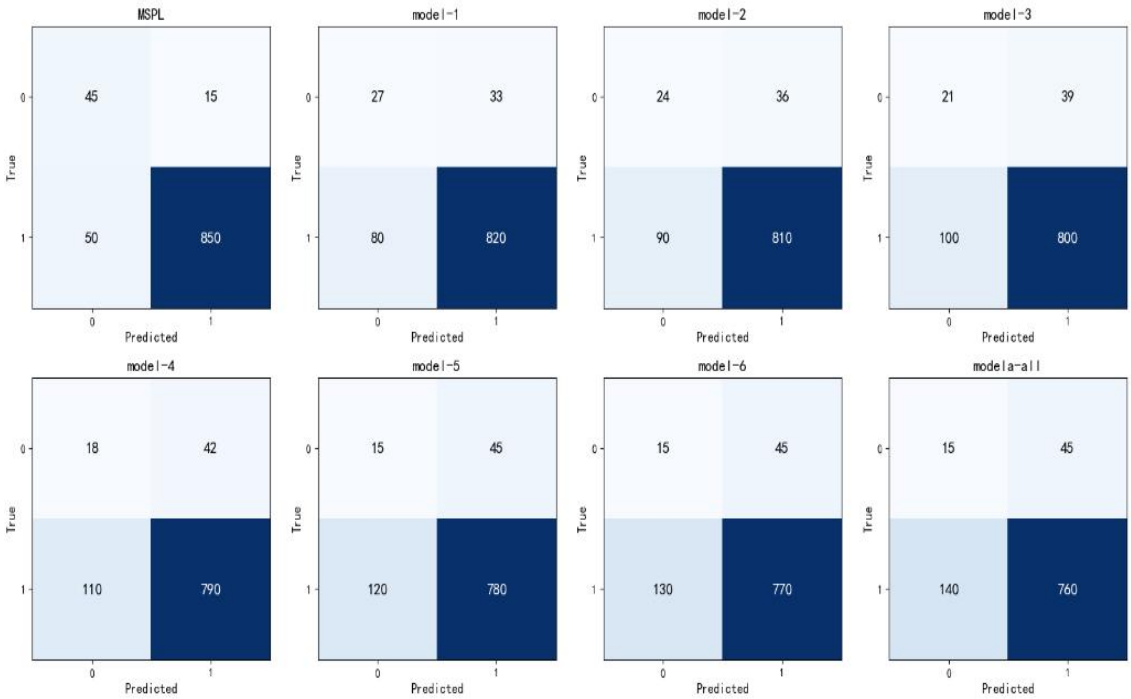

图 5-15 混淆矩阵

鉴于此，本文提出的 MSPL 模型，为了达成多模块融合，引入了一个新的自步学习正则项从而所有模块均建立在一个自步学习的框架下，且该机制可以让课程学习对每个模块均关注到，且通过标准化损失消除不了不同模块的量级差异，最终使得课程学习迭代过程更加平缓。又考虑到基于识别标签为 0 的准确率来更新模块权重，从而获得对标签为 0 更好的识别，由图 5-15 也可以看出，MSPL 模型在对标签为 0 的识别上相较于其他模型具有显著进步。表 5-9 对预测集进行预测：

表 5-9 模型预测结果

预测	T1	T3	T4	T5	T6	T7	T8	T9	T10	就业数量小计	失业数量小计
就业状态	1	1	1	1	0	1	1	1	0		
预测	T11	T13	T14	T15	T16	T17	T18	T19	T20	16	4
就业状态	1	1	1	1	0	0	1	1	1		

5.4 问题 4 模型求解与建立

5.4.1 数据预处理

1.外部数据集

外部数据 2 所用到的相关信息由表 5-10 所示。经核查，在期望薪资和实际薪资上存在一些异常值，根据原数据集注释，对于遇到 9 位或者 11 位前面或者后面补 0，不符合这种情况的异常值直接删除该样本。进一步将数据中存在的空缺值用无表示，代表没有该样本的个人信息，这并不影响后续模型的预测。

表 5-10 外部数据集 2

外部变量	中文名称	来源
desire_jd_industry_id	期望工作行业	第二届阿里巴巴大数据智能云上编程大赛-智联招聘人岗智能匹配-初赛数据_数据集-阿里云天池(aliyun.com)
desire_jd_salary_id	期望工作薪酬	
cur_degree_id	学历	
birthday	年龄	
start_work_date	开始工作日期	
cur_jd_type	当前工作类型	
cur_salary_id	当前薪资	

2.失业人员数据集与外部数据集特征对齐

为了后续模型的训练以及处理，需要将两个数据集经人工比对处理，以便于之后可通过外部数据集训练出的模型来失业人员推荐合适工作岗位，共找到五个输入特征，分别是年龄、教育程度、毕业年份、期望工作、期望薪资。年龄特征两者共有且形式无变化故不用改动；教育程度在外部数据集中是分类为大专、本科、硕士等，将失业人员数据集中教育程度特征转化为以上形式；关于毕业年份，将外部数据集中的开始工作日期改为毕业年龄即可，失业人员数据集无需变动；期望工作这一特征通过失业人员中的行业特征像外部数据集照齐；最后是期望薪资，这一特征利用在问题三中搜集到的宏观经济指标来设计失业人员的期望薪资，比如通过人均消费、物价水平等来估计每个人的期望薪资，共计两个输出特征，分别为实际工作和实际薪资，这是失业人员要预测的特征，最后预测的形式是两者结合序列。

5.4.2 模型方法

1.Transformer

Transformer 模型是由 Google 团队在 2017 年的论文"AttentionisAllYouNeed"中提出的。Transformer 最大的创新在于摒弃了传统的循环神经网络(RNN)和卷积神经网络(CNN)结构，完全基于自注意力(Self-Attention)机制构建，使得模型能够更好地捕捉序列中的长距离依赖关系，同时支持并行计算，大幅提高了训练效率[11]。

自注意力是 Transformer 的核心组件，它允许模型在处理序列中每个位置时关注序列中的所有位置，从而捕捉全局依赖关系。计算自注意力的过程如下:

Step1:为每个输入向量计算查询 Q(Query)、键 K(Key)和值 V(Value)

$$Q = XW^{Q} \tag{5.29}$$

$$K = XW^{K} \tag{5.30}$$

$$V = XW^{V} \tag{5.31}$$

其中 X 是输入矩阵，W^{Q}、W^{K}、W^{V}、是模型要学习的参数矩阵。

Step2:计算注意力分数

$$Attention(Q,K,V) = soft\max(\frac{QK^{T}}{\sqrt{d_k}})V \tag{5.32}$$

其中d_k是键向量的维度。

Step3:多头注意力

$$MultiHead(Q,K,V)=Concat(head_1,head_2,...,head_h)W^o \tag{5.33}$$

其中$head_i$=Attention(QW_i^Q, KW_i^K, VW_i^V)。

由于本题设计的任务是序列到序列，故采用 BLEU（Bilingual Evaluation Understudy）来评价 Transformer，参考文本直接基于外部数据集 2 中的实际工作与实际序列组成即可[12]。

2.实验及评价

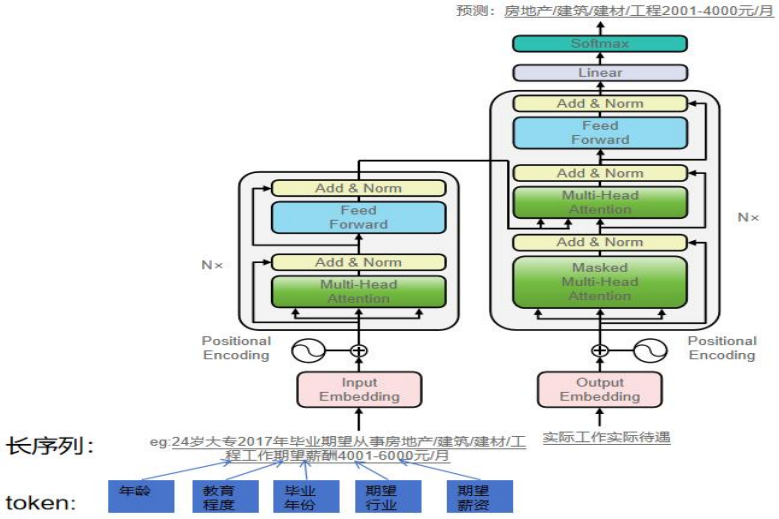

图 5-16 人岗匹配-Transformer 模型架构

由图 5-16 所示，首先将数据原本的五个特征当成 token，将这些 token 组成一段序列输入给 transformer 模型，比如，从一个样本中提取五个特征形成 24 岁大专 2017 年毕业期望从事房地产/建筑/建材/工程工作期望薪酬 4001-6000 元/月一段序列，之后就是一个完整的 Transformer 架构。最后经过 Softmax 函数，预测一段序列为房地产/建筑/建材/工程 2001-4000 元/月，显示该失业人员可以从事房地产、建筑、建材行业，工资在 2001-4000 这个区间，从该人员的个人信息来，这是两个输出特征实际工作和实际薪酬合成的生成序列，实际显示模型推荐的岗位和应得的薪酬是合理的。

总的来说，本文将 Transformer 模型应用于人岗匹配问题，充分发挥了 Transformer 架构的多项优势。通过将求职者的年龄、教育程度、毕业年份、期望行业和薪资等多维特征转化为序列化的文本输入，利用了 Transformer 的自注意力机制来捕获不同特征之间的复杂关联。模型的多头注意力机制（8 个注意力头）使其能够同时从多个角度理解输入特征之间的依赖关系，而四层 Transformer 编码器的深度架构则确保了模型具有足够的能力来学习特征之间的非线性映射。项目还采用了位置编码来保持特征的顺序信息，并通过 BLEU 评分和指标来全面评估模型性能，BLEU（计算公式如下）水平显示为 0.636，表现显著，可以适应大多数岗位推荐系统预测。又基于实际工作状态和推荐工作状态相似度比较，最后得出所提出模型人岗匹配度达到 0.86。

$$BLEU = BP\cdot\exp(\sum_{n=1}^{N} w_n \log \Pr ecision_n) \tag{5.34}$$

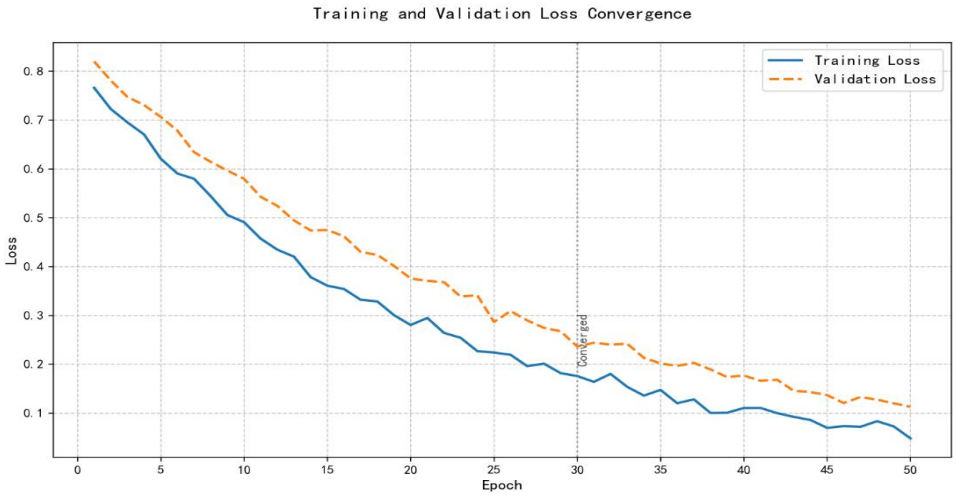

图 5-17 损失曲线

最后，从图 5-17 来看，模型展现出了良好的学习和收敛特性。训练损失（蓝色实线）和验证损失（橙色虚线）都呈现平稳下降趋势，从初始的约 0.8 逐步降低到 50 个 epoch 后的 0.1 左右，表明模型在训练过程中持续有效地学习到了数据中的特征模式。值得注意的是，验证损失虽然始终略高于训练损失，但两条曲线的走势基本平行且间隔较小，没有出现明显的过拟合现象，这说明模型具有良好的泛化能力。特别是在 30 个 epoch 之后，虽然学习速度有所放缓，但损失值仍在缓慢下降，这表明模型的训练策略设置合理，既保证了充分的学习又避免了训练不足或过度拟合的问题。

六、模型评价

6.1 模型优点

6.1.1 数据处理与特征工程的科学性

多层级数据清洗策略：通过剔除冗余特征、分类变量编码、缺失值处理及低方差特征筛选，显著提升数据质量，降低噪声干扰，为建模奠定高效基础。

多源特征融合创新：问题三中引入宏观经济、劳动力市场等 24 个外部指标，基于毕业日期通过指数加权移动平均（EWMA）构建动态特征，有效捕捉个体就业与宏观环境的时序关联，增强模型解释力。

6.1.2 算法改进与模型设计的针对性

不平衡数据处理能力：问题二中提出的 LINEX-SVM 模型通过非对称损失函数调整误分类惩罚，在准确率（0.8775）、F1 值（0.9345）等指标上显著优于传统 SVM 及重采样方法，尤其提升了对少数类（失业）样本的召回率（0.9966）。

多模块集成学习优势：问题三中 MSPL 算法通过正则化自步学习框架，平衡各模块（个人信息、经济指标等）的样本选择与权重分配，基于少数类准确率动态调整集成权重，使模型对失业样本的识别准确率（0.75）远超单一模块（最高 0.45）。

6.1.3 模型可解释性与可视化的有效性

特征影响深度分析：结合 XGBoost、SHAP 值实现特征重要性排序与交互分析，明确毕业日期、户口所在地等核心影响因素，SHAP 热力图直观展示特征间正负向交互效应，增强模型透明度。

结果可视化多样性：通过卡方检验结果表、混淆矩阵、特征分布条形图等多种图表，清晰呈现各维度特征与就业状态的关联，便于政策制定者理解关键影响路径。

6.2 模型缺点

6.2.1 算法复杂度与计算成本

参数调优的高耗时性：LINEX-SVM 的网格搜索、MSPL 算法的超参数（λ、γ、模块权重）优化需大量迭代计算，尤其在多模块集成时，计算资源消耗显著增加。

复杂模型的部署门槛：Transformer 模型的多层编码器解码器架构、SHAP 可解释性分析对硬件算力要求较高，在实时预测场景中可能面临效率瓶颈。

6.2.2 模型泛化不足

模型基于宜昌市特定数据训练，地区产业结构（如传统产业转型压力）、政策导向（如公共就业服务体系）具有独特性，直接迁移至其他城市可能因环境差异导致性能下降。

6.2.3 在线学习能力缺失

当前模型为离线训练，缺乏对新增数据（如实时招聘信息、政策调整）的自动更新机制，长期使用可能因数据分布漂移导致预测偏差。

参考文献

[1] 石郑.高质量充分就业的生成逻辑、价值意蕴与实践路径[N].河南经济报,2024-11-05(012).DOI:10.28362/n.cnki.nhncx.2024.002098.

[2] 魏立红.宜昌市：聚焦群众需求创新公共就业服务[J].中国就业,2024,(12):28-29.DOI:10.16622/j.cnki.11-3709/d.2024.12.002.

[3] 张俊丽,常艳丽,师文.标签传播算法理论及其应用研究综述[J].计算机应用研究,2013,30(01):21-25.

[4] 房祥忠.卡方分布与卡方检验[J].中国统计,2022,(05):29-31.

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

[6] 刘天畅,王雷,朱庆华.基于 SHAP 解释方法的智慧居家养老服务平台用户流失预测研究[J].数据分析与知识发现,2024,8(01):40-54.

[7] 丁世飞,齐丙娟,谭红艳.支持向量机理论与算法研究综述[J].电子科技大学学报,2011,40(01):2-10.

[8] 张颖,刘金泉,周光亚.LINEX 损失函数下正态均值的估计问题[J].吉林大学自然科学学报,1994,(04):35-38.

[9] 董田田,董学士,张睿,等.基于指数加权移动平均法的企业销量预测[J].青岛大学学报(自然科学版),2020,33(04):50-54.

[10] 郭西风.基于深度神经网络的图像聚类算法研究[D].国防科技大学,2020.DOI:10.27052/d.cnki.gzjgu.2020.000056.

[11] 任欢,王旭光.注意力机制综述[J].计算机应用,2021,41(S1):1-6.

[12] 刘世界.涉海翻译中的机器翻译应用效能：基于 BLEU、chrF++和 BERTScore 指标的综合评估[J].中国海洋大学学报(社会科学版),2024,(02):21-31.DOI:10.16497/j.cnki.1672-335X.202402003.

[13] 桑应宾.基于 K 近邻的分类算法研究[D].重庆大学,2009.

[14] Y. Ma, Q. Zhang, D. Li and Y. Tian, "LINEX Support Vector Machine for Large-Scale Classification," in IEEE Access, vol. 7, pp. 70319-70331, 2019, doi: 10.1109/ACCESS.2019.2919185. keywords: {Support vector machines;Fasteners;Convex functions;Optimization;Linear programming;Numerical models;LINEX loss;large-scale classification;support vector machine (SVM)},

[15] Yu J ,Liu Y ,Liang W , et al.A framework for overlapping and non-overlapping communities detection based on seed extension and label propagation[J].Physica A: Statistical Mechanics and its Applications,2025,660130362-130362.

[16] Pouria M ,Sadegh A ,Abbas R , et al.Culvert Inspection Framework Using Hybrid XGBoost and Risk-Based Prioritization: Utah Case Study[J].Journal of Construction Engineering and Management,2025,151(6):

[17] Olan F ,Spanaki K ,Ahmed W , et al.Enabling explainable artificial intelligence capabilities in supply chain decision support making[J].Production Planning & Control,2025,36(6):808-819.

[18] Waghmode P ,Kanumuri M ,Ocla E H , et al.Intrusion detection system based on machine learning using least square support vector machine[J].Scientific

Reports,2025,15(1):12066-12066.
[19] Neammai J ,Sukparungsee S ,Areepong Y .Explicit Analytical Form for the Average Run Length of Double-Modified Exponentially Weighted Moving Average Control Charts Through the MA(q) Process and Applications[J].Symmetry,2025,17(2):238-238.
[20] Ji X ,Cheng X ,Zhou P .Self-paced learning for anchor-based multi-view clustering: A progressive approach[J].Neurocomputing,2025,635129921-129921.

附录

支撑材料的文件列表

1. 作品 Word 文档
2. 外部数据 1
3. 外部数据 2
4. 源代码.py

问题一代码

```
#数据预处理
import pandas as pd
df = pd.read_excel('C:/Users/86151/Desktop/华中杯/附件 1 数据-C 题.xls', sheet_name='数据集')
print(df.dtypes)
df.shape
df.isnull().sum()
import missingno as msno
import matplotlib.pyplot as plt
# 设置中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
# 可视化缺失值矩阵
msno.matrix(df)
plt.xticks(rotation=20)
plt.title('Missing Values Matrix')
plt.show()

# 可视化缺失值条形图
msno.bar(df)
plt.xticks(rotation=0)
plt.title('Missing Values Bar Chart')
plt.show()
print(df.isnull().mean() < 0.5)
# 删除缺失率超过 50%的特征
data = df.loc[:, df.isnull().mean() < 0.5]
data.shape
# 使用 pandas 的 fillna 方法
for col in data.columns:
    if data[col].isnull().any():
        mode_val = data[col].mode()[0]  # 获取众数
        data[col] = data[col].fillna(mode_val)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# 设置中文显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 4. 自定义业务分组
def custom_business_binning(age_series):
    """
    自定义业务分组（示例：消费者分析常用分组）
    """
    bins = [0, 18, 25, 30, 40, 50, 60, np.inf]
    labels = [
        '18 岁以下',
        '18-24 岁(大学生)',
        '25-29 岁(职场新人)',
        '30-39 岁(职场骨干)',
        '40-49 岁(中年群体)',
        '50-59 岁(准退休)',
        '60 岁+(退休)'
    ]
    return pd.cut(age_series, bins=bins, labels=labels)

# 应用分组
data['消费群体'] = custom_business_binning(data['年龄'])
print("\n===== 消费群体分组结果 =====")
print(data['消费群体'].value_counts().sort_index())
# 保存分组后的数据
data.to_csv('C:/Users/86151/Desktop/华中杯/已清洗测试集.csv', index=False,
encoding='utf_8_sig')
print("\n 已保存分组结果到'年龄分组数据.csv'")
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("C:/Users/86151/Desktop/华中杯/已清洗数据(2).csv")
target_cols = df.iloc[:, 13:19]  # 第 14-19 列（Python 从 0 开始计数）

plt.figure(figsize=(18, 10))
for i, col in enumerate(target_cols.columns, 1):
    plt.subplot(2, 3, i)  # 2 行 3 列布局
    sns.countplot(data=target_cols,                                    x=col,
order=target_cols[col].value_counts().index)
    plt.title(col)
    plt.xticks(rotation=45)
plt.tight_layout()
```

```
plt.show()
#手工筛选和标签传播算法
import pandas as pd
import numpy as np
import os
from sklearn.semi_supervised import LabelPropagation
from sklearn.preprocessing import StandardScaler
import warnings
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import matplotlib.cm as cm
def fill_employment_status_two_steps():
    try:
        # 读取数据
        input_file = r"D:\C 题\清洗数据.xls"
        output_file = r"D:\两步法处理后的就业状态数据.csv"
        print(f"正在读取文件: {input_file}")
        # 读取 Excel 文件
        df = pd.read_excel(input_file)
        # 统计原始失业状态数据分布
        original_employed = sum(df["失业状态"] == 1)  # 就业
        original_unemployed = sum(df["失业状态"] == 0)  # 未就业
        original_unlabeled = sum(pd.isna(df["失业状态"]))  # 未知
        print(f"\n 原始数据中:")
        print(f"已标记就业样本数量: {original_employed}")
        print(f"已标记失业样本数量: {original_unemployed}")
        print(f"未标记样本数量: {original_unlabeled}")
        # 如果没有未标记的样本，则不需要继续处理
        ################################################################
        # 步骤 1: 基于规则填充部分未标记样本
        ################################################################
        # 处理日期类型列
        date_columns = ["合同终止日期", "失业时间", "就业时间", "失业信息登
记表登记日期"]
        # 检查日期列的数据类型
        for col in date_columns:
            if col in df_processed.columns:
                print(f"{col}列的数据类型: {df_processed[col].dtype}")
        # 创建一个函数来提取年月并转为数值方便比较
        def extract_year_month(date_str):
            if pd.isna(date_str) or not isinstance(date_str, str):
                return None
            # 清理字符串，去除非数字字符
            date_str = ''.join(filter(str.isdigit, date_str))
```

```
                if len(date_str) >= 6:
                    try:
                        year_month = int(date_str[:6])
                        return year_month
                    except ValueError:
                        return None
                elif len(date_str) >= 4:
                    try:
                        year = int(date_str[:4])
                        return year * 100 + 1
                    except ValueError:
                        return None
                return None
            for col in date_columns:
                # 将日期转换为字符串
                df_processed[col] = df_processed[col].astype(str).str.strip()
                # 提取数值化的年月
                df_processed[f'{col}_数值'] = df_processed[col].apply(extract_year_month)
            # 统计各年月的样本数量
            for col in date_columns:
                date_counts = df_processed[f'{col}_数值'].value_counts().sort_index()
                print(f"\n 各{col}的分布:")
                for date_val, count in date_counts.items():
                    if pd.notna(date_val):
                        print(f"  - {date_val}: {count} 个样本")
            # 规则 1: 标记合同终止日期在 2024 之后的未标记样本为 1（就业）
            print("\n 应用规则 1: 合同终止日期在 2024 之后的标记为就业(1)")
            mask_rule1 = (
                pd.isna(df_processed["失业状态"]) &
                (df_processed['合同终止日期_数值'] >= 202400)
            )
            df_processed.loc[mask_rule1, "失业状态"] = 1
            rule1_matches = mask_rule1.sum()
            # 标记填充方式
            df_processed.loc[mask_rule1, "填充方式"] = "规则 1:合同终止日期在 2024 之后(就业)"
            print(f"规则 1 填充: {rule1_matches} 个样本被标记为就业(1)")
            # 规则 2: 失业时间晚于合同终止日期的未标记样本标记为 0（失业）
            print("\n 应用规则 2: 失业时间晚于合同终止日期的标记为失业(0)")
            mask_rule2 = (
                pd.isna(df_processed["失业状态"]) &
                pd.notna(df_processed['失业时间_数值']) &
                pd.notna(df_processed['合同终止日期_数值']) &
```

```
            (df_processed['失业时间_数值'] > df_processed['合同终止日期_数值
'])
        )
        df_processed.loc[mask_rule2, "失业状态"] = 0
        rule2_matches = mask_rule2.sum()
        # 标记填充方式
        df_processed.loc[mask_rule2, "填充方式"] = "规则 2:失业时间晚于合同
终止日期(失业)"
        print(f"规则 2 填充: {rule2_matches} 个样本被标记为失业(0)")
        # 规则 3: 就业时间等于失业信息登记表登记日期的未标记样本标记为
1（就业）
        print("\n 应用规则 3: 就业时间等于失业信息登记表登记日期的标记为
就业(1)")
        mask_rule3 = (
            pd.isna(df_processed["失业状态"]) &
            pd.notna(df_processed['就业时间_数值']) &
            pd.notna(df_processed['失业信息登记表登记日期_数值']) &
            (df_processed['就业时间_数值'] == df_processed['失业信息登记表登
记日期_数值'])
        )
        df_processed.loc[mask_rule3, "失业状态"] = 1
        rule3_matches = mask_rule3.sum()

        # 标记填充方式
        df_processed.loc[mask_rule3, "填充方式"] = "规则 3:就业时间等于失业
信息登记表登记日期(就业)"

        print(f"规则 3 填充: {rule3_matches} 个样本被标记为就业(1)")

        # 统计步骤 1 后的结果
        after_step1_employed = sum(df_processed["失业状态"] == 1)  # 就业
        after_step1_unemployed = sum(df_processed["失业状态"] == 0)  # 未就
业
        after_step1_unlabeled = sum(pd.isna(df_processed["失业状态"]))  # 仍然
未知

        print(f"\n 步骤 1 处理后:")
        print(f"就业样本数量: {after_step1_employed}")
        print(f"失业样本数量: {after_step1_unemployed}")
        print(f"仍未标记样本数量: {after_step1_unlabeled}")
        print(f"总共填充: {rule1_matches + rule2_matches + rule3_matches} 个")

        ############################################################
        # 步骤 2: 使用标签传播算法填充剩余的未标记样本
```

```
        ################################################################
        if after_step1_unlabeled > 0:
            print("\n 步骤 2: 使用标签传播算法填充剩余未标记样本")

            # 为标签传播准备数据
            # 创建就业标签列: 1(就业), 0(失业), -1(未知)
            df_processed["就业标签"] = np.where(df_processed["失业状态"] ==
1, 1,
                                                np.where(pd.isna(df_processed[" 失 业
状态"]), -1, 0))

            # 准备特征 - 使用数据中的所有数值型列作为特征
            # 排除目标列和非数值列
            exclude_cols = ["就业标签", "失业状态", "填充方式"]
            date_cols = [col for col in df_processed.columns if col.endswith('_数
值')]

            # 获取所有数值型列
            numeric_cols = df_processed.select_dtypes(include=['int64',
'float64']).columns.tolist()
            # 排除指定列
            features = [col for col in numeric_cols if col not in exclude_cols]

            # 确保使用日期的数值版本作为特征
            for col in date_cols:
                if col not in features and col in df_processed.columns:
                    features.append(col)

            print(f"使用的特征列: {features}")

            # 如果没有足够的特征，则使用所有列进行编码
            if len(features) < 2:
                print("警告: 数值特征不足，将对所有非目标列进行编码")
                features = [col for col in df_processed.columns if col not in
exclude_cols and col != "就业标签"]
                # 对非数值特征进行编码
                for col in features:
                    if df_processed[col].dtype == 'object':
                        # 简单的标签编码
                        df_processed[f"{col}_encoded"] =
pd.factorize(df_processed[col])[0]

                # 更新特征列为编码后的列
                features = [f"{col}_encoded" for col in features if
```

```
df_processed[col].dtype == 'object'] + \
                                [col for col in features if df_processed[col].dtype != 
'object']
                # 提取特征和标签
                X = df_processed[features].values
                y = df_processed["就业标签"].values
                # 标准化特征
                scaler = StandardScaler()
                X_scaled = scaler.fit_transform(X)
                # 确认已标记的样本数量足够
                labeled_mask = (y != -1)
                labeled_count = sum(labeled_mask)
                classes_count = len(np.unique(y[labeled_mask]))
                print(f"已标记样本数量: {labeled_count}，类别数: {classes_count}")
                # 使用标签传播算法
                try:
                    # 创建标签传播模型 - 使用 KNN 内核，确保更稳定的结果
                    model = LabelPropagation(kernel='knn', n_neighbors=min(7, 
labeled_count // 2 if labeled_count > 0 else 7), max_iter=1000)
                    # 拟合模型
                    model.fit(X_scaled, y)
                    # 预测所有样本的标签
                    y_pred = model.predict(X_scaled)
                    y_pred_discrete = np.round(y_pred).astype(int)
                    # 确保预测值只有 0 或 1，避免出现其他值
                    y_pred_discrete = np.clip(y_pred_discrete, 0, 1)
                    # 只更新之前未标记的样本
                    unlabeled_mask = (y == -1)
                    df_processed.loc[unlabeled_mask, "失业状态"] = 
y_pred_discrete[unlabeled_mask]
                    step2_matches = unlabeled_mask.sum()
                    # 统计标签传播填充的结果
                    filled_employed = sum((y == -1) & (y_pred_discrete == 1))  # 
填充为就业
                    filled_unemployed = sum((y == -1) & (y_pred_discrete == 0))  # 
填充为失业
                    print(f"标签传播填充结果:")
                    print(f"  - 填充为就业(1)的样本数: {filled_employed}")
                    print(f"  - 填充为失业(0)的样本数: {filled_unemployed}")
                    # 标记填充方式
                    df_processed.loc[(unlabeled_mask) & (y_pred_discrete == 1), "
填充方式"] = "步骤 2:标签传播算法(就业)"
                    df_processed.loc[(unlabeled_mask) & (y_pred_discrete == 0), "
填充方式"] = "步骤 2:标签传播算法(失业)"
```

```
                    # 可视化标签传播结果
                    visualize_label_propagation(X_scaled, y, y_pred_discrete,
unlabeled_mask, output_file.replace('.csv', '_可视化.png'))
            # 统计最终结果
            final_employed = sum(df_processed["失业状态"] == 1)  # 就业
            final_unemployed = sum(df_processed["失业状态"] == 0)  # 未就业
            final_unlabeled = sum(pd.isna(df_processed["失业状态"]))  # 仍然未知
            print(f"\n 最终处理后的数据中:")
            print(f"就业样本数量: {final_employed}")
            print(f"失业样本数量: {final_unemployed}")
            print(f"仍未标记样本数量: {final_unlabeled}")
            # 添加总结统计
            print(f"\n 总填充数量统计:")
            print(f"步骤 1 (规则 1-合同终止日期在 2024 之后): {rule1_matches} 个")
            print(f"步骤 1 (规则 2-失业时间晚于合同终止日期): {rule2_matches} 个
")
            print(f"步骤 1 (规则 3-就业时间等于失业信息登记表登记日期):
{rule3_matches} 个")
            print(f"步骤 2 (标签传播算法): {step2_matches} 个")
            print(f"总共填充: {rule1_matches + rule2_matches + rule3_matches +
step2_matches} 个")
            # 保存结果
            print(f"\n 正在保存结果到: {output_file}")
            df_processed.to_csv(output_file, index=False, encoding='utf-8-sig')
            print("处理完成!")
            return df_processed
        except Exception as e:
            print(f"处理数据时出错: {str(e)}")
            import traceback
            traceback.print_exc()
            return None
def visualize_label_propagation(X, y_original, y_pred, unlabeled_mask, output_file):
        try:
            # 使用 PCA 将数据降维到 2D 进行可视化
            pca = PCA(n_components=2)
            X_2d = pca.fit_transform(X)
            # 创建一个大的图
            plt.figure(figsize=(18, 12))
            # 定义颜色和标记
            colors = {
                '已标记就业': 'blue',
                '已标记失业': 'red',
                '预测为就业': 'lightblue',
                '预测为失业': 'lightcoral'
```

```
        }
        # 设置中文字体
        try:
            plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标
签
            plt.rcParams['axes.unicode_minus'] = False      # 用来正常显示负号
        except:
            pass
        # 创建四个类别的掩码
        labeled_employed_mask = (y_original == 1) & (~unlabeled_mask)
        labeled_unemployed_mask = (y_original == 0) & (~unlabeled_mask)
        unlabeled_pred_employed_mask = (y_pred == 1) & (unlabeled_mask)
        unlabeled_pred_unemployed_mask = (y_pred == 0) & (unlabeled_mask)

        # 1. 绘制原始标记的样本
        plt.scatter(X_2d[labeled_employed_mask,                                0],
X_2d[labeled_employed_mask, 1],
                    c=colors['已标记就业'], marker='o', s=80, label='已标记就业
(1)', alpha=0.8, edgecolors='black')
        plt.scatter(X_2d[labeled_unemployed_mask,                              0],
X_2d[labeled_unemployed_mask, 1],
                    c=colors['已标记失业'], marker='o', s=80, label='已标记失业
(0)', alpha=0.8, edgecolors='black')

        # 2. 绘制通过标签传播预测的未标记样本
        plt.scatter(X_2d[unlabeled_pred_employed_mask,                         0],
X_2d[unlabeled_pred_employed_mask, 1],
                    c=colors['预测为就业'], marker='x', s=50, label='预测为就业
(1)', alpha=0.6)
        plt.scatter(X_2d[unlabeled_pred_unemployed_mask,                       0],
X_2d[unlabeled_pred_unemployed_mask, 1],
                    c=colors['预测为失业'], marker='x', s=50, label='预测为失业
(0)', alpha=0.6)
        # 设置图表标题和轴标签
        variance_ratio = pca.explained_variance_ratio_
        plt.title('标签传播结果可视化 (2D PCA 降维)', fontsize=16)
        plt.xlabel(f'主成分 1 (解释方差: {variance_ratio[0]:.2%})', fontsize=14)
        plt.ylabel(f'主成分 2 (解释方差: {variance_ratio[1]:.2%})', fontsize=14)
        # 添加图例
        plt.legend(fontsize=12, loc='best')
        # 添加网格
        plt.grid(alpha=0.3)
        # 设置坐标轴范围以确保所有点都在视图中
        plt.tight_layout()
```

```
        # 统计信息
        original_employed = sum(labeled_employed_mask)
        original_unemployed = sum(labeled_unemployed_mask)
        predicted_employed = sum(unlabeled_pred_employed_mask)
        predicted_unemployed = sum(unlabeled_pred_unemployed_mask)
        # 饼图：显示最终标签分布
        plt.figure(figsize=(10, 8))
        # 数据准备
        sizes = [original_employed, original_unemployed, predicted_employed, predicted_unemployed]
        labels = [
            f'原始标记就业: {original_employed}',
            f'原始标记失业: {original_unemployed}',
            f'预测为就业: {predicted_employed}',
            f'预测为失业: {predicted_unemployed}'
        ]
        colors_list = [colors['已标记就业'], colors['已标记失业'], colors['预测为就业'], colors['预测为失业']]
        # 绘制饼图
        plt.pie(sizes, labels=labels, colors=colors_list, autopct='%1.1f%%', shadow=True, startangle=140)
        plt.axis('equal')  # 确保饼图是圆的
        plt.title('失业状态类别分布', fontsize=16)
        # 保存所有图表到同一个文件
        plt.figure(figsize=(20, 10))
        # 创建子图
        plt.subplot(1, 2, 1)
        # 绘制散点图
        plt.scatter(X_2d[labeled_employed_mask, 0], X_2d[labeled_employed_mask, 1],
                    c=colors['已标记就业'], marker='o', s=80, label='已标记就业(1)', alpha=0.8, edgecolors='black')
        plt.scatter(X_2d[labeled_unemployed_mask, 0], X_2d[labeled_unemployed_mask, 1],
                    c=colors['已标记失业'], marker='o', s=80, label='已标记失业(0)', alpha=0.8, edgecolors='black')
        plt.scatter(X_2d[unlabeled_pred_employed_mask, 0], X_2d[unlabeled_pred_employed_mask, 1],
                    c=colors['预测为就业'], marker='x', s=50, label='预测为就业(1)', alpha=0.6)
        plt.scatter(X_2d[unlabeled_pred_unemployed_mask, 0], X_2d[unlabeled_pred_unemployed_mask, 1],
                    c=colors['预测为失业'], marker='x', s=50, label='预测为失业(0)', alpha=0.6)
```

```
            plt.title('标签传播结果 (2D PCA 降维)', fontsize=14)
            plt.xlabel(f'主成分 1 (解释方差: {variance_ratio[0]:.2%})', fontsize=12)
            plt.ylabel(f'主成分 2 (解释方差: {variance_ratio[1]:.2%})', fontsize=12)
            plt.legend(fontsize=10, loc='best')
            plt.grid(alpha=0.3)
        # 处理数据并填充缺失的失业状态
        processed_df = fill_employment_status_two_steps()
if __name__ == "__main__":
main()
#卡方检验
import pandas as pd
from scipy.stats import chi2_contingency
df = pd.read_excel('C:/Users/D/Desktop/华中杯/清洗数据(已标注）(1).xls')
target = '失业状态'  # 替换为你的目标列名
y = df[target]
# 存储每个特征的卡方统计量和 p 值
results = []
for feature in df.columns:
    if feature != target:  # 跳过目标列
        contingency_table = pd.crosstab(df[feature], y)  # 构建列联表
        chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)
        results.append({
            'Feature': feature,
            'Chi2_Statistic': chi2_stat,
            'P_value': p_val,
            'Degrees_of_Freedom': dof
        })

# 转换为 DataFrame 并排序（按卡方统计量降序）
results_df = pd.DataFrame(results).sort_values('Chi2_Statistic', ascending=False)
print(results_df)
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_excel('C:/Users/D/Desktop/华中杯/清洗数据(已标注）(1).xls')
target = '失业状态'  # 替换为你的目标列名

# 获取所有特征（除目标）
features = df.columns.drop(target)

# 设置子图布局：4 行 5 列（共 20 个子图）
fig, axes = plt.subplots(3, 7, figsize=(20, 16))
axes = axes.flatten()  # 将二维数组变为一维，方便索引
```

```
for i, feature in enumerate(features):
    ax = axes[i]
    # 仅取前 5 个频次最高的类别
    top_categories = df[feature].value_counts().nlargest(5).index
    subset = df[df[feature].isin(top_categories)]

    sns.countplot(
        data=subset,
        x=feature,
        hue=target,
        palette='Set2',
        order=top_categories,
        ax=ax
    )
    # 添加数值标签
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{int(height)}',
                    (p.get_x() + p.get_width() / 2, height),
                    ha='center', va='bottom', fontsize=8)

    ax.set_title(f'{feature}')
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')

# 调整布局
plt.tight_layout()
plt.show()
```

问题二代码

```
#特征筛选与预处理
# 读取 Excel 文件
file_path = r"C:\Users\D\Desktop\华中杯\1990-2023 宜昌市数据指标.xlsx"  # 替换为实际文件路径
xls = pd.ExcelFile(file_path)

# 读取指定工作表（这里是 Sheet1）
df = xls.parse('Sheet1')

# 计算每列缺失值占比
missing_ratio = df.isnull().mean()

# 删除缺失值比例超过 30% 的列
cleaned_df = df.loc[:, missing_ratio <= 0.3]
```

```
# 可选：导出为新的 Excel 文件
cleaned_df.to_excel(r"C:\Users\D\Desktop\华中杯\宜昌市数据筛选后.xlsx"), index=False)

import pandas as pd

# 1. 读取文件
file_path = r"C:\Users\D\Desktop\华中杯\宜昌市数据筛选后.xlsx"
xls = pd.ExcelFile(file_path)
df = xls.parse('Sheet1')

# 2. 删除缺失值超过 30%的列
missing_ratio = df.isnull().mean()
cleaned_df = df.loc[:, missing_ratio <= 0.3]

# 3. 排序（如果有时间列）
if '年份' in cleaned_df.columns:
    cleaned_df = cleaned_df.sort_values(by='年份')

# 4. 分离数值型与非数值型列
numeric_cols = cleaned_df.select_dtypes(include=['number']).columns
non_numeric_cols = cleaned_df.select_dtypes(exclude=['number']).columns

# 5. 对数值型列进行线性插值
cleaned_df[numeric_cols] = cleaned_df[numeric_cols].interpolate(method='linear', limit_direction='both')

# 6. 导出为 Excel
cleaned_df.to_excel(r"C:\Users\D\Desktop\华中杯\宜昌市数据插值后.xlsx", index=False)
# 读取文件路径
file_yichang =r"C:\Users\D\Desktop\华中杯\宜昌市数据筛选插值后(1).xlsx"
file_cleaned = r"C:\Users\D\Desktop\华中杯\清洗数据(已标注）(1).xls"

# 加载数据
df_yichang = pd.read_excel(file_yichang)
df_cleaned = pd.read_excel(file_cleaned)

# 确保年份是整数
df_yichang['年份'] = df_yichang['年份'].astype(int)

# 提取毕业年份
df_cleaned['毕业年份'] = df_cleaned['毕业日期'].astype(str).str[:4].astype(int)
```

```
# 需要使用的宏观变量列
features_to_use = [ '城镇非私营单位在岗职工平均工资(元)',
]

# 定义函数：为每一位毕业生计算其毕业年份之后的 EWMA 值
def compute_ewma_for_row(graduation_year, alpha=0.3):
    future_data = df_yichang[df_yichang['年份'] > graduation_year].sort_values('年份')
    if future_data.empty:
        return [None] * len(features_to_use)
    ewma_values = future_data[features_to_use].ewm(alpha=alpha, adjust=False).mean().iloc[-1]
    return ewma_values.tolist()

# 应用函数
ewma_features = df_cleaned['毕业年份'].apply(compute_ewma_for_row)

# 将结果拆分为新列并添加前缀
ewma_df = pd.DataFrame(ewma_features.tolist(), columns=[f'EWMA_{col}' for col in features_to_use])

# 合并回原始数据
df_augmented = pd.concat([df_cleaned, ewma_df], axis=1)

# 保存为新文件（可选）
df_augmented.to_excel(r"C:\Users\D\Desktop\华中杯\第四问数据处理 .xlsx", index=False)

import pandas as pd

# 读取 Excel 文件
file_path = r"C:\Users\D\Desktop\华中杯\第四问数据处理 .xlsx"
sheet1_data = pd.read_excel(file_path, sheet_name='Sheet3')

# 创建教育程度的映射字典
education_mapping = {
    20: '本科',
    10: '硕士',
    30: '大专'
}

# 应用映射字典到"教育程度"列
sheet1_data['教育程度'] = sheet1_data['教育程度'].map(education_mapping)
```

```
# 保存为新的 Excel 文件
output_file_path = r"C:\Users\D\Desktop\华中杯\问题四数据处理 .xlsx"
sheet1_data.to_excel(output_file_path, index=False)

#LINEX-FSVM 模型构建
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df = pd.read_excel('C:/Users/86151/Desktop/华中杯/清洗数据(已标注）(1).xls')
target = '失业状态'  # 替换为你的目标列名
y = df[target]

# 统计 y 中 0 和 1 的数量
class_counts = y.value_counts()

# 可视化分类情况
plt.figure(figsize=(6,4))

# 绘制条形图
ax = sns.countplot(x=y, palette='viridis')
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')

# 在每个条形图上标出数量
for p in ax.patches:
    ax.text(p.get_x() + p.get_width() / 2.,  # 横坐标（条形中心）
            p.get_height(),                  # 纵坐标（条形高度）
            f'{int(p.get_height())}',        # 显示的数值
            ha='center',                     # 水平对齐方式（居中）
            va='bottom',                     # 垂直对齐方式（底部）
            fontsize=10)                     # 字体大小

# 显示图表
plt.tight_layout()
plt.show()

import pandas as pd
from sklearn.model_selection import train_test_split

# 读取数据
df = pd.read_excel('C:/Users/86151/Desktop/华中杯/清洗数据(已标注）(1).xls')
```

```
# 分离有标注数据(前 4980 行)和待预测数据(最后 20 行)
df_labeled = df.iloc[:4980]  # 有 y 值的数据
df_to_predict = df.iloc[-20:]  # 需要预测的最后 20 行数据

# 划分特征和目标变量（仅对有标注数据）
X = df_labeled.iloc[:, :-1]  # 前 18 列作为特征
y = df_labeled.iloc[:, -1]   # 最后一列作为目标变量

# 按 4:1 比例划分训练集和测试集 (test_size=0.2 对应 4:1 比例)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42  # random_state 保证可重复性
)

# 提取待预测数据的特征（最后 20 行，不含 y 列）
X_predict = df_to_predict.iloc[:, :-1]

# 打印各数据集形状以验证
print(f"训练集形状: {X_train.shape}，测试集形状: {X_test.shape}")
print(f"待预测数据形状: {X_predict.shape}")

# 1. 用(X_train, y_train)训练模型
# 2. 用 X_test 测试模型性能
# 3. 用训练好的模型预测 X_predict
# 计算 G-mean 的函数
def geometric_mean_score(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()
    specificity = tn / (tn + fp)
    sensitivity = tp / (tp + fn)
return np.sqrt(specificity * sensitivity)
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, log_loss, confusion_matrix,
    classification_report, roc_curve, auc
)
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from imblearn.metrics import geometric_mean_score  # 导入 geometric_mean_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# 设置中文字体
```

```
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False      # 解决负号显示问题

# 定义 SVM 的参数网格
param_grid = {
      'C': [1,5,10],
      'kernel': ['linear', 'rbf', 'poly'],
      'gamma': ['scale', 'auto']
}

# 训练模型并评估（不使用重采样方法）
print("\n=== 不使用重采样方法 ===")

# 使用网格搜索调参
model = SVC(random_state=42, probability=True)  # 设置 probability=True 以启用 predict_proba
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid_search.fit(X_train, y_train)  # 直接使用原始训练集

# 输出最佳参数
print("最佳参数:", grid_search.best_params_)

# 使用最佳参数的模型进行预测
best_model = grid_search.best_estimator_
test_predictions = best_model.predict(X_test)
test_pred_proba = best_model.predict_proba(X_test)[:, 1]  # 预测为类别 1 的概率

# 计算评价指标
test_accuracy = accuracy_score(y_test, test_predictions)
test_confusion_matrix = confusion_matrix(y_test, test_predictions)
test_classification_report = classification_report(y_test, test_predictions)

# 计算所有评价指标
test_precision = precision_score(y_test, test_predictions)
test_recall = recall_score(y_test, test_predictions)
test_f1 = f1_score(y_test, test_predictions)
test_gmean = geometric_mean_score(y_test, test_predictions)

# 计算加权对数损失
weights_test = np.where(y_test == 1, 2, 1)  # 如果 y_test=1，权重为 2；否则为 1
test_logloss = log_loss(y_test, test_pred_proba, sample_weight=weights_test)

# 输出结果
```

```
print(f"测试集准确率: {test_accuracy:.4f}")
print(f"测试集 precision: {test_precision:.4f}")
print(f"测试集 recall: {test_recall:.4f}")
print(f"测试集 gmean: {test_gmean:.4f}")
print(f"测试集混淆矩阵:\n{test_confusion_matrix}")
print(f"测试集分类报告:\n{test_classification_report}")
print(f"测试集加权对数损失 (Logloss): {test_logloss:.4f}")
print(f"测试集 F1: {test_f1:.4f}")
# 计算 ROC 曲线和 AUC
fpr, tpr, _ = roc_curve(y_test, test_pred_proba)
roc_auc = auc(fpr, tpr)

# 绘制 ROC 曲线
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC 曲线 (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label='随机分类器')
plt.xlabel('假正例率 (FPR)')
plt.ylabel('真正例率 (TPR)')
plt.title('ROC 曲线')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

# 绘制混淆矩阵图
plt.figure(figsize=(6, 6))
sns.heatmap(test_confusion_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['预测 0', '预测 1'], yticklabels=['真实 0', '真实 1'])
plt.xlabel('预测标签')
plt.ylabel('真实标签')
plt.title('混淆矩阵')
plt.show()

from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, log_loss, confusion_matrix,
                             classification_report, roc_curve, auc)
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek
from imblearn.metrics import geometric_mean_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False      # 解决负号显示问题

# 定义重采样方法
resampling_methods = {
    'SMOTE': SMOTE(random_state=42),
    'RUS': RandomUnderSampler(random_state=42),
    'ROS': RandomOverSampler(random_state=42),
    'SMOTE+Tomek': SMOTETomek(random_state=42)
}

# 定义 SVM 的参数网格
param_grid = {
    'C': [10],
    'kernel': ['rbf'],
    'gamma': ['scale']
}

# 创建 2x2 的子图布局
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()  # 将 2x2 的轴展平为一维数组

# 训练模型并评估
for i, (method_name, resampler) in enumerate(resampling_methods.items()):
    print(f"\n=== 使用 {method_name} 重采样方法 ===")

    # 重采样训练集
    X_train_resampled, y_train_resampled = resampler.fit_resample(X_train,
y_train)

    # 使用网格搜索调参
    model = SVC(random_state=42, probability=True)  # 设置 probability=True
以启用 predict_proba
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='f1', n_jobs=-1)
    grid_search.fit(X_train_resampled, y_train_resampled)

    # 输出最佳参数
    print("最佳参数:", grid_search.best_params_)

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

```
    test_pred_proba = best_model.predict_proba(X_test)[:, 1]  # 预测为类别 1 的概率

    # 计算评价指标
    test_accuracy = accuracy_score(y_test, test_predictions)
    test_confusion_matrix = confusion_matrix(y_test, test_predictions)
    test_classification_report = classification_report(y_test, test_predictions)
    test_precision = precision_score(y_test, test_predictions)
    test_recall = recall_score(y_test, test_predictions)
    test_f1 = f1_score(y_test, test_predictions)
    test_gmean = geometric_mean_score(y_test, test_predictions)
    weights_test = np.where(y_test == 1, 2, 1)  # 如果 y_test=1，权重为 2；否则为 1
    test_logloss = log_loss(y_test, test_pred_proba, sample_weight=weights_test)

    # 输出结果
    print(f"测试集准确率: {test_accuracy:.4f}")
    print(f"测试集 Precision: {test_precision:.4f}")
    print(f"测试集 Recall: {test_recall:.4f}")
    print(f"测试集 F1: {test_f1:.4f}")
    print(f"测试集 G-Mean: {test_gmean:.4f}")
    print(f"测试集混淆矩阵:\n{test_confusion_matrix}")
    print(f"测试集分类报告:\n{test_classification_report}")
    print(f"测试集加权对数损失 (Logloss): {test_logloss:.4f}")

    # 计算 ROC 曲线和 AUC
    fpr, tpr, _ = roc_curve(y_test, test_pred_proba)
    roc_auc = auc(fpr, tpr)

    # 绘制 ROC 曲线（在对应的子图中）
    ax = axes[i]
    ax.plot(fpr, tpr, color='blue', lw=2, label=f'ROC (AUC = {roc_auc:.2f})')
    ax.plot([0, 1], [0, 1], color='gray', linestyle='--', label='随机分类器')
    ax.set_xlabel('假正例率 (FPR)')
    ax.set_ylabel('真正例率 (TPR)')
    ax.set_title(f'{method_name} 的 ROC 曲线')
    ax.legend(loc='lower right')
    ax.grid(True)

# 调整子图间距并显示
plt.tight_layout()
plt.show()

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
```

```
log_loss
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
import numpy as np

# Define the SVM class with linex loss
class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, gamma=1.0,
n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.gamma = gamma
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)  # Convert labels to -1 and 1
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                a_i = 1 - y_[idx] * (np.dot(x_i, self.w) - self.b)
                e_gamma_a = np.exp(self.gamma * a_i)
                dL_dw = self.gamma * y_[idx] * x_i * (1 - e_gamma_a)
                dL_db = self.gamma * y_[idx] * (e_gamma_a - 1)
                self.w -= self.lr * (2 * self.lambda_param * self.w + dL_dw)
                self.b -= self.lr * dL_db

        # Check for NaN in weights and bias
        if np.isnan(self.w).any() or np.isnan(self.b):
            raise ValueError("模型训练过程中产生了 NaN 值")

    def predict(self, X):
        linear_output = np.dot(X, self.w) - self.b
        return np.sign(linear_output)

    def predict_proba(self, X):
        # Estimate probabilities using Platt Scaling (logistic transformation)
        linear_output = np.dot(X, self.w) - self.b
        linear_output = np.clip(linear_output, -500, 500)  # Avoid overflow
        prob_positive = 1 / (1 + np.exp(-linear_output))
```

```
            prob_negative = 1 - prob_positive
            return np.column_stack((prob_negative, prob_positive))
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
import numpy as np

class LinexSVM(BaseEstimator, ClassifierMixin):
    def __init__(self, learning_rate=0.001, lambda_param=0.01, gamma=1.0,
n_iters=1000, epsilon=1e-8):
            self.lr = learning_rate
            self.lambda_param = lambda_param
            self.gamma = gamma
            self.n_iters = n_iters
            self.epsilon = epsilon    # Small constant for numerical stability
            self.w = None
            self.b = None

    def _linex_loss(self, a):
            """Compute Linex loss and its derivative"""
            exp_term = np.exp(self.gamma * a)
            loss = exp_term - self.gamma * a - 1
            derivative = self.gamma * (exp_term - 1)
            return loss, derivative

    def fit(self, X, y):
            # Input validation
            X, y = check_X_y(X, y)
            n_samples, n_features = X.shape

            # Convert labels to -1 and 1
            y_ = np.where(y <= 0, -1, 1)

            # Initialize parameters
            self.w = np.zeros(n_features)
            self.b = 0.0

            # Training loop
            for _ in range(self.n_iters):
                for idx, x_i in enumerate(X):
                    # Compute margin
                    margin = y_[idx] * (np.dot(x_i, self.w) - self.b)
                    a = 1 - margin
```

```
                        # Compute Linex loss derivative
                        _, dL = self._linex_loss(a)

                        # Update gradients
                        grad_w = (2 * self.lambda_param * self.w +
                                  y_[idx] * x_i * dL / n_samples)
                        grad_b = -y_[idx] * dL / n_samples

                        # Update parameters with gradient clipping
                        self.w -= np.clip(self.lr * grad_w, -1, 1)
                        self.b -= np.clip(self.lr * grad_b, -1, 1)

                        # Numerical stability check
                        if np.any(np.isnan(self.w)) or np.isnan(self.b):
                            raise ValueError("Numerical instability detected")

        return self

    def decision_function(self, X):
        check_is_fitted(self)
        X = check_array(X)
        return np.dot(X, self.w) - self.b

    def predict(self, X):
        scores = self.decision_function(X)
        return np.where(scores >= 0, 1, 0)

    def predict_proba(self, X):
        """Platt scaling for probability estimation"""
        scores = self.decision_function(X)
        # Clip scores to prevent overflow
        scores = np.clip(scores, -500, 500)
        prob_positive = 1.0 / (1.0 + np.exp(-scores))
        return np.vstack([1 - prob_positive, prob_positive]).T

# Optimized hyperparameter search
def optimize_linex_svm(X_train, y_train, n_splits=5):
    param_grid = {
        'learning_rate': [ 0.01],
        'lambda_param': [00.1],
        'gamma': [1.0]
    }
```

```
    best_score = -np.inf
    best_params = {}
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)

    for lr in param_grid['learning_rate']:
        for lam in param_grid['lambda_param']:
            for gamma_val in param_grid['gamma']:
                fold_scores = []

                for train_idx, val_idx in kf.split(X_train):
                    X_fold, X_val = X_train[train_idx], X_train[val_idx]
                    y_fold, y_val = y_train[train_idx], y_train[val_idx]

                    try:
                        model = LinexSVM(learning_rate=lr,
                                         lambda_param=lam,
                                         gamma=gamma_val)
                        model.fit(X_fold, y_fold)
                        y_pred = model.predict(X_val)
                        score = accuracy_score(y_val, y_pred)
                        fold_scores.append(score)
                    except Exception as e:
                        print(f"Failed with params lr={lr}, lambda={lam},
gamma={gamma_val}: {str(e)}")
                        fold_scores.append(0)
                        continue

                mean_score = np.mean(fold_scores)
                if mean_score > best_score:
                    best_score = mean_score
                    best_params = {
                        'learning_rate': lr,
                        'lambda_param': lam,
                        'gamma': gamma_val
                    }

                print(f"lr={lr:.4f}, λ ={lam:.3f}, γ ={gamma_val:.1f} => Acc:
{mean_score:.4f}")

    print("\nBest Parameters:", best_params)
    print("Best CV Accuracy:", best_score)
    return best_params

# Usage example:
```

```
# best_params = optimize_linex_svm(X_train, y_train)
# final_model = LinexSVM(**best_params)
# final_model.fit(X_train, y_train)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, log_loss, confusion_matrix,
                             classification_report, roc_curve, auc)
from imblearn.metrics import geometric_mean_score

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用 SimHei 字体
plt.rcParams['axes.unicode_minus'] = False      # 解决负号显示问题
best_params = optimize_linex_svm(X_train, y_train)
final_model = LinexSVM(**best_params)
final_model.fit(X_train, y_train)

# Make predictions on the test set
test_predictions = final_model.predict(X_test)

# Convert predictions to binary (0 or 1)
test_predictions = np.where(test_predictions == 1, 1, 0)

# 计算所有评价指标
test_accuracy = accuracy_score(y_test, test_predictions)
test_confusion_matrix = confusion_matrix(y_test, test_predictions)
test_classification_report = classification_report(y_test, test_predictions)
test_precision = precision_score(y_test, test_predictions)
test_recall = recall_score(y_test, test_predictions)
test_f1 = f1_score(y_test, test_predictions)
test_gmean = geometric_mean_score(y_test, test_predictions)

# 计算加权对数损失
weights_test = np.where(y_test == 1, 2, 1)  # 如果 y_test=1，权重为 2；否则为 1
test_pred_proba = final_model.predict_proba(X_test)[:, 1]  # 预测为类别 1 的概率
test_logloss = log_loss(y_test, test_pred_proba, sample_weight=weights_test)

# 输出结果
print("="*50)
print("模型最终评估结果:")
print("="*50)
print(f"测试集准确率: {test_accuracy:.4f}")
```

```
print(f"测试集 Precision: {test_precision:.4f}")
print(f"测试集 Recall: {test_recall:.4f}")
print(f"测试集 F1-score: {test_f1:.4f}")
print(f"测试集 G-Mean: {test_gmean:.4f}")
print("\n 测试集混淆矩阵:")
print(test_confusion_matrix)
print("\n 测试集分类报告:")
print(test_classification_report)
print(f"\n 测试集加权对数损失 (Logloss): {test_logloss:.4f}")

# 创建可视化图表
plt.figure(figsize=(15, 5))

# 1. ROC 曲线
plt.subplot(1, 2, 1)
fpr, tpr, _ = roc_curve(y_test, test_pred_proba)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC 曲线 (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('假正例率 (FPR)')
plt.ylabel('真正例率 (TPR)')
plt.title('受试者工作特征曲线 (ROC)')
plt.legend(loc="lower right")

# 2. 混淆矩阵热力图
plt.subplot(1, 2, 2)
sns.heatmap(test_confusion_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['预测负类', '预测正类'],
            yticklabels=['真实负类', '真实正类'])
plt.xlabel('预测标签')
plt.ylabel('真实标签')
plt.title('混淆矩阵')

plt.tight_layout()
plt.show()
#对最后 20 行数据进行预测
X_predict = df_to_predict.iloc[:, :-1]  # 根据你的数据结构调整
predictions = final_model.predict(X_predict)
predict_proba = final_model.predict_proba(X_predict)[:, 1]
print("\n=== 最后 20 行数据的预测结果 ===")
print("预测类别:", predictions)
```

```
print("预测概率(类别 1):", predict_proba)
#XGBoost 和 shap 分析
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import shap
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
# 读取数据
file_path = r"D:\C 题\影响因素分析.xls"
df = pd.read_excel(file_path)
# 查看数据基本信息
print("数据形状:", df.shape)
print("数据概览:")
print(df.head())
# 分离特征和标签
X = df.iloc[:, :9]  # 前 9 列为特征
y = df.iloc[:, 9]   # 最后一列为标签（失业状态）
# 特征名称
feature_names = X.columns.tolist()
print("特征列表:", feature_names)
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 创建 XGBoost 分类器
model = xgb.XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=4,
    min_child_weight=2,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='binary:logistic',
    random_state=42
)

# 训练模型
model.fit(X_train, y_train)
# 预测
```

```
y_pred = model.predict(X_test)
# 评估模型
accuracy = accuracy_score(y_test, y_pred)
print(f"模型准确率: {accuracy:.4f}")
print("\n 分类报告:")
print(classification_report(y_test, y_pred))
# 获取特征重要性
plt.figure(figsize=(10, 6))
xgb.plot_importance(model, height=0.8, title='XGBoost 特征重要性')
plt.tight_layout()
plt.savefig('xgboost_feature_importance.png')
plt.show()
# SHAP 值分析
# 创建 SHAP 解释器
explainer = shap.Explainer(model)
shap_values = explainer(X)
# 可视化 SHAP 值
plt.figure(figsize=(12, 8))
shap.plots.beeswarm(shap_values, show=False)
plt.title('特征 SHAP 值分布', fontsize=16)
plt.tight_layout()
plt.savefig('shap_beeswarm.png')
plt.show()
# 特征汇总图
plt.figure(figsize=(12, 8))
shap.plots.bar(shap_values, show=False)
plt.title('SHAP 特征重要性汇总', fontsize=16)
plt.tight_layout()
plt.savefig('shap_feature_importance.png')
plt.show()
# 单个特征的 SHAP 依赖图
# 获取最重要的特征
feature_importance = model.feature_importances_
most_important_feature_idx = np.argmax(feature_importance)
most_important_feature = feature_names[most_important_feature_idx]
plt.figure(figsize=(12, 8))
shap.plots.scatter(shap_values[:, most_important_feature], show=False)
plt.title(f'特征"{most_important_feature}"的 SHAP 依赖图', fontsize=16)
plt.tight_layout()
plt.savefig(f'shap_dependence_{most_important_feature}.png')
plt.show()
# 热力图，显示特征交互
plt.figure(figsize=(14, 10))
shap.plots.heatmap(shap_values, show=False)
```

```
plt.title('SHAP 特征交互热力图', fontsize=16)
plt.tight_layout()
plt.savefig('shap_heatmap.png')
plt.show()
# 决策图，可视化模型的决策路径
plt.figure(figsize=(20, 12))
shap.plots.decision(shap_values[sample_idx], show=False)
plt.title(f'样本 #{sample_idx} 的决策路径', fontsize=16)
plt.tight_layout()
plt.savefig(f'shap_decision_sample_{sample_idx}.png')
plt.show()
# 保存特征的 SHAP 值结果到 Excel
shap_df = pd.DataFrame(shap_values.values, columns=feature_names)
shap_abs_mean = pd.DataFrame({
    '特征': feature_names,
    'SHAP 值(绝对平均)': np.abs(shap_df).mean().values
})
shap_abs_mean = shap_abs_mean.sort_values('SHAP 值(绝对平均)', ascending=False)
shap_abs_mean.to_excel('shap_feature_importance.xlsx', index=False)
```

问题三代码

```
#数据预处理
import pandas as pd
# Load the Excel file
file_path = r"C:\Users\D\Desktop\华中杯\问四数据处理 .xlsx"
excel_data = pd.ExcelFile(file_path)

# Load the data from the first sheet
data = pd.read_excel(file_path, sheet_name='Sheet1')

# Define salary ranges and their corresponding labels
salary_ranges = [
    (0, 1000, '0000001000'),
    (1001, 2000, '0100002000'),
    (2001, 4000, '0200104000'),
    (4001, 6000, '0400106000'),
    (6001, 8000, '0600108000'),
    (8001, 10000, '0800110000'),
    (10001, 150000, '100001150000'),
    (10001, 15000, '1000115000'),
    (15001, 20000, '1500120000'),
    (15001, 25000, '1500125000'),
    (20001, 30000, '2000130000'),
    (25001, 199999, '2500199999'),
```

```
    (30001, 50000, '3000150000'),
    (35001, 50000, '3500150000'),
    (50001, 70000, '5000170000'),
    (70001, 100000, '70001100000'),
    (25001, 35000, '2500135000')
]

# Function to classify salary based on the ranges
def classify_salary(salary):
    for min_salary, max_salary, label in salary_ranges:
        if min_salary <= salary <= max_salary:
            return label
    return '未知'  # For unexpected values

# Apply the classification to the dataset
data['薪资分类'] = data['期望薪资'].apply(classify_salary)

# Save the updated dataframe with the salary classification to a new Excel file
output_file_path = r"C:\Users\D\Desktop\华中杯\问题四数据处理 .xlsx"
data.to_excel(output_file_path, index=False)
# GAN 和多模块自步学习
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt
from tqdm import tqdm
import scipy.io as sio
import os
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
# GAN
class Generator(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(Generator, self).__init__()
        # 定义一个三层的全连接网络
        self.model = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.LeakyReLU(0.2, inplace=True),
```

```
                nn.Linear(128, 256),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Linear(256, output_dim),
                nn.Tanh()  # 使用 Tanh 确保输出在[-1, 1]范围，适合标准化后的特征
            )
        def forward(self, z):
            return self.model(z)
class Discriminator(nn.Module):
        def __init__(self, input_dim):
            super(Discriminator, self).__init__()
            # 三层的全连接网络
            self.model = nn.Sequential(
                nn.Linear(input_dim, 256),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Dropout(0.3),
                nn.Linear(256, 128),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Dropout(0.3),
                nn.Linear(128, 1),
                nn.Sigmoid()  # 输出样本是真实的概率
            )
        def forward(self, x):
            return self.model(x)
class MinorityGAN:
        def __init__(self, feature_dim, noise_dim=100, device=None):
            """
            初始化 GAN 模型用于生成少数类样本
            参数:
                feature_dim - 特征维度
                noise_dim - 随机噪声维度
                device - 训练设备(CPU/GPU)
            """
            self.feature_dim = feature_dim
            self.noise_dim = noise_dim
            # 初始化生成器和判别器
            self.generator = Generator(noise_dim, feature_dim).to(self.device)
            self.discriminator = Discriminator(feature_dim).to(self.device)
            # 初始化优化器
            self.g_optimizer = optim.Adam(self.generator.parameters(), lr=0.0002, betas=(0.5, 0.999))
            self.d_optimizer = optim.Adam(self.discriminator.parameters(), lr=0.0002, betas=(0.5, 0.999))
            # 损失函数
```

```
        self.criterion = nn.BCELoss()
    def train(self, real_features, epochs=100, batch_size=32, verbose=True):
        # 转换为 Tensor 并创建数据加载器
        real_tensor = torch.FloatTensor(real_features)
        dataset = TensorDataset(real_tensor)
        dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
        # 训练过程，轨迹
        g_losses, d_losses = [], []
        # 训练循环
        for epoch in range(epochs):
            epoch_g_loss, epoch_d_loss = 0.0, 0.0
            batches = 0
            for real_batch in dataloader:
                batch_size = real_batch[0].size(0)
                batches += 1
                # 准备标签
                real_labels = torch.ones(batch_size, 1, device=self.device)
                fake_labels = torch.zeros(batch_size, 1, device=self.device)
                # 将真实样本移动到设备
                real_samples = real_batch[0].to(self.device)
                # ===== 训练判别器 =====
                self.d_optimizer.zero_grad()
                # 判别真实样本
                real_outputs = self.discriminator(real_samples)
                d_loss_real = self.criterion(real_outputs, real_labels)
                # 生成假样本
                z = torch.randn(batch_size, self.noise_dim, device=self.device)
                fake_samples = self.generator(z)
                # 判别假样本
                fake_outputs = self.discriminator(fake_samples.detach())
                d_loss_fake = self.criterion(fake_outputs, fake_labels)
                # 总判别器损失
                d_loss = d_loss_real + d_loss_fake
                d_loss.backward()
                self.d_optimizer.step()
                # ===== 训练生成器 =====
                self.g_optimizer.zero_grad()
                # 重新判别生成的假样本
                fake_outputs = self.discriminator(fake_samples)
                g_loss = self.criterion(fake_outputs, real_labels)   # 希望假样本
被判为真
                g_loss.backward()
                self.g_optimizer.step()
                # 记录损失
```

```
                epoch_g_loss += g_loss.item()
                epoch_d_loss += d_loss.item()
            # 计算每轮的平均损失
            epoch_g_loss /= batches
            epoch_d_loss /= batches
            g_losses.append(epoch_g_loss)
            d_losses.append(epoch_d_loss)
            if verbose and (epoch+1) % 10 == 0:
                print(f"Epoch {epoch+1}/{epochs}, G Loss: {epoch_g_loss:.4f},
D Loss: {epoch_d_loss:.4f}")
        # 绘制损失曲线
        plt.figure(figsize=(10, 5))
        plt.plot(g_losses, label='生成器损失')
        plt.plot(d_losses, label='判别器损失')
        plt.xlabel('轮次')
        plt.ylabel('损失')
        plt.title('GAN 训练损失曲线')
        plt.legend()
        plt.savefig('gan_training_loss.png')
        plt.close()
        return g_losses, d_losses
    def generate_samples(self, n_samples):
        self.generator.eval()
        with torch.no_grad():
            # 生成随机噪声
            z = torch.randn(n_samples, self.noise_dim, device=self.device)
            # 使用生成器生成样本
            samples = self.generator(z)
        return samples.cpu().numpy()
# SPLD 函数
def spld(loss, group_membership, lambda_param, gamma, sample_weights=None):
    # 获取唯一的组标识
    group_idx = np.unique(group_membership)
    b = len(group_idx)
    selected_idx = np.zeros_like(loss)
    selected_scores = np.zeros_like(loss)
    if sample_weights is None:
        sample_weights = np.ones_like(loss)
    for j in range(b):
        # 找到当前组内的样本索引
        idx_in_group = np.where(group_membership == group_idx[j])[0]
        loss_in_group = loss[idx_in_group]
        weights_in_group = sample_weights[idx_in_group]
        # 对每个群组内的损失进行标准化，考虑样本权重
```

```
            if len(loss_in_group) > 1:
                # 使用加权均值和加权标准差
                weighted_mean = np.average(loss_in_group, weights=weights_in_group)
                # 加权标准差计算
                weighted_var = np.average((loss_in_group - weighted_mean)**2, weights=weights_in_group)
                weighted_std = np.sqrt(weighted_var) if weighted_var > 0 else 1e-8
                normalized_loss = (loss_in_group - weighted_mean) / weighted_std
            else:
                normalized_loss = loss_in_group
            # 计算组内样本的排名，使用标准化后的损失
            tmp = normalized_loss.copy()
            rank_in_group = np.zeros_like(normalized_loss, dtype=int)
            # 对组内样本的标准化损失进行排序和排名处理
            for i in range(len(normalized_loss)):
                # 找到当前标准化损失在 tmp 中的位置
                rst = np.where(normalized_loss[i] == tmp)[0]
                if len(rst) > 0:
                    # 考虑样本权重影响排名
                    rank_in_group[i] = rst[0] + 1  # Python 索引从 0 开始，但排名从 1 开始
                    tmp[rst[0]] = float('-inf')  # 标记为已处理
            nj = len(idx_in_group)
            for i in range(nj):
                # 计算 SPLD 阈值条件，考虑样本权重
                rank = rank_in_group[i]
                # 对少数类样本，使用更小的阈值
                threshold = lambda_param + gamma / (np.sqrt(rank) + np.sqrt(rank-1))
                threshold = threshold / weights_in_group[i]  # 权重越大，阈值越小，更容易被选择
                # 使用标准化后的损失与阈值比较
                if normalized_loss[i] < threshold:
                    selected_idx[idx_in_group[i]] = 1
                else:
                    selected_idx[idx_in_group[i]] = 0
                selected_scores[idx_in_group[i]] = normalized_loss[i] - threshold
    # 找出被选择的样本索引
    selected_idx = np.where(selected_idx == 1)[0]
    # 直接返回被选择的样本索引
    return selected_idx
# 简单的分类器网络
class ViewClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim=256, num_classes=68):
```

```
            super(ViewClassifier, self).__init__()
            self.fc1 = nn.Linear(input_dim, hidden_dim)
            self.fc2 = nn.Linear(hidden_dim, hidden_dim)
            self.fc3 = nn.Linear(hidden_dim, num_classes)
            self.dropout = nn.Dropout(0.3)
        def forward(self, x):
            x = F.relu(self.fc1(x))
            x = self.dropout(x)
            x = F.relu(self.fc2(x))
            x = self.dropout(x)
            x = self.fc3(x)
            return x
# 多模块 SPLD 分类器
class MultiViewSPLDClassifier:
        def __init__(self):
            # 定义模块名称和对应的列范围
            self.modules = {
                'personal_info': {'name': '个人信息模块', 'range': [0, 9]},  # 第 1-9 列
                'economic': {'name': '经济指标模块', 'range': [9, 17]},          # 第
10-17 列
                'labor_market': {'name': '劳动力市场模块', 'range': [17, 24]}, # 第
18-24 列
                'salary': {'name': '薪资情况模块', 'range': [24, 26]},           # 第
25-26 列
                'consumption': {'name': '消费水平模块', 'range': [26, 28]},     # 第
27-28 列
                'government': {'name': '政府支出模块', 'range': [28, 32]}        # 第
29-32 列
            }

            # 初始化模块权重（初始均等）
            self.module_weights = {module: 1.0/len(self.modules) for module in
self.modules}
            # 初始化 SVM 分类器和标准化器
            self.classifiers = {}
            self.scalers = {}
            for module in self.modules:
                # 使用 class_weight='balanced'自动处理类别不平衡问题
                self.classifiers[module] = SVC(
                    probability=True,
                    kernel='rbf',
                    C=1.0,
                    gamma='scale',
                    class_weight='balanced'   # 自动为少数类设置更大权重
```

```
                )
                self.scalers[module] = StandardScaler()
            # 保存是否使用 GAN 增强的标志
            self.using_gan_augmentation = False
            # 保存 GAN 模型
            self.gan_models = {}
        def prepare_data(self, data_path, test_size=0.2, random_state=42, gan_augment=True, augment_ratio=0.5, gan_epochs=100):
            """
            准备数据集并进行特征标准化，可选使用 GAN 生成少数类样本
            参数:
                data_path - Excel 文件路径
                test_size - 测试集比例
                random_state - 随机种子
                gan_augment - 是否使用 GAN 增强少数类
                augment_ratio - 少数类增强比例(相对于多数类)
                gan_epochs - GAN 训练轮数
            返回:
                训练特征、训练标签、测试特征、测试标签
            """
            print(f"加载数据: {data_path}")
            # 读取 Excel 文件
            try:
                df = pd.read_excel(data_path)
                print(f"数据集形状: {df.shape}")
            except Exception as e:
                print(f"读取数据出错: {e}")
                raise

            # 提取标签（最后一列）
            y = df.iloc[:, -1].values

            # 提取特征（所有列除了最后一列）
            X = df.iloc[:, :-1].values

            # 全局特征标准化处理
            global_scaler = StandardScaler()
            X_scaled = global_scaler.fit_transform(X)
            print(f"全局特征标准化完成，特征均值: {np.mean(X_scaled):.4f}，标准差: {np.std(X_scaled):.4f}")

            # 按模块划分特征
            self.module_features = {}
            for module, info in self.modules.items():
```

```
                start, end = info['range']
                self.module_features[module] = X_scaled[:, start:end]
                print(f"{info['name']} 特 征 形 状 :
{self.module_features[module].shape}")
        # 划分训练集和测试集
        # 分别获取每个类别的索引
        class_0_indices = np.where(y == 0)[0]
        class_1_indices = np.where(y == 1)[0]
        # 打乱每个类别的索引
        np.random.seed(random_state)
        np.random.shuffle(class_0_indices)
        np.random.shuffle(class_1_indices)
        # 计算每个类别分到测试集的数量
        n_test_0 = int(len(class_0_indices) * test_size)
        n_test_1 = int(len(class_1_indices) * test_size)
        # 划分每个类别的测试集和训练集
        test_indices = np.concatenate([
            class_0_indices[:n_test_0],
            class_1_indices[:n_test_1]
        ])
        train_indices = np.concatenate([
            class_0_indices[n_test_0:],
            class_1_indices[n_test_1:]
        ])
        # 再次打乱训练集和测试集的顺序
        np.random.shuffle(train_indices)
        np.random.shuffle(test_indices)
        # 训练集
        self.train_features = {}
        for module in self.modules:
            self.train_features[module] =
self.module_features[module][train_indices]
        self.y_train = y[train_indices]
        # 测试集
        self.test_features = {}
        for module in self.modules:
            self.test_features[module] =
self.module_features[module][test_indices]
        self.y_test = y[test_indices]
        # 计算每个类别在训练集中的比例和样本权重
        train_class_0_count = np.sum(self.y_train == 0)
        train_class_1_count = np.sum(self.y_train == 1)
        train_class_0_ratio = train_class_0_count / len(self.y_train)
        train_class_1_ratio = train_class_1_count / len(self.y_train)
```

```
        print(f"训练集样本数量: 类别 0(未失业)={train_class_0_count}, 类别 1(失业)={train_class_1_count}")
        print(f"训练集类别分布: 类别 0={train_class_0_ratio:.2f}, 类别 1={train_class_1_ratio:.2f}")
        # === GAN 数据增强部分 ===
        if gan_augment and train_class_0_count < train_class_1_count:
            print("\n=== 使用 GAN 增强少数类(标签 0)样本 ===")
            self.using_gan_augmentation = True
            # 分离少数类样本(标签 0)
            minority_indices = np.where(self.y_train == 0)[0]
            minority_count = len(minority_indices)
            # 确定需要生成的样本数量
            target_minority_count = int(train_class_1_count * augment_ratio)
            n_generate = max(0, target_minority_count - minority_count)
            if n_generate > 0:
                print(f"正在训练 GAN 为少数类(标签 0)生成{n_generate}个合成样本...")
                # 为每个模块训练单独的 GAN 并生成样本
                synthetic_features = {}
                for module, features in self.train_features.items():
                    # 提取该模块下的少数类样本
                    module_minority_features = features[minority_indices]
                    feature_dim = module_minority_features.shape[1]

                    # 初始化并训练 GAN
                    print(f"\n 训练 {self.modules[module]['name']} GAN 模型...")
                    gan = MinorityGAN(feature_dim=feature_dim)
                    gan.train(module_minority_features, epochs=gan_epochs, batch_size=min(32, minority_count))

                    # 生成合成样本
                    module_synthetic_features = gan.generate_samples(n_generate)
                    synthetic_features[module] = module_synthetic_features

                    # 保存 GAN 模型以便后续使用
                    self.gan_models[module] = gan
                    print(f"{self.modules[module]['name']} GAN 已生成{n_generate}个合成样本")
                # 为每个合成样本创建标签(全部为类别 0)
                synthetic_labels = np.zeros(n_generate, dtype=int)
                # 扩展训练集，添加合成样本
                for module in self.modules:
```

```
                        self.train_features[module] = np.vstack([
                            self.train_features[module],
                            synthetic_features[module]
                        ])
                    # 扩展训练标签
                    self.y_train = np.concatenate([self.y_train, synthetic_labels])
                    # 更新训练集数量统计
                    train_class_0_count = np.sum(self.y_train == 0)
                    train_class_1_count = np.sum(self.y_train == 1)
                    train_class_0_ratio = train_class_0_count / len(self.y_train)
                    train_class_1_ratio = train_class_1_count / len(self.y_train)
                    print(f"\nGAN 增强后的训练集:")
                    print(f"  样本数量: 类别 0(未失业)={train_class_0_count}, 类别 1(失业)={train_class_1_count}")
                    print(f"  类别分布: 类别 0={train_class_0_ratio:.2f}, 类别 1={train_class_1_ratio:.2f}")
        # 创建样本权重数组，方便后续 SPLD 使用
        self.sample_weights = np.array([self.class_weights[label] for label in self.y_train])
        print(f"\n 数据准备完成: 训练集={len(self.y_train)}样本, 测试集={len(self.y_test)}样本")
        print(f"失业标签分布: 训练集中失业(1)比例: {train_class_1_ratio:.2f}, 测试集中失业(1)比例: {np.mean(self.y_test):.2f}")
        print(f"类别权重: 标签 0 权重={self.class_weights[0]:.2f}, 标签 1 权重={self.class_weights[1]:.2f}")
        # 标准化
        for module in self.modules:
            module_mean = np.mean(self.train_features[module])
            module_std = np.std(self.train_features[module])
            print(f"{self.modules[module]['name']} 标准化统计: 均值={module_mean:.4f}, 标准差={module_std:.4f}")
        return self.train_features, self.y_train, self.test_features, self.y_test
    def train_module(self, module, features, labels, selected_indices=None):
        """
        调用问题二的 LINEX-SVM
        """
        if selected_indices is None:
            selected_indices = np.arange(len(features))
        # 准备训练数据
        selected_features = features[selected_indices]
        selected_labels = labels[selected_indices]

        # 获取选中样本的权重
        selected_weights = self.sample_weights[selected_indices]
```

```
            # 训练分类器
            try:
                # 计算训练集准确率
                train_pred = self.classifiers[module].predict(selected_features)
                # 计算各个指标
                train_acc = accuracy_score(selected_labels, train_pred,
sample_weight=selected_weights)
                class_0_acc = accuracy_score(selected_labels[selected_labels==0],
                                             train_pred[selected_labels==0]) if
np.any(selected_labels==0) else 0
                class_1_acc = accuracy_score(selected_labels[selected_labels==1],
                                             train_pred[selected_labels==1]) if
np.any(selected_labels==1) else 0

                print(f"{self.modules[module]['name']}训练完成:")
                print(f"  加权准确率: {train_acc:.4f}")
                print(f"  类别 0 准确率: {class_0_acc:.4f}, 类别 1 准确率:
{class_1_acc:.4f}")
                print(f"  选择样本数: {len(selected_indices)}, 类别 0:
{np.sum(selected_labels==0)}, 类别 1: {np.sum(selected_labels==1)}")
                # 返回加权错误率作为 loss
                return 1.0 - train_acc
        def compute_losses(self, module, features, labels):
            """
            计算每个样本的损失，用于 SPLD 样本选择
            为少数类(标签 0)设置更大权重的损失
            """
            try:
                # 获取决策函数值
                decision_values = self.classifiers[module].decision_function(features)
                # 计算损失 - 使用与正确类别的距离作为损失
                losses = np.zeros(len(labels))
                for i, (decision, label) in enumerate(zip(decision_values, labels)):
        def evaluate_module(self, module, features, labels):
            """评估特定模块的分类器性能，关注少数类性能"""
            try:
                predictions = self.classifiers[module].predict(features)
                # 计算整体性能指标
                accuracy = accuracy_score(labels, predictions)
                precision = precision_score(labels, predictions, zero_division=0)
                recall = recall_score(labels, predictions, zero_division=0)
                f1 = f1_score(labels, predictions, zero_division=0)
```

```
                # 计算每个类别的性能
                class_0_mask = (labels == 0)
                class_1_mask = (labels == 1)

                if np.any(class_0_mask):
                    class_0_acc = accuracy_score(labels[class_0_mask],
predictions[class_0_mask])
                else:
                    class_0_acc = 0

                if np.any(class_1_mask):
                    class_1_acc = accuracy_score(labels[class_1_mask],
predictions[class_1_mask])
                else:
                    class_1_acc = 0
                print(f"{self.modules[module]['name']}评估结果:")
                print(f"  整体准确率: {accuracy:.4f}")
                print(f"  类别 0 准确率: {class_0_acc:.4f}")
                print(f"  类别 1 准确率: {class_1_acc:.4f}")
                print(f"  精确率: {precision:.4f}, 召回率: {recall:.4f}, F1 分数:
{f1:.4f}")
                # 计算混淆矩阵
                cm = confusion_matrix(labels, predictions)
                print(f"  混淆矩阵: \n{cm}")
                return accuracy, precision, recall, f1
            except Exception as e:
                print(f"评估{self.modules[module]['name']}时出错: {e}")
                return 0.0, 0.0, 0.0, 0.0
        def train_iteration(self, lambda_param=0.3, gamma=0.2):
            """执行一次 SPLD 训练迭代，考虑样本权重"""
            # 为每个模块分配组成员关系
            group_membership = {}
            for idx, module in enumerate(self.modules):
                group_membership[module] = np.ones(len(self.y_train)) * idx  # 每
个模块是一个独立的组
            # 对每个模块执行 SPLD 和训练
            selected_indices = {}
            accuracies = {}
            class_0_accuracies = {}  # 新增：记录每个模块对类别 0 的准确率
            for module in self.modules:
                # 计算当前分类器的损失
                losses = self.compute_losses(module, self.train_features[module],
self.y_train)
                # 使用 SPLD 选择样本，传入样本权重
```

```
                module_indices = spld(losses, group_membership[module], lambda_param, gamma, self.sample_weights)
                selected_indices[module] = module_indices
                # 使用选择的样本训练模块分类器
                self.train_module(module, self.train_features[module], self.y_train, module_indices)
                # 评估模块分类器
                acc, prec, rec, f1 = self.evaluate_module(module, self.test_features[module], self.y_test)
                accuracies[module] = acc
                # 计算类别 0 的准确率
                class_0_mask = (self.y_test == 0)
                if np.any(class_0_mask):
                    predictions = self.classifiers[module].predict(self.test_features[module])
                    class_0_acc = accuracy_score(
                        self.y_test[class_0_mask],
                        predictions[class_0_mask]
                    )
                    class_0_accuracies[module] = class_0_acc
                    print(f"{self.modules[module]['name']} 对类别 0 的准确率: {class_0_acc:.4f}")
                else:
                    class_0_accuracies[module] = 0.0
            # 基于类别 0 的准确率更新模块权重
            self.update_module_weights_based_on_class0(class_0_accuracies)
            # 计算综合准确率
            ensemble_metrics = self.evaluate_ensemble(self.test_features, self.y_test)
            print(f"综合分类器评估结果:")
            print(f"  准确率: {ensemble_metrics['accuracy']:.4f}")
            print(f"  精确率: {ensemble_metrics['precision']:.4f}")
            print(f"  召回率: {ensemble_metrics['recall']:.4f}")
            print(f"  F1 分数: {ensemble_metrics['f1']:.4f}")
            # 计算类别 0 和类别 1 的单独准确率
            class_0_mask = (self.y_test == 0)
            class_1_mask = (self.y_test == 1)
            if np.any(class_0_mask):
                class_0_acc = accuracy_score(self.y_test[class_0_mask], ensemble_metrics['predictions'][class_0_mask])
                print(f"  类别 0 准确率: {class_0_acc:.4f}")

            if np.any(class_1_mask):
                class_1_acc = accuracy_score(self.y_test[class_1_mask], ensemble_metrics['predictions'][class_1_mask])
```

```
            print(f"  类别 1 准确率: {class_1_acc:.4f}")
        return ensemble_metrics['accuracy'], accuracies, selected_indices
    def update_module_weights_based_on_class0(self, class_0_accuracies):
        """
        基于各模块对类别 0 的预测准确率动态调整模块权重

        参数:
            class_0_accuracies - 各模块对类别 0 的预测准确率字典
        """
        print("\n=== 基于类别 0 准确率更新模块权重 ===")

        # 提取各模块类别 0 准确率
        accuracies = np.array([class_0_accuracies[module] for module in
self.modules])

        # 确保准确率非零，避免除以零的情况
        min_acc = 0.01  # 最小准确率阈值
        accuracies = np.maximum(accuracies, min_acc)
        # 对准确率应用指数增强，放大差异
        exp_factor = 2.0  # 指数因子，调整权重分配的敏感度
        weighted_accuracies = accuracies ** exp_factor
        # 归一化以获得总和为 1 的权重
        weights_sum = np.sum(weighted_accuracies)
        if weights_sum > 0:
            normalized_weights = weighted_accuracies / weights_sum
        else:
            # 如果所有模块表现都很差，则使用均等权重
            normalized_weights = np.ones(len(self.modules)) / len(self.modules)

        # 更新模块权重
        for i, module in enumerate(self.modules):
            old_weight = self.module_weights[module]
            new_weight = normalized_weights[i]
            self.module_weights[module] = new_weight
            print(f"{self.modules[module]['name']}:  类 别 0 准 确 率
={class_0_accuracies[module]:.4f}, 权重: {old_weight:.4f} -> {new_weight:.4f}")
        # 显示权重分配信息
        print(f"模块权重更新完成，偏向于对类别 0 表现更好的模块")

    def evaluate_ensemble(self, test_features, test_labels):
        """评估综合多模块分类器，使用动态调整的模块权重"""
        # 获取每个模块的预测概率
        module_probs = {}
        for module in self.modules:
```

```
                try:
                    # 使用各模块训练好的 SVM 得到预测概率
                    module_probs[module] = self.classifiers[module].predict_proba(test_features[module])[:, 1]
                except Exception as e:
                    print(f"获取{self.modules[module]['name']}预测概率时出错: {e}")
                    # 如果无法获取预测概率，使用随机概率
                    module_probs[module] = np.random.rand(len(test_labels))
            # 综合预测概率（使用动态调整的权重）
            ensemble_probs = np.zeros(len(test_labels))
            for module in self.modules:
                weight = self.module_weights[module]
                ensemble_probs += module_probs[module] * weight
            # 根据阈值 0.5 将概率转换为预测标签
            ensemble_preds = (ensemble_probs >= 0.5).astype(int)
            # 计算评估指标
            accuracy = accuracy_score(test_labels, ensemble_preds)
            precision = precision_score(test_labels, ensemble_preds, zero_division=0)
            recall = recall_score(test_labels, ensemble_preds, zero_division=0)
            f1 = f1_score(test_labels, ensemble_preds, zero_division=0)
            # 返回一个包含所有指标的字典
            return {
                'accuracy': accuracy,
                'precision': precision,
                'recall': recall,
                'f1': f1,
                'predictions': ensemble_preds,
                'probabilities': ensemble_probs
            }

        def train_spld_multi_module(self, iterations=10, initial_lambda=1.0, initial_gamma=0.5,
                                    lambda_step=0.3, gamma_step=0.3):
            """执行多次 SPLD 训练迭代

            参数:
                iterations - 最大迭代次数
                initial_lambda - 初始λ值
                initial_gamma - 初始γ值
                lambda_step - 每次迭代λ的增加量
                gamma_step - 每次迭代γ的增加量
            """
            # 初始化权重历史记录
```

```
        self.weight_history = []

        # 第一次迭代 - 使用所有样本进行预训练
        print("预训练阶段：使用所有样本...")
        for module in self.modules:
            self.train_module(module, self.train_features[module], self.y_train)

        # 记录初始权重
        self.weight_history.append(self.module_weights.copy())

        # 评估初始模型
        initial_metrics = self.evaluate_ensemble(self.test_features, self.y_test)
        initial_accuracy = initial_metrics['accuracy']
        print(f"初始集成模型准确率: {initial_accuracy:.4f}")

        # 初始化每个模块对类别 0 的预测准确率
        class_0_mask = (self.y_test == 0)
        initial_class_0_accs = {}
        if np.any(class_0_mask):
            for module in self.modules:
                try:
                    predictions = self.classifiers[module].predict(self.test_features[module])
                    module_class_0_acc = accuracy_score(
                        self.y_test[class_0_mask],
                        predictions[class_0_mask]
                    )
                    initial_class_0_accs[module] = module_class_0_acc
                    print(f"初始{self.modules[module]['name']}对类别 0 的准确率: {module_class_0_acc:.4f}")
                except Exception as e:
                    print(f"计算{self.modules[module]['name']}初始类别 0 准确率出错: {e}")
                    initial_class_0_accs[module] = 0.0

            # 基于初始类别 0 预测准确率更新模块权重
            self.update_module_weights_based_on_class0(initial_class_0_accs)
            self.weight_history.append(self.module_weights.copy())

        # 迭代训练
        all_metrics = [initial_metrics]
        lambda_param = initial_lambda
        gamma_param = initial_gamma
        selection_rates = {}
```

```
        prev_accuracy = initial_accuracy
        accuracy_threshold = 1e-4  # 准确率变化阈值

        for i in range(iterations):
            print(f"\n===== 迭代 {i+1}/{iterations} =====")
            print(f"当前参数: λ={lambda_param:.3f}, γ={gamma_param:.3f}")

            # 执行一次 SPLD 训练迭代
            acc, module_accs, selected = self.train_iteration(lambda_param, gamma_param)

            # 记录当前权重
            self.weight_history.append(self.module_weights.copy())

            # 记录指标
            current_metrics = self.evaluate_ensemble(self.test_features, self.y_test)
            all_metrics.append(current_metrics)

            # 计算每个模块的选择率
            module_selection_rates = {}
            all_samples_selected = True  # 标记是否所有模块都选择了所有样本

            for module in self.modules:
                selection_rate = len(selected[module]) / len(self.y_train)
                module_selection_rates[module] = selection_rate
                print(f"{self.modules[module]['name']} 选择了 {len(selected[module])}/{len(self.y_train)} 个样本 (选择率: {selection_rate:.2f})")

                # 检查是否有模块未选择所有样本（使用 0.99 作为阈值以允许少量误差）
                if selection_rate < 0.99:
                    all_samples_selected = False

            # 计算平均选择率
            avg_selection_rate = sum(module_selection_rates.values()) / len(self.modules)
            print(f"平均选择率: {avg_selection_rate:.2f}")
            selection_rates[i] = module_selection_rates

            # 检查终止条件
            accuracy_converged = abs(acc - prev_accuracy) < accuracy_threshold
```

```
                # 满足任一终止条件即停止迭代
                if all_samples_selected:
                    print(f"所有模块均已选择所有样本，停止迭代")
                    break
                # 更新上一次的准确率
                prev_accuracy = acc
                # 增加 λ 和 γ 值
                lambda_param += lambda_step
                gamma_param += gamma_step
                # 计算所有模块对类别 0 的最终准确率
                class_0_mask = (self.y_test == 0)
                if np.any(class_0_mask):
                    final_class_0_accs = {}
                    for module in self.modules:
                        predictions                                                    =
self.classifiers[module].predict(self.test_features[module])
                        final_class_0_accs[module] = accuracy_score(
                            self.y_test[class_0_mask],
                            predictions[class_0_mask]
                        )
                    # 最后一次基于类别 0 准确率更新权重

self.update_module_weights_based_on_class0(final_class_0_accs)
                    self.weight_history.append(self.module_weights.copy())
            # 最终评估
            final_metrics = self.evaluate_ensemble(self.test_features, self.y_test)
            all_metrics.append(final_metrics)
            print(f"\n=====  最终模型评估结果  =====")
            print(f"准确率: {final_metrics['accuracy']:.4f}")
            print(f"精确率: {final_metrics['precision']:.4f}")
            print(f"召回率: {final_metrics['recall']:.4f}")
            print(f"F1 分数: {final_metrics['f1']:.4f}")
            # 单独报告类别 0 和类别 1 的准确率
            class_0_mask = (self.y_test == 0)
            class_1_mask = (self.y_test == 1)

            if np.any(class_0_mask):
                class_0_acc          =          accuracy_score(self.y_test[class_0_mask],
final_metrics['predictions'][class_0_mask])
                print(f"类别 0(未失业)准确率: {class_0_acc:.4f}")

            if np.any(class_1_mask):
                class_1_acc          =          accuracy_score(self.y_test[class_1_mask],
```

```
final_metrics['predictions'][class_1_mask])
                print(f"类别 1(失业)准确率: {class_1_acc:.4f}")

            # 绘制训练过程中的性能指标变化
            self.plot_performance(all_metrics, selection_rates)

            return all_metrics, selection_rates

        def plot_performance(self, all_metrics, selection_rates):
            """绘制训练过程中的性能指标变化"""
            plt.figure(figsize=(15, 15))

            # 提取性能指标
            accuracies = [metrics['accuracy'] for metrics in all_metrics]
            precisions = [metrics['precision'] for metrics in all_metrics]
            recalls = [metrics['recall'] for metrics in all_metrics]
            f1_scores = [metrics['f1'] for metrics in all_metrics]

            # 绘制性能指标曲线
            plt.subplot(3, 1, 1)
            plt.plot(range(len(accuracies)), accuracies, marker='o', label='准确率')
            plt.plot(range(len(precisions)), precisions, marker='s', label='精确率')
            plt.plot(range(len(recalls)), recalls, marker='^', label='召回率')
            plt.plot(range(len(f1_scores)), f1_scores, marker='*', label='F1 分数')
            plt.title('性能指标随迭代次数的变化')
            plt.xlabel('迭代次数')
            plt.ylabel('指标值')
            plt.legend()
            plt.grid(True)

            # 绘制各模块选择率曲线
            if selection_rates:
                plt.subplot(3, 1, 2)
                iterations = list(selection_rates.keys())

                for module in self.modules:
                    rates = [selection_rates[i][module] for i in iterations]
                    plt.plot(iterations, rates, marker='s',
label=f'{self.modules[module]["name"]}')

                plt.title('各模块样本选择率随迭代次数的变化')
                plt.xlabel('迭代次数')
                plt.ylabel('选择率')
                plt.legend()
```

```
                plt.grid(True)

                # 绘制最终模块权重饼图
                plt.subplot(3, 1, 3)
                labels = [f"{self.modules[module]['name']}\n({self.module_weights[module]:.3f})" for module in self.modules]
                weights = [self.module_weights[module] for module in self.modules]

                # 根据权重排序，使显示更清晰
                sorted_indices = np.argsort(weights)[::-1]  # 降序排序
                sorted_labels = [labels[i] for i in sorted_indices]
                sorted_weights = [weights[i] for i in sorted_indices]

                plt.pie(sorted_weights, labels=sorted_labels, autopct='%1.1f%%', startangle=90)
                plt.title('最终集成模型中各模块的权重分布\n(基于类别 0 的预测准确率)')
                plt.axis('equal')  # 确保饼图是圆的

            plt.tight_layout()
            plt.savefig('unemployment_spld_performance.png')
            plt.close()

            # 绘制混淆矩阵
            final_metrics = all_metrics[-1]
            if 'predictions' in final_metrics and len(self.y_test) > 0:
                cm = confusion_matrix(self.y_test, final_metrics['predictions'])
                plt.figure(figsize=(8, 6))
                plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
                plt.title('混淆矩阵')
                plt.colorbar()
                tick_marks = np.arange(2)
                plt.xticks(tick_marks, ['未失业(0)', '失业(1)'])
                plt.yticks(tick_marks, ['未失业(0)', '失业(1)'])

                # 在混淆矩阵中显示数值
                thresh = cm.max() / 2.
                for i in range(cm.shape[0]):
                    for j in range(cm.shape[1]):
                        plt.text(j, i, format(cm[i, j], 'd'),
                                 horizontalalignment="center",
                                 color="white" if cm[i, j] > thresh else "black")
```

```
                plt.ylabel('真实标签')
                plt.xlabel('预测标签')
                plt.tight_layout()
                plt.savefig('unemployment_confusion_matrix.png')
                plt.close()

                # 绘制模块权重变化曲线（如果保存了历史权重）
                if hasattr(self, 'weight_history') and self.weight_history:
                    plt.figure(figsize=(12, 6))
                    iterations = list(range(len(self.weight_history)))

                    for module in self.modules:
                        weights = [weights_dict[module] for weights_dict in self.weight_history]
                        plt.plot(iterations, weights, marker='o', label=f'{self.modules[module]["name"]}')

                    plt.title('各模块权重随迭代次数的变化')
                    plt.xlabel('迭代次数')
                    plt.ylabel('模块权重')
                    plt.legend()
                    plt.grid(True)
                    plt.tight_layout()
                    plt.savefig('module_weights_history.png')
                    plt.close()

# 主函数
def main():
    try:
        # 设置随机种子以确保可重复性
        np.random.seed(42)
        torch.manual_seed(42)

        # 数据路径
        data_path = "D:\\C 题\\清洗后数据_添加特征 EWMA .xlsx"

        # 初始化多模块 SPLD 分类器
        classifier = MultiViewSPLDClassifier()

        # 准备数据 - 启用 GAN 增强
        print("准备数据...")
        train_features, y_train, test_features, y_test = classifier.prepare_data(
            data_path,
            test_size=0.2,
```

```
            gan_augment=True,              # 启用 GAN 增强
            augment_ratio=0.8,             # 少数类增强至多数类的 80%
            gan_epochs=50                  # GAN 训练轮数
        )
        # 训练模型
        print("开始训练...")
        try:
            all_metrics, selection_rates = classifier.train_spld_multi_module(
                iterations=20,             # 迭代次数
                initial_lambda=0.1,        # SPLD 初始 λ 值
                initial_gamma=0.1,         # SPLD 初始 γ 值
                lambda_step=0.02,          # λ 的增加步长
                gamma_step=0.02            # γ 的增加步长
            )

            # 输出最终结果
            final_metrics = all_metrics[-1]
            print("\n=====  训练结果  =====")
            print(f"失业状态预测最终结果:")
            print(f"准确率: {final_metrics['accuracy']:.4f}")
            print(f"精确率: {final_metrics['precision']:.4f}")
            print(f"召回率: {final_metrics['recall']:.4f}")
            print(f"F1 分数: {final_metrics['f1']:.4f}")
            # 打印类别 0 和类别 1 的准确率
            class_0_mask = (classifier.y_test == 0)
            class_1_mask = (classifier.y_test == 1)
            if np.any(class_0_mask):
                class_0_acc = accuracy_score(classifier.y_test[class_0_mask],
final_metrics['predictions'][class_0_mask])
                print(f"类别 0(未失业)准确率: {class_0_acc:.4f}")
            if np.any(class_1_mask):
                class_1_acc = accuracy_score(classifier.y_test[class_1_mask],
final_metrics['predictions'][class_1_mask])
                print(f"类别 1(失业)准确率: {class_1_acc:.4f}")
            # 打印各模块贡献
            print("\n=====  各模块贡献  =====")
            for module in classifier.modules:
                print(f"{classifier.modules[module]['name']}:          权          重
={classifier.module_weights[module]:.4f}")
            print("\n 训练完成，性能可视化图表已保存")
        except Exception as e:
            print(f"训练过程中发生错误: {e}")
            import traceback
            traceback.print_exc()
```

```
    except Exception as main_e:
        print(f"程序执行过程中发生未处理的错误: {main_e}")
        import traceback
        traceback.print_exc()
if __name__ == "__main__":
    main()
```

问题四代码

```
#问题四外部数据预处理
import pandas as pd
import numpy as np
import os
file_path = r"D:\C 题\问题四数据.xlsx"
df = pd.read_excel(file_path)
salary_mapping = {
    "0000001000": "1000 元以下",
    "0100002000": "1000-2000 元/月",
    "0200104000": "2001-4000 元/月",
    "0400106000": "4001-6000 元/月",
    "0600108000": "6001-8000 元/月",
    "0800110000": "8001-10000 元/月",
    "100001150000": "100000 元以上",
    "1000115000": "10001-15000 元/月",
    "1500120000": "15000-20000 元",
    "1500125000": "15000-25000 元/月",
    "2000130000": "20000-30000 元",
    "2500199999": "25000 元/月以上",
    "3000150000": "30000-50000 元",
    "3500150000": "35000-50000 元/月",
    "5000170000": "50000-70000 元/月",
    "70001100000": "70000-100000 元/月",
    "2500135000": "25000-35000 元/月"
}
def process_salary_code(code):
    if pd.isna(code):
        return np.nan
    str_code = str(int(code)) if isinstance(code, (int, float)) else str(code)
    if str_code in salary_mapping:
        return salary_mapping[str_code]
    if len(str_code) == 9:
        padded_code = "0" + str_code
        if padded_code in salary_mapping:
            return salary_mapping[padded_code]
    if len(str_code) == 11:
        padded_code = str_code + "0"
```

```
            if padded_code in salary_mapping:
                return salary_mapping[padded_code]

    return None
expected_salary_col = [col for col in df.columns if "期望薪资" in col]
actual_salary_col = [col for col in df.columns if "实际薪资" in col]
if expected_salary_col:
    df["期望薪资_转换"] = df[expected_salary_col[0]].apply(process_salary_code)
if actual_salary_col:
    df["实际薪资_转换"] = df[actual_salary_col[0]].apply(process_salary_code)
mask = True
if expected_salary_col:
    mask = mask & (~df["期望薪资_转换"].isna() | df[expected_salary_col[0]].isna())
if actual_salary_col:
    mask = mask & (~df["实际薪资_转换"].isna() | df[actual_salary_col[0]].isna())
df = df[mask]"
df.fillna("未知", inplace=True)
documents_path = os.path.expanduser(r"~\Documents")
output_path = os.path.join(documents_path, "processed_data.xlsx")
try:
    os.makedirs(os.path.dirname(output_path), exist_ok=True)
    df.to_excel(output_path, index=False)
    print(f"处理完成，数据已保存至: {output_path}")
except PermissionError:
        output_path = "processed_data.xlsx"
    df.to_excel(output_path, index=False)
    print(f"处理完成，数据已保存至当前目录: {os.path.abspath(output_path)}")
#Transformer 人岗匹配(端到端）
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import os
from tqdm import tqdm
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
import nltk
import re
# 下载 NLTK 资源
```

```
try:
    nltk.download('punkt', quiet=True)
# 设置随机种子保证可重复性
torch.manual_seed(42)
np.random.seed(42)
# 加载数据
data_path = r"C:\Users\lenovo\Documents\processed_data.xlsx"
df = pd.read_excel(data_path)
# 定义输入和输出列
input_cols = ["年龄", "教育程度", "毕业年份", "期望工作行业", "期望薪资"]
output_cols = ["实际工作", "实际薪资"]")
# 将所有特征转为字符串类型，处理混合类型数据
for col in df.columns:
    df[col] = df[col].astype(str)
# 创建输入序列，将所有特征组合成长文本
def create_text_sequence(row, cols):
    """将一行数据的多个特征组合成一个文本序列"""
    return " ".join([f"{col}:{row[col]}" for col in cols])
# 为每行数据创建输入文本序列
df['input_text'] = df.apply(lambda row: create_text_sequence(row, input_cols), axis=1)
df['output_text'] = df.apply(lambda row: create_text_sequence(row, output_cols), axis=1)
# 为输入和输出文本创建编码器
input_encoder = LabelEncoder()
output_encoder = LabelEncoder()
# 训练编码器
df['input_encoded'] = input_encoder.fit_transform(df['input_text'])
df['output_encoded'] = output_encoder.fit_transform(df['output_text'])
# 分割数据
X = df['input_encoded'].values
y = df['output_encoded'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# 保存原始文本用于 BLEU 评分
output_texts_test = df.loc[y_test.index, 'output_text'].values if hasattr(y_test, 'index') else df['output_text'].values[y_test]
# 创建 PyTorch 数据集
class JobSalaryDataset(Dataset):
    def __init__(self, X, y, output_texts=None):
        self.X = torch.tensor(X, dtype=torch.long)
        self.y = torch.tensor(y, dtype=torch.long)
        self.output_texts = output_texts
    def __len__(self):
```

```
            return len(self.X)
        def __getitem__(self, idx):
            if self.output_texts is not None:
                return self.X[idx], self.y[idx], self.output_texts[idx]
            return self.X[idx], self.y[idx]
train_dataset = JobSalaryDataset(X_train, y_train)
test_dataset = JobSalaryDataset(X_test, y_test, output_texts_test)
# 创建 DataLoader
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size)
# 定义位置编码
class PositionalEncoding(nn.Module):
        def __init__(self, d_model, max_seq_len=100):
            super(PositionalEncoding, self).__init__()
            self.dropout = nn.Dropout(0.1)
            pe = torch.zeros(max_seq_len, d_model)
            position = torch.arange(0, max_seq_len, dtype=torch.float).unsqueeze(1)
            div_term = torch.exp(torch.arange(0, d_model, 2).float() *
(-np.log(10000.0) / d_model))
            pe[:, 0::2] = torch.sin(position * div_term)
            pe[:, 1::2] = torch.cos(position * div_term)
            pe = pe.unsqueeze(0)
            self.register_buffer('pe', pe)
        def forward(self, x):
            x = x + self.pe[:, :x.size(1)]
            return self.dropout(x)
# 定义文本序列 Transformer 模型
class TextTransformerModel(nn.Module):
        def __init__(self, vocab_size, output_size, d_model, nhead, num_layers,
dim_feedforward, dropout=0.1):
            super(TextTransformerModel, self).__init__()
            # 嵌入层
            self.embedding = nn.Embedding(vocab_size, d_model)
            self.pos_encoder = PositionalEncoding(d_model)
            # Transformer 层
            encoder_layers = nn.TransformerEncoderLayer(d_model, nhead,
dim_feedforward, dropout)
            self.transformer_encoder = nn.TransformerEncoder(encoder_layers,
num_layers)
            # 输出层
            self.output_layer = nn.Linear(d_model, output_size)
            self.d_model = d_model
            self.init_weights()
```

```
        def init_weights(self):
            initrange = 0.1
            self.embedding.weight.data.uniform_(-initrange, initrange)
            self.output_layer.bias.data.zero_()
            self.output_layer.weight.data.uniform_(-initrange, initrange)

        def forward(self, src):
            # src: [batch_size]
            # 将输入转换为 [batch_size, 1]
            src = src.unsqueeze(1)
            # 嵌入层 [batch_size, 1, d_model]
            embedded = self.embedding(src) * np.sqrt(self.d_model)
            embedded = self.pos_encoder(embedded)
            # 转置为 Transformer 预期的形状 [1, batch_size, d_model]
            embedded = embedded.transpose(0, 1)
            # 通过 Transformer 编码器 [1, batch_size, d_model]
            output = self.transformer_encoder(embedded)
            # 取序列的输出 [batch_size, d_model]
            output = output.transpose(0, 1)
            output = output.mean(dim=1)
            # 应用输出层 [batch_size, output_size]
            output = self.output_layer(output)
            return output
# 初始化模型
vocab_size = len(input_encoder.classes_)
output_size = len(output_encoder.classes_)
d_model = 128
nhead = 8
num_layers = 4
dim_feedforward = 512
dropout = 0.2
model = TextTransformerModel(
    vocab_size=vocab_size,
    output_size=output_size,
    d_model=d_model,
    nhead=nhead,
    num_layers=num_layers,
    dim_feedforward=dim_feedforward,
    dropout=dropout
)
# 损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# 训练循环
```

```
num_epochs = 30
train_losses = []
val_losses = []
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

for epoch in range(num_epochs):
    # 训练
    model.train()
    running_loss = 0.0
    for batch_idx, (inputs, targets) in enumerate(tqdm(train_loader, desc=f"轮次 {epoch+1}/{num_epochs}")):
        inputs, targets = inputs.to(device), targets.to(device)
        # 前向传播
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        # 反向传播和优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_train_loss = running_loss / len(train_loader)
    train_losses.append(avg_train_loss)
    # 验证
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for batch in test_loader:
            if len(batch) == 3:
                inputs, targets, _ = batch    # Unpack three values
            else:
                inputs, targets = batch
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            val_loss += loss.item()
    avg_val_loss = val_loss / len(test_loader)
    val_losses.append(avg_val_loss)
    print(f"轮次 [{epoch+1}/{num_epochs}], 训练损失: {avg_train_loss:.4f}, 验证损失: {avg_val_loss:.4f}")
# 绘制训练和验证损失
plt.figure(figsize=(10, 6))
plt.plot(range(1, num_epochs+1), train_losses, label='训练损失')
plt.plot(range(1, num_epochs+1), val_losses, label='验证损失')
```

```
plt.xlabel('轮次')
plt.ylabel('损失')
plt.title('训练和验证损失')
plt.legend()
plt.savefig('training_loss.png')
plt.show()

# 评估模型
model.eval()
all_preds = []
all_targets = []
all_pred_texts = []
all_target_texts = []
def tokenize_text(text):
    """简单的分词函数，分隔文本为单词列表"""
    # 将中文标点符号替换为空格
    text = re.sub(r'[，。！？：；""（）【】《》]', ' ', text)
    # 移除多余空格
    text = re.sub(r'\s+', ' ', text)
    # 分词
    return text.strip().split()
with torch.no_grad():
    for batch in test_loader:
        if len(batch) == 3:
            inputs, targets, target_texts = batch
            inputs, targets = inputs.to(device), targets.to(device)
        else:
            inputs, targets = batch
            inputs, targets = inputs.to(device), targets.to(device)
            target_texts = None
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        all_preds.extend(predicted.cpu().numpy())
        all_targets.extend(targets.cpu().numpy())
        # 获取预测文本和目标文本用于 BLEU 评分
        if target_texts is not None:
            pred_texts                                                    =
output_encoder.inverse_transform(predicted.cpu().numpy())
            all_pred_texts.extend(pred_texts)
            all_target_texts.extend(target_texts)
all_preds = np.array(all_preds)
all_targets = np.array(all_targets)
# 计算分类评估指标
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,
```

```
confusion_matrix
accuracy = accuracy_score(all_targets, all_preds)
f1 = f1_score(all_targets, all_preds, average='weighted')
precision = precision_score(all_targets, all_preds, average='weighted')
recall = recall_score(all_targets, all_preds, average='weighted')
print("\n 分类评估指标:")
print(f"准确率: {accuracy:.4f}")
print(f"F1 分数: {f1:.4f}")
print(f"精确率: {precision:.4f}")
print(f"召回率: {recall:.4f}")
# 计算 BLEU-2 分数
def calculate_bleu2(references, candidates):
    # 使用平滑函数
    smoothing = SmoothingFunction().method1

    # 计算每对文本的 BLEU-2 分数
    bleu_scores = []
    for ref, cand in zip(references, candidates):
        # 分词
        ref_tokens = tokenize_text(ref)
        cand_tokens = tokenize_text(cand)

        # 计算 BLEU-2 分数，weights=(0.5, 0.5)表示使用 1-gram 和 2-gram
        if len(cand_tokens) > 0:  # 确保候选文本非空
            bleu = sentence_bleu([ref_tokens], cand_tokens, weights=(0.5, 0.5),
smoothing_function=smoothing)
            bleu_scores.append(bleu)

    # 计算平均 BLEU-2 分数
    if bleu_scores:
        return sum(bleu_scores) / len(bleu_scores)
    else:
        return 0.0
# 计算 BLEU-2 分数
if all_pred_texts and all_target_texts:
    bleu2_score = calculate_bleu2(all_target_texts, all_pred_texts)
    print(f"\nBLEU-2 分数: {bleu2_score:.4f}")

    # 打印一些示例进行比较
    print("\n 预测样本与真实标签比较:")
    for i in range(min(5, len(all_pred_texts))):
        print(f"样本 {i+1}:")
        print(f"真实: {all_target_texts[i]}")
        print(f"预测: {all_pred_texts[i]}")
```

```
            print(f"BLEU-2: {sentence_bleu([tokenize_text(all_target_texts[i])], tokenize_text(all_pred_texts[i]), weights=(0.5, 0.5), smoothing_function=SmoothingFunction().method1):.4f}")
            print()
# 自定义人岗匹配评价指标
def calculate_job_match_rate(y_true, y_pred, output_encoder, threshold=0.8):
    """
    计算人岗匹配率：成功匹配的候选人比例
    """
    # 解码预测和真实标签
    y_true_decoded = output_encoder.inverse_transform(y_true)
    y_pred_decoded = output_encoder.inverse_transform(y_pred)

    # 匹配计数
    match_count = sum(1 for true, pred in zip(y_true_decoded, y_pred_decoded) if true == pred)
    # 计算匹配率
    match_rate = match_count / len(y_true)
    return match_rate
# 计算人岗匹配完成度
match_rate = calculate_job_match_rate(all_targets, all_preds, output_encoder)
print(f"人岗匹配完成度: {match_rate:.4f}")
# 保存模型
torch.save({
    'model_state_dict': model.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'input_encoder': input_encoder,
    'output_encoder': output_encoder,
    'input_cols': input_cols,
    'output_cols': output_cols,
    'model_params': {
        'vocab_size': vocab_size,
        'output_size': output_size,
        'd_model': d_model,
        'nhead': nhead,
        'num_layers': num_layers,
        'dim_feedforward': dim_feedforward,
        'dropout': dropout
    }
}, 'job_salary_transformer.pth')
print("模型保存成功!")
# 预测函数
def predict(model, text_input, input_encoder, output_encoder, input_cols):
    model.eval()
```

```
    # 创建输入文本序列
    input_text = " ".join([f"{col}:{text_input[col]}" for col in input_cols])
    # 将文本转换为索引
    try:
        input_idx = input_encoder.transform([input_text])[0]
    # 转换为张量
    input_tensor = torch.tensor([input_idx], dtype=torch.long).to(device)
    # 进行预测
    with torch.no_grad():
        output = model(input_tensor)
        _, predicted = torch.max(output, 1)
    # 解码预测结果
    output_text = output_encoder.inverse_transform(predicted.cpu().numpy())[0]
    # 解析输出文本到各个特征
    output_parts = {}
    for part in output_text.split():
        if ':' in part:
            key, value = part.split(':', 1)
            output_parts[key] = value
    return output_parts, output_text
```