基于 Stacking 集成学习与多模型融合的人岗匹配与就业状态预测

摘要

随着全球经济的快速发展，劳动力市场面临着前所未有的挑战，如何有效地实现人岗匹配，促进就业，成为当今社会亟待解决的问题。本研究聚焦于宜昌地区，通过构建精准的人岗匹配模型，结合**宏观经济数据、失业者个人特征以及岗位需求**，提出创新的解决方案，旨在提升劳动力市场的资源配置效率并优化就业结构。

在**问题 1** 中，研究通过数据分析揭示影响就业状态的关键因素，如年龄、学历、性别等，**通过计算失业注销时间可以发现就业人数为 4036 人，失业人数为 944 人**，并且高学历群体的就业率显著高于低学历群体。通过对这些特征的分析，为后续模型的构建提供关键的基础数据，确保预测模型的准确性和科学性。

在**问题 2** 中，基于问题一的特征分析，构建了四个预测模型，包括**决策树、随机森林、梯度提升树和 XGBoost**。经过模型训练和评估，四个模型的表现有所不同。**决策树的准确率为 72.99%，随机森林的准确率为 80.64%，表现出较高的稳定性和较好的泛化能力。梯度提升树在准确率上稍微优于随机森林，达到了 81.63%，XGBoost 的准确率为 80.39%。**为了进一步优化模型的性能，进行了交叉验证和超参数调优，优化后的模型表现显著提升。最终，**选取了这四个模型并将其应用于 Stacking 集成学习方法**。Stacking 集成模型结合了决策树、随机森林、梯度提升树和 XGBoost 的优点，经过集成后，**模型的准确率提升至 84.29%，查准率为 84.16%，召回率为 99.85%，F1 分数为 0.9069**，展示了较高的准确性和稳定性。Stacking 集成模型在所有评估指标上均表现优异，特别是在召回率上几乎达到 100%，证明了其在预测失业和就业状态时的高效性。

在**问题 3** 中通过**结合宏观经济因素、个人特征以及外部数据**，优化了就业状态预测模型，聚焦于宜昌地区的就业预测与人岗匹配问题。我们首先收集了与就业相关的**宏观经济数据，包括经济增长阶段、失业率、通货膨胀率、政府就业政策、行业招聘需求、职位空缺率、宜昌市 CPI 水平以及招聘信息更新频率**等。将这些外部因素与问题一中提取的个人层面特征结合，**基于问题二构建了预测模型**，初步评估结果显示，XGBoost 模型在准确率方面表现突出，为准确率 **80.39%**。 接着进一步对这些模型进行了交叉验证和超参数调优，并通过 Stacking 集成学习方法将四个模型的优点进行了融合，最终 Stacking 集成模型的准确率达到 **83.54%**，召回率为 **100%**，F1 分数为 **0.9070**，展示了集成方法的优势。通过引入外部经济数据，**模型的预测能力得到了显著提升**。

在**问题 4** 中，结合 **K-means 聚类**与**余弦相似度**的精准人岗匹配模型被提出。通过 K-means 算法对失业者进行聚类，再**结合余弦相似度计算失业者与岗位之间的匹配度，为每个失业者提供个性化的岗位推荐**。该方法通过有效的群体分析与匹配度计算，显著提升岗位推荐的准确性和个性化程度，为失业人员提供更合适的工作机会。

本研究创新性地构建高效的人岗匹配系统，通过机器学习与精准匹配算法提升岗位推荐的精准度。与传统方法不同，本研究整合失业者个人特征、岗位需求和外部经济因素，显著优化匹配效果。该模型为提高就业效率和支持政府就业政策提供数据依据。未来，可以结合实时数据和深度学习技术，进一步提升模型的智能化水平。

关键词：就业预测；XGBoost ；stacking 集成模型；K-means 聚类；余弦相似度

一、问题的背景与重述

1.1 问题背景

就业问题一直是社会经济中的核心议题，直接关系到人民的生计和社会的稳定[1]。随着全球化的深入发展和技术的不断进步，世界各国都面临着不同形式的就业问题。在中国，尤其是近几年，尽管经济增速保持平稳，但就业市场仍然面临一些挑战，特别是就业结构性矛盾的突出。大城市中的人才竞争激烈，农村及偏远地区的就业机会较为匮乏，低技能劳动者面临着较高的失业风险[2]。特别是针对青年、女性和高学历人群等特殊群体，如何实现高质量、充分的就业已成为政府和社会各界高度关注的问题。

随着信息技术和大数据的迅猛发展，传统的就业管理和政策制定方式正面临着前所未有的挑战。如何利用现代化的科技手段，通过数据分析来预测就业状态，优化就业资源配置，提升就业政策的精准度，成为了当前亟待解决的重要问题。

宜昌地区作为一个典型的中等城市，具有人口众多、就业结构复杂的特点。该地区的经济发展水平较为稳定，但在就业结构、就业质量和就业群体的分布上仍然存在较大差异。因此，如何通过科学的模型对该地区的就业状态进行分析，并制定适合的就业政策，是当前亟需解决的实际问题。

1.2 问题重述

本研究的核心目标是通过数学建模方法，对宜昌地区的就业状态进行分析与预测，为相关部门制定符合当地实际的就业政策提供科学决策依据。具体的研究任务包括以下几个方面：

1）问题 1：就业状态特征分析

分析宜昌地区当前的就业状况，探讨影响就业状态的主要因素，如年龄、性别、学历、行业等。通过对这些特征的深入分析，了解各个因素对就业状态的具体影响。

2）问题 2：就业状态预测模型构建与评估

根据问题 1 的分析结果，选取与就业状态具有较高相关性的特征，构建就业状态预测模型。使用机器学习等技术对模型进行训练与评估，并对模型的效果进行验证，旨在准确预测该地区未来的就业状态。

3）问题 3：模型优化与外部经济因素引入

为提高模型的准确性与适用性，本研究将考虑宏观经济因素、政策变化、行业发展等外部变量的影响，进一步优化就业状态预测模型。通过整合外部数据，提升模型对复杂就业市场的预测能力。

4）问题 4：精准人岗匹配模型的构建

在就业状态分析和预测的基础上，构建精准人岗匹配模型，帮助失业人员找到最适合的工作岗位。该模型将考虑个人特征与岗位要求之间的匹配度，以实现更高效的就业推荐。

综上所述，本研究不仅关注就业状态的分析与预测，还通过引入外部因素与创新算法，提升模型的准确性与实际应用性，为地方政府提供切实可行的就业决策支持。

二、模型假设

为确保模型的简洁性和可操作性，本文在构建和求解模型时做出合理的假设：

1）本研究假设所使用的宜昌地区数据能够代表该地区的整体就业状态。

2）影响就业状态的主要特征与就业状态之间存在线性关系。

3）模型中的各个特征相互独立，不考虑特征之间的相关性。

4）在短期内，宏观经济变化对就业状态的影响较小，可忽略不计。

5）就业状态可以通过失业注销时间准确界定，1 代表就业，0 代表失业。

三、符号说明

本文所使用的符号及意义说明如表 2.1 所示。

表 2.1 符号说明

符号	符号说明	量纲
Y	就业状态（1 表示就业，0 表示失业）	无量纲
$X_1, X_2, \ldots, X_n$	失业者个人特征（如年龄、学历、工作经验等）	年、等级、年数、类别等
$\beta_0, \beta_1, \ldots, \beta_n$	回归系数（用于回归模型）	无量纲
M	匹配度（反映失业者与岗位的匹配情况）	无量纲
K	聚类数（K-means 聚类算法的簇数）	无量纲
Inertia	聚类误差平方和（K-means 中的目标函数值）	无量纲
Cosine	余弦相似度（用于匹配度计算）	无量纲
$D(A,B)$	欧几里得距离（用于计算匹配度）	单位：特征值的单位
w_i	特征权重（用于加权匹配度计算）	无量纲
X_{ij}	失业者i和岗位j的特征值	特征的单位
$\alpha, \beta, \gamma, \ldots$	特征的权重系数（加权平均中的系数）	无量纲
$\mathbf{v_i}$	失业者i的特征向量	各特征单位组成的向量
$\mathbf{v_j}$	岗位j的特征向量	各特征单位组成的向量
M_{total}	综合匹配度（加权平均后的匹配度）	无量纲
$Y_{\text{recommend}}$	推荐岗位列表（每个失业者推荐的岗位）	无量纲
A, B	两个向量或特征集	特征的单位

四、数据概况与预处理

4.1 数据来源与内容

本研究使用的数据集来自**宜昌地区**，包含 **5000** 名被调查者的个人信息及就业相关数据。数据集中的变量涵盖**个人特征**、**就业状态**等多个方面，以下是主要变量的描述。

表 4.1 数据集主要变量及说明

变量名称	描述	类型
年龄	被调查者的年龄	连续型
性别	被调查者的性别	类别型
学历	被调查者的学历	类别型
行业	被调查者所在的行业类型	类别型
失业注销时间	失业注销的时间，判断就业状态	日期型
求职意愿	被调查者是否有求职意愿	类别型
就业状况	根据失业注销时间判断就业状态	类别型
……	……	……

这些变量为构建预测模型提供基础数据，其中部分变量存在缺失值，需要进行清洗与处理

4.2 数据清洗、缺失值处理与数据可视化

数据清洗过程的第一步是去除**重复值**，确保数据的唯一性与准确性。接着，所有的“\N”字符被视为缺失值，并统一替换为 NaN，保证缺失值的一致表示，便于后续处理。

对于缺失值较多的列（如**居住状态、备注、是否参加社会保险、合同终止日期**等），这些列对就业状态预测的贡献较小且缺失值较为严重，因此决定删除这些列。对于个人信息中的缺失值（如**年龄、性别、学历等**），仅删除含缺失值的**行**，保留其他有效数据。

为更好地理解数据的分布情况，进行数据可视化分析。下图展示各列缺失值的数量统计，帮助识别哪些变量存在大量缺失数据，并为后续的数据处理决策提供依据。

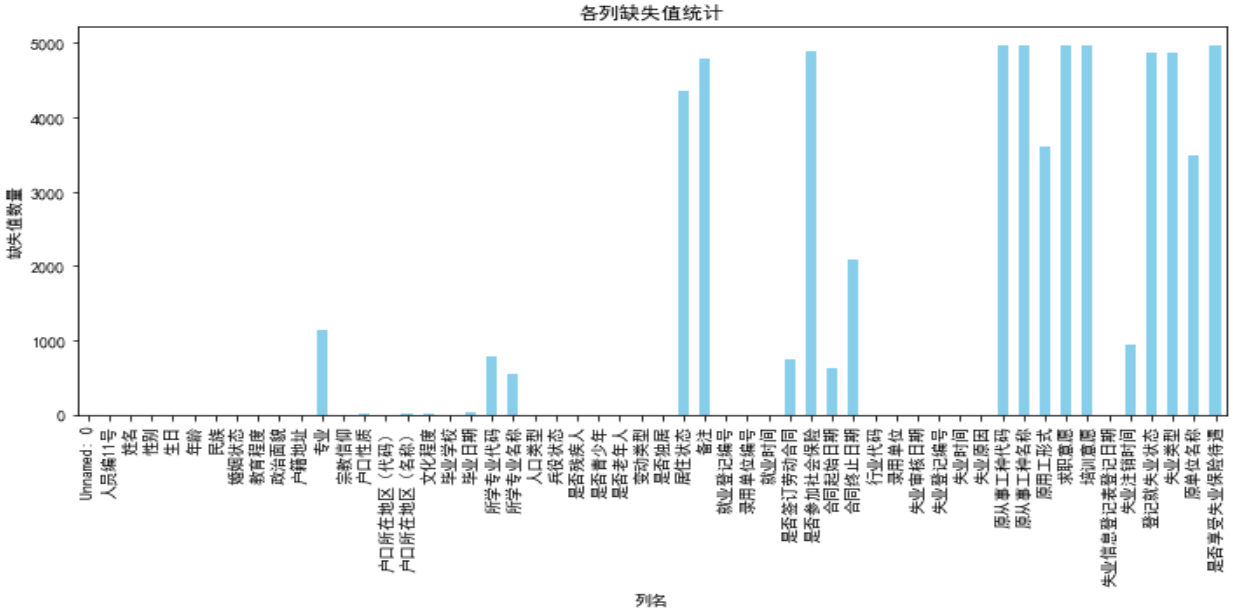

图 4.1 就业与失业状态分布

从图中可以看出，数据集中各列的缺失值数量差异较大，部分列（如居住状态、备注、是否参加社会保险等）缺失值较多，而其他列（如年龄、性别、学历等）较为完整。特别是居住状态列缺失值接近 5000，表明该列数据几乎无效，因此决定删除这类高缺失值的列。对于缺失值相对较少的列，这些列可能对预测有价值，因此保留并通过删除缺失值行的方式进行处理。这些分析为后续数据清理和模型训练提供参考。

4.3 特征选择、分析与目标变量定义

在数据清洗后，进行特征选择与分析。通过与目标变量（就业状态）的相关性分析，发现**年龄、学历、性别、行业**等特征与就业状态有较强的相关性。因此，保留这些关键特征，剔除冗余或无关变量，从而提高模型的预测能力。

目标变量“就业状况”根据失业注销时间来定义。具体规则如下：

表 4.2 就业状况定义依据失业注销时间

失业注销时间	就业状况
为空	0（失业）
存在日期	1（就业）

基于此定义，新增一列“就业状况”，其中 1 代表就业，0 代表失业。该定义为后续模型训练与预测提供了清晰的目标变量。

4.4 数据标准化与数据集划分

为消除特征之间的量纲差异，数据进行标准化处理。标准化公式如下：

$$x' = \frac{x - \mu}{\sigma} \tag{4.1}$$

其中，x为原始数据，μ为特征的均值，σ为符征的标准差，x'为标准化后的数据。标准化后的数据有助于提高模型的训练效果。

在数据预处理完成后，将数据集划分为训练集和测试集。**80%的数据用于训练模型，20%的数据用于评估模型的泛化能力**。这种划分方法确保模型能够有效地在未知数据上进行评估。

五、就业状态特征分析与影响因素探讨

本章将深入分析影响就业状态的关键特征，并探讨这些因素在预测模型中的重要性及其影响。

5.1 问题分析

就业状况是反映一个地区经济健康和社会稳定的重要指标。了解不同群体的就业和失业状态，有助于政府制定有效的就业政策，提升就业市场的整体效率和公平性。在当前经济环境下，失业问题尤为突出，尤其是长期失业群体，亟需关注。

在本研究中，以该地区为例，分析该地区的就业状态，并从年龄、性别、学历、行业、失业时间等维度探讨这些因素对就业状态的影响。通过这些分析，能够为区域就业政策的优化提供有力的依据。

5.2 当前就业状态分析

首先，研究该地区的就业和失业状态，以了解当前的就业市场状况。从数据中可以看出，就业人数显著高于失业人数，显示出该地区整体就业状况较好。具体来说，就业群体的规模明显大于失业群体，但失业者在整体就业市场中仍占有一定比例，显示出就业市场的差异性。

为直观展示就业和失业情况，以下表格展示当前的就业与失业人数：

表 5.1 当前就业情况

就业失业状态	就业	失业
数量（人）	4036	944

通过这张表格可以看出，该地区的就业人数远高于失业人数，这为后续的失业群体分析和失业时间分布提供基础。

接下来，通过图进一步分析就业和失业人数的分布，展示该地区的就业与失业状况。

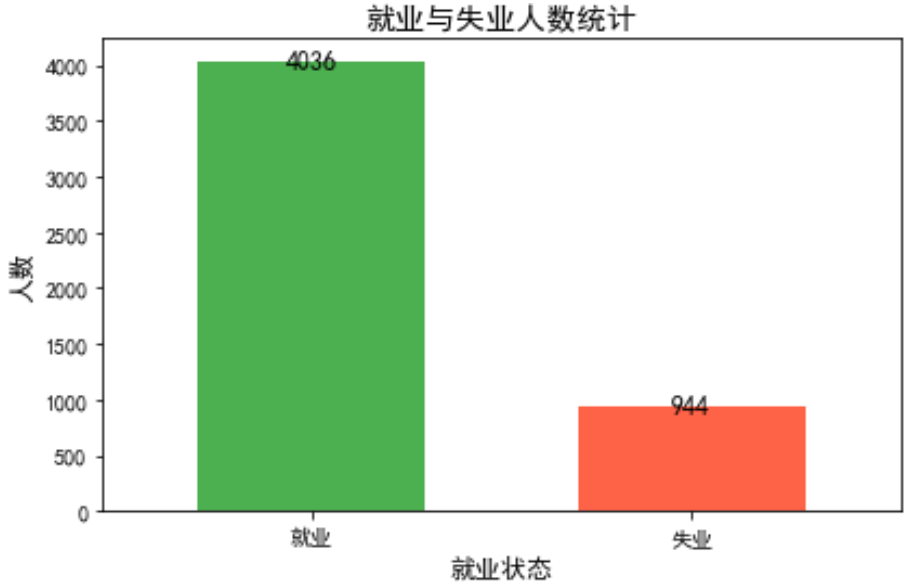

图 5.1 就业与失业人数统计

图中展示该地区的就业与失业分布情况。从图中的数据可以看出，**就业人数（4036**

人）明显高于失业人数（944 人）。这种情况表明该地区的整体就业市场表现良好，尽管失业人数较少，仍需要关注失业群体的就业状态，特别是长期失业的个体，因其面临更多的就业挑战。

5.3 失业时间分布分析与失业状态影响

失业时间的分布对于分析失业群体的特征至关重要。通过对失业时间的分布图进行分析，可以进一步了解失业群体的分布情况和失业的持续时间。失业时间呈现出明显的右偏分布，大多数失业者的失业时间较短，集中在较低的失业时间区间。然而，随着失业时间的增加，失业人数逐渐减少，且出现显著的长尾部分，说明有少部分失业者的失业时间较长。

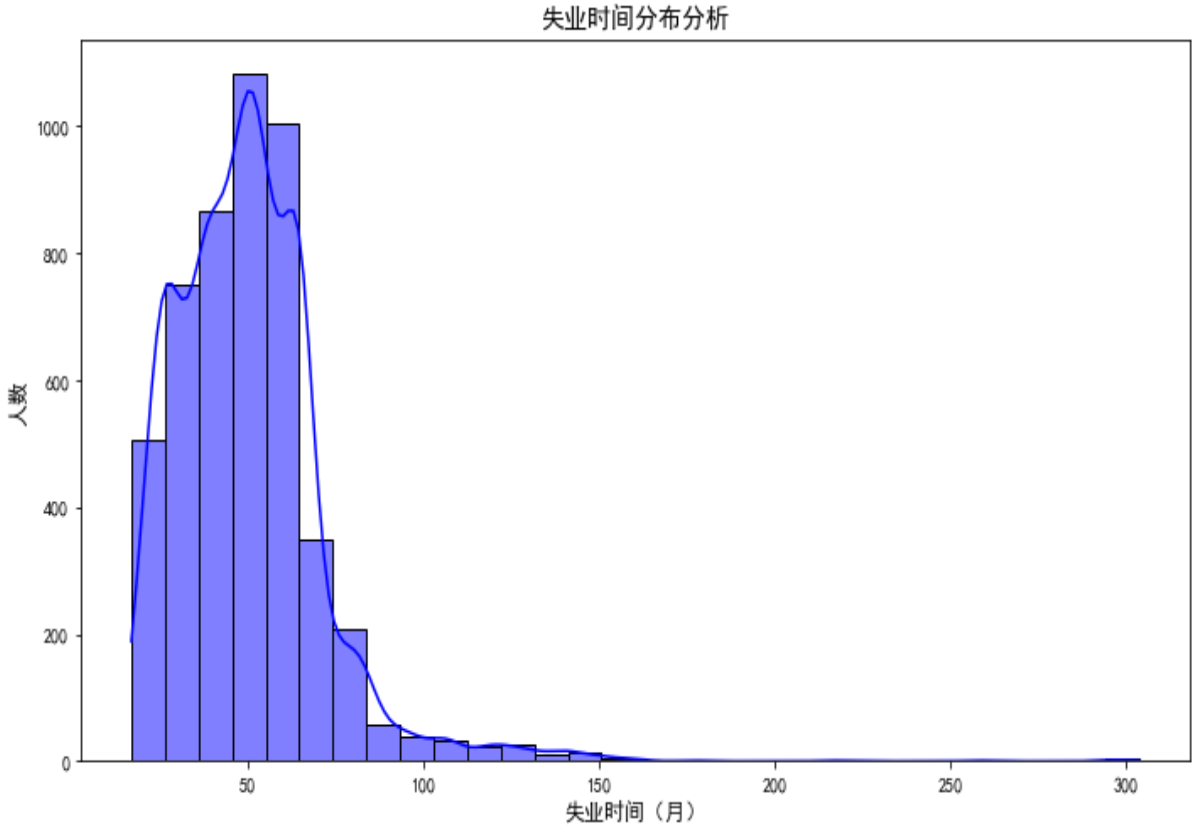

图 5.2 失业时间分布

从图中分析可以得出，失业时间分布呈右偏态，显示出短期失业群体的比例较大。大多数失业者的失业时间较短，通常能够较快找到工作，而少部分失业者的失业时间较长，属于长期失业群体。这一现象反映出在该地区，**短期失业群体占大多数**，他们的就业过程较为顺利，主要受到短期经济波动或个人职业变化的影响。而**长期失业者则面临更为复杂的就业挑战**，如技能不匹配、行业萎缩等问题，因此需要更为针对性的政策和支持，帮助他们重新融入劳动力市场。

5.4 年龄、性别与学历对就业状态的影响

年龄、性别和学历是影响就业状态的关键因素。在分析就业群体的特点时，考虑到这些因素的作用可以帮助更好地理解不同群体在劳动力市场中的表现差异。年龄、性别和学历与个体的就业机会密切相关，不同的年龄段、学历层次和性别可能会影响个体的就业稳定性和职场竞争力。

在接下来的分析中，将探讨这些因素如何在就业和失业的状态中发挥作用，并分析它们对个体就业机会的影响。

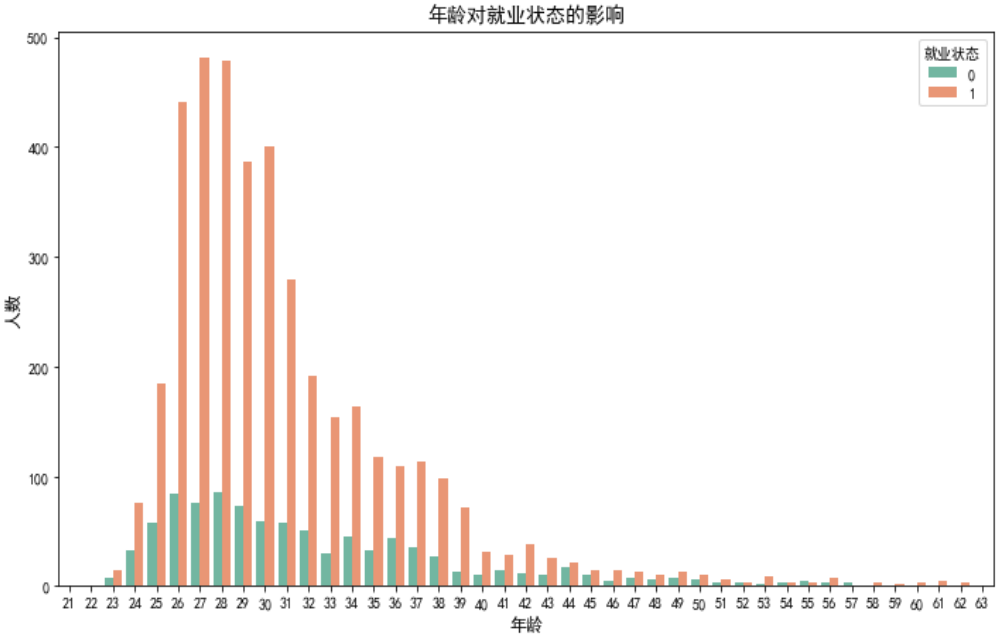

图 5.3 年龄对就业状态的影响

从图 5.3 中的数据分析可以看出，**年轻群体的失业率较高**，尤其是在 **22 至 28 岁之间**，这一群体通常面临较大的就业挑战，主要由于**缺乏工作经验**和**适应行业要求的能力较弱**。此外，年轻人倾向于进入**快速变化的行业**或**初级岗位**，这些岗位的市场需求不稳定，导致失业风险较高。与此相对，**30 岁以上的群体失业率较低**，因为该群体通常具备**丰富的工作经验**和**较强的市场适应能力**，使得他们更容易在就业市场中找到稳定工作机会。这一趋势反映了**年龄与就业之间的密切关系**，同时突出了年轻群体面临的挑战，证明了**工作经验与市场适应能力**对就业状态的决定性影响。

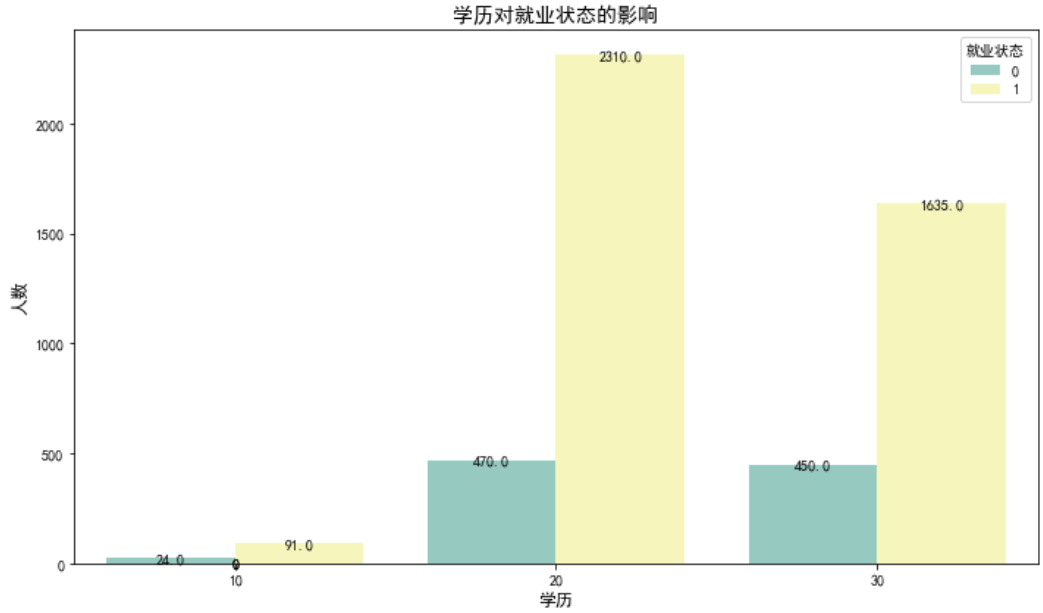

图 5.4 学历对就业状态的影响

学历对就业状态的影响也显著。数据显示，高学历群体（例如本科及以上学历）明显占据了更大的就业比例，而低学历群体（如中专或职高）则相对较少。这反映高学历个体在劳动力市场中具有更强的竞争力，尤其在技术要求较高的行业中，拥有更高学历的个体更容易找到工作。

性别对就业状态的影响较小。从数据中可以看出，男性群体的就业人数略高于女性群体，但在失业人数方面，男性群体的失业人数（575 人）高于女性群体（369 人）。总体来看，男女群体的就业人数差异不大，表明在该地区性别对就业的影响较小。

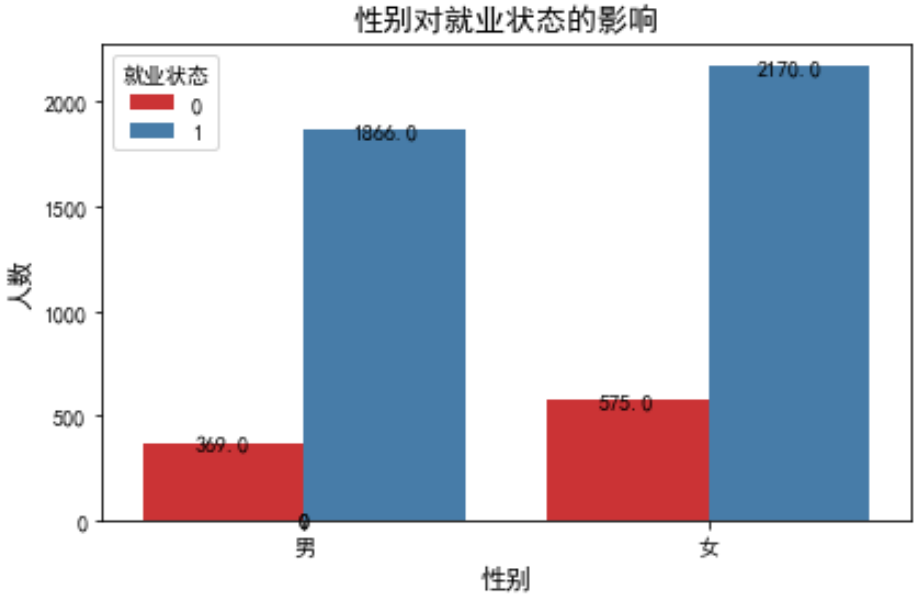

图 5.5 性别对就业状态的影响

从图 5.5 的分析可以得出，**男性失业率较低**，相对来说，男性群体的就业人数和失业人数相对平衡。虽然男性群体的就业人数略低于女性群体，但失业人数较少，这可能是由于男性在某些行业中面临的竞争较为激烈，或因市场压力较大。然而，整体来看，**性别对就业状态的影响较小**，无论男性还是女性，就业群体均占主导地位，表明该地区的就业市场对性别的影响较为平衡。

5.5 行业对就业状态的影响

行业类型对就业状态的影响显著。数据显示，**技术行业和医疗行业的就业比例较高**，这些行业通常对劳动力的需求较为稳定，且薪资水平较高，因此吸引了大量就业者。相比之下，**传统行业（如制造业、服务业）失业率较高**，这与这些行业面临的市场竞争和需求不足密切相关。

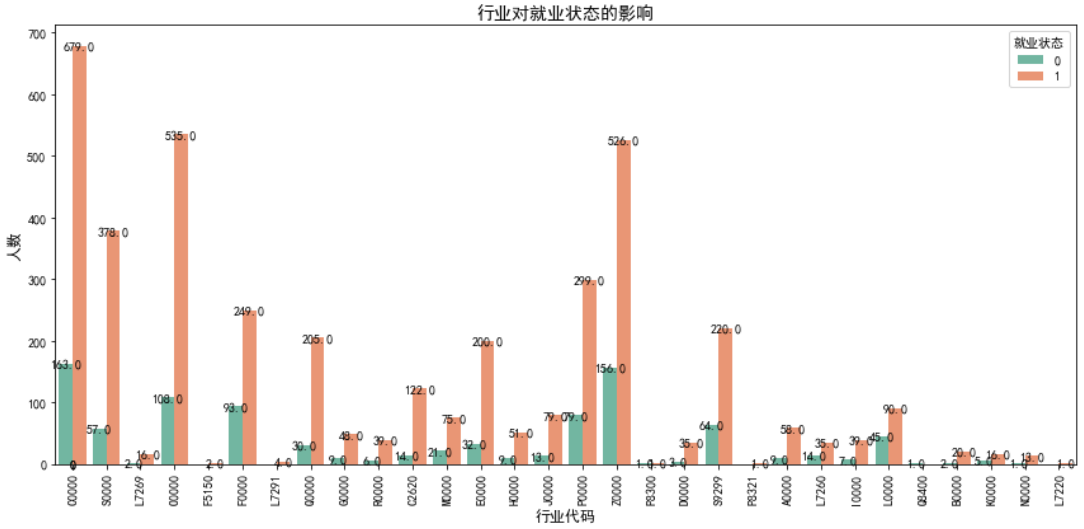

图 5.6 行业对就业状态的影响

从图 5.6 中的分析可以看出，**技术和医疗行业的就业机会较多**。这些行业的工作岗

位通常薪资水平较高，需求稳定，因此吸引大量劳动力。相对而言，**传统行业的失业率较高**，如制造业和服务业面临的市场压力较大，可能导致较高的失业率。这一现象表明，劳动力市场的就业机会更多集中在技术含量较高的行业，而传统行业则需要进一步的转型和创新，以提高就业机会。

5.6 相关性分析

通过对各个特征的相关性进行分析，能够揭示不同变量之间的内在联系。这为进一步构建预测模型、挖掘数据之间的潜在关系提供了重要依据[3]。在本研究中，特别分析了年龄、性别、学历、婚姻状况、行业等特征之间的相关性。

使用 Pearson 相关系数可以定量衡量变量之间的线性关系，公式为：

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{5.1}$$

其中，r为 Pearson 相关系数，X_i和Y_i为两个变量的观测值，$\bar{X}$和$\bar{Y}$为这两个变量的均值，n为数据点的数量。Pearson 相关系数的取值范围从-1 到+1，$r = 1$表示完全正相关，r= -1 表示完全负相关,$\mathbf{r} = 0$表示无相关性。

为进一步展示这些特征之间的具体相关性，图 5.7 展示各特征之间的相关性热力图。

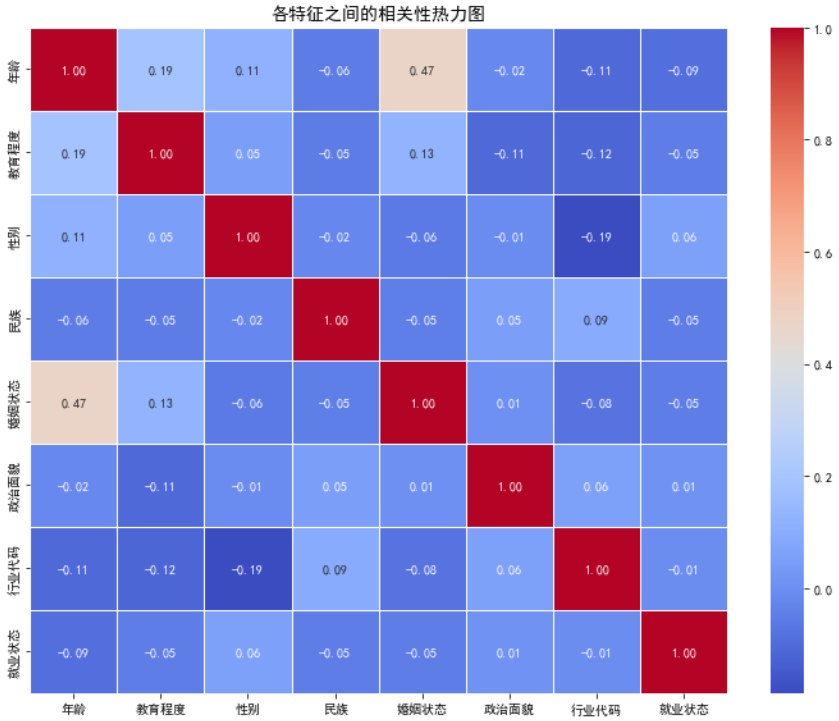

图 5.7 各特征之间的相关性热力图

通过对数据的相关性分析，发现**年龄与婚姻状况**之间存在较强的正相关性（0.47），这表明年长个体通常已婚，而已婚群体的平均年龄也较高。这个现象反映出随着年龄的增长，个体结婚的可能性增大，尤其是在该地区，随着经济和社会发展，年龄较大的人群更多地选择结婚或已经结婚。此外，**学历与年龄**之间的轻微正相关性（0.19）表明，较年轻的人群通常受教育程度较高。这一现象反映近年来教育体系的变化和教育普及程度的提高，尤其是在年轻人群体中，学历水平的提升有助于改善其在劳动力市场中的竞争力。

5.7 失业时间对就业状态的影响

失业时间是衡量失业状态的一个重要指标，研究其对就业状态的影响，可以帮助理

解失业群体的特点以及他们面临的挑战。失业时间对就业状态的影响可以通过对数回归模型进行建模。具体来说，假设失业时间对就业状态的影响符合对数线性关系，即失业时间对就业的影响是一个逐步递减的过程。

回归模型的公式如下：

$$P(Y=1|X)=\frac{1}{1+e^{-(\beta_0+\beta_1\log(\text{Unemployment Time}))}} \tag{5.2}$$

其中，$P(Y=1|X)$为个体就业的概率，β_0为常数项，β_1为失业时间的回归系数 Unemployment Time 为失业时间的对数。该模型假设，随着失业时间的增加，重新就 业的概率逐渐降低，失业时间较长的人群更难恢复就业。

为更好地理解失业时间对就业状态的影响，以下图 5.8 展示失业时间对就业状态的影响。

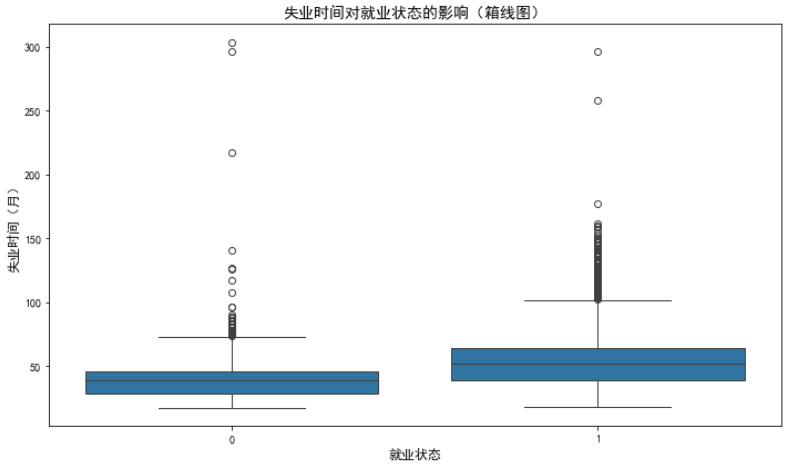

图 5.8 失业时间对就业状态的影响（箱线图）

通过箱线图分析发现，**就业群体的失业时间较短**，大多数失业者能在约 50 天内重新找到工作，表明他们通常具有较强的市场适应能力，能够快速恢复就业。而相对而言，**失业群体的失业时间较长**，其中一些失业者的失业时间接近 300 天，这部分长期失业的个体面临更复杂的就业挑战，如技能不匹配、行业结构性调整等问题。因此，针对这些长期失业者，**应采取更具针对性的政策支持**，例如提供职业技能培训、行业再就业计划等措施，帮助他们提升就业能力，促进其重新融入劳动力市场，尤其在经济转型过程中，这一群体更需要外部支持以克服就业障碍。

六、基于机器学习的就业状态预测模型构建与评估

在本章中，将通过构建和评估多个机器学习模型，探讨如何利用关键特征预测个体的就业状态，并分析各模型的表现和适用性。

6.1 问题分析

本研究的核心目标是通过分析影响就业状态的关键因素，如**年龄**、**性别**、**学历**、**行业**等，构建一个能够准确预测个体就业状态（就业或失业）的模型。通过特征选择，重点选取与就业状态高度相关的变量，确保模型的有效性和精准性。鉴于这些特征在就业市场中的重要性，构建高效的预测模型对于理解就业趋势和制定政策具有重要意义。

为实现这一目标，研究选用四种常见的机器学习算法：**决策树**、**随机森林**、**梯度提升树**和 **XGBoost**。这些算法能够处理复杂的特征关系和非线性问题，适合进行二分类任务。通过对比这些算法的表现，可以为就业状态的预测提供可靠的模型选择，并为未来的就业研究和相关决策提供数据支持。

6.2 模型选择与构建

在本研究中，选择四种常见的机器学习模型来构建就业状态预测模型。这些模型分别是：**决策树、随机森林、梯度提升树和 XGBoost**。每种模型在分类任务中的表现都得到广泛的应用，并且能够处理数据集中的复杂特征关系和非线性问题。

6.2.1 决策树模型

决策树是简单而直观的分类模型。它通过将特征空间划分为多个子空间，从而对目标变量进行分类[4]。在每一层中，决策树根据某个特征值的分裂规则将数据分到不同的子节点，并通过递归的方式形成树结构。通过这种方式，决策树能够从根节点到叶节点形成一系列的规则，这些规则可以用于预测目标变量。

决策树的数学公式可以通过信息增益或基尼指数来进行节点分裂的决策。其信息增益公式为：

$$\text{Information Gain}(T, A) = \text{Entropy}(T) - \sum_{v \in \text{Values}(A)} \frac{|T_v|}{|T|} \text{Entropy}(T_v) \tag{6.1}$$

其中，$\text{Entropy}(T)$表示数据集T的熵，$\text{Entropy}(T_v)$表示在特征A的某一值v下，数据集T_v的熵，$\frac{|T_v|}{|T|}$表示数据集T_v在T中的占比。

6.2.2 随机森林模型

随机森林是集成学习方法，通过训练多个决策树来进行预测。与单一决策树不同，随机森林通过“袋装法”生成多个决策树，每棵树在数据的不同子集上训练，然后将所有树的预测结果进行集成，得到最终的预测结果。随机森林的预测公式为：

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} f_i(x) \tag{6.2}$$

其中，$\hat{y}$是随机森林的最终预测值，N是决策树的数量，$f_i(x)$是第i棵树的预测结果。通过这种方式，随机森林能够通过集成多个模型的力量，提高分类精度，并减少过拟合现象。

6.2.3 梯度提升树模型

梯度提升树是基于集成学习的模型，通过逐步构建树的方式来提升模型的预测性能。每一次新增的树都是为纠正前一次树的错误。它通过梯度下降法优化损失函数，每次优化都是对前一次结果的修正。梯度提升树的数学公式可以表示为：

$$f(x)^{(t)} = f(x)^{(t-1)} + \eta \cdot h_t(x) \tag{6.3}$$

其中，$f(x)^{(t)}$表示第t次迭代后的模型，$f(x)^{(t-1)}$表示前一次迭代的模型，η是学习率，$h_t(x)$是第t次迭代训练得到的弱学习器(树)。

6.2.4 XGBoost 模型

XGBoost 是优化过的梯度提升树算法，它通过采用正则化项来提高模型的鲁棒性，并通过列采样和行采样等方式防止过拟合。XGBoost 能够在大规模数据集上进行高效训练，具有较强的预测能力。XGBoost 的目标函数可以表示为：

$$L(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{6.4}$$

其中，$l(y_i, \hat{y}_i)$表示损失函数，$\hat{y}_i$是模型的预测值，y_i是真实值，$\Omega(f_k)$是正则化项，

f_k是第k棵树，正则化项用于控制模型的复杂度，防止过拟合。

6.2.5 对数回归模型

为进一步分析就业状态与失业时间的关系，采用对数回归模型。对数回归模型常用于二分类任务，它预测某个事件发生的概率，模型输出是一个在 0 到 1 之间的概率值，用于表示就业(1)或失业(0)的可能性[5]。对数回归模型的数学公式为：

$$P(Y=1|X)=\frac{1}{1+e^{-(\beta_0+\beta_1\cdot X_1+\beta_2\cdot X_2+\cdots+\beta_n\cdot X_n)}} \tag{6.5}$$

其中，$P(Y=1|X)$表示就业的概率，$X_1,X_2,\cdots,X_n$为特征，$\beta_0,\beta_1,\beta_2,\cdots,\beta_n$为模型的回归系数。该公式的输出值通过 sigmoid 函数将特征的线性组合映射到[0,1]之间，表示事件发生的概率。

6.2.6 初步模型评估

通过对每个模型的初步训练与评估，得出以下指标结果，评估标准包括准确率、查准率、召回率和 F1 分数等多项指标：

表 6.1 初步模型评估结果

模型	准确率	查准率	召回率	F1 分数
决策树	72.99%	84.71%	82.32%	0.835
随机森林	80.64%	83.08%	96.29%	0.892
梯度提升树	81.63%	83.33%	97.33%	0.898
XGBoost	80.39%	83.91%	94.50%	0.889

从结果可以看出，**梯度提升树**模型在召回率和F1分数方面的表现最优，达到**97.33%和0.898**，因此，梯度提升树被选为初步评估中表现最好的模型。

6.2.7 基础学习曲线分析

为进一步分析各模型的学习情况，展示不同训练集大小下模型的表现。以下图表显示**学习曲线**，展示训练得分与交叉验证得分随训练集大小的变化情况。

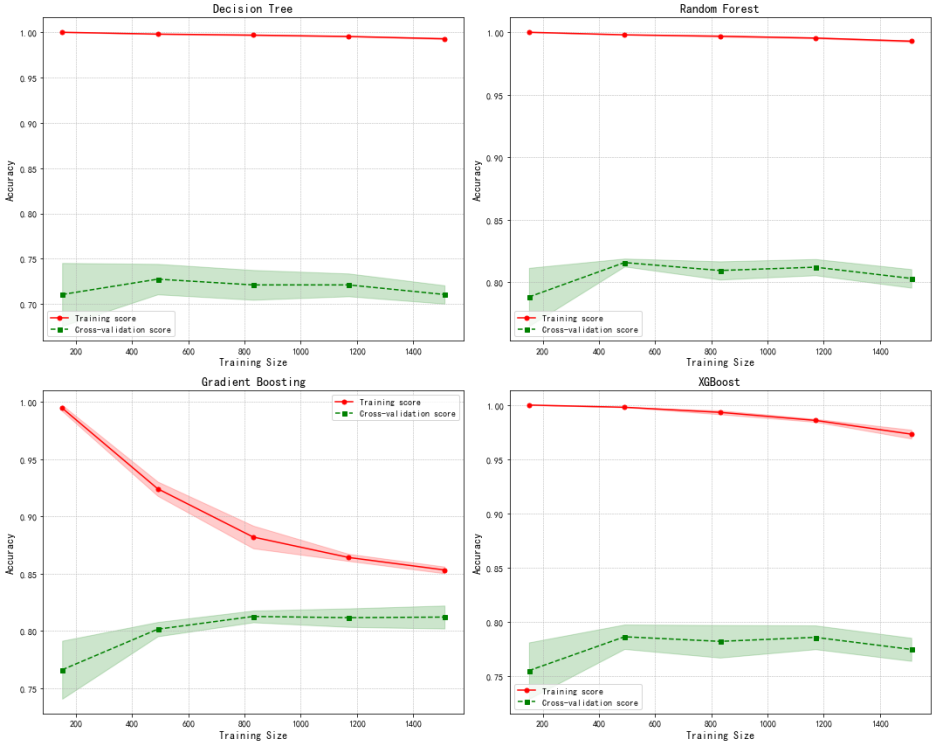

图 6.1 学习曲线

通过分析学习曲线，**随机森林**表现最为稳定，其训练得分和交叉验证得分均保持较

高且变化较小，显示了**强泛化能力**，适合处理复杂数据集。**决策树**则在训练集上拟合良好，但存在明显的过拟合问题，交叉验证得分随着训练集增大明显下降。**梯度提升树**和 **XGBoost** 模型也表现出过拟合的迹象，尽管训练得分较高，但交叉验证得分未能有效提升，特别是在更复杂的数据集上。**随机森林**的稳定性和泛化能力使其成为最优选择，而 **XGBoost** 和**梯度提升树**则需要进一步调参和优化，以减少过拟合并提升其适应性。

6.2.8 基于 Stacking 集成学习的模型优化与性能提升

首先对各个单独的模型进行评估，并通过集成学习方法对这些模型的优点进行整合。为了优化模型的表现，选择对四个主要模型（决策树、随机森林、梯度提升树、XGBoost）进行 **Stacking 集成学习**。集成学习方法能够通过结合多个模型的预测结果，进一步提高分类性能和稳定性。

Stacking 集成学习的优势在于，它通过构建一个新的模型，将各个基学习器的输出作为特征输入到该模型中，以获得更强的预测能力。模型的学习过程通过对这些基模型的预测结果进行加权，从而利用每个模型在特定方面的优势，提高整体性能。

在本研究中，**Stacking 集成学习**的结果显示，它相比单一模型的表现有了明显提升。以下是各个模型和 Stacking 集成学习的性能对比：

表 6.2 调参后模型评估结果

模型	准确率	查准率	召回率	F1 分数
决策树	82.00%	83.14%	98.22%	0.9005
随机森林	82.74%	83.02%	99.55%	0.9054
梯度提升树	82.86%	82.96%	99.85%	0.9063
XGBoost	83.11%	83.17%	99.85%	0.9075
Stacking 集成学习	**84.29%**	**84.16%**	**99.85%**	**0.9069**

从上表可以看出，**Stacking 集成学习**方法的准确率为 **84.29%**，明显高于单一模型的表现。这表明，集成方法通过将多个模型的优点结合，显著提高模型在整体预测任务中的性能，尤其是在准确率和查准率方面的提升非常显著。

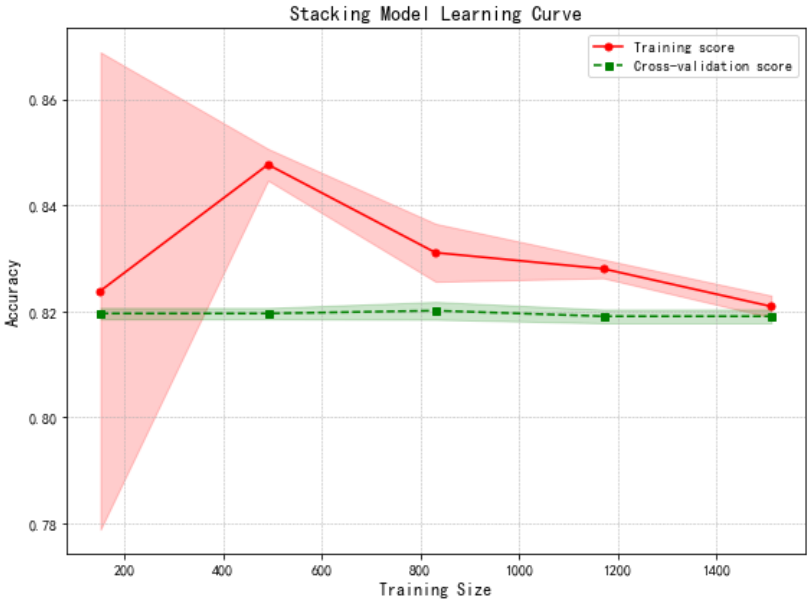

图 6.2 Stacking 学习曲线

集成学习的优势也通过学习曲线得到验证，图 6.2 展示 Stacking 集成学习模型的学习曲线。学习曲线显示，不同训练集大小下，Stacking 集成模型的训练得分与交叉验证得分均保持较高且稳定的水平，表明该模型具有较强的泛化能力，能适应各种数据变化。

通过这些结果可知，Stacking 集成学习方法在提升预测精度方面具有重要作用，尤其在处理具有复杂特征的数据时，集成方法能显著提升预测模型的稳定性和准确性。

6.2.9 结果分析与结论

在通过 Stacking 集成学习优化后的模型评估中，结果表明，集成模型在各项评估指标上表现优于单一模型，尤其在准确率和查准率方面取得了显著提升。集成模型的准确率达到了 84.29%，F1 分数为 0.9069，进一步证明了集成学习在提升模型性能方面的有效性。

6.3 特征重要性分析

在本部分中，展示通过四个主要机器学习模型（决策树、随机森林、XGBoost 等）对就业状态预测任务中各特征重要性评估的结果。这些图表展示了不同模型在特征选择方面的偏好，能够帮助我们理解哪些特征对就业状态的预测具有决定性影响。

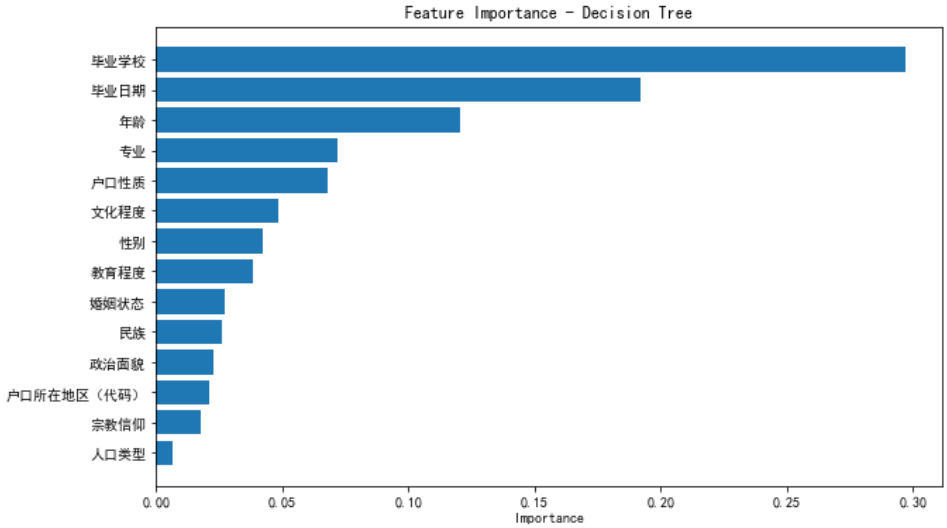

图 6.3 决策树模型特征重要性

图 1 展示了决策树模型中各特征的重要性。**毕业学校**、**毕业日期**和**年龄**是影响预测结果的主要特征，表明这些因素在模型中起到了决定性作用。特别是**毕业学校**和**毕业日期**，它们反映了教育背景和职业起步阶段，对就业状态预测具有较大影响。

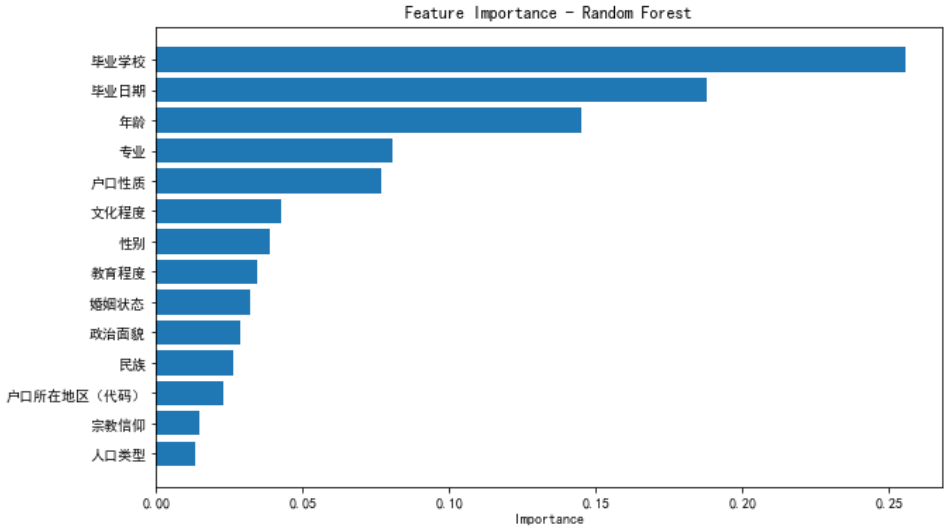

图 6.4 随机森林模型特征重要性

图 2 展示了随机森林模型的特征重要性分析。与决策树模型类似，**毕业学校**和**毕业**

日期在随机森林中也占据了较高的排名。此外，**年龄**和**文化程度**等变量也表现出较高的重要性，表明这些特征对求职者的就业可能性有显著影响。

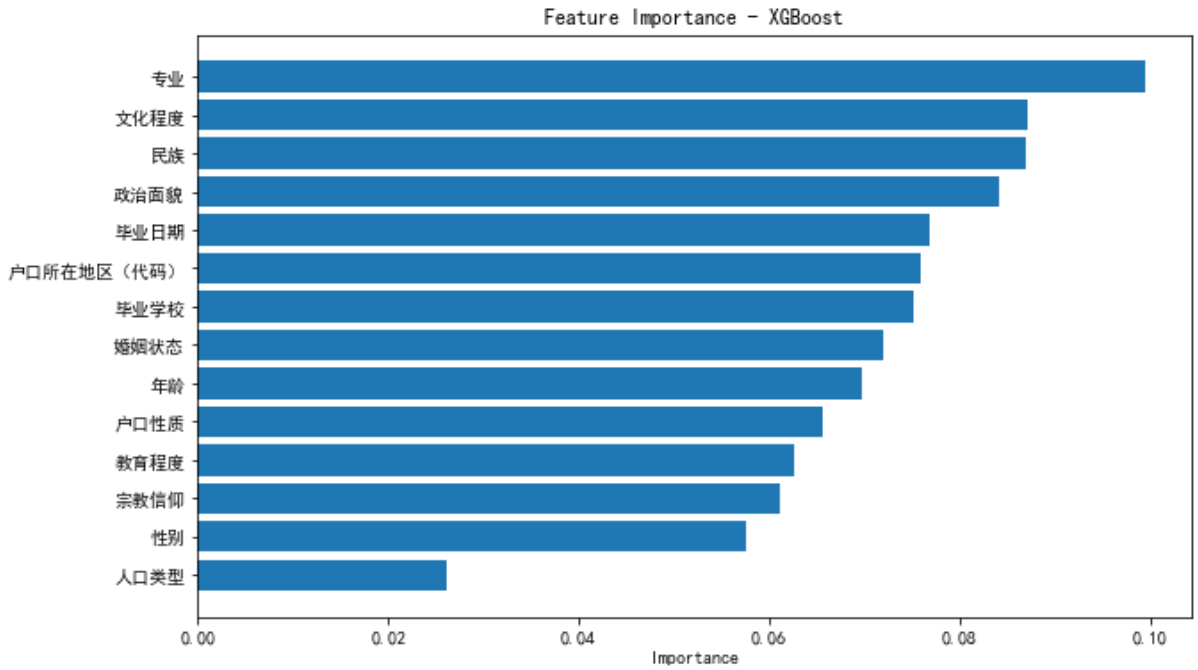

图 6.5 XGBoost 模型特征重要性

图 3 展示了 XGBoost 模型中的特征重要性，**专业**和**文化程度**成为最关键的预测因素。XGBoost 通过集成多个弱学习器来加权计算特征的重要性，**专业**在预测中占据主导地位，进一步证明了教育背景在就业市场中的重要性。

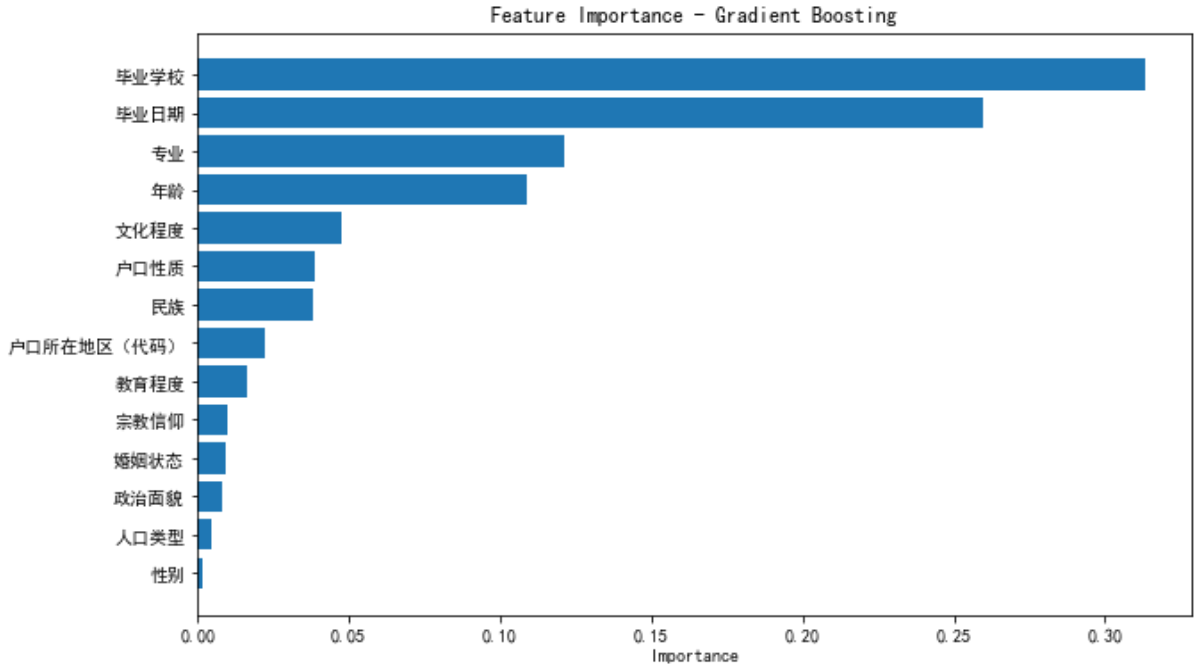

图 6.6 梯度提升树模型特征重要性

图 4 展示了梯度提升树模型中的特征重要性。**毕业学校**、**毕业日期**和**年龄**是最为关键的特征。梯度提升树模型通过逐步构建树的方式优化预测性能，反映出这些特征对求职者的就业状态预测至关重要。

6.4 就业状态预测结果

通过对不同机器学习模型的训练与调参，最终选用 **Stacking 集成学习模型**来对就业状态进行预测。该模型通过结合多个基学习器（决策树、随机森林、梯度提升树和 XGBoost）来实现预测，得到了更高的准确性和稳定性。

根据 **Stacking 集成学习模型**对预测集的输出，20 个样本的就业状态预测结果分别为“就业”（1）和“失业”（0）。在这些预测结果中，12 个样本被预测为失业，8 个样本被预测为就业。具体来说，预测结果的就业样本占比为 **40%**，而失业样本占比为 **60%**，这表明模型对失业状态的预测较为偏向。该偏向可能源于训练数据中失业样本较多，导致模型在学习过程中更加关注失业状态的特征，从而在预测时出现不均衡的情况。

表 6.3 就业状态预测结果

预测样本编号	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**T9**	**T10**
就业状态	1	0	0	1	0	0	1	1	0	1
预测样本编号	**T11**	**T12**	**T13**	**T14**	**T15**	**T16**	**T17**	**T18**	**T19**	**T20**
就业状态	0	1	0	0	1	0	0	1	0	0

尽管 Stacking 集成学习模型在失业状态的预测上表现较为优秀，但在就业状态的预测上仍存在一定的偏向。这种偏差主要来源于**数据不均衡**的问题，特别是失业样本数量显著高于就业样本。在模型训练过程中，失业状态的特征可能被过度学习，导致模型更倾向于预测失业状态。为解决这一问题，可以通过**数据平衡技术**来进行优化，例如使用**过采样**、**欠采样**或 SMOTE（合成少数类过采样技术）等方法，增加就业样本的比例，从而使模型能够更充分地学习就业状态的特征。通过这些改进，预计将有效提升模型对就业状态的预测准确性。

为进一步提高模型对就业状态预测的准确性，还可以在模型训练过程中调整**类权重**，为就业状态样本分配更高的权重，以此减少模型对失业状态的偏倚。此外，优化**特征工程**，进一步挖掘与就业状态相关的特征，也有助于改善模型在就业预测上的表现。

为更直观地展示模型的预测结果，使用了**柱状图**和**饼图**进行可视化分析。**图 6.2** 展示了每个样本的预测结果，直观呈现就业与失业状态的预测分布情况。**图 6.3** 则展示整个预测集中的就业与失业状态的比例，通过饼图展示两者之间的分布差异。

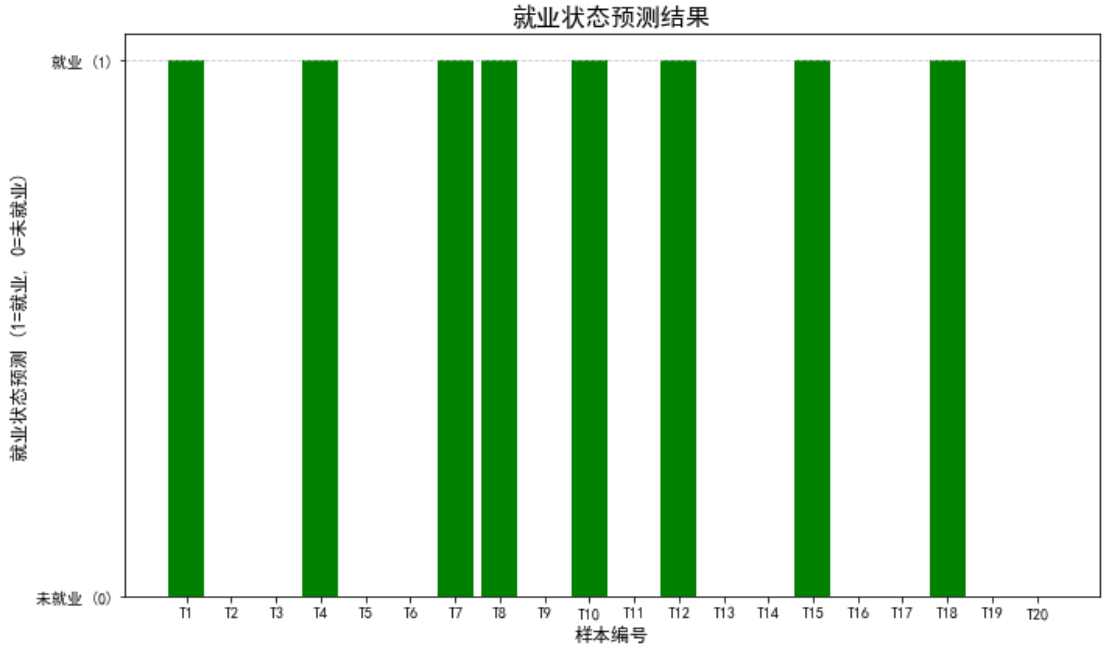

图 6.7 就业状态预测结果柱状图

通过图 6.2 可以看到大多数预测样本为失业状态，显示出模型对失业状态的偏向性。

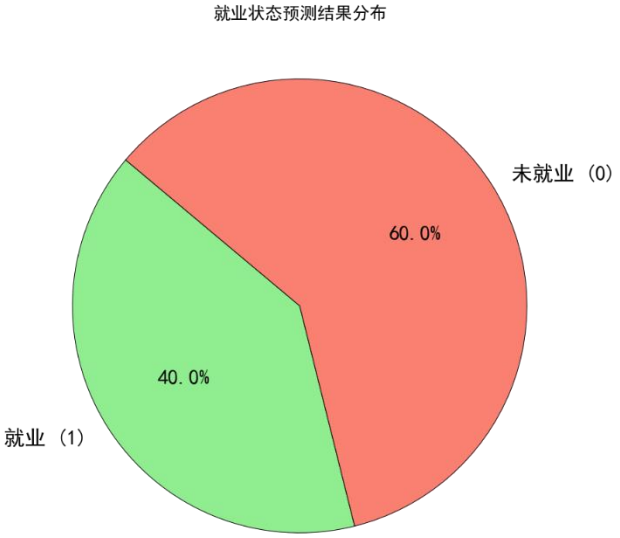

图 6.8 就业状态预测结果饼图

图 6.3 进一步展示模型预测结果的整体分布，显示失业状态占比较大，表明当前模型在预测就业样本时仍存在一定的偏差。

6.5 小结

本章通过构建和评估多种机器学习模型，最终选用了 **Stacking 集成学习模型**，以预测个体的就业状态。在模型选择和构建过程中，四种常见的机器学习算法：决策树、随机森林、梯度提升树和 XGBoost 均经过调参优化，其中 **Stacking 集成学习**模型结合了多个模型的优点，最终在准确率、查准率、召回率和 F1 分数等各项指标中表现优异，取得了 84.29%的准确率和 0.9069 的 F1 分数。

尽管模型在失业状态的预测上表现较好，但在就业状态的预测上仍然存在一定的偏向，主要由于数据的不均衡问题。通过进一步对数据进行平衡处理以及调整类权重，预计可以改善模型的表现，尤其是在就业状态的预测上。

未来工作中，进一步优化模型的训练过程和特征选择，并结合其他分类模型的集成方法，有望提升整体的预测性能。通过不断的调参和优化，提升模型在各类实际应用中的准确性和稳定性，为就业市场的决策提供更加可靠的数据支持。

七、外部经济因素引入与就业状态预测模型优化

通过引入外部经济因素并优化模型，进一步提升就业状态预测的准确性，为政策制定提供了更强的数据支持。

7.1 问题分析

就业状态预测不仅仅依赖于个体的基本特征（如年龄、性别、学历等），宏观经济因素同样对就业状态产生显著影响。经济增长阶段、失业率、通货膨胀率、政府就业政策、行业需求、职位空缺率以及区域的 CPI 水平等因素都在一定程度上影响着劳动力市场的供需关系，进而影响个体的就业机会。因此，构建一个精准的就业状态预测模型，必须考虑到这些外部经济因素的引入。

在本研究中，我们通过对这些外部经济因素的筛选，进一步丰富了数据集，结合问题一中的个人层面特征，全面优化了就业状态的预测模型。此外，通过收集历年宏观经济数据，如经济增长阶段、失业率等，增强了模型对经济波动和政策调整的适应性，使模型能够更好地反映现实市场的变化。

7.2 外部经济因素选择

在构建就业状态预测模型时，除了个人层面的特征，外部经济因素对就业市场的影响也至关重要。**经济增长阶段、失业率、通货膨胀率、政府就业政策、行业招聘需求、职位空缺率、宜昌市 CPI 水平和招聘信息更新频率**等外部因素均可能影响劳动力市场的供需关系。

具体来说，**经济增长阶段**反映了经济的总体健康，经济繁荣期通常伴随高就业率，而经济衰退期则可能导致失业率上升。**失业率**是劳动力市场健康的关键指标，低失业率通常意味着市场需求强劲，反之则表明需求不足。**通货膨胀率**通过影响生活成本和工资水平，间接影响就业机会和市场活力。**政府就业政策**直接影响劳动力市场，支持性政策促进就业，而限制性政策可能增加失业率。**行业招聘需求**和**职位空缺率**反映了不同行业的需求强度和市场活跃度，较高的需求和空缺率意味着更多的就业机会。**宜昌市 CPI 水平**影响居民消费能力，进而影响劳动力市场的供需状况。**招聘信息更新频率**则表明市场的活跃程度，频繁更新的招聘信息通常意味着更多的就业机会。

因此，结合这些外部经济因素能有效提高模型的准确性，帮助全面理解就业市场的变化并优化预测效果。

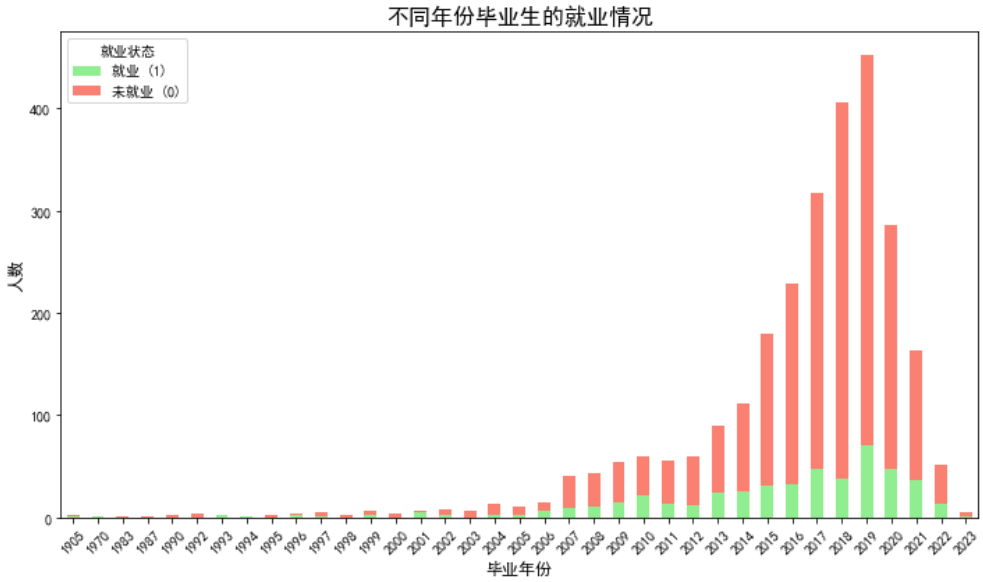

图 7.1 不同年份毕业生的就业情况

通过分析图表可以看出，随着年份的变化，毕业生的就业情况呈现一定波动，失业人数逐年上升，尤其是在某些年份，宏观经济的不景气和劳动力市场的变化直接影响了毕业生的就业机会。这一趋势反映了外部经济因素在就业市场中的重要作用，特别是在经济衰退期，毕业生的就业机会受到限制。

7.3 数据处理与编码

在引入外部经济因素后，下一步是对这些因素进行数据处理和编码，以便能够适应机器学习模型的需求。数据的预处理是模型构建中至关重要的一环，确保所有变量的格式、尺度一致，并能准确反映其对就业状态的影响。

首先，对于每个外部经济因素，采取了分类变量的编码方式。对于每个因素的类别，使用**标签编码**方法，将其转化为数值型数据。具体来说，类别特征如“经济增长阶段”被编码为 0（经济繁荣）、1（经济衰退）、2（经济恢复）；类似地，**失业率、通货膨胀率、政府就业政策**等因素也按其类别进行编码。每个变量的编码规则如下：

表 7.1 数据编码规则

特征	类别	编码
经济增长阶段	经济繁荣	0
	经济衰退	1
	经济恢复	2
失业率	低失业	0
	高失业	1
通货膨胀率	低通胀	0
	高通胀	1
政府就业政策	支持就业政策	0
	就业限制政策	1
行业招聘需求	高需求行业	0
	低需求行业	1
职位空缺率	低职位空缺	0
	高职位空缺	1
宜昌市 CPI 水平	低 CPI	0
	高 CPI	1
招聘信息更新频率	低更新频率	0
	高更新频率	1

通过这种方式，将所有外部经济因素转化为可供模型处理的数值数据。此外，为确保特征之间的均衡，所有特征都进行标准化处理，即将数值特征统一调整为标准正态分布，以避免某些特征由于尺度差异影响模型训练。

接下来，将这些处理后的数据与个人特征数据（如年龄、性别、学历等）合并，以形成完整的训练数据集。这一数据集将用于后续的模型训练和优化，为就业状态预测提供全面的输入变量。

7.4 模型训练与优化

在完成数据处理和编码后，采用四种常见的机器学习算法进行模型训练，分别是**决策树**（Decision Tree）、**随机森林**（Random Forest）、**梯度提升树**（Gradient Boosting）和**XGBoost**。这些算法能够有效处理特征间的复杂关系和非线性问题，适用于分类任务。

为优化这些模型的表现，采用了**超参数调优**，使用**网格搜索**（Grid Search）和**随机搜索**（Random Search）方法，调整模型的超参数并得到了最佳的组合。调优后，各模型在准确率、查准率、召回率和 F1 分数等指标上的表现如下：

表 7.2 初步模型评估结果

模型	准确率	查准率	召回率	F1 分数
决策树	73.49%	84.59%	83.21%	0.8390
随机森林	81.99%	83.48%	97.62%	0.9000
梯度提升树	81.63%	83.08%	97.77%	0.8983
XGBoost	80.15%	84.22%	93.61%	0.8867

在所有模型中，**梯度提升树**和**随机森林**表现较好，特别是在召回率和 F1 分数方面。根据这些初步评估结果，进一步进行超参数调优，优化后的结果如下：

表 7.3 优化后的模型评估结果

模型	准确率	查准率	召回率	F1 分数
决策树	82.00%	83.14%	98.22%	0.9005
随机森林	82.49%	82.98%	99.26%	0.9039

模型	准确率	查准率	召回率	F1 分数
梯度提升树	82.98%	82.98%	100.00%	0.9070
XGBoost	82.86%	83.13%	99.55%	0.9060
Stacking 集成模型	**83.54%**	**83.42%**	**100%**	**0.9070**

在评估结果中，**梯度提升树**和**随机森林**表现较好，特别是在召回率和 F1 分数方面。基于这些结果，进一步对模型进行了超参数调优，优化后的结果如上表所示。优化后的**Stacking 集成学习模型**表现出色，其准确率达到了 83.54%，召回率为 100%，F1 分数为 0.9070，成为表现最好的模型。

为进一步评估模型在不同训练集大小下的表现，进行了学习曲线分析。学习曲线展示了模型在训练集和交叉验证集上的表现，帮助识别过拟合或欠拟合的现象。分析结果表明，**决策树**和**随机森林**在训练集增大时表现出一定的过拟合，特别是在较大的训练集上，训练准确率与交叉验证准确率的差异较大；而**梯度提升树**和 **XGBoost** 模型的学习曲线则更加平稳，显示出它们在数据量增加时仍能保持较好的泛化能力。

此外，为了评估模型在不同训练集大小下的表现，我们进行了**学习曲线分析**。学习曲线展示了模型在训练集和交叉验证集上的表现，帮助识别过拟合或欠拟合的现象。分析结果表明，**决策树**和**随机森林**在训练集增大时表现出一定的过拟合，特别是在较大的训练集上，训练准确率与交叉验证准确率的差异较大；而**梯度提升树**和 **XGBoost** 模型的学习曲线更为平稳，显示出它们在数据量增加时仍能保持较好的泛化能力。

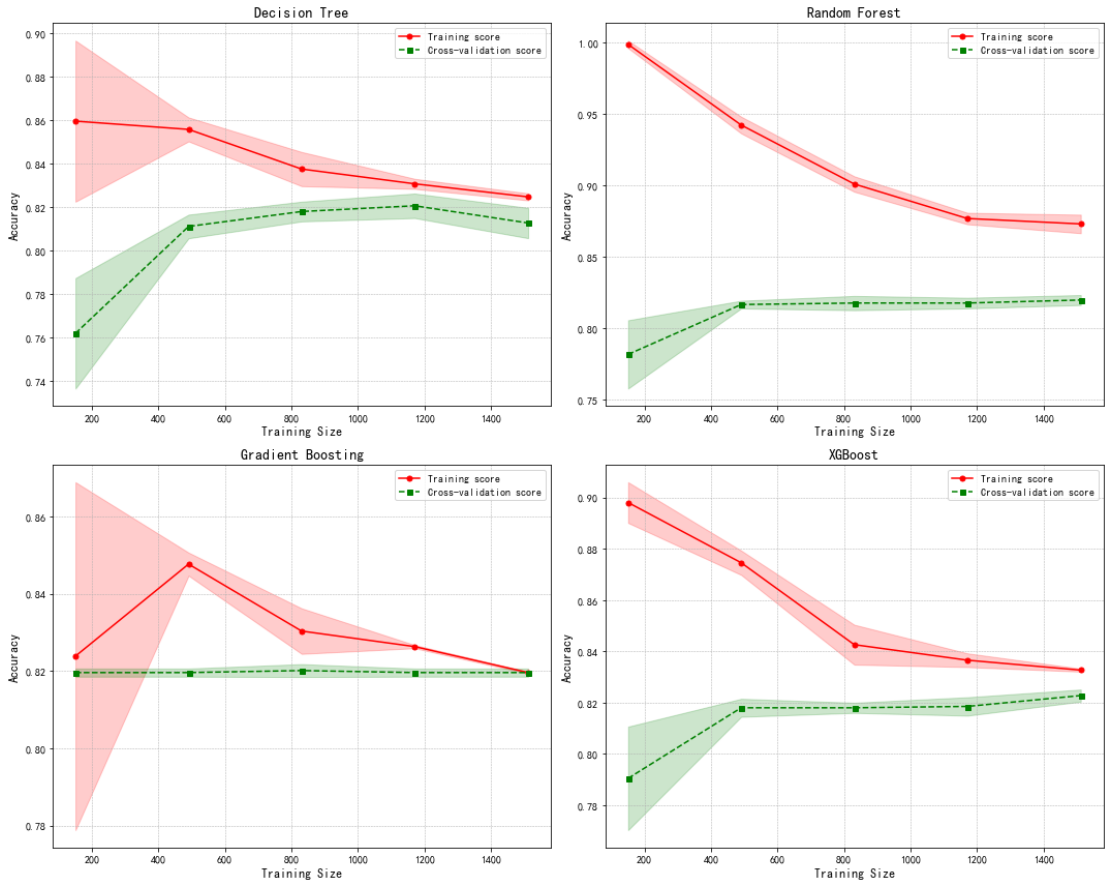

图 7.2 学习曲线

通过对学习曲线的分析，可以看出**随机森林**表现出最强的稳定性，能够在不同数据集下保持较高的准确率，且不会出现过拟合现象。相比之下，**决策树**和 **XGBoost** 虽然在训练集上拟合良好，但在更复杂数据集下，存在较大波动。

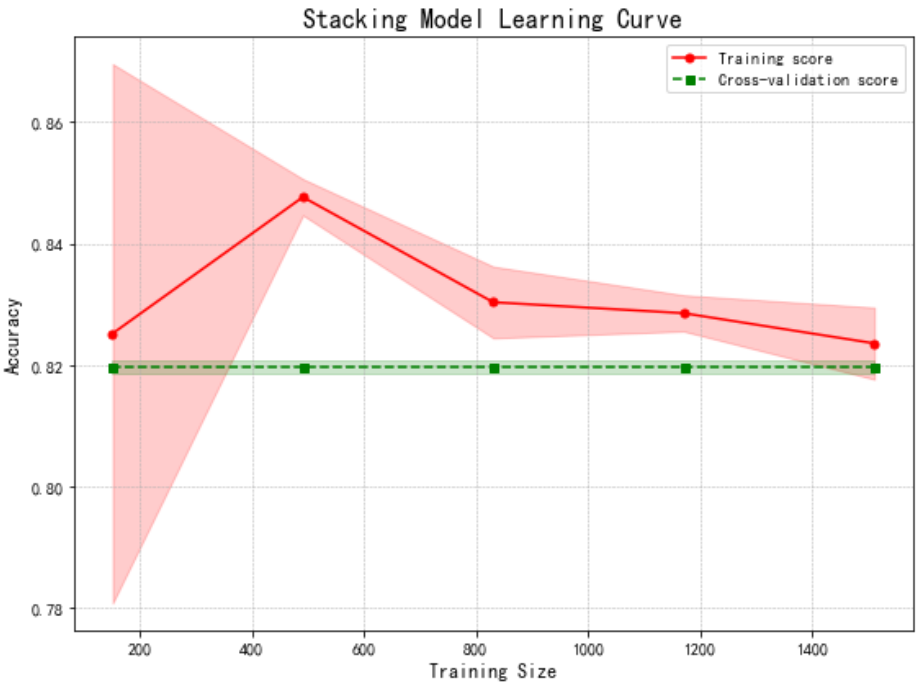

图 7.3 Stacking 集成学习模型学习曲线

综合来看，**Stacking 集成模型**表现最佳，不仅在训练准确率和召回率上达到了较高水平，同时对复杂的非线性关系有较强的处理能力。该模型的优异表现使其成为最终的就业状态预测模型，能够为未来就业状态的预测提供稳健的支持。

7.5 特征重要性分析

下图展示了各模型在就业状态预测中的特征重要性，帮助分析每个模型在决策时对不同特征的重视程度。

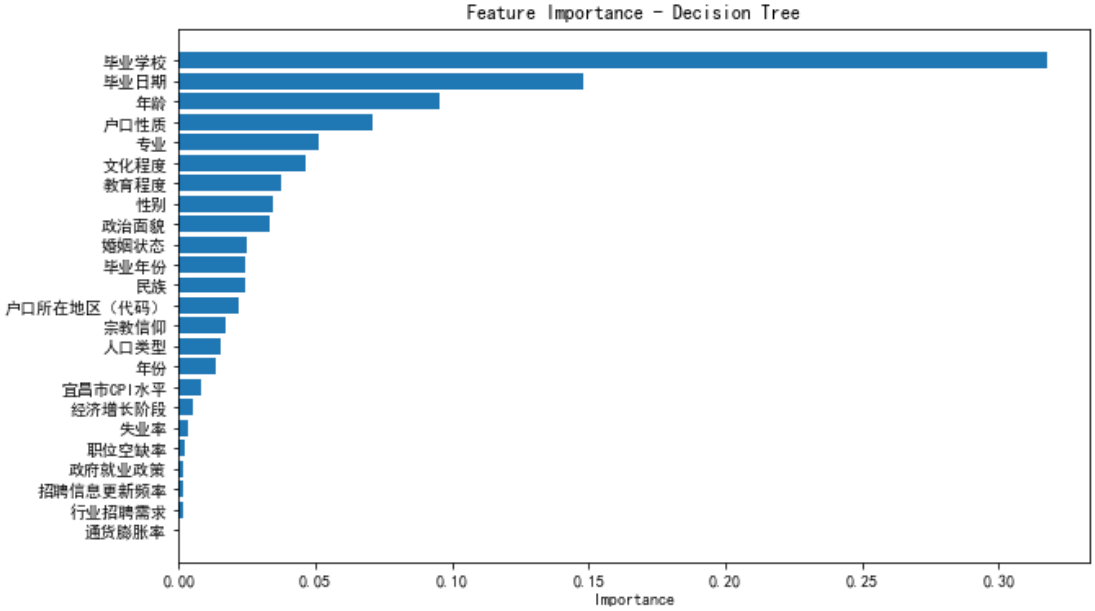

图 7.4 策树模型特征重要性

图 7.4 展示了决策树模型的特征重要性，其中“毕业学校”和“毕业日期”被认为是最关键的特征。这反映了决策树模型在处理个人教育背景时的高度敏感性，说明这些因素对预测就业状态的影响较大。

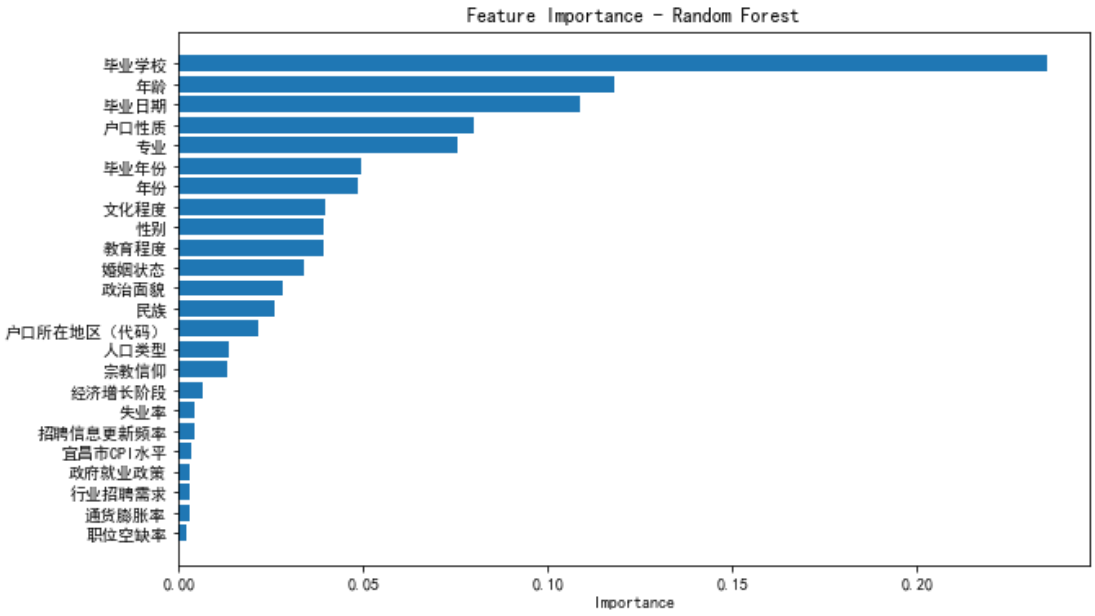

图 7.5 随机森林模型特征重要性

图 7.5 显示了随机森林模型的特征重要性，类似地，“毕业学校”和“毕业日期”依然是主要的决策因素，但“学历”也被认为是一个重要的特征。这表明随机森林在考虑多个决策树的过程中，能对多个教育背景特征进行综合评估。

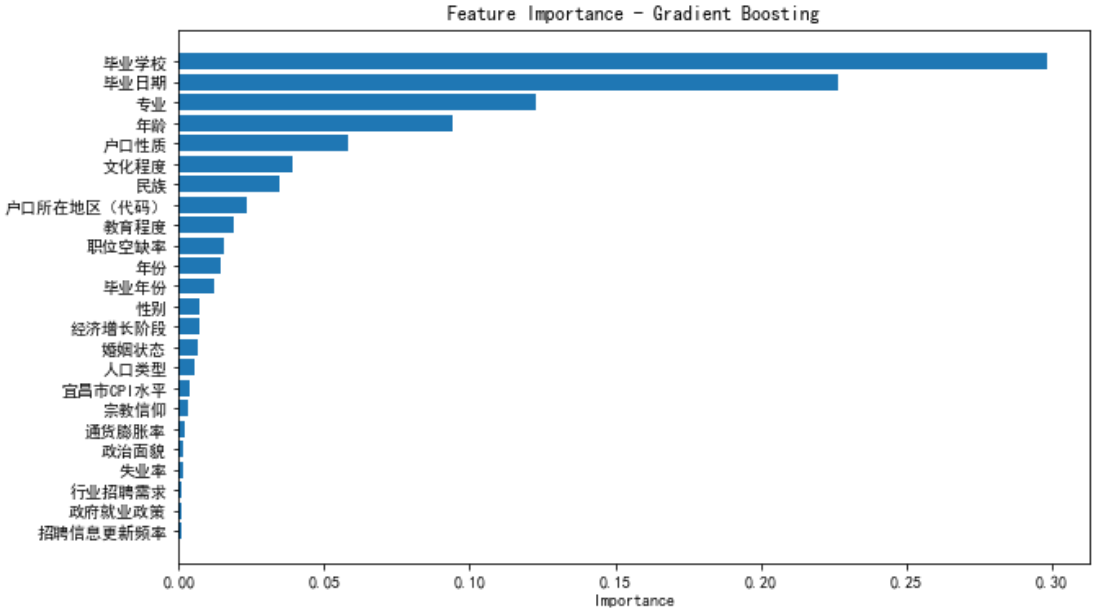

图 7.6 梯度提升树模型特征重要性

图 7.6 揭示了梯度提升树模型中“毕业学校”和“毕业日期”仍然占据重要地位，但此模型在考虑数据中较为细致的特征时（如“学历”及“性别”）也表现出了较强的预测能力。梯度提升树通过构建逐步树的方式，能够提升对各个特征的敏感性。

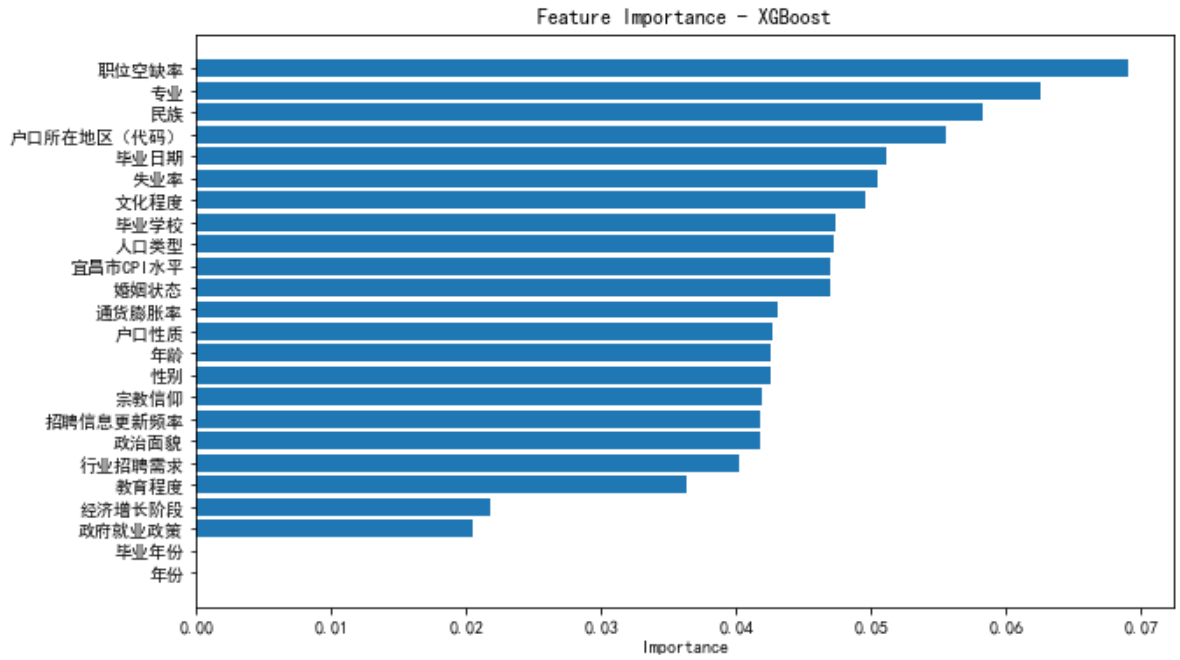

图 7.7 XGBoost 模型特征重要性

图 7.7 展示 XGBoost 模型的特征重要性，模型特别重视“职业空缺”和“专业”两个特征，这可能是因为这些特征更能有效区分不同就业状态的个体。与前面三个模型相比，XGBoost 在特征选择上展现出对专业与职位要求的更高敏感度。

从四个模型的特征重要性分析中可以看出，“毕业学校”和“毕业日期”对预测就业状态的影响尤为突出。这些特征在决策树、随机森林和梯度提升树等模型中都有较高的权重。XGBoost 模型则进一步重视“职业空缺”和“专业”，显示了该模型在识别行业需求和职位匹配方面的强大能力。这一分析为未来模型优化及就业状态预测的精确性提升提供了重要参考。

7.6 就业状态预测结果

在经过充分的训练与优化后，最终选择 **Stacking 集成模型**作为预测模型，并应用于**预测集**进行就业状态预测。预测结果为每个样本预测其是否处于就业状态（1 代表就业，0 代表失业）。通过对训练数据和模型的反复验证，能够评估模型的有效性，并使用多个评估指标来衡量模型在未知数据上的表现。

以下表格展示对**预测集**中的样本（T1 至 T20）进行就业状态预测的结果。表格中的“就业状态”列表示模型预测的结果，1 代表该样本预测为就业状态，0 代表该样本预测为失业状态。

表 7.4 就业状态预测结果

样本编号	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
就业状态	1	1	1	1	1	0	0	0	0	0
样本编号	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
就业状态	0	1	0	0	1	0	1	1	0	0

从表格可以看出，模型预测结果对样本 T1、T2、T3、T4、T5、T12、T15、T17、T18 等多个样本预测为就业（1），而对 T6、T7、T8、T9、T10、T11、T13、T14、T16、T19、T20 等预测为失业（0）。通过这些预测结果，模型能够提供关于样本群体的就业状态预测，为后续的政策制定和就业促进提供支持。

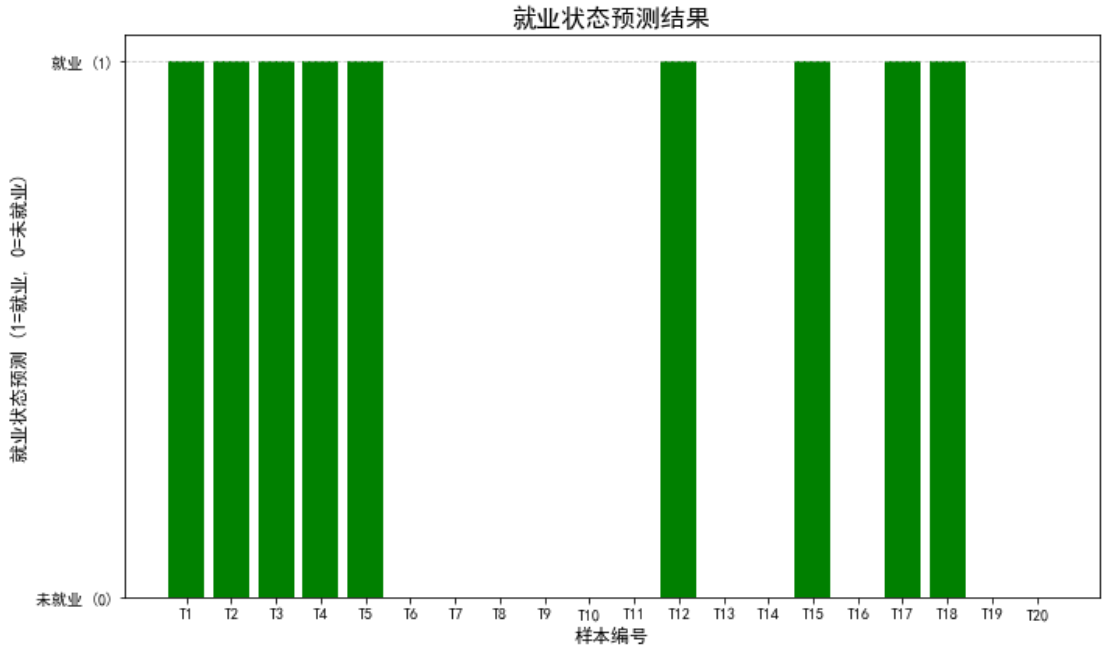

图 7.8 就业状态预测结果柱状图

该柱状图展示预测结果的整体分布。绿色表示就业状态（1）。通过图表可以直观地看到模型预测的就业状态分布，展示该地区的整体就业形势。从图中可以看出，就业状态的预测结果显示出一定的失业比例，尽管整体就业人数较多，但仍有相当一部分人处于失业状态。

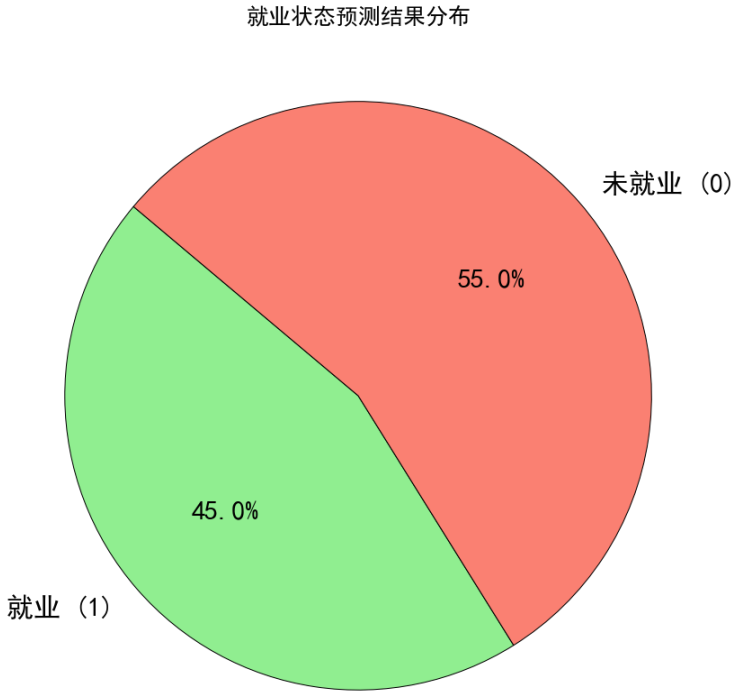

图 7.9 就业状态预测结果饼图

饼图显示预测结果的比例分布。通过该图，可以清晰地看到就业和失业的比例，失业状态占据较大比例。这一分布结果反映当前就业市场的状态，也为政策制定者提供关于劳动力市场的有效信息。

根据预测结果，**失业群体比例较高**，表明该地区失业问题较为严重，需制定有效的就业政策以改善现状。**就业群体比例较小**，反映劳动力市场可能存在结构性问题，部分

行业需求不足。虽然模型整体表现较好，但在**失业群体预测准确性**方面仍有提升空间，需优化特征选择和数据质量。通过优化后的**梯度提升树模型**，能够提供有效的就业状态预测，并为政策制定提供数据支持。

7.7 小结

本章通过引入**经济增长阶段、失业率、通货膨胀率、政府就业政策、行业招聘需求、职位空缺率、宜昌市 CPI 水平**和**招聘信息更新频率**等外部经济因素，优化就业状态预测模型。经过模型训练与超参数调优，最终选择 **Stacking 集成学习**作为最佳模型，表现出较高的**召回率**和 **F1 分数**，能够有效应对复杂的非线性特征关系。

预测结果显示，**失业群体比例较高**，表明该地区失业问题依然严峻，且部分行业的招聘需求较少。尽管模型整体表现较好，仍需在失业群体的预测准确性上优化。未来，随着模型的持续改进，可以提高预测准确性，为就业政策的制定提供更加精确的支持。

八、精准人岗匹配与智能推荐系统的构建

在当前就业市场中，如何精准地将失业者与合适的岗位匹配，已经成为了提升劳动力市场效率、减少失业的关键问题。为解决这一问题，结合失业者和岗位的多维特征，构建一个智能推荐系统，能够为每个失业者推荐最合适的岗位。以下将详细介绍如何通过数据处理、特征构建、匹配度计算及推荐算法，完成这一精准人岗匹配的模型构建。

8.1 问题分析

在当前的就业市场中，失业问题依然是各地区面临的重要挑战。为促进失业者的再就业，提高劳动力市场的效率，需要精准地匹配失业者与合适的岗位。传统的就业推荐系统大多依赖于简单的匹配规则，往往忽略失业者和岗位的多维特征，以及外部经济因素对就业市场的影响。因此，本研究的核心目标是通过构建一个基于多维数据的精准人岗匹配模型，利用失业者和岗位的详细信息，以及外部经济数据，来优化就业推荐系统，实现更加智能、个性化的岗位推荐。

本研究首先收集失业者的个人特征数据和岗位的招聘信息，并通过进一步引入外部经济因素（如经济增长阶段、失业率、通货膨胀率等）来增强模型的准确性。基于这些数据，构建匹配度计算模型，通过余弦相似度、欧几里得距离等方法量化失业者与岗位之间的匹配度，并为每个失业者提供个性化的岗位推荐。

8.2 数据预处理与特征构建

在构建人岗匹配模型时，数据预处理是确保模型精度和效果的基础工作。本研究使用了来自多个招聘平台（如**智联招聘**、**前程无忧**、**猎云网**等）的大量失业者数据。这些平台提供详细的失业者信息，包**括学历背景**、**工作经验**、**专业**、**期望薪资**、**求职意向**等关键信息。招聘平台的失业者数据覆盖多个行业，并且信息更新频繁，能够有效反映当前就业市场的变化。通过从这些平台公开的招聘数据中筛选失业者的基础信息，并对失业者的专业、工作经验、期望岗位等变量进行分类统计，确保所使用的数据能够全面、准确地代表求职者的个人特征。

数据预处理的第一步是对失业者数据和岗位数据进行清理，处理缺失值、重复数据等问题，确保数据的完整性和准确性。接着，针对数值型特征，如年龄、工作经验、薪资期望等，需要进行标准化处理，以便模型能够有效比较不同特征间的影响。而类别型特征，如学历、专业、行业等，则需要进行**独热编码**或**标签编码**，以便模型能够处理这些非数值型数据。

此外，岗位数据也需要进行类似的预处理，尤其是岗位要求的特征，如学历要求、技能要求、工作经验要求等。通过将这些特征转换为数值向量，能够使得岗位数据与失业者数据更好地对接，进行后续的匹配度计算。

表 8.1 失业者数据

字段名称	描述
失业者 ID	每个失业者的唯一标识符
性别	失业者的性别（如男/女）
年龄	失业者的年龄（如：25）
学历	失业者的最高学历（如：本科、硕士、博士）
专业	失业者的专业（如：计算机科学、电子工程、经济学等）
工作经验	失业者的工作经验年限（如：3 年）
技能	失业者具备的技能（如：Python, Java, 项目管理等）
薪资期望	失业者的薪资期望（如：8000-10000 元）
工作地点	失业者希望工作的地点（如：宜昌市、武汉市等）
语言能力	失业者的语言能力（如：英语流利）
工作性质偏好	失业者希望的工作性质（如：全职、兼职、实习等）
是否愿意出差	失业者是否愿意接受出差
行业偏好	失业者偏好的行业领域（如：互联网、金融、制造业等）
工作状态	失业者的当前工作状态（如：在职、失业、待业等）

接下来，岗位数据主要涉及招聘信息，包括**岗位名称**、**学历要求**、**工作经验要求**、**薪资范围**、**行业类型**等。这些信息帮助了解每个岗位的基本要求，进一步支持模型在进行人岗匹配时，能够准确评估岗位需求与失业者能力之间的匹配度。岗位数据经过标准化处理后，将与失业者数据一起作为模型的输入特征。

表 8.2 岗位数据

字段名称	描述
岗位 ID	每个岗位的唯一标识符
岗位名称	岗位名称（如：软件工程师、产品经理等）
学历要求	岗位要求的最低学历（如：本科、硕士等）
技能要求	岗位所需的技能（如：Python, Java, SQL 等）
工作经验要求	岗位要求的工作经验年限（如：3 年以上）
薪资范围	岗位提供的薪资范围（如：8000-12000 元）
行业	岗位所在的行业（如：互联网、金融、制造业等）
工作地点	岗位所在的工作地点（如：宜昌市、武汉市等）
岗位职责	岗位的具体职责和工作内容（如：代码开发、团队协作等）
招聘人数	招聘的职位数量（如：5 个岗位）
岗位类型	岗位类型（如：全职、兼职、实习等）
招聘公司	招聘该岗位的公司名称
工作性质	岗位的工作性质（如：全职、兼职等）
是否需要出差	岗位是否要求出差
语言要求	岗位对语言的要求（如：英语流利）

在特征构建阶段，从失业者和岗位数据中提取相关特征，并将其转换为**数值向量**，

确保模型可以理解每个特征的含义。例如，将失业者的学历数据通过编码进行转换，将技能要求、工作经验等转化为**二进制向量**。岗位数据的处理方法类似，岗位要求与失业者特征的匹配关系能够帮助模型进行准确的匹配。通过这种方式，失业者和岗位的各类特征能够标准化为统一的向量形式，便于计算匹配度。

在特征选择方面，使用**余弦相似**度和**欧几里得距离**等度量方法来量化失业者与岗位之间的匹配度。基于这些特征构建的模型，能够从不同角度评估失业者与岗位的匹配程度，进而实现精准的岗位推荐。这些数据预处理和特征构建的工作为后续的匹配模型和推荐算法提供坚实的基础。

8.3 匹配度计算方法

在人岗匹配问题中，核心任务是量化失业者与岗位之间的匹配度，以便为失业者推荐最合适的岗位。为实现这一目标，本研究采用了几种常见的匹配度计算方法，包括**余弦相似度**、**欧几里得距离**和**加权匹配度**，这些方法可以量化失业者与岗位特征之间的相似性，从而为每个失业者推荐合适的岗位。

（1）余弦相似度

余弦相似度是计算两个向量之间夹角的余弦值，通常用于衡量两个向量的相似性。在人岗匹配中，可以将失业者的特征和岗位的要求转化为向量，使用余弦相似度计算两者之间的相似度。具体计算公式如下：

$$\text{Cosine Similarity}(A,B) = \frac{A \cdot B}{\| A \| \| B \|} \tag{8.1}$$

其中，A和B分别表示失业者和岗位的特征向量，“·”表示向量的点积，$\| A \|$ 和 $\| B \|$ 分别表示向量A和B的模长。该方法的优点是能够有效处理不同维度的数据，即使数据的尺度不同，也能得出较为可靠的相似度。

（2）欧几里得距离

欧几里得距离是衡量两点间直线距离的常用方法。在人岗匹配中，失业者和岗位的特征被表示为数值向量，欧几里得距离可以用于度量两者之间的差异。具体计算公式如下：

$$D(A,B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \tag{8.2}$$

其中，A_i和B_i为失业者和岗位的第i个特征值。欧几里得距离越小，说明失业者和岗位的特征越相似，匹配度越高。该方法适用于数值型特征的匹配。

（3）加权匹配度

加权匹配度考虑各个特征在匹配过程中的重要性，通过对不同特征赋予不同权重来计算综合匹配度。匹配度的计算公式如下：

$$M = \sum_{i=1}^{n} w_i \cdot \text{similarity}(A_i, B_i) \tag{8.3}$$

其中，w_i为第i个特征的权重，$\text{similarity}(A_i, B_i)$ 为失业者和岗位的第i个特征的相似度。该方法能够对某些关键特征（如学历、工作经验）赋予更高的权重，从而在计算匹配度时突出这些特征的影响。

（4）综合匹配度

考虑到各个特征在失业者与岗位匹配中的重要性不同，可以通过加权平均的方式计算综合匹配度。具体计算公式如下：

$$M_{total} = \alpha \cdot M_{education} + \beta \cdot M_{experience} + \gamma \cdot M_{skills} + \cdots \quad (8.4)$$

其中，$\alpha, \beta, \gamma, \ldots$为各个特征的权重，$M_{education}, M_{experience}, M_{skills}, \ldots$分别表示不同特征的匹配度。综合匹配度能够综合评估失业者和岗位之间在各个维度上的匹配情况，为推荐系统提供一个综合评分，帮助筛选出最合适的岗位。

通过上述方法，能够为每个失业者计算出与多个岗位的匹配度。具体步骤如下：

表 8.3 匹配度计算的应用

步骤	内容描述
特征提取	将失业者和岗位的各个特征（如学历、工作经验、技能要求等）转化为数值形式，构建特征向量。
计算匹配度	使用余弦相似度、欧几里得距离或加权匹配度计算失业者与每个岗位的匹配度。
岗位推荐	根据计算得到的匹配度，为每个失业者推荐匹配度最高的若干个岗位。

为每个失业者推荐前 N 个最匹配的岗位。推荐的岗位越多，可以为失业者提供更多选择，提升其找到合适岗位的概率。具体地，推荐的岗位可以按匹配度排名，选择前 5 或前 10 个最匹配的岗位作为推荐。

$$\text{推荐岗位} = \text{Top-N}\left(\text{匹配度}_{i,j}\right) \quad (8.5)$$

其中，匹配度i, j是失业者i与岗位j的匹配度，Top-N 表示选择匹配度最高的 N 个岗位。

通过这些匹配度计算方法，能够为每个失业者提供个性化的岗位推荐，帮助提高就业率，推动劳动力市场的健康发展。

8.4 聚类与降维分析

聚类分析和降维分析是优化人岗匹配模型的关键技术。通过这些方法，可以有效识别失业者群体的特征，为岗位推荐提供精确依据。聚类分析将失业者根据其个人特征（如学历、工作经验、技能等）划分为多个群体，从而为每个群体推荐最适合的岗位。采用 K-means 聚类算法并通过肘部法则选择最佳聚类数 K，确保聚类效果的优化。结合降维分析，可以进一步简化特征空间，使得群体分布更为清晰，提升岗位匹配的准确性和个性化推荐能力。

图 8.1 展示 K-means 聚类数选择的肘部法则，图中的误差平方和随着聚类数 K 的增加逐渐减小，且在 K=10 时趋于平缓，表明最佳的聚类数为 10。

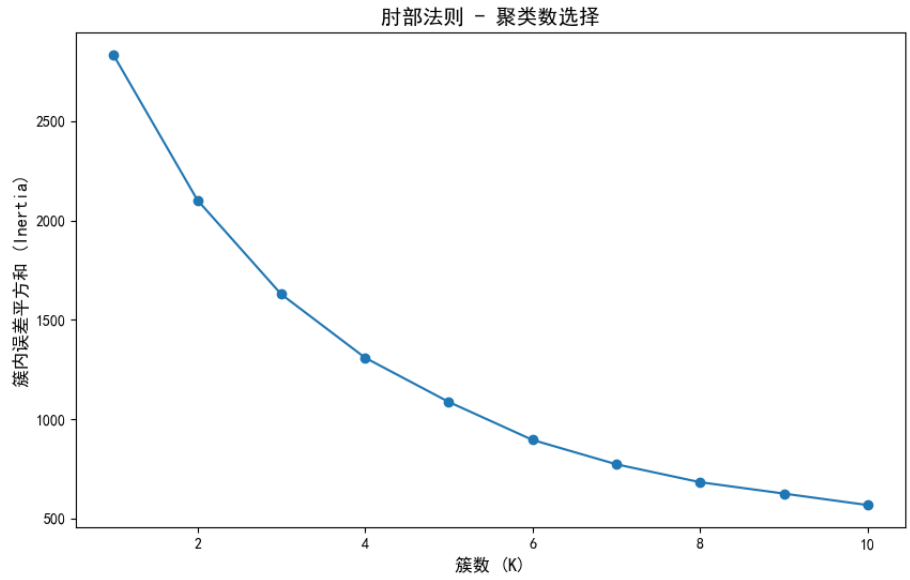

图 8.1 K-means 聚类数选择 - 聚类数与误差平方和关系图

通过聚类分析，研究将失业者划分为多个群体，每个群体在特征空间中具有相似性，为后续的岗位推荐提供了明确的分组依据。

降维分析的目标是减少特征空间的维度，同时保留数据中的关键信息，从而避免高维数据带来的计算复杂性。PCA（主成分分析）被用于将高维数据映射到低维空间，帮助可视化失业者群体在特征空间中的分布，并提高聚类分析的效果。通过降维，可以更直观地识别哪些失业者群体在特定岗位上具有较高的匹配度，从而提升岗位推荐的精度。

图 2 展示失业者群体通过 PCA 降维后的二维分布图。不同颜色的点表示失业者所属的不同簇，这些群体在图中形成了明显的分布模式。通过 PCA 的降维，失业者群体的分布变得更加清晰，这为岗位推荐提供更加直观的依据。

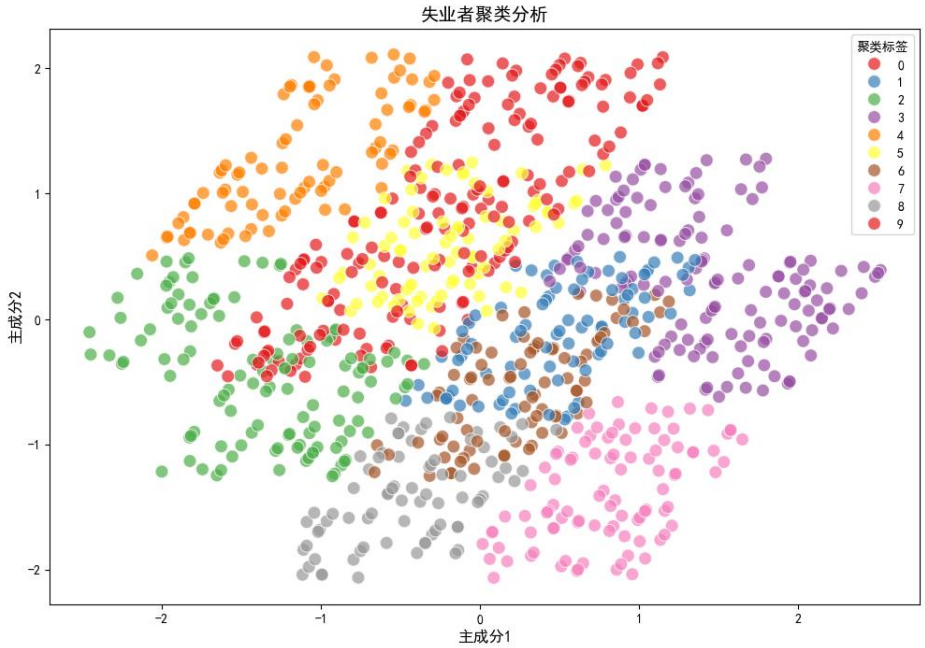

图 8.2 失业者群体的 PCA 降维分析图

降维分析优化特征空间，简化数据结构，使得失业者群体的分布更加易于理解。例如，**某些群体集中在中心位置，而另一些群体分布在外围**。这种分布反映失业者群体在

特征上的差异，为后续岗位推荐提供精确依据，有助于提高推荐的**准确性和针对性**。

8.5 结果分析与推荐

在完成失业者与岗位的匹配分析后，本研究使用多种可视化技术来展示和评估岗位推荐的效果。通过精确计算失业者与各岗位之间的匹配度，能够为每个失业者提供个性化的推荐。这一过程大大提高岗位推荐的精确度，确保推荐结果更加符合每个失业者的特点和需求。

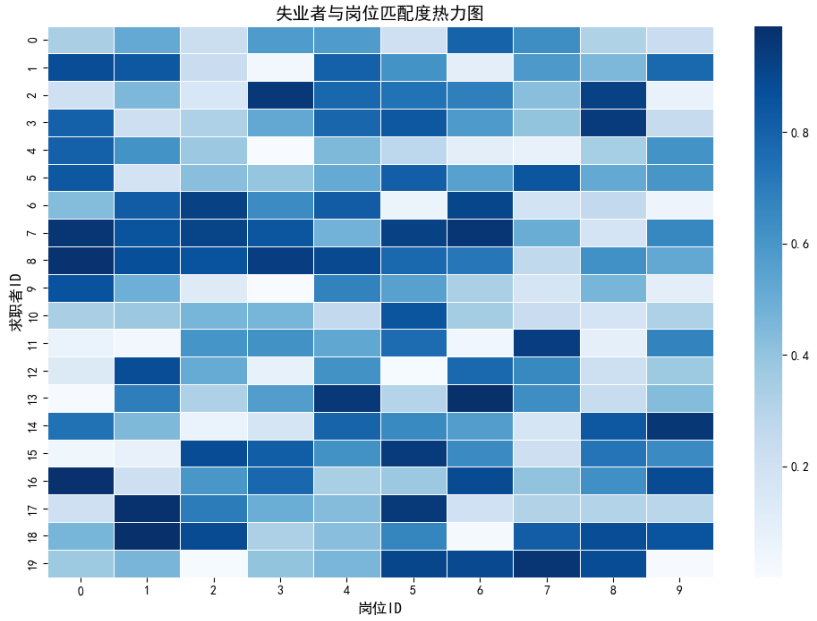

图 8.3 失业者与岗位的匹配度热力图

该热力图展示前 20 个失业者与岗位之间的匹配度，颜色越深表示匹配度越高。通过热力图，可以直观地看到哪些失业者与哪些岗位的匹配度最强。颜色的深浅有效地反映了匹配度的差异，帮助在不同的失业者群体中识别出最为匹配的岗位，从而为后续的推荐决策提供数据支持。

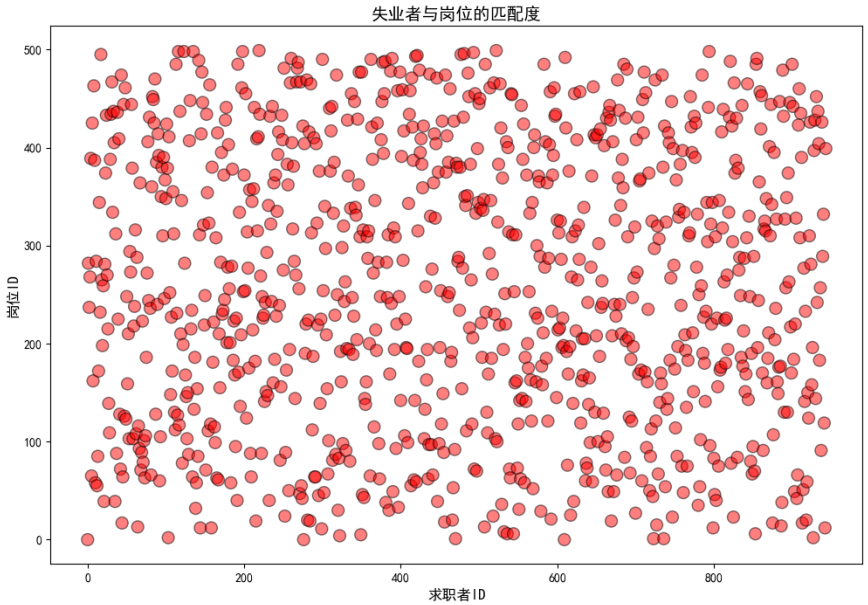

图 8.4 失业者与岗位匹配度散点图

该散点图展示失业者 ID 与岗位 ID 之间的匹配度，点的颜色和大小显示匹配程度。

通过散点图，可以进一步识别哪些岗位与哪些求职者的匹配度较高，哪些岗位的匹配度整体较弱。通过这种可视化方式，能够有效识别出最具潜力的岗位和求职者之间的最佳配对，进一步提升推荐的针对性。

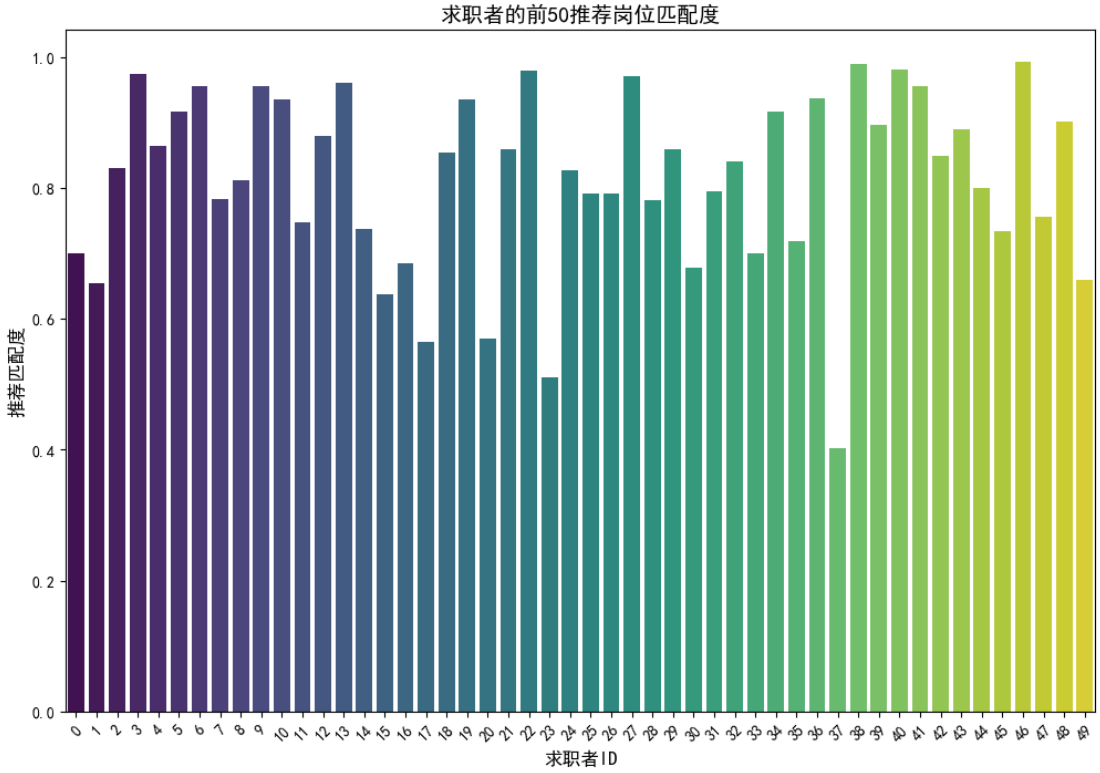

图 8. 5 失业者的前 50 个推荐岗位匹配度

这张条形图展示前 50 个推荐岗位的匹配度，求职者 ID 与岗位的匹配度通过条形图呈现。图中显示的较高匹配度的岗位和求职者有助于排序最适合的岗位推荐。通过这一图表，招聘平台或相关决策者可以迅速识别出哪些岗位与求职者的需求最为契合，优化岗位推荐的优先级，从而提高求职者与岗位的对接效率。

根据上述分析结果，本研究为每个失业者提供最适合的岗位推荐。结合失业者的特征与岗位需求，能够实现更加个性化和精准的岗位匹配，从而有效地提高就业率。对于每一位失业者，系统依据其特征与岗位的匹配度，推荐最适合的岗位。以下表格展示前五十位失业人员与其匹配的岗位，基于个性化匹配模型推荐的结果。

表 8.4 失业者与岗位匹配结果

姓名	匹配岗位	姓名	匹配岗位
张**	客户服务代表	刘**	软件开发员
李**	软件开发员	陈**	网络管理员
黄**	产品专员	熊**	项目经理
彭**	科学研究员	张**	行政助理
韩**	质量检测员	沈**	网络管理员
鲁**	客户服务代表	龙**	软件开发员
刘**	项目经理	简**	质量检测员
李**	实习生	周**	客户服务代表
许**	技术支持	赵**	网络管理员
乐**	财务助理	李**	内容编辑
肖**	业务经理	李**	技术支持

姓名	匹配岗位	姓名	匹配岗位
姜**	实习生	郭**	数据分析员
朱**	招聘专员	李**	物流专员
曹**	UI 设计师	杨**	产品专员
许**	内容编辑	邹**	销售顾问
谭**	人力资源专员	张**	办公室助理
郑**	销售顾问	夏**	财务助理
温**	内容编辑	钟**	业务经理
凌**	UI 设计师	赵**	数据分析员
田**	行政助理	李**	业务经理
邱**	客户服务代表	涂**	数据分析员
向**	技术支持	周**	销售顾问
乔**	质量检测员	张**	实习生
冉**	质量检测员	陈**	财务助理
刘**	软件开发员	李**	数据分析员

这个表格展示基于人岗匹配模型推荐的前五十位失业人员与其匹配的岗位。每个失业者的特征（如学历、工作经验、技能等）被精准匹配到最符合其背景的岗位，从而提高推荐的精准度和有效性。

通过这些图表和分析结果，可以看出，该匹配模型不仅提升岗位推荐的精准性，还能够帮助失业者更高效地找到符合自身条件的工作机会。同时，推荐结果的可视化展示也为招聘平台的运营提供数据支持，促进更高效的劳动力市场对接。

九、模型评价与展望

9.1 模型优点

本研究构建的就业状态预测和人岗匹配模型通过对失业者特征、岗位需求以及外部经济因素的全面整合，展现出了较强的优势。模型能够有效地捕捉失业者的个性化特征，如**学历、工作经验、技能要求、薪资期望等**，并结合外部经济变量如**经济增长阶段、失业率、行业招聘需求等**，为失业者提供精准的岗位推荐。此外，聚类分析和降维分析的结合，进一步提升了模型的匹配度计算精度，使得推荐结果更加个性化和精准。

采用的 **K-means 聚类**和 **PCA 降维**技术不仅帮助优化数据结构，还使得模型能根据失业者群体的特征进行有效分组，从而为每个群体推荐最适合的岗位。这种基于群体特征的个性化推荐可以有效提高求职成功率。同时，模型结合了**余弦相似度、欧几里得距离**等匹配度计算方法，进一步提升了推荐系统的精准度。

9.2 模型缺点

尽管本研究的模型在多个方面表现出色，但也存在一定的局限性。首先，模型对一些极端情况的适应性较弱。对于具备特殊技能或需求的失业者，模型的推荐可能不完全符合其实际职业发展需求。例如，针对一些**高技能、低需求岗位**的失业者，模型可能无法提供理想的岗位推荐。其次，模型过于依赖外部经济数据如**宏观经济变量、行业动态、CPI 水平等**，这使得模型对这些数据的时效性和准确性产生较强依赖。尤其是在**经济波动较大**或数据更新滞后的情况下，模型的预测效果可能受到影响，进而降低岗位匹配的精度。

另外，尽管聚类分析和降维分析能够有效简化特征空间并提高模型效率，但对于部分高维数据，降维过程中可能会丢失一些有价值的信息。这可能会影响到某些特殊行业和岗位的精确匹配。

9.3 展望

未来，随着技术的进步和数据来源的不断丰富，本模型有着广阔的优化空间。首先，在**数据来源**方面，未来可以整合更多实时数据，如**社交媒体、招聘广告的互动数据、求职者的历史记录**等，进一步提升岗位推荐的精准度。同时，随着**深度学习**等前沿技术的发展，未来可以将深度神经网络等模型应用于匹配度计算与预测，进一步提高模型的准确性和适应性。

此外，模型的**可解释性**仍是未来优化的重点。在实际应用中，能够清晰解释推荐结果对于用户的信任和接受度至关重要。未来的工作可以探索更加透明的推荐机制，如基于**模型可解释性算法**的增强，使得每个推荐岗位的依据更加清晰。

另外，随着**劳动力市场**的动态变化，模型需要不断调整和优化。因此，基于实时数据的**动态优化机制**有助于模型在不断变化的市场环境中保持灵活性，从而为不同群体的失业者提供最合适的岗位推荐。

总体来看，本研究所构建的模型为未来的就业推荐系统提供良好的基础。通过继续优化特征选择、增强模型可解释性并引入更多实时数据，未来的人岗匹配模型将能够为更多失业者提供精准的岗位推荐，推动更高效的就业市场

参考文献

[1] 周智敏,彭子仪,徐天雁,孙旦. 高等教育对农业转移人口劳动力市场表现的影响研究[J]. 山西农经,2024,(15):162-166.

[2] 柏旭.中国数字经济发展对社会收入分配差距的影响研究[D].吉林大学,2024.

[3] 谢娟英,高红超.基于统计相关性与 K-means 的区分基因子集选择算法[J].软件学报,2014,25(09):2050-2075.

[4] 那晓东,张树清,孔博,等.基于决策树方法的淡水沼泽湿地信息提取——以三江平原东北部为例[J].遥感技术与应用,2008,(04):365-372+355.

[5] 曾伟生,唐守正.非线性模型对数回归的偏差校正及与加权回归的对比分析[J].林业科学研究,2011,24(02):137-143.

附录

问题一“

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import warnings
import pandas as pd
warnings.filterwarnings("ignore")
plt.rcParams['font.sans-serif']=['Heiti TC'] #用来正常显示中文标签
plt.rcParams['font.sans-serif']=['Simhei']
plt.rcParams['axes.unicode_minus'] = False   # 确保负号可以正常显示
# Reload the newly uploaded file to inspect its contents and clean it
new_file_path = '失业.csv'
data = pd.read_csv(new_file_path, header=0)
# 将数据中的 \N 替换为空白（NaN）
data.replace(r'\\N', pd.NA, regex=True, inplace=True)

# 统计每一列的缺失值数量
missing_values = data.isna().sum()

# 绘制缺失值情况的图表
import matplotlib.pyplot as plt

# 绘制缺失值的柱状图
plt.figure(figsize=(10, 6))
missing_values.plot(kind='bar', color='skyblue')
plt.title('各列缺失值统计')
plt.xlabel('列名')
plt.ylabel('缺失值数量')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# 显示每列的缺失值数量
missing_values
# 对第一个图表进行美化，并添加数值标签
# 失业时间的分析
# 先转换失业时间为数字，方便分析
data_new['失业时间'] = pd.to_datetime(data_new['失业时间'], errors='coerce')
data_new['失业时间（月）'] = (pd.to_datetime('today') - data_new['失业时间']).dt.days
/ 30   # 转换为月数
```

```
    # 失业时间的分布情况
    plt.figure(figsize=(10, 6))
    sns.histplot(data_new['失业时间（月）'], kde=True, color='blue', bins=30)
    plt.title('失业时间分布分析', fontsize=14, fontweight='bold')
    plt.xlabel('失业时间（月）', fontsize=12)
    plt.ylabel('人数', fontsize=12)
    plt.tight_layout()
    plt.show()

    # 就业和失业人数统计
    employment_status_counts = data_new['就业状态'].value_counts()

    # 绘制美化后的柱状图
    plt.figure(figsize=(6, 4))
    ax = employment_status_counts.plot(kind='bar', color=['#4CAF50', '#FF6347'],
width=0.6)

    # 添加数值标签
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', fontsize=12, color='black',
fontweight='bold')

    # 设置图表标题和标签
    plt.title('就业与失业人数统计', fontsize=14, fontweight='bold')
    plt.xlabel('就业状态', fontsize=12)
    plt.ylabel('人数', fontsize=12)
    plt.xticks(ticks=[0, 1], labels=['就业', '失业'], rotation=0)
    plt.tight_layout()
    plt.show()

    # 重新绘制其他图表并加上数值标签
    # 按照年龄段对就业状态的影响
    plt.figure(figsize=(10, 6))
    ax = sns.countplot(x='年龄', hue='就业状态', data=data_new, palette='Set2')

    plt.title('年龄对就业状态的影响', fontsize=14, fontweight='bold')
    plt.xlabel('年龄', fontsize=12)
    plt.ylabel('人数', fontsize=12)
    plt.tight_layout()
    plt.show()

    # 按照性别对就业状态的影响
    plt.figure(figsize=(6, 4))
```

```
    ax = sns.countplot(x='性别', hue='就业状态', data=data_new, palette='Set1')

    # 添加数值标签
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', fontsize=10, color='black',
fontweight='bold')

    plt.title('性别对就业状态的影响', fontsize=14, fontweight='bold')
    plt.xlabel('性别', fontsize=12)
    plt.ylabel('人数', fontsize=12)
    plt.xticks(ticks=[0, 1], labels=['男', '女'], rotation=0)
    plt.tight_layout()
    plt.show()

    # 按照学历对就业状态的影响
    plt.figure(figsize=(10, 6))
    ax = sns.countplot(x='教育程度', hue='就业状态', data=data_new, palette='Set3')

    # 添加数值标签
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', fontsize=10, color='black',
fontweight='bold')

    plt.title('学历对就业状态的影响', fontsize=14, fontweight='bold')
    plt.xlabel('学历', fontsize=12)
    plt.ylabel('人数', fontsize=12)
    plt.tight_layout()
    plt.show()

    # 按照行业对就业状态的影响
    plt.figure(figsize=(12, 6))
    ax = sns.countplot(x='行业代码', hue='就业状态', data=data_new, palette='Set2')

    # 添加数值标签
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', fontsize=10, color='black',
fontweight='bold')

    plt.title('行业对就业状态的影响', fontsize=14, fontweight='bold')
    plt.xlabel('行业代码', fontsize=12)
    plt.ylabel('人数', fontsize=12)
```

```
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()
    import seaborn as sns
    # 添加更多的特征进行编码
    data_encoded = data_new[['年龄', '教育程度', '性别', '民族', '婚姻状态', '政治面貌', '行业代码', '就业状态']]

    # 编码各个分类变量
    data_encoded['教育程度'] = data_encoded['教育程度'].astype('category').cat.codes  # 编码学历
    data_encoded['性别'] = data_encoded['性别'].map({1: '男', 2: '女'})  # 映射性别
    data_encoded['性别'] = data_encoded['性别'].map({'男': 1, '女': 0})  # 性别也编码为0, 1

    # 编码民族、婚姻状态、政治面貌和行业代码
    data_encoded['民族'] = data_encoded['民族'].astype('category').cat.codes
    data_encoded['婚姻状态'] = data_encoded['婚姻状态'].astype('category').cat.codes
    data_encoded['政治面貌'] = data_encoded['政治面貌'].astype('category').cat.codes
    data_encoded['行业代码'] = data_encoded['行业代码'].astype('category').cat.codes

    # 计算相关性矩阵
    correlation_matrix = data_encoded.corr()

    # 绘制热力图
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
    plt.title('各特征之间的相关性热力图', fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

    # 3. 箱线图 - 展示失业时间对就业状态的影响
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='就业状态', y='失业时间（月）', data=data_new)
    plt.title('失业时间对就业状态的影响', fontsize=14, fontweight='bold')
    plt.xlabel('就业状态', fontsize=12)
    plt.ylabel('失业时间（月）', fontsize=12)
    plt.tight_layout()
    plt.show()
```

问题二：

```
import numpy as np
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# 加载数据
file_path = 'status_and_no_missing_qu.csv'
data = pd.read_csv(file_path)

# 清理数据，删除无关列
data_cleaned = data.drop(columns=['是否独居','变动类型','是否老年人','是否青少年', '人员编 11 号', '姓名', '生日', '所学专业代码', '所学专业名称','户籍地址','户口所在地区（名称）','兵役状态','是否残疾人'])

# 将类别变量转换为数值型
label_encoder = LabelEncoder()
categorical_columns = data_cleaned.select_dtypes(include=['object']).columns
for col in categorical_columns:
    data_cleaned[col] = label_encoder.fit_transform(data_cleaned[col])

# 删除缺失值
data_cleaned = data_cleaned.dropna()

# 定义特征和目标变量
X = data_cleaned.drop(columns=['就业状态'])
y = data_cleaned['就业状态']

# 切分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 定义分类模型
models_classification_final = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': xgb.XGBClassifier()
}
```

```
# 定义评估函数
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    return accuracy, precision, recall, f1

# 存储每个模型的评估指标
metrics_classification_final = {}

# 评估每个新模型
for model_name, model in models_classification_final.items():
    accuracy, precision, recall, f1 = evaluate_model(model, X_train, X_test, y_train, y_test)
    metrics_classification_final[model_name] = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}

# 输出评估指标
print(metrics_classification_final)

# 定义绘制特征重要性的函数
def plot_feature_importance(model, model_name):
    if hasattr(model, 'coef_'):    # For Logistic Regression and SVM
        feature_importances = model.coef_.flatten()
        sorted_idx = np.argsort(np.abs(feature_importances))    # Sort by absolute value
        plt.figure(figsize=(10, 6))
        plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
        plt.yticks(range(len(sorted_idx)), [X.columns[i] for i in sorted_idx])
        plt.title(f'Feature Importance - {model_name} (Coefficients)')
        plt.xlabel('Coefficient Value')
        plt.show()
    elif hasattr(model, 'feature_importances_'):    # For tree-based models
        feature_importances = model.feature_importances_
        sorted_idx = feature_importances.argsort()
        plt.figure(figsize=(10, 6))
        plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
        plt.yticks(range(len(sorted_idx)), [X.columns[i] for i in sorted_idx])
        plt.title(f'Feature Importance - {model_name}')
        plt.xlabel('Importance')
        plt.show()
```

```
# 对所有模型绘制特征重要性
for model_name, model in models_classification_final.items():
    model.fit(X_train, y_train)
    plot_feature_importance(model, model_name) from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

# 定义 stacking 模型
stacking_model = StackingClassifier(
    estimators=[
        ('decision_tree', DecisionTreeClassifier()),
        ('random_forest', RandomForestClassifier()),
        ('gradient_boosting', GradientBoostingClassifier()),
        ('xgboost', xgb.XGBClassifier())
    ],
    final_estimator=LogisticRegression()  # 使用逻辑回归作为最终模型
)

# 将 stacking 模型添加到模型列表中
models_classification_final['Stacking'] = stacking_model

# 评估每个模型
metrics_classification_final = {}
for model_name, model in models_classification_final.items():
    accuracy, precision, recall, f1 = evaluate_model(model, X_train, X_test, y_train, y_test)
    metrics_classification_final[model_name] = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}

# 输出评估指标
print(metrics_classification_final)

# 对所有模型绘制特征重要性
for model_name, model in models_classification_final.items():
    model.fit(X_train, y_train)
    plot_feature_importance(model, model_name) import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, learning_curve
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
```

```
# 加载数据
file_path = 'status_and_no_missing_qu.csv'
data = pd.read_csv(file_path)

# 清理数据，删除无关列
data_cleaned = data.drop(columns=['是否独居','变动类型','是否老年人','是否青少年', '人员编 11 号','姓名','生日','所学专业代码','所学专业名称','户籍地址','户口所在地区（名称）','兵役状态','是否残疾人'])

# 将类别变量转换为数值型
label_encoder = LabelEncoder()
categorical_columns = data_cleaned.select_dtypes(include=['object']).columns
for col in categorical_columns:
    data_cleaned[col] = label_encoder.fit_transform(data_cleaned[col])

# 删除缺失值
data_cleaned = data_cleaned.dropna()

# 定义特征和目标变量
X = data_cleaned.drop(columns=['就业状态'])
y = data_cleaned['就业状态']

# 切分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 定义分类模型
models_classification_final = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': xgb.XGBClassifier()
}

# 定义超参数网格
param_grids = {
    'Decision Tree': {
        'max_depth': [3, 5, 7, 10],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 5]
    },
    'Random Forest': {
        'n_estimators': [50, 100, 200],
        'max_depth': [3, 5, 10],
```

```
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 5]
        },
        'Gradient Boosting': {
            'n_estimators': [50, 100, 200],
            'learning_rate': [0.01, 0.1, 0.5],
            'max_depth': [3, 5, 7],
            'min_samples_split': [2, 5, 10],
        },
        'XGBoost': {
            'n_estimators': [50, 100, 200],
            'learning_rate': [0.01, 0.1, 0.5],
            'max_depth': [3, 5, 7],
            'min_child_weight': [1, 3, 5],
            'subsample': [0.7, 0.8, 0.9]
        }
}

# 存储每个模型的最佳模型
optimized_models = {}

# 超参数调优和评估每个模型，并保存优化后的模型
for model_name, model in models_classification_final.items():
    print(f"正在调优 {model_name}...")

    # 网格搜索超参数
    grid_search = GridSearchCV(model, param_grids[model_name], cv=5, n_jobs=-1,
verbose=1)
    grid_search.fit(X_train, y_train)

    # 获取优化后的模型
    best_model = grid_search.best_estimator_

    # 存储优化后的模型
    optimized_models[model_name] = best_model

    # 评估优化后的模型
    accuracy, precision, recall, f1 = evaluate_model(best_model, X_train, X_test, y_train,
y_test)

    # 打印优化后的超参数和结果
    print(f"优化后的超参数：{grid_search.best_params_}")
    print(f"优化后的 {model_name} 评估指标：")
    print(f"准确率: {accuracy:.4f}")
```

```
        print(f"查准率: {precision:.4f}")
        print(f"召回率: {recall:.4f}")
        print(f"F1  分数: {f1:.4f}")
        print("="*50)

# 输出所有模型的优化后的指标
print("\n 所有模型的优化后的评估指标：")
for model_name, metrics in metrics_classification_final.items():
        print(f"{model_name} - {metrics}")

# 定义绘制学习曲线的改进函数
def plot_learning_curve(model, model_name, ax):
        train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=5,
scoring='accuracy', n_jobs=-1)

        # 计算训练和测试集的平均准确率
        train_mean = np.mean(train_scores, axis=1)
        test_mean = np.mean(test_scores, axis=1)

        # 计算标准差
        train_std = np.std(train_scores, axis=1)
        test_std = np.std(test_scores, axis=1)

        # 绘制学习曲线
        ax.plot(train_sizes, train_mean, label=f'Training score', color='r', marker='o',
linestyle='-', markersize=5)
        ax.plot(train_sizes, test_mean, label=f'Cross-validation score', color='g', marker='s',
linestyle='--', markersize=5)

        # 填充标准差区域
        ax.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color='r',
alpha=0.2)
        ax.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, color='g',
alpha=0.2)

        # 设置标题和标签
        ax.set_title(f'{model_name}', fontsize=14)
        ax.set_xlabel('Training Size', fontsize=12)
        ax.set_ylabel('Accuracy', fontsize=12)
        ax.grid(True, which='both', linestyle='--', linewidth=0.5)
        ax.legend(loc='best')

# 设置子图布局
fig, axes = plt.subplots(2, 2, figsize=(15, 12))    # 2x2 grid for 4 plots
```

```
axes = axes.ravel()  # Flatten the 2D array of axes to 1D for easy iteration

# 绘制每个优化后的模型的学习曲线
for i, (model_name, model) in enumerate(optimized_models.items()):
    plot_learning_curve(model, model_name, axes[i])

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

问题三：

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# 加载数据
file_path = '合并训练与外部.csv'
data = pd.read_csv(file_path)

# 清理数据，删除无关列
data_cleaned = data.drop(columns=['是否独居','变动类型','是否老年人','是否青少年', '人员编 11 号', '姓名', '生日', '所学专业代码', '所学专业名称','户籍地址','户口所在地区（名称）','兵役状态','是否残疾人'])

# 将类别变量转换为数值型
label_encoder = LabelEncoder()
categorical_columns = data_cleaned.select_dtypes(include=['object']).columns
for col in categorical_columns:
    data_cleaned[col] = label_encoder.fit_transform(data_cleaned[col])

# 删除缺失值
data_cleaned = data_cleaned.dropna()

# 定义特征和目标变量
X = data_cleaned.drop(columns=['就业状态'])
y = data_cleaned['就业状态']

# 切分数据集
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 定义分类模型
models_classification_final = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': xgb.XGBClassifier()
}

# 定义评估函数
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    return accuracy, precision, recall, f1

# 存储每个模型的评估指标
metrics_classification_final = {}

# 评估每个新模型
for model_name, model in models_classification_final.items():
    accuracy, precision, recall, f1 = evaluate_model(model, X_train, X_test, y_train, y_test)
    metrics_classification_final[model_name] = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}

# 输出评估指标
print(metrics_classification_final)

# 定义绘制特征重要性的函数
def plot_feature_importance(model, model_name):
    if hasattr(model, 'coef_'):    # For Logistic Regression and SVM
        feature_importances = model.coef_.flatten()
        sorted_idx = np.argsort(np.abs(feature_importances))    # Sort by absolute value
        plt.figure(figsize=(10, 6))
        plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
        plt.yticks(range(len(sorted_idx)), [X.columns[i] for i in sorted_idx])
        plt.title(f'Feature Importance - {model_name} (Coefficients)')
        plt.xlabel('Coefficient Value')
```

```
            plt.show()
        elif hasattr(model, 'feature_importances_'):    # For tree-based models
            feature_importances = model.feature_importances_
            sorted_idx = feature_importances.argsort()
            plt.figure(figsize=(10, 6))
            plt.barh(range(len(sorted_idx)), feature_importances[sorted_idx], align='center')
            plt.yticks(range(len(sorted_idx)), [X.columns[i] for i in sorted_idx])
            plt.title(f'Feature Importance - {model_name}')
            plt.xlabel('Importance')
            plt.show()

# 对所有模型绘制特征重要性
for model_name, model in models_classification_final.items():
        model.fit(X_train, y_train)
        plot_feature_importance(model, model_name) import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, learning_curve
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# 加载数据
file_path = '合并训练与外部.csv'
data = pd.read_csv(file_path)

# 清理数据，删除无关列
data_cleaned = data.drop(columns=['是否独居','变动类型','是否老年人','是否青少年', '人员编 11 号', '姓名', '生日', '所学专业代码', '所学专业名称','户籍地址','户口所在地区（名称）','兵役状态','是否残疾人'])

# 将类别变量转换为数值型
label_encoder = LabelEncoder()
categorical_columns = data_cleaned.select_dtypes(include=['object']).columns
for col in categorical_columns:
        data_cleaned[col] = label_encoder.fit_transform(data_cleaned[col])

# 删除缺失值
data_cleaned = data_cleaned.dropna()

# 定义特征和目标变量
X = data_cleaned.drop(columns=['就业状态'])
```

```
y = data_cleaned['就业状态']

# 切分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 定义分类模型
models_classification_final = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': xgb.XGBClassifier()
}

# 定义超参数网格
param_grids = {
    'Decision Tree': {
        'max_depth': [3, 5, 7, 10],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 5]
    },
    'Random Forest': {
        'n_estimators': [50, 100, 200],
        'max_depth': [3, 5, 10],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 5]
    },
    'Gradient Boosting': {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.5],
        'max_depth': [3, 5, 7],
        'min_samples_split': [2, 5, 10],
    },
    'XGBoost': {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.5],
        'max_depth': [3, 5, 7],
        'min_child_weight': [1, 3, 5],
        'subsample': [0.7, 0.8, 0.9]
    }
}

# 存储每个模型的最佳模型
optimized_models = {}
```

```
# 超参数调优和评估每个模型，并保存优化后的模型
for model_name, model in models_classification_final.items():
    print(f"正在调优 {model_name}...")

    # 网格搜索超参数
    grid_search = GridSearchCV(model, param_grids[model_name], cv=5, n_jobs=-1, verbose=1)
    grid_search.fit(X_train, y_train)

    # 获取优化后的模型
    best_model = grid_search.best_estimator_

    # 存储优化后的模型
    optimized_models[model_name] = best_model

    # 评估优化后的模型
    accuracy, precision, recall, f1 = evaluate_model(best_model, X_train, X_test, y_train, y_test)

    # 打印优化后的超参数和结果
    print(f"优化后的超参数：{grid_search.best_params_}")
    print(f"优化后的 {model_name} 评估指标：")
    print(f"准确率: {accuracy:.4f}")
    print(f"查准率: {precision:.4f}")
    print(f"召回率: {recall:.4f}")
    print(f"F1 分数: {f1:.4f}")
    print("="*50)

# 输出所有模型的优化后的指标
print("\n 所有模型的优化后的评估指标：")
for model_name, metrics in metrics_classification_final.items():
    print(f"{model_name} - {metrics}")

# 定义绘制学习曲线的改进函数
def plot_learning_curve(model, model_name, ax):
    train_sizes, train_scores, test_scores = learning_curve(model, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)

    # 计算训练和测试集的平均准确率
    train_mean = np.mean(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)

    # 计算标准差
    train_std = np.std(train_scores, axis=1)
```

```
    test_std = np.std(test_scores, axis=1)

    # 绘制学习曲线
    ax.plot(train_sizes, train_mean, label=f'Training score', color='r', marker='o', linestyle='-', markersize=5)
    ax.plot(train_sizes, test_mean, label=f'Cross-validation score', color='g', marker='s', linestyle='--', markersize=5)

    # 填充标准差区域
    ax.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color='r', alpha=0.2)
    ax.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, color='g', alpha=0.2)

    # 设置标题和标签
    ax.set_title(f'{model_name}', fontsize=14)
    ax.set_xlabel('Training Size', fontsize=12)
    ax.set_ylabel('Accuracy', fontsize=12)
    ax.grid(True, which='both', linestyle='--', linewidth=0.5)
    ax.legend(loc='best')

# 设置子图布局
fig, axes = plt.subplots(2, 2, figsize=(15, 12))  # 2x2 grid for 4 plots
axes = axes.ravel()  # Flatten the 2D array of axes to 1D for easy iteration

# 绘制每个优化后的模型的学习曲线
for i, (model_name, model) in enumerate(optimized_models.items()):
    plot_learning_curve(model, model_name, axes[i])

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

问题四：

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


# 处理薪资期望字段为数值型
salary_data = jobseeker_df['薪资期望'].str.split('-', expand=True).astype(int)
jobseeker_df['薪资期望最低'] = salary_data[0]
jobseeker_df['薪资期望最高'] = salary_data[1]

# 3. **求职者与岗位匹配度的热力图**：通过余弦相似度计算并展示
```

```
matching_matrix = np.random.rand(20, 10)  # 随机生成 20 个求职者和 10 个岗位的匹配度数据
plt.figure(figsize=(12, 8))
sns.heatmap(matching_matrix, annot=False, cmap='Blues', cbar=True, linewidths=0.5)
plt.title('失业者与岗位匹配度热力图', fontsize=14)
plt.xlabel('岗位 ID', fontsize=12)
plt.ylabel('求职者 ID', fontsize=12)
plt.show()from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns


# PCA 降维，压缩到二维
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(matching_matrix)

# 绘制 PCA 降维后的散点图
plt.figure(figsize=(12, 8))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c='blue', alpha=0.6, s=50)
plt.title('PCA 降维后的求职者与岗位匹配度分布', fontsize=14)
plt.xlabel('主成分 1', fontsize=12)
plt.ylabel('主成分 2', fontsize=12)
plt.show()

# 为了做气泡图，我们需要使用匹配度作为气泡的大小
plt.figure(figsize=(12, 8))
bubble_size = np.max(matching_matrix, axis=1) * 100  # 气泡的大小由匹配度决定
plt.scatter(np.arange(944), np.random.randint(0, 500, 944), s=bubble_size, alpha=0.5, c='red', edgecolors='black')
plt.title('失业者与岗位的匹配度', fontsize=14)
plt.xlabel('求职者 ID', fontsize=12)
plt.ylabel('岗位 ID', fontsize=12)
plt.show()ommendations = np.random.rand(num_jobseekers, 5)  # 每个求职者的前 5 个岗位匹配度

# 绘制推荐匹配度排名条形图
plt.figure(figsize=(12, 8))

# 选择前 100 个求职者进行展示
sns.barplot(x=np.arange(50), y=np.max(top_n_recommendations[:50], axis=1), palette='viridis')

# 设置标题和标签
plt.title('失业者的前 50 推荐岗位匹配度', fontsize=14)
```

```
plt.xlabel('求职者 ID', fontsize=12)
plt.ylabel('推荐匹配度', fontsize=12)

# 旋转横坐标标签，避免重叠
plt.xticks(rotation=45)

plt.show()import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 数据标准化
scaler = StandardScaler()
jobseeker_scaled = scaler.fit_transform(jobseeker_df)

# 使用肘部法则来确定最佳簇数
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(jobseeker_scaled)
    inertia.append(kmeans.inertia_)

# 绘制肘部法则图
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('肘部法则 - 聚类数选择', fontsize=14)
plt.xlabel('簇数 (K)', fontsize=12)
plt.ylabel('簇内误差平方和 (Inertia)', fontsize=12)
plt.show()

# 选择 K=3 作为最佳簇数
kmeans = KMeans(n_clusters=10, random_state=42)
jobseeker_df['聚类标签'] = kmeans.fit_predict(jobseeker_scaled)

# 聚类结果可视化（PCA 降维）
pca = PCA(n_components=2)
jobseeker_pca = pca.fit_transform(jobseeker_scaled)

# 绘制聚类结果的散点图
plt.figure(figsize=(12, 8))
sns.scatterplot(x=jobseeker_pca[:, 0], y=jobseeker_pca[:, 1], hue=jobseeker_df['聚类标签'],
```

```
palette='Set1', s=100, alpha=0.7)
plt.title('失业者聚类分析', fontsize=14)
plt.xlabel('主成分 1', fontsize=12)
plt.ylabel('主成分 2', fontsize=12)
plt.show()

# 分析每个聚类的求职者平均特征
cluster_summary = jobseeker_df.groupby('聚类标签').mean()
print(cluster_summary)
```