

队伍编号	MCB2201112
赛道	B

基于多模型调参优化的 Stacking 用户评分预测集成学习

摘要

随着移动通信技术的迅猛发展和网络工程的不断建设，在信息透明、产品同质化的今天，提升语音通话及网络服务的质量，满足用户对高质量语音通话、网络服务的需求显得尤为重要。本文旨在建立一个基于多模型调参优化的 Stacking 集成学习，完善且合理地预测用户评分的普适性模型，从已有数据中心获得有效信息，更高效地提升服务质量，从而完善业务服务体系。

针对问题一，主要需要对用户语音及上网业务评分影响因素的程度进行量化分析。本文首先对数据集进行统一处理，包括：初步剔除相关列数据、学习数据与预测数据指标一致化、指标规范化、空缺值处理、标签编码、特征构造、数据标准化、学习数据与预测数据一致化、学习数据训练集与测试集划分。之后在处理好的数据集上建立熵权法、灰色关联度分析、随机森林分类模型，多方面综合考虑，量化分析各影响因素对评分的影响程度，最终结果见表 2、表 3 及表 4，并依此来确定影响用户两项业务满意度的主要因素。量化结果接近于实际生活，效果良好，且可为后续问题奠定基础。

针对问题二，主要需要根据已有影响因素对用户的评分进行预测，并解释预测的合理性。本文首先结合问题一量化结果以及建立主成分分析模型，对数据累计方差进行解释，确定特征个数；之后建立 XGBoost 模型，并得出各影响因素的重要性，与随机森林模型结合分析，确定特征的选择；再建立 KNN、SVM、LightGBM 以及多分类逻辑回归模型，对数据进行学习分析；随后，对各个模型进行超参数调优，模型准确率均有大幅度提升，如随机森林较原先提升了 11.69%，最高提升较原先可达到 14.25%，效果良好。再者，以模型的准确率、平均绝对误差、均方误差为标准，选择表现较优的模型作为 Stacking 集成学习的基模型，同时选择余下的一个模型作为第二层模型，在提升准确率的同时，避免过拟合。同时对其采用五折交叉验证，验证其稳健性。Stacking 集成学习结果符合预期效果，各评分预测模型效果见表 10，明显优于单一模型。在保证准确率的同时，预测的平均绝对误差、均方误差均有一定优化，同时本文还注重结果的可解释性及模型的现实意义。最后，本文进行可视化分析，绘制原始数据及预测数据评分人数南丁格尔玫瑰图，查看数据分布，绘制模型的混淆矩阵热力图、分类报告、ROC/AUC 曲线，多方面评估模型效果及解释模型的合理性。综合上述分析，可以确认模型效果良好，具有良好的稳健性、泛化能力。

最后，本文对所建立的模型的优缺点进行了中肯的评价、提出了模型的改进措施以及对模型进行了一定推广。

关键词：影响程度量化分析；特征工程；Stacking 集成学习；评分预测；可视化评估

目录

一、 问题的提出	1
1.1 问题背景	1
1.2 问题要求	1
二、 问题的分析	1
2.1 问题的整体分析	1
2.2 问题一的分析	2
2.3 问题二的分析	2
三、 符号说明	2
四、 模型的假设	3
五、 模型的建立与求解	3
5.1 数据的准备	3
5.2 问题一模型的建立与求解	6
5.2.1 模型的建立	6
5.2.2 语音业务求解	9
5.2.3 上网业务求解	13
5.3 问题二模型的建立与求解	16
5.3.1 累计解释方差	16
5.3.2 多种多分类模型的建立	17
5.3.3 Stacking 集成学习原理解释	20
5.3.4 模型的超参数调优	20
5.3.5 特征选择	22
5.3.6 模型的选择及集成	22
5.3.7 客户评分预测	24
5.3.8 模型预测结果合理性分析	24
六、 模型的评价与推广	30
6.1 模型的评价	30
6.2 模型的推广	31
参考文献	32
附录	33

一、问题的提出

1.1 问题背景

随着移动通信技术的迅猛发展和网络工程的不断建设，在信息透明、产品同质化的今天，提升语音通话及网络服务的质量，满足用户对高质量语音通话、网络服务的需求显得尤为重要。由于当今用户数量的不断增多、用户需求不断提高、运营商业务不断广泛化，因此点对点、传统方法解决问题逐渐困难化。而现在有来自移动通信集团北京分公司根据用户对语音业务及上网业务的满意度进行的评分及相关影响因素的数据，我们需要对其进行分析、建立相关数学模型，以便从数据中心获得有效信息，更高效地提升服务质量，为客户提供更好的服务。

1.2 问题要求

- **问题一：**研究并量化分析影响用户对语音及上网业务满意度的主要因素；
- **问题二：**建立基于影响用户评分影响因素的数学模型，并依据附件 3、4 中相关因素对其评分进行预测，并解释预测评分的合理性。

二、问题的分析

2.1 问题的整体分析

该题是一个关于移动用户对语音及上网业务体验评分的数据分析、预测类问题。

从分析目的看，本题需要分析用户对语音与上网业务的评分及各个影响因素，筛选出影响用户评分的主要因素，并量化结果。同时需要对用户的评分进行预测及研究，为运营商提供参考，从而提升用户语音及上网的优质体验。因此本题主要需完成两方面任务：**其一**，研究影响用户语音及上网业务满意度的主要因素，并对各因素进行量化分析；**其二**，根据上述的分析，建立合理模型，对用户的评分进行预测及研究，确保分类模型的准确性、稳健性、可靠性，并有一定的泛化能力，且能够包容用户真实评分的主观性。

从数据来源、特征看，本题的数据来源于北京移动用户的语音与上网业务评分数据，数据包括用户对语音业务下“语音通话整体满意度”“网络覆盖与信号强度”“语音通话清晰度”“语音通话稳定性”，上网业务下“手机上网整体满意度”“网络覆盖与信号强度”“手机上网速度”“手机上网稳定性”方面的评分，以及相关的影响评分的因素。评分数据具有主观性，影响因素数据具有高维、多样、标准体系不一致、量纲不一致等特点，且数据量较大。因此，本题数据相对特殊且复杂，需要对数据进行一定的预处理，以便于后续的分析。

从模型的选择看，本题数据量较大、维度较高，且分析目的是分析影响用户评分的主要因素，并对用户的评分进行预测及研究。本文将评分视为多分类，且评分具有一定主观性、分类种类多，因此，在模型的选择上，本文结合多种分类预测模型，构建集成学习模型，尽可能多地学习到用户评分特点，提升模型的准确性、稳健性及可泛化性能。

从软件的选择看，本题为数据类型，且需要进行大量的数据分析、预测等，因此我们选择 Python Jupyter 对问题进行求解，其交互式的编程范式，方便且高效。

2.2 问题一的分析

问题一的核心目的在于研究并量化分析影响用户对语音及上网业务满意度的主要因素。对于已给的数据集，数据在完整度、指标标准等方面存在一定缺陷。这导致在原数据上我们不可直接进行分析，需要对原数据集进行数据的预处理。此外附件数据集在语音及上网业务中，每一业务均有四项评分，因此我们需要对每一项评分进行分析，对各因素进行量化。结合数据来源、与特征方面，我们综合皮尔逊相关系数、熵权法、灰色关联度分析、随机森林分类，构建多元量化分析模型，尽可能准确挖掘到影响用户评分的因素，为构建后续预测模型提供优质依据。

2.3 问题二的分析

问题二的核心目的在于建立基于影响用户评分影响因素的数学模型，并依据附件 3、4 中相关因素对用户评分进行预测，并解释模型预测的合理性。但是在附件 1 与附件 2，附件 3 与附件 4 中，影响因素存在不配对的情况。这导致在给定用户评分的数据中，部分因素不可作为模型建立的基础特征数据，因此在数据预处理的同时，还需要对附件 1 与附件 2，附件 3 与附件 4 中的影响因素列取交集，使得学习数据与预测数据的特征数据一致。此外，在已给的存在用户评分的数据集中，用户对每一项的评分均为整数，不存在小数，且评分范围为 [1, 10]。因此，我们在建立预测模型时，应尽量避免使用回归模型，而应使用分类模型，但部分分类模型需要分类标签量值从 0 开始，因此需要对所有评分进行标签编码，规范数据。同时分类种类较多，对于单一模型，其预测准确率较低，平均绝对误差较高、泛化能力较弱……因此，本文结合多种机器学习模型，构建集成学习模型，尽可能准确预测用户评分。最后，在此基础上，结合模型的分类混淆矩阵热力图、分类报告、ROC/AUC 曲线等对于预测结果进行解释，叙述模型的合理性，同时考虑集成学习模型对预测误差的包容性，对模型的泛化能力进行分析。

三、符号说明

符号	符号说明
μ	样本平均值
σ	样本方差
x_{standard}	经过标准化后的数据
$R(x)_{m \times n}$	经过某项处理后的数据特征集
ρ	皮尔逊相关系数
x'	经过某项处理后的数据
$Gini$	样本集合基尼系数
\hat{y}	预测值
$L^{(t)}$	目标函数
Ω	叶节点正则项惩罚系数
P	某事件发生的概率
ω	权重

四、模型的假设

- **假设一：**语音与上网业务的八项评分中，存在个别用户乱评、错评现象；
- **假设二：**除个别用户的部分评分外，其余所有数据真实且符合实际情况；
- **假设三：**用户评分还受到除附件中因素之外的因素的影响；
- **假设四：**给定的数据集可全面体现用户整体情况；
- **假设五：**对于同一业务，学习数据与预测数据的内在规律是一致的。

五、模型的建立与求解

模型的建立与求解¹部分主要分为数据的准备，模型建立与求解、结果分析。²

- **数据的准备：**对于给定的数据集，对数据进行预处理。
- **模型的建立、求解、结果分析：**对于给定的数据集，建立相关模型，研究并量化分析影响用户对语音及上网业务满意度的主要因素。此外还需要建立基于影响用户评分影响因素的数学模型，并依据附件 3、4 中相关因素对其评分进行预测，并解释预测评分的合理性。

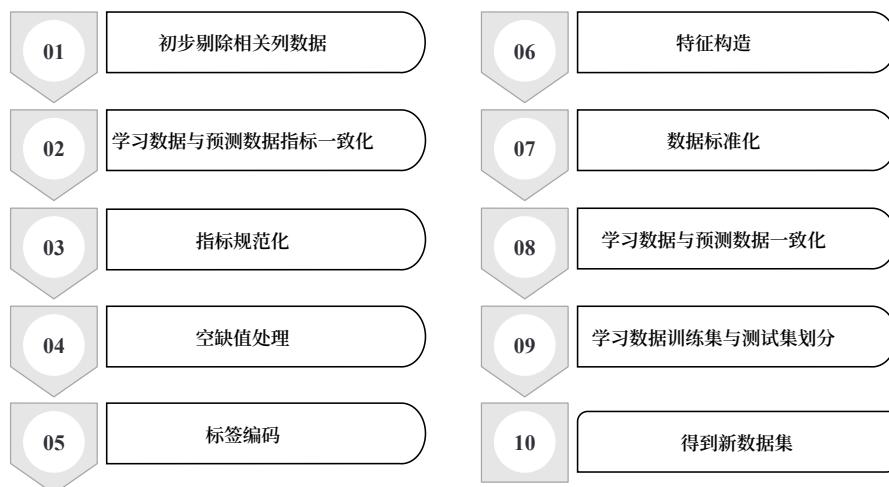


图 1 数据的准备主要过程

5.1 数据的准备

为方便、准确、高效解决问题，我们需要对数据进行预处理，其主要过程见图 1，包括：初步剔除相关列数据、学习数据与预测数据指标一致化、指标规范化、空缺值处理、标签编码、特征构造、标准化、学习数据与预测数据一致化、学习数据训练集与测试集划分。**本文后续的模型建立都在此基础之上。**

¹本文中所有解决问题的源程序均可在附录 [D] 或支撑材料中查看

²本文中除词云图及南丁格尔玫瑰图示外，其余均为矢量图。由于数据较多，在 PDF 阅读时，表格、图示中字体可能较小，读者可适当放大进行查看，所有图表放大后，均可清晰查看。

Step1 初步剔除相关列数据

由于“用户 id”为连续编号，且与评分无任何关系，故本文将该列数据剔除；同时对于“用户描述”等文字性叙述指标，由于其均为文本，且描述特征难以提取，难以量化，本文将该列数据剔除，但为了获得客户相关描述，本文将绘制用户描述高频词汇云图；此外，对于“终端品牌类型”等多类别指标，由于其类别较多，量化后难以提取出有效信息，故也将其剔除，其余列暂时保留。

Step2 学习数据与预测数据指标一致化

附件 1 与附件 3 为用户语音业务数据，但两表数据影响的因素存在不一致的现象，需要对指标取交集，确保两者一致，附件 2 与附件 4 同理。这里我们利用 Python 中集合 set 容器元素唯一性特征及 pandas 库，筛选出相同因素。而对于可能重合的指标，我们在下文也会进行一定处理。这样即可确保在学习数据上建立的模型依据的指标存在于预测数据集上，避免两者指标不一的情况。

Step3 指标规范化

经过上述处理后，我们发现大部分影响因素为分类指标，其划分“是”“否”的字段不统一，故依据“附件 5 附件 1、2、3、4 的字段说明.xlsx”文件，本文对附件 1、2、3、4 中的分类指标进行规范化，记“是”类别为 1，“否”类别为 0，方便后续模型的建立。

Step4 空缺值处理

在给定数据集中，部分空缺值可以依据附件 5 的解释进行填充。经过一定处理后，附件 3 与附件 4 中无空缺值，故本文对附件 1 与附件 2 中的空缺值进行分析与处理：

- 对于附件 1：据附件 5 解释进行填充后，还存在个别用户的空缺值，如图 2 所示。空缺

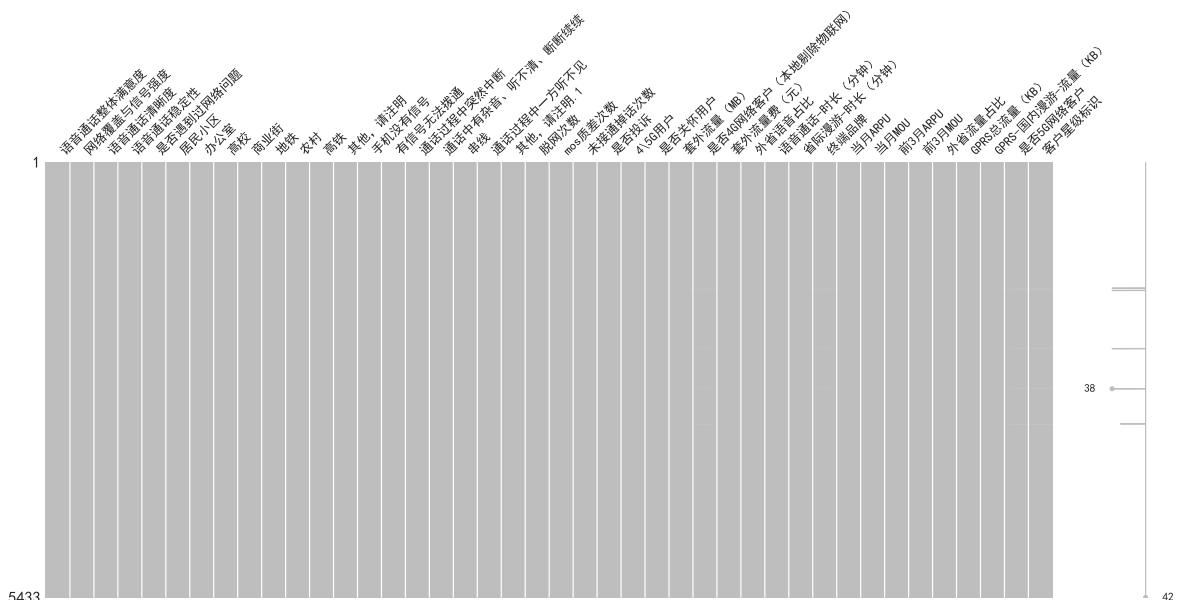


图 2 附件 1 初步处理后缺失值查看

值的列名为：“是否 4G 网络客户（本地剔除物联网）”“终端品牌”“是否 5G 网络客户”“客户星级标识”，且这些空缺值均在同一用户中出现，用户 id 分别为 1573、1601、2326、2827、3265。附件 1 的空缺值有集中、个数少的特点，存有空缺值的用户仅有 5 个，占整体用户的 0.0920%，对于模型的建立影响较小，因此我们将这 5 行用户剔除。

- **对于附件 2：** 经过指标一致化后及初步数据空缺值的填补后，附件 2 中仅剩“终端品牌”列指标存在 14 个空缺值，根据该列数据其余特征，我们将这个 14 个空缺值以 0 填充。

Step5 标签编码

首先，对用户的评分进行编码，由于部分分类模型需要分类量值从 0 开始，因此，为方便后续集成学习等，本文将评分从 [1, 10] 映射至 [0, 9]，且仍均为整数，即将评分减 1。其次，对“终端品牌”“4\5G 用户”指标利用 Python 的 sklearn 库中的 LabelEncoder 进行标签编码。此外，对于“客户星级标识”指标，我们依据移动公司对客户星级标识的划分进行编码，编码值对应见表 1。

表 1 客户星级标识编码对应表

未评级	准星	一星	二星	三星	银卡	金卡	白金卡	钻石卡
0	1	2	3	4	5	6	7	8

Step6 特征构造

观察并分析给定的数据，我们可以初步构造以下特征：

- **对于附件 1 与附件 3：** 观察到附件 1 中有“家宽投诉”与“资费投诉”两项，而在附件 3 中有“是否投诉”一项，因此，我们在附件 1 中构造“是否投诉”一项。若“家宽投诉”与“资费投诉”均为 0，则“是否投诉”记为 0，否则记为 1。并同时删去“家宽投诉”与“资费投诉”。
- **对于附件 2 与附件 4：** 观察数据，我们构造三项新指标，分别为“出现问题场所或应用总”“网络卡速度慢延时大上不了网总”“质差总”。其来源为对应列指标按行求和。

Step7 数据标准化

该处标准化处理为 **Z-score** 方法，仅用于后续机器学习模型的使用。而在问题一的熵权法、灰色关联度分析中我们采用 **Min-Max** 方法，该方法在后文模型中会具体说明。

对于某一列数据 $x = [x_1, x_2, \dots, x_m]^T$ ，其平均值为

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i \quad (1)$$

标准差为

$$\sigma = \sqrt{\frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2} \quad (2)$$

则标准化后的数据为

$$(x_{\text{standard}})_i = \frac{x_i - \mu}{\sigma} \quad (3)$$

利用上述计算公式，我们对非分类指标进行处理，使得原数据经过处理后，其值聚集于 0 附近，即均值为 0，标准差为 1。这样处理，利于机器学习模型的建立、学习与预测，加快模型的收敛速度，并在一定程度上提升模型的准确性。同时该标准化处理方法适合当代嘈杂的大数据场景^[1]。因此对于大样本的数据，如出现部分异常值，使用该方法对最终结果影响较小。

Step8 学习数据与预测数据一致化

经过上述几项处理后，我们还需要将附件 1 与附件 3 数据集一致化，包括指标一致化以及数据字段、分布排列一致化，从而保证对于需要预测的数据集附件 3 利用在附件 1 中建立的模型所利用到的数据集的一致性，避免造成数据的不一致，导致预测错误。本文对附件 2 与附件 4 进行上述相同的操作。

Step9 学习数据训练集与测试集划分

为计算问题二中建立的模型的准确性等指标，需要在附件 1 与附件 2 中均划分训练集与测试集。对于语音业务划分训练集与测试集比例为 8:2；而对于上网业务，其比例设为 9:1。对于比例的设置，本文将在后文解释其合理性。且上述划分利用 sklearn 库中的 train_test_split 函数实现，且任意设定随机种子为 2022，确保多次调试结果的一致性。该函数可确保划分的随机性，确保训练集与测试集数据分布规律大致相同。

5.2 问题一模型的建立与求解

对于问题一，我们综合皮尔逊相关系数、熵权法、灰色关联度分析法，以及随机森林分类模型，构建多元量化分析模型，尽可能准确挖掘到影响用户评分的因素。

5.2.1 模型的建立

皮尔逊相关系数 (Pearson Correlation Coefficient) 可衡量两个变量之间的相似度，我们不妨用 $\rho(x, y)$ 表示，计算公式^[2]如下

$$\rho(x, y) = \frac{\sum_{i=1}^n (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^n (x_i - \mu_x)^2} \sqrt{\sum_{i=1}^n (y_i - \mu_y)^2}} \quad (4)$$

其中 μ 计算见⁽¹⁾式。

由该定义，显然 $\rho \in [-1, 1]$ 。当 $\rho > 0$ 时，上述两变量呈正相关；当 $\rho = 0$ 时，上述两变量不相关；当 $\rho < 0$ 时，上述两变量呈负相关。当 $|\rho|$ 越接近于 1 时，则上述两变量相关性就越强^[3]。

熵权法 (Entropy Weight Method, EWM) 是一种指标客观影响程度的量化方法。当信息熵越大时，信息的无序程度越大，此时，信息价值越小，指标权重就越小^[4]。其计算步骤如下：

- **Step1**，指标正向化。由于数据集构成的指标类别不一，部分指标可能数值越大越好，部分指标可能越小越好，而有的可能在某一点取值最优，为方便、高效评价，我们需要进行指标正向化处理^[5]。其处理方法如下

- 越大越优指标

$$x'_{ij} = x_{ij} \quad (5)$$

- 越小越优指标

$$x'_{ij} = \max(x_{ij}) - x_{ij} \quad (6)$$

- 在 β 处取值最优指标

$$x'_{ij} = 1 - \frac{|x_{ij} - \beta|}{\max(|x_{ij} - \beta|)} \quad (7)$$

- **Step2**，数据标准化。由于数据集构成的指标数据数量级存在差异、量纲不一，为消除上述情况对结果的影响，我们需要将各指标进行标准化处理，这里我们使用 Min-Max 方法。处理方法计算公式如下

$$r_{ij} = \frac{x'_{ij} - \min(x'_j)}{\max(x'_j) - \min(x'_j)} \quad (8)$$

- **Step3**，计算信息熵。进行上述处理后可得到由特征数据构成的矩阵 $R(r_{ij})_{m \times n}$ ，对于某一项指标的数据 r_j ，其信息熵为

$$E_j = -\frac{1}{\ln m} \cdot \sum_{i=1}^m p_{ij} \ln p_{ij} \quad (9)$$

其中

$$p_{ij} = \frac{r_{ij}}{\sum_{i=1}^m r_{ij}} \quad (10)$$

观察到(10)式中分母不可为0，且(9)式对数真数部分不能为0，因此，我们在进行 Step2 时，标准化区间的最小值设为 0.002，可避免计算时的不合定义。

- **Step4**，计算指标权重。其计算公式如下

$$\omega_j = \frac{1 - E_j}{\sum_{j=1}^n (1 - E_j)} \quad (11)$$

灰色关联度分析 (Grey Relation Analysis, GRA) 是通过指标之间的关联系数来判断指标对系统的影响程度^[6]。其计算方法如下：

- **Step1**, 指标无量纲化。这里指标无量纲化与熵权法的 **Step2** 中的数据标准化方法处理相同，此处不再赘述。
- **Step2**, 计算每一项比较序列与参考序列绝对值差

$$\Delta_i(k) = |R_0(k) - R_i(k)| \quad (12)$$

- **Step3**, 确定二级最小差与二级最大差

$$\max_{i=1}^n \max_{k=1}^m |R_0(k) - R_i(k)| \quad \min_{i=1}^n \min_{k=1}^m |R_0(k) - R_i(k)| \quad (13)$$

- **Step4**, 计算关联系数，记分辨系数为 η ，本文我们取 $\eta = 0.5$

$$\xi_i(k) = \frac{\min_{i=1}^n \min_{k=1}^m |R_0(k) - R_i(k)| + \eta \cdot \max_{i=1}^n \max_{k=1}^m |R_0(k) - R_i(k)|}{|R_0(k) - R_i(k)| + \eta \cdot \max_{i=1}^n \max_{k=1}^m |R_0(k) - R_i(k)|} \quad (14)$$

- **Step5**, 计算灰色关联度

$$\gamma_{0i} = \frac{1}{m} \sum_{k=1}^m \xi_i(k) \quad (15)$$

随机森林 (Random Forest, RF) 是由多棵决策树 (Decision Tree) 进行组合后对预测结果投票或取均值的一种算法^[7]。其有分类和回归两种模型，对于本题，我们选择分类模型。其简要过程如图 3 所示，算法伪代码如 Algorithm1 所示。

Algorithm 1: 随机森林 (RF)

Data: 数据集 \mathcal{D}

```

1 function DTree( $\mathcal{D}$ )
2 if Termination then
3   return base( $g_t$ )
4 else
5   learn  $b(x)$  并且依据  $b(x)$  划分  $\mathcal{D}$  为  $\mathcal{D}_C$ 
6   build  $G_C \leftarrow \text{DTree}(\mathcal{D}_C)$ 
7   return  $G(x) = \sum_{C=1}^C \llbracket b(x) = C \rrbracket G_C(x)$ 
8 end
9 function RandomForest( $\mathcal{D}$ )
10 for  $t = 1, 2, 3, \dots, T$  do
11   request 数据集  $\tilde{\mathcal{D}}_t \leftarrow \text{BoostStrapping}(\mathcal{D})$ 
12   obtain DTree  $g_t \leftarrow \text{DTree}(\tilde{\mathcal{D}}_t)$ 
13   return  $G = \text{Uniform}(g_t)$ 
14 end

```

Result: 随机森林模型 $G = \text{Uniform}(g_t)$

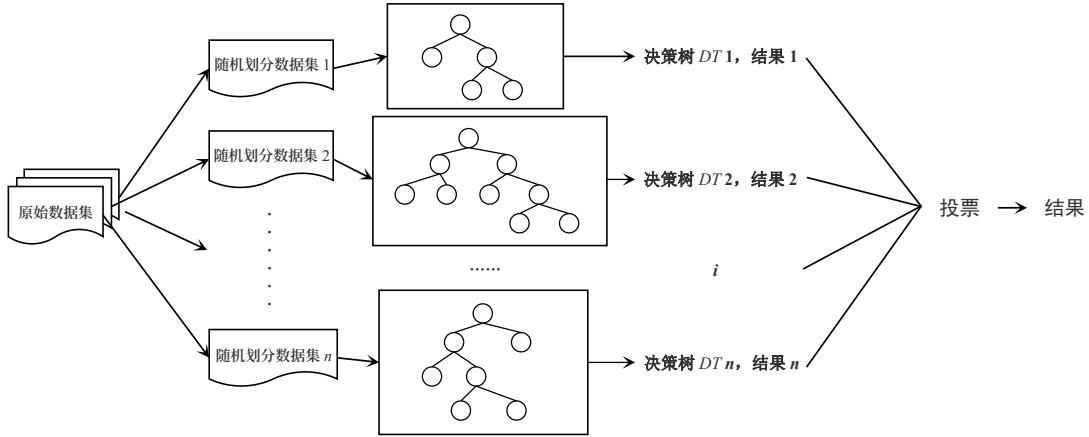


图 3 随机森林算法简图

对于单棵决策树而言，本文利用 **CART** 算法^[7]构建。基尼系数是衡量样本集合纯度的指标，当该值越小时，其纯度也就越高。计算公式如下

$$Gini(R_i) = \sum_{k=1}^K \sum_{k \neq k'} P_k P_{k'} = 1 - \sum_{k=1}^K P_k^2 \quad (16)$$

其中， R 为选取出的特征（影响因素）， K 表示在该特征中包含的类别数， P_k 表示该特征中第 k 类别的出现概率。

由上述分析可知，对于单棵决策树而言，其叶子节点的分裂特征为选择的所有特征中基尼系数最小的特征。

5.2.2 语音业务求解

首先我们绘制语音业务用户四项评分箱线图，如图 4 所示（图中数据是原评分数据通过标签编码转化而来的）。之后依据(4) 式，可以计算出各影响因素与语音业务四项评分之间的皮尔逊相关系数，并通过 Python 绘制其热力图，如图 6 所示，图中仅显示相关性排名前 10 的指标。由图 6 可以直观发现，用户对于语音业务的满意度影响较大排名在前的因素依次为

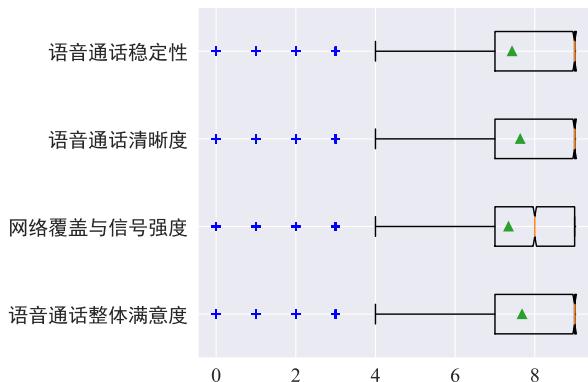


图 4 语音业务用户四项评分箱线图

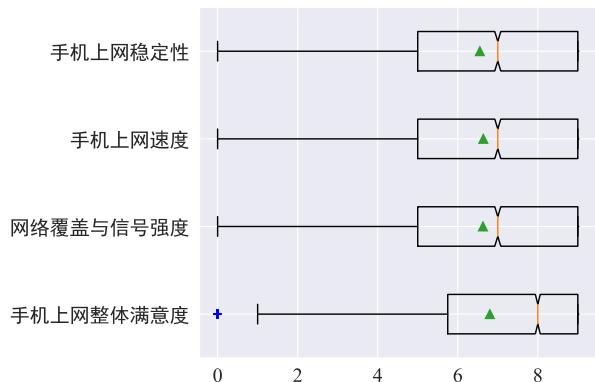


图 5 上网业务用户四项评分箱线图

“是否遇到过网络问题”、“居民小区”、“手机没有信号”、“有信号无法拨通”、“通话过程中突

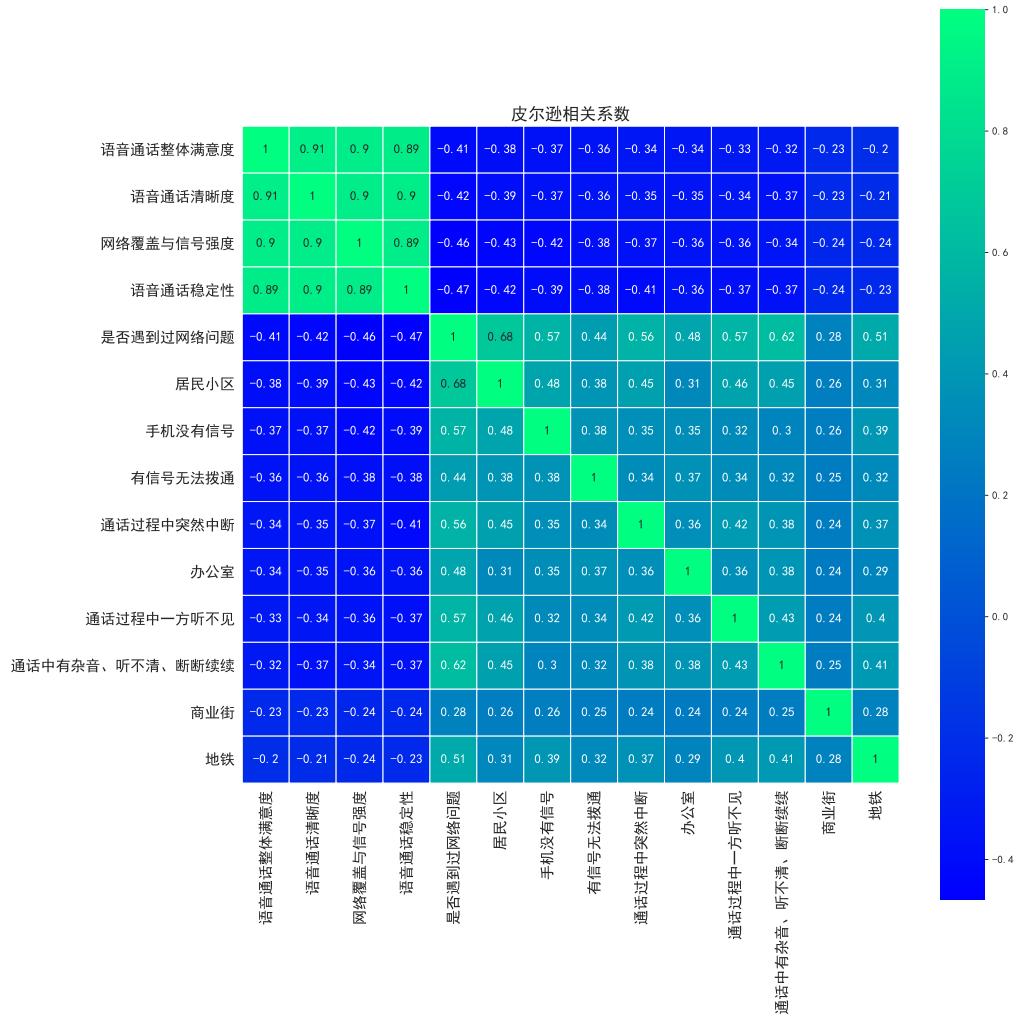


图 6 语音业务评分与其影响因素皮尔逊相关系数热力图

然中断”、“办公室”、“通话过程中一方听不见”、“通话中有杂音、听不清、断断续续”、“商业街”、“地铁”。为了更好地分析影响语音业务的因素，我们分别绘制出上述 10 个指标与语音业务四项评分之间的 RidViz 图，如图 7~图 10 所示。由此我们可以发现，用户对于语音业务的四项评分，分布规律大致一致，且主要影响满意度的因素非常相似，同时噪声较小。因此我们在此分析基础上，建立熵权法、灰色关联度分析法、随机森林模型在整体上求得量化结果并得出主要因素。结果见表 12、表 13、图 11，其中图 11 中数据见表 2 最后一列。

通过分析上述图表，我们可以发现利用熵权法、灰色关联度分析法、随机森林模型得到的排序结果大致相同，主要影响因素有：“省际漫游-时长（分钟）”、“前 3 月 ARPU”、“套外流量（MB）”、“GPRS-总流量（KB）”、“套外流量费（元）”、“脱网次数”、“未接通掉话次数”、“mos 质差次数”、“其他，请注明.1”（该指标对应的是通话过程中遇到其他问题部分的注明）、“是否投诉”（特征构造出的因素）、“是否关怀用户”、“串线”、“其他，请注明”（该指标对应的是出现问题的场所部分的注明），以及在各场所遇到语音通话各个问题的情况。而在随机森林模型中，我们可以发现有部分因素量化值也排名较前，如“当月 ARPU”、“是否遇到过网络问题”、“前 3 月 MOU”、“语音通话-时长（分钟）”、“当月 MOU”、“客户星级标识”、“终端品牌”等因素。可以发现，随机森林模型可以给出更多的影响因素，避免了熵权法及灰色关联度分析的局限性，并且量化值可以用于后续模型的建立。

因此我们在此分析基础上，将语音业务的四项评分分别作为因变量，所有指标作为自变量（不包括评分），建立随机森林模型，绘制出其特征重要性图示，由于篇幅原因，读者可翻阅附录，见图 29~图 32。以及量化各影响因素影响程度，量化结果见表 2。

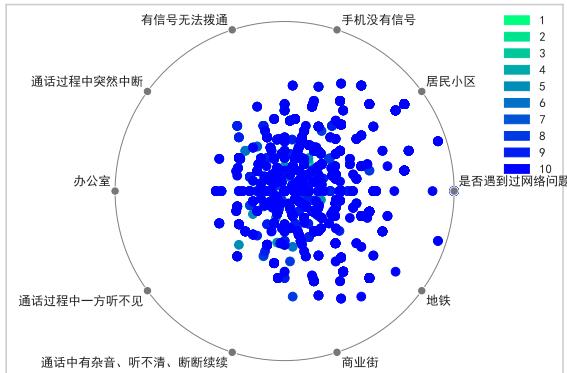


图 7 语音通话整体满意度与指标 RidViz

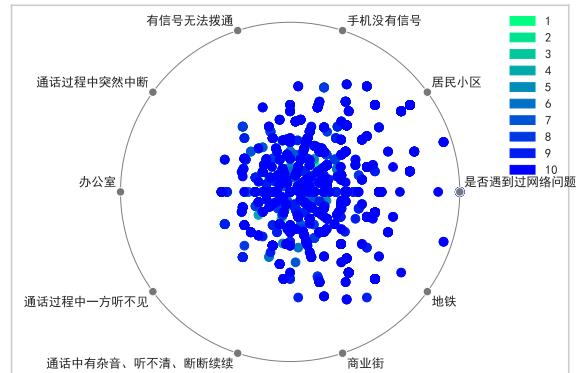


图 8 网络覆盖与信号强度与指标 RidViz

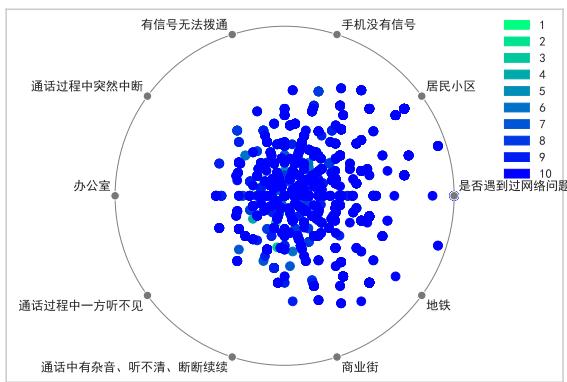


图 9 语音通话清晰度与指标 RidViz

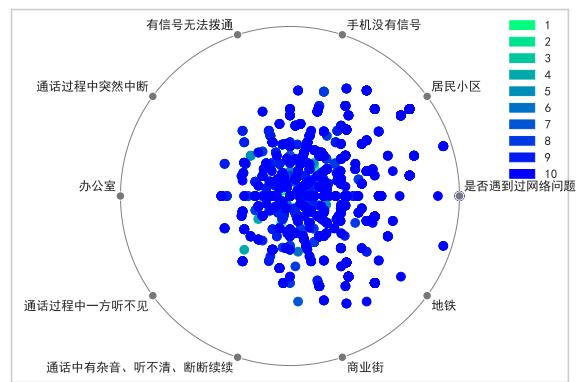


图 10 语音通话稳定性 RidViz

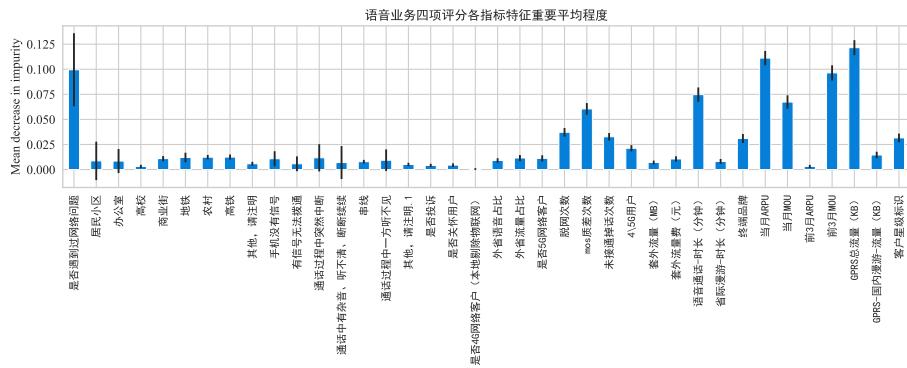


图 11 语音业务四项评分各指标特征重要平均程度

通过分析图 29~图 32以及表 2，我们可以得到以下结论：

- 对于语音通话整体满意度，影响客户满意度的主要因素有：“当月 ARPU”、“GPRS 总流量 (KB)”、“语音通话-时长 (分钟)”、“mos 质差次数”、“手机没有信号”、“未接通掉话次数”、“有信号无法拨通”、“通话中有杂音、听不清、断断续续”、“通话过程中突然中断”、“通话过程中一方听不见”，以及在各场所发生上述情况；
- 对于网络覆盖与信号强度，影响客户满意度的主要因素有：“GPRS 总流量 (KB)”、“前 3 月 MOU”、“当月 ARPU”、“是否遇到过网络问题”、“mos 质差次数”、“脱网次

数”、“语音通话-时长（分钟）”、“手机没有信号”，以及在各场所发生上述情况；

- 对于**语音通话清晰度**，影响客户满意度的主要因素有：“GPRS 总流量（KB）”、“前 3 月 MOU”、“当月 ARPU”、“通话中有杂音、听不清、断断续续”、“mos 质差次数”、“语音通话-时长（分钟）”、“未接通掉话次数”，以及在各场所发生上述情况；
- 对于**语音通话稳定性**，影响客户满意度的主要因素有：“GPRS 总流量（KB）”、“当月 ARPU”、“语音通话-时长（分钟）”、“前 3 月 MOU”、“mos 质差次数”、“终端品牌”、“未接通掉话次数”、“客户星级标识”、“脱网次数”、“通话过程中突然中断”、“通话过程中一方听不见”、“手机没有信号”、“通话中有杂音、听不清、断断续续”、“有信号无法拨通”，以及在各场所发生上述情况；
- 对于**语音业务四项评分**，我们可以发现，影响客户满意度的主要因素大致相同，在不同的评分中，其量化出的影响程度不相同，但大致趋势一致。

通过上述模型的建立，我们得到了符合预期的效果。本文进行综合比较，可以更全面地分析出影响的主要因素，而对于量化值，我们以随机森林结果为准，这是由于该量化值更贴近于生活、可以更好地挖掘出其中隐含的因素，同时在后续模型的建立中，本文也会利用该结果进行分析，对于影响语音业务用户评分的因素量化最终结果见表 2。

表 2 语音业务总体以及四项评分各个指标影响程度量化结果

因素	语音通话整体满意度	网络覆盖与信号强度	语音通话清晰度	语音通话稳定性	语音业务总
GPRS 总流量 (KB)	0.1266	0.1151	0.1181	0.1263	0.1215
当月 ARPU	0.1154	0.1116	0.1000	0.1032	0.1111
是否遇到过网络问题	0.0922	0.1015	0.0953	0.1080	0.0995
前 3 月 MOU	0.0996	0.1232	0.1028	0.0993	0.0964
语音通话-时长（分钟）	0.0726	0.0762	0.0754	0.0680	0.0747
当月 MOU	0.0671	0.0715	0.0722	0.0689	0.0673
mos 质差次数	0.0645	0.0512	0.0617	0.0670	0.0604
脱网次数	0.0388	0.0406	0.0361	0.0444	0.0372
未接通掉话次数	0.0334	0.0319	0.0328	0.0302	0.0327
客户星级标识	0.0315	0.0233	0.0259	0.0243	0.0316
终端品牌	0.0356	0.0297	0.0368	0.0382	0.0310
4/5G 用户	0.0114	0.0137	0.0177	0.0119	0.0212
GPRS-国内漫游-流量 (KB)	0.0105	0.0112	0.0162	0.0136	0.0145
高铁	0.0107	0.0114	0.0141	0.0118	0.0124
农村	0.0117	0.0106	0.0105	0.0076	0.0124
地铁	0.0151	0.0126	0.0152	0.0129	0.0121
通话过程中突然中断	0.0081	0.0134	0.0096	0.0075	0.0117
外省流量占比	0.0133	0.0125	0.0110	0.0094	0.0115
是否 5G 网络客户	0.0116	0.0090	0.0128	0.0091	0.0112
商业街	0.0096	0.0080	0.0124	0.0071	0.0109
手机没有信号	0.0167	0.0076	0.0142	0.0177	0.0108
套外流量费 (元)	0.0063	0.0077	0.0089	0.0102	0.0106
通话过程中一方听不见	0.0112	0.0121	0.0171	0.0085	0.0093
外省语音占比	0.0023	0.0069	0.0051	0.0066	0.0091
居民小区	0.0115	0.0112	0.0091	0.0148	0.0086
办公室	0.0073	0.0145	0.0117	0.0085	0.0085
省际漫游-时长 (分钟)	0.0086	0.0114	0.0078	0.0060	0.0081
串线	0.0053	0.0036	0.0035	0.0057	0.0079
套外流量 (MB)	0.0071	0.0070	0.0037	0.0070	0.0071
通话中有杂音、听不清、断断续续	0.0105	0.0078	0.0067	0.0106	0.0070
其他，请注明	0.0040	0.0052	0.0050	0.0065	0.0058
有信号无法拨通	0.0076	0.0051	0.0087	0.0088	0.0058
其他，请注明.1	0.0036	0.0065	0.0040	0.0048	0.0051
是否关怀用户	0.0070	0.0041	0.0055	0.0059	0.0044
是否投诉	0.0028	0.0029	0.0047	0.0032	0.0041
高校	0.0039	0.0038	0.0049	0.0033	0.0031
前 3 月 ARPU	0.0044	0.0037	0.0022	0.0030	0.0031
是否 4G 网络客户 (本地剔除物联网)	0.0008	0.0005	0.0009	0.0004	0.0005

此外，为了更好地利用原数据集中用户描述文本数据，我们对其提取出高频词汇，并利用 Python 中 wordcloud 库绘制出语音业务用户描述高频词汇云图，见图 12。



图 12 语音业务用户描述高频词汇云图



图 13 上网业务用户描述高频词汇云图

根据图 12 结果，我们可以发现，对于原数据集中尚未提及的因素，用户进行描述，语音方面更多的问题体现在：信号、网络、断断续续、突然中断等；场所更多地出现在：车库、小区、地下室、家里、山区、电梯、高速等。

5.2.3 上网业务求解

对于上网业务的分析，与语音业务求解类似。首先我们绘制上网业务用户四项评分箱线图，如图 5 所示。再计算出皮尔逊相关系数，绘制其热力图，如图 27 所示，图中仅显示相关度排名前 10 的指标。

由图 27 可以直观发现，用户对于上网业务的满意度影响较大排名在前的因素依次为：“网络卡速度慢延时大上不了网总”、“出现问题场所或应用总”、“网络信号差/没有信号”、“手机上网速度慢”、“上网过程中网络时断时续或时快时慢”、“打开网页或 APP 图片慢”、“居民小区”、“显示有信号上不了网”、“办公室”、“看视频卡顿”。同时为了更好地分析影响上网业务的因素，我们与语音业务分析类似，绘制出上述 10 个指标与上网业务四项评分之间的 RidViz 图，由于篇幅原因，这里不再展示，读者可以在附件中查看，见图 33~图 36。

通过观察上述所得到的图，我们可以发现，用户对于上网业务的四项评分，分布规律也大致一致，且主要影响满意度的因素同样非常相似。因此，建立熵权法、灰色关联度分析法、随机森林模型在整体上得出主要因素并求得量化结果。结果见表 14、图 28。其中，图 28 中数据见表 3 最后一列及表 4 最后一列。

结合分析结果，我们可以发现，利用 EWM 和灰色关联度分析，量化的结果大致相似，间接说明结果的准确性。对于上网业务，影响用户评分的主要因素有：“套外流量”、“其他，请注明.5”（该指标对应的是游戏部分的注明）、“火山”（视频）、“全部都卡”（特征构造出的因素）、“全部游戏都卡顿”、“看视频卡顿”、“性别”、“客户星级标识”、“上网过程中网络时断时续或时快时慢”、“下载速度慢”，“当月 MOU”，以及各种游戏、APP、视频的卡顿。这些因素在日常生活中也明显影响到用户对于上网体验的满意度，可以依此验证上述分析结果的准确性。对于随机森林量化出的结果，我们可以发现，影响到用户体验的主要因素有：“当有 MOU”（当月的通话时长），“网络卡速度慢延时大上不了网总”（特征构造出的因素）、“终端品牌”、“客户星级标识”、“出现问题场所或应用总”（特征构造出的因素）、“性别”、“脱网次数”、“质差总”（特征构造出的因素），“是否 5G 网络客户”、“是否不限量套餐到达用户”、“显示有信号上不了网”、“上网过程中网络时断时续或时快时慢”，以及在各场所发生网络卡顿等问题。与 EWM 及灰色关联度分析比较，可以发现，有部分指标不一致，这可能是

由于在该模型中，部分因素对于模型的训练贡献较大，有部分潜在因素未被 EWM 及灰色关联度分析发现，但大致结果相同，符合预期效果。

其次，将语音业务的四项评分分别作为因变量，所有指标作为自变量，建立随机森林模型，绘制出其特征重要性图。由于图示与上文语音业务求解出的图为同一类型，且篇幅限制，读者可在附录中查看，见图 37~图 40。量化数据见表 3 及表 4。通过分析上述图表，我们可以得出以下结论：

- 对于**手机上网整体满意度**，影响客户满意度的主要因素有：“当月 MOU”、“出现问题场所或应用总”（特征构造出的因素），“终端品牌”、“脱网次数”、“网络卡速度慢延时大上不了网总”（特征构造出的因素），“性别”、“客户星级标识”、“质差总”（特征构造出的因素），“微信质差次数”、“居民小区”、“显示有信号上不了网”、“地铁”等；而类似于：“龙之谷”、“阴阳师”、“梦幻诛仙”等游戏或是 APP，影响度量化值趋近于 0，对于客户满意度影响较小；
- 对于**网络覆盖与信号强度**，影响客户满意度的主要因素有：“当月 MOU”、“网络卡速度慢延时大上不了网总”、“客户星级标识”、“终端品牌”、“出现问题场所或应用总”、“性别”、“是否 5G 网络客户”、“脱网次数”、“质差总”、“微信质差次数”、“套外流量费（元）”、“上网质差次数”、“农村”、“居民小区”等；而类似于：“龙之谷”、“阴阳师”、“梦幻诛仙”等游戏或是 APP，影响度量化值趋近于 0，对于客户满意度影响较小；
- 对于**手机上网速度**，影响客户满意度的主要因素有：“当月 MOU”、“出现问题场所或应用总”、“终端品牌”、“客户星级标识”、“质差总”、“是否 5G 网络客户”、“网络卡速

表 3 上网业务总体以及四项评分各个指标影响程度量化结果 [续表见表 4]

因素	手机上网整体满意度	网络覆盖与信号强度	手机上网速度	手机上网稳定性	上网业务总
当月 MOU	0.1563	0.2447	0.2530	0.2436	0.2278
网络卡速度慢延时大上不了网总	0.0899	0.1259	0.0292	0.1295	0.1244
终端品牌	0.0332	0.0484	0.0643	0.0594	0.0551
客户星级标识	0.0223	0.0596	0.0531	0.0549	0.0522
出现问题场所或应用总	0.4678	0.0402	0.1258	0.0407	0.0394
性别	0.0242	0.0440	0.0286	0.0315	0.0387
脱网次数	0.0383	0.0282	0.0284	0.0323	0.0318
质差总	0.0107	0.0289	0.0334	0.0330	0.0317
是否 5G 网络客户	0.0092	0.0301	0.0350	0.0267	0.0296
微信质差次数	0.0104	0.0251	0.0205	0.0210	0.0233
是否不限量套餐到达用户	0.0078	0.0183	0.0189	0.0172	0.0226
套外流量费（元）	0.0089	0.0206	0.0249	0.0213	0.0221
上网质差次数	0.0058	0.0185	0.0208	0.0183	0.0215
农村	0.0067	0.0154	0.0150	0.0154	0.0163
显示有信号上不了网	0.0105	0.0131	0.0146	0.0150	0.0161
居民小区	0.0138	0.0185	0.0185	0.0170	0.0161
地铁	0.0100	0.0141	0.0145	0.0169	0.0151
高铁	0.0016	0.0138	0.0123	0.0123	0.0150
商业街	0.0090	0.0124	0.0127	0.0149	0.0139
上网过程中网络时断时续或时快时慢	0.0038	0.0102	0.0107	0.0103	0.0137
网络信号差/没有信号	0.0046	0.0109	0.0138	0.0139	0.0133
办公室	0.0091	0.0136	0.0156	0.0144	0.0120
套外流量（MB）	0.0029	0.0107	0.0089	0.0134	0.0119
手机支付较慢	0.0000	0.0073	0.0069	0.0058	0.0086
其他，请注明	0.0031	0.0123	0.0090	0.0100	0.0079
下载速度慢	0.0023	0.0073	0.0071	0.0067	0.0074
拼多多	0.0000	0.0035	0.0033	0.0038	0.0063
打游戏延时大	0.0094	0.0044	0.0033	0.0025	0.0063
抖音	0.0015	0.0081	0.0072	0.0051	0.0059
百度	0.0000	0.0064	0.0048	0.0061	0.0059
其他，请注明.1	0.0033	0.0041	0.0028	0.0056	0.0059
看视频卡顿	0.0020	0.0056	0.0054	0.0055	0.0056
微信	0.0012	0.0054	0.0042	0.0051	0.0052

表 4 上网业务总体以及四项评分各个指标影响程度量化结果 [表 3续表]

因素	手机上网整体满意度	网络覆盖与信号强度	手机上网速度	手机上网稳定性	上网业务总
高校	0.0000	0.0046	0.0063	0.0052	0.0051
腾讯视频	0.0022	0.0042	0.0044	0.0040	0.0049
打开网页或 APP 图片慢	0.0054	0.0042	0.0045	0.0042	0.0044
全部网页或 APP 都慢	0.0023	0.0031	0.0030	0.0028	0.0044
京东	0.0000	0.0064	0.0041	0.0060	0.0043
快手	0.0000	0.0026	0.0058	0.0042	0.0041
淘宝	0.0022	0.0044	0.0043	0.0054	0.0040
手机上网速度慢	0.0022	0.0031	0.0032	0.0034	0.0038
今日头条	0.0000	0.0047	0.0044	0.0048	0.0035
王者荣耀	0.0000	0.0030	0.0021	0.0020	0.0034
新浪微博	0.0000	0.0039	0.0033	0.0041	0.0029
爱奇艺	0.0016	0.0040	0.0049	0.0035	0.0027
优酷	0.0025	0.0021	0.0014	0.0026	0.0026
芒果 TV	0.0000	0.0014	0.0016	0.0026	0.0026
全部都卡顿	0.0020	0.0033	0.0040	0.0029	0.0026
手机 QQ	0.0000	0.0027	0.0040	0.0023	0.0026
其他, 请注明.2	0.0000	0.0017	0.0017	0.0010	0.0019
搜狐视频	0.0000	0.0011	0.0006	0.0013	0.0018
咪咕视频	0.0000	0.0016	0.0009	0.0019	0.0017
其他, 请注明.3	0.0000	0.0016	0.0030	0.0024	0.0015
其他, 请注明.5	0.0000	0.0011	0.0016	0.0004	0.0013
全部游戏都卡顿	0.0000	0.0010	0.0006	0.0005	0.0013
火山	0.0000	0.0004	0.0003	0.0000	0.0011
和平精英	0.0000	0.0015	0.0014	0.0011	0.0008
欢乐斗地主	0.0000	0.0006	0.0003	0.0010	0.0008
其他, 请注明.4	0.0000	0.0012	0.0002	0.0004	0.0007
梦幻西游	0.0000	0.0000	0.0009	0.0000	0.0003
穿越火线	0.0000	0.0005	0.0003	0.0004	0.0002
炉石传说	0.0000	0.0000	0.0007	0.0003	0.0001
部落冲突	0.0000	0.0004	0.0000	0.0002	0.0001
梦幻诛仙	0.0000	0.0000	0.0000	0.0000	0.0001
阴阳师	0.0000	0.0000	0.0000	0.0000	0.0000
龙之谷	0.0000	0.0000	0.0000	0.0000	0.0000

度慢延时大上不了网总”、“性别”、“脱网次数”、“套外流量费（元）”、“上网质差次数”等；而类似于：“龙之谷”、“阴阳师”、“梦幻诛仙”等游戏或是 APP，影响度量化值趋近于 0，对于客户满意度影响较小；

- 对于**手机上网稳定性**，影响客户满意度的主要因素有：“当月 MOU”、“网络卡速度慢延时大上不了网总”、“终端品牌”、“客户星级标识”、“出现问题场所或应用总”、“脱网次数”、“质差总”、“性别”等；而类似于：“龙之谷”、“阴阳师”、“梦幻诛仙”等游戏或是 APP，影响度量化值趋近于 0，对于客户满意度影响较小；
- 对于**上网业务四项评分**，我们可以发现，影响客户满意度的主要因素大致相同，在不同的评分中，其量化出的影响程度不相同，但大致趋势一致。

通过上述模型的建立，得到的效果符合预期。本文对 EWM、灰色关联度分析、随机森林的综合比较分析，可以更全面地分析出影响的主要因素。而对于量化值，我们以随机森林结果为准，这是由于，在上述的分析中，我们发现随机森林模型可以更好地解释主要影响因素，并且可以更好地利用数据挖掘出潜在信息，大大提高决策的高效性，准确性及可靠性，同时在后续模型的建立中，本文也会利用该结果进行分析，对于上网业务用户评分的因素量化最终结果见表 3 及表 4。

此外，为了更好地分析影响语音业务的因素，我们对用户描述列提取出高频词汇，并绘制出上网业务用户描述高频词汇云图，见图 13。根据图 13 结果，我们可以发现，对于原数据集中尚未提及的因素，用户进行描述，上网业务部分问题还体现在：信号、网络、流量、网速、经常没有、打不卡 APP、网页、在小区、电梯、医院、公交、家里等多个场所出现一系

列问题，提及的关键词还有：5G、速度、经常、断网、单位、微信、哔哩哔哩、小红书、突然、上网、信号不好等，这一系列关键词，都能有效反映出用户在实际体验中遇到的问题。

5.3 问题二模型的建立与求解

对于问题二，我们在问题一的基础上，对附件指标采用**主成分分析法**，对结果进行**累计方差解释**，绘制累计解释方差图，根据问题一结果及累计解释方差图。再建立多种用于**分类**的模型，包括**随机森林**、**XGBoost**、**KNN**、**SVM**、**LightGBM**、**改进的逻辑回归**。并对上述模型进行**超参数调优**，得到其最优模型。之后综合各个模型的表现，如准确率、平均绝对误差、均方误差等，有目的地进行合理组合，利用**Stacking**方法，进行集成学习，得到各评分预测模型，并且根据多方面指标对构建的集成学习进行评价分析，且解释预测的合理性。

5.3.1 累计解释方差

这里我们使用**主成分分析**（Principal Component Analysis, PCA）对附件 1 与附件 3 影响因素进行累计解释方差分析。其可对较高维度数据进行降维处理，并且保证原数据集信息丢失量最小化^[11]，从而方便本文后续模型的建立。对于语音及上网业务，我们分别绘制出其各指标的累计解释方差图，如图 14、图 15 所示。通过分析可知：

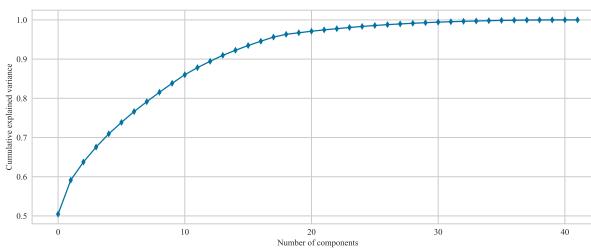


图 14 语音业务指标累计解释方差

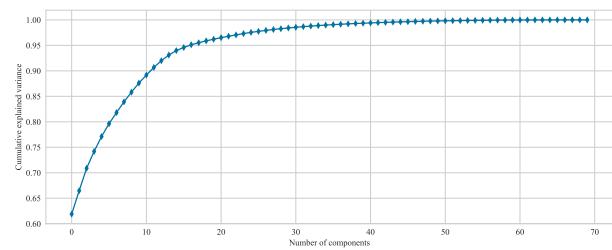


图 15 上网业务指标累计解释方差

- 对于**语音业务**：指标个数少于 20 个时，累计解释方差增幅较大，且约可累计解释 96% 以上方差，在 20 个指标后，增幅较小，当指标个数为 34 个时，可解释近于 100% 的方差。通过问题一的分析，我们可以将在其中量化结果中选择平均影响度量化值小于 0.0050 的进行特征剔除，包括：“是否关怀用户”、“是否投诉”、“高校”、“前 3 月 ARPU”、“是否 4G 网络客户（本地剔除物联网）”；
- 对于**上网业务**：指标个数少于 20 个时，累计解释方差增幅较大，且约可累计解释 96% 以上方差，在 20 个指标后，增幅较小，当指标个数为 50 个时，可解释近于 100% 的方差。通过问题一的分析，我们可以将在其中量化结果中选择平均影响度量化值小于 0.0010 的进行特征剔除，包括：“和平精英”、“欢乐斗地主”、“其他，请注明.4”（视频类 APP 的特殊注明列）、“梦幻西游”、“穿越火线”、“炉石传说”、“部落冲突”、“梦幻诛仙”、“阴阳师”、“龙之谷”；
- 对于**剔除的因素的合理性分析**：综合问题分析，我们发现：这些因素在业务上的解释性较弱，且在量化结果中的影响度较小，且在 PCA 分析中，这些因素的累计解释方差较

小，因此，我们认为这些因素对于业务的影响较小，故将其剔除分析。同时，我们对剔除前后的模型的准确率进行分析，发现，对于语音及上网业务的分析，模型准确率均有一定提升，介于 $1.24\% \sim 2.74\%$ ；

- 而对于下文**各个评分模型的建立**：我们会逐一分析，进行特征剔除，提升模型的准确性，以及模型的各项指标。上述的分析，首先在整体上确定合理方向。

5.3.2 多种多分类模型的建立

- **极端梯度提升 (eXtreme Gradient Boosting, XGBoost)**。XGBoost 算法是一种基于树模型的优化模型，其将弱分类器组合，训练出一个较强的分类器。该算法通过多次迭代，生成一个新的树模型用于优化前一个树模型，随着迭代次数的增多，该模型的预测精度也会相应提高^[8]。

记通过数据处理后的数据集特征为 $R(x_{ij})_{m \times n}$ ，表示其包含 m 个用户， n 个特征，在训练中形成的 CART 树的集合记为 $F = \{f(x) = w_{q(x)}, q : \mathbf{R}^n \rightarrow T, w \in \mathbf{R}^T\}$ ，其中 q 为树模型的叶节点决策规划， T 为某一树模型叶节点数量， w 为叶节点对应的得分^[9]。对于预测的 y 值，其计算公式为

$$\hat{y} = \varphi(x_i) = \sum_{k=1}^K f_k(x_i) \quad (17)$$

XGBoost 算法在每一次迭代过程中会保存前面所学习的模型，会将这些模型加入到新一轮迭代过程中，因此我们记第 i 个模型为预测结果为

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i) \quad (18)$$

XGBoost 算法的目标函数计算公式如下

$$L^{(t)} = \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 + \text{const} \quad (19)$$

上述公式中， l 为模型误差损失，描述在该模型下预测值与实际值之间的出差异损失， Ω 为模型叶节点的正则项惩罚系数， γ 与 λ 为模型的超参数^[9]。通常情况下，我们难以用枚举法得到在模型中所训练出来的树结构，因此这里采用贪婪算法，从单叶子节点开始，通过迭代方法，将其加入到树结构中，从而得到最优解，其计算公式^[10]如下

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \quad (20)$$

其中 $I_j = \{i | q(x_i) = j\}$ 为叶节点 j 上的样本集合^[9]，且有

$$g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right) \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 l\left(y_i, \hat{y}_i^{(t-1)}\right) \quad (21)$$

通过上述分析，我们可以得到 XGBoost 算法简图，如图 16 所示。

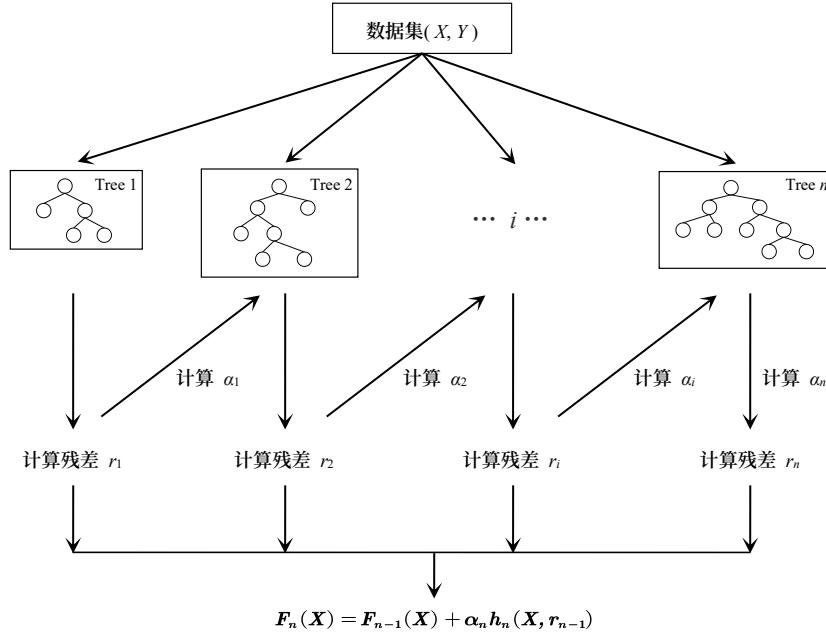


图 16 XGBoost 算法简图

- **K-近邻 (K Nearest Neighbor, KNN)**。KNN 算法的主要思想为：在出现新样本时从现有的训练数据中找到与其相对应的最接近的 K 个样本，并根据最相似的类别出现的样本进行分类。基于多数 K 个样本所属的类别来分辨待分类的数据集所属的类别^[12]。接近度由两点之间的距离函数给出属性空间中的点决定。距离函数通常使用两个点之间的标准欧几里得距离。欧氏距离的计算公式如下

$$d(X, Y) = \sqrt{\sum_{i=1}^n (X_i - Y_i)^2} \quad (22)$$

其中 $X = (x_1, x_2, \dots, x_m)^T$ 和 $Y = (y_1, y_2, \dots, y_m)^T$ 表示两个样本列数据， m 为样本数量。

- **支持向量机 (Support Vector Machine, SVM)**。SVM 建立在结构风险最小原理及 Vapnik-Chervonenkis 理论基础之上^[13]，以有限的数据信息，在数据样本中找出合适区分类别的决策分界面，且保证边界点与分界面尽可能远，即需要再找出合适的边界分界面，该算法示意图如图 17 所示。而由于 SVM 多应用于解决二分类问题，且我们需要建立多分类模型，因此需要对其进行相应的改进。本文采用 OVR (One Versus Rest) 方法，将该问题改进为多个二分类问题^[13]。在模型的训练时，任意将某一类别记为一类，其余类别记为另一类别，依次下去，建立出多分类的 SVM 模型。而对于核函数的选择，本文选择高斯核函数进行求解，其定义公式如下

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) = \exp(-\gamma \|x_i - x_j\|^2) \quad (23)$$

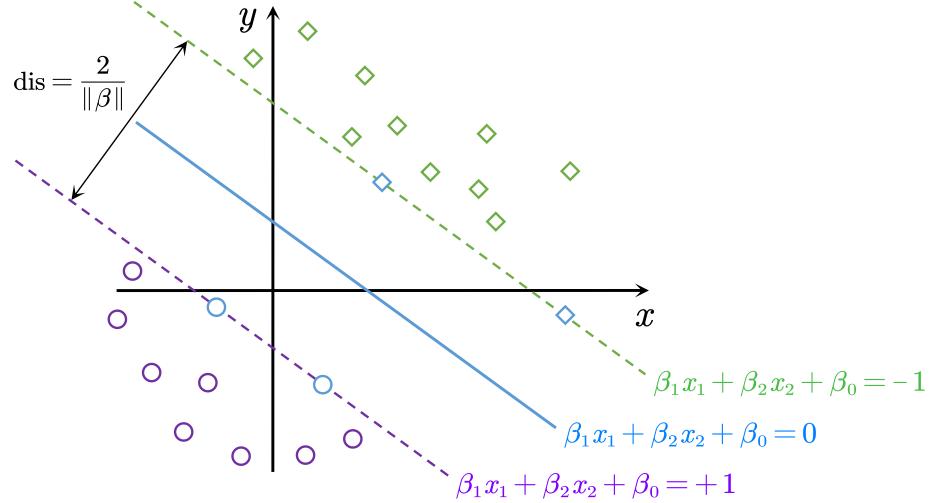


图 17 SVM 示意图

对于高斯核函数，其可以反映出样本两点之间的相似度大小。当 σ 确定后，若两点之间距离越小，则相似度趋近于 1；若距离越大，则相似度趋近于 0。

- **LightGBM (Light Gradient Boosting Machine)**。LightGBM 模型是基于决策树算法构建的一种高效的机器学习算法^[14]。其为 XGBoost、直方图算法 (Histogram)、基于梯度的单边采样 (GOSS) 算法以及互斥特征捆绑 (EFB) 算法的结合的一种算法。
- **多分类逻辑回归 (Multinomial Logistic Regression)**。多分类逻辑回归是基于逻辑回归 (Logistic Regression) 进行学习的分类模型。对于逻辑回归模型，其属于分类模型，多用于二分类问题。若数据集为 $(\mathbf{A}, \mathbf{B}) = ((\mathbf{a}_1, b_1), (\mathbf{a}_2, b_2), \dots, (\mathbf{a}_m, b_m))^T$ ，其中 $\mathbf{a}_i = (a_i^1, a_i^2, \dots, a_i^j)$ ， a_i^j 为样本 \mathbf{a}_i 的第 j 个特征， \mathbf{B} 为因变量标签矩阵，该模型使用 Sigmoid 函数，同时构建样本 \mathbf{a}_i 所属类别的概率，对于标签为 1 的结果，其概率可写为

$$P(b_i = 1 | \mathbf{a}_i, \boldsymbol{\omega}) = \frac{1}{1 + e^{-\mathbf{a}_i b_i \boldsymbol{\omega}^T}} \quad (24)$$

其中 $\boldsymbol{\omega} = (\omega^0, \omega^1, \dots, \omega^n)^T$ 为权重向量，即为优化模型的超参数。逻辑回归中利用损失函数来评估模型的预测结果与实际值之间的误差，其计算公式如下

$$L(\mathbf{A}, \mathbf{B}, \boldsymbol{\omega}) = \frac{1}{m} \sum_{i=1}^m \log \left(1 + e^{-\mathbf{a}_i b_i \boldsymbol{\omega}^T} \right) \quad (25)$$

而对于 $\boldsymbol{\omega}$ ，常采用梯度下降法来获得模型参数的最优解，其通过

$$\boldsymbol{\omega}^{\alpha+1} = \boldsymbol{\omega}^\alpha - \frac{\gamma}{m} \sum_{i=1}^m \left(\frac{1}{1 + e^{-\mathbf{a}_i b_i \boldsymbol{\omega}^T}} - 1 \right) \mathbf{a}_i b_i \quad (26)$$

进行迭代更新，其中 γ 为模型的学习率，当 $|\boldsymbol{\omega}^\alpha - \boldsymbol{\omega}^{\alpha+1}| < \eta$ 或达到最大迭代次数时，停止训练，输出最终模型，其中 η 为人为给定的阈值^[15]。

5.3.3 Stacking 集成学习原理解释

Stacking 集成学习是通过建立多个机器学习模型，将其有目的地进行合理组合，从各模型中学到优点，有利于模型的效果的提升。其基本过程为，首先将已经经过处理的原数据集划分成若干个子集数据，在第一层建立多个模型的融合模型，输入数据，并采用五折交叉验证，获得每个模型的对于因变量标签的预测结果；之后第一层的输出结果作为第二层较弱分类模型的输入数据，第二层单个模型进行训练学习，得到最终预测结果^[16]。算法示意图如图 18 所示，算法伪代码如 Algorithm 2 所示。

Algorithm 2: Stacking 集成学习

Input: 训练集 \mathcal{D}

第一层学习模型 $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_n$

第二层学习模型 \mathcal{S}

```

1 for  $t = 1, 2, 3, \dots, n$  do
2   |  $h_n = \mathcal{F}_n(\mathcal{D})$ 
3 end
4  $\mathcal{D}' = \emptyset$ 
5 for  $i = 1, 2, \dots, m$  do
6   | for  $t = 1, 2, \dots, n$  do
7     |   |  $z_{in} = h_n(x_i)$ 
8   | end
9   |  $\mathcal{D}' = \mathcal{D}' \cup ((z_{i1}, z_{i2}, \dots, z_{in}), y_i)$ 
10 end
11  $h' = \mathcal{S}(\mathcal{D}')$ 

```

Output: $\mathcal{H}(x) = h'(h_1(x), h_2(x), \dots, h_n(x))$

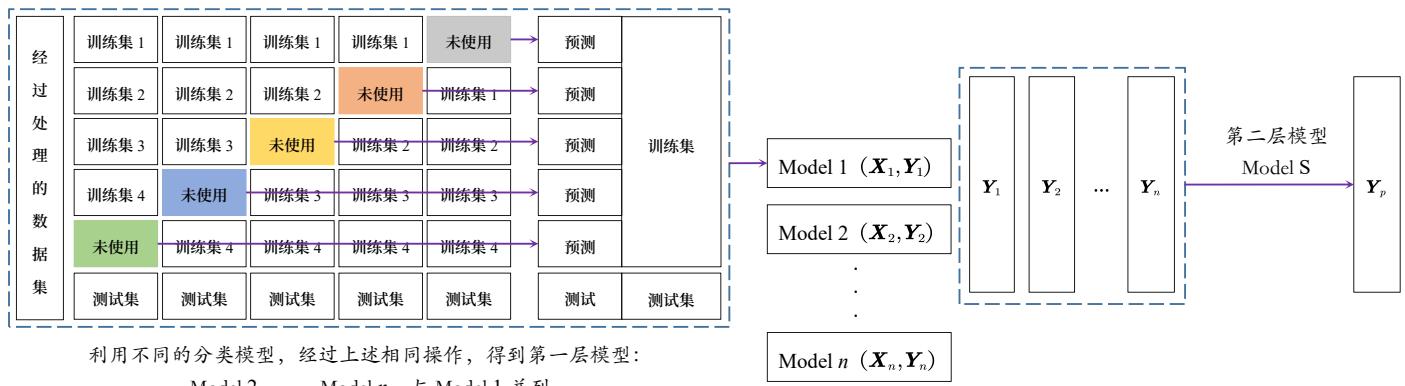


图 18 Stacking 集成学习示意图

5.3.4 模型的超参数调优

为了更好地预测用户评分，我们在建立上述模型的基础上，对模型进行超参数调优，以提高模型的预测准确性，减小平均绝对误差以及均方误差。

- **随机森林调优：**这边我们以语音业务中“网络覆盖与信号强度”评分为研究对象，进行调参，其他评分下建立的模型调参与此大致相同。随机森林最需要调节的参数为树的棵数（n_estimators），而森林深度、信息计算方法、叶节点等为辅助调参参数。首先我

们绘制模型“Acc-n_estimators”图像，见图 19，图像纵坐标为模型准确率，横坐标为 n_estimators 值。通过观察可以发现，该模型对于该数据在 n_estimators=161 附近时，

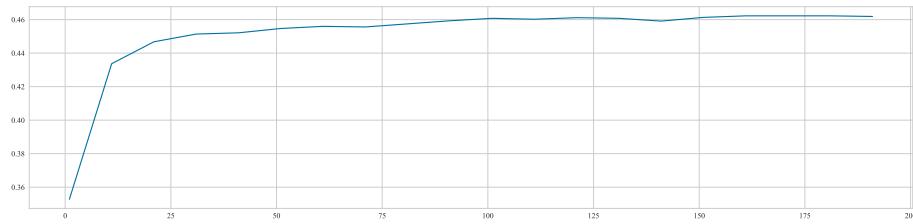


图 19 随机森林 n_estimators 调优图像，第一次

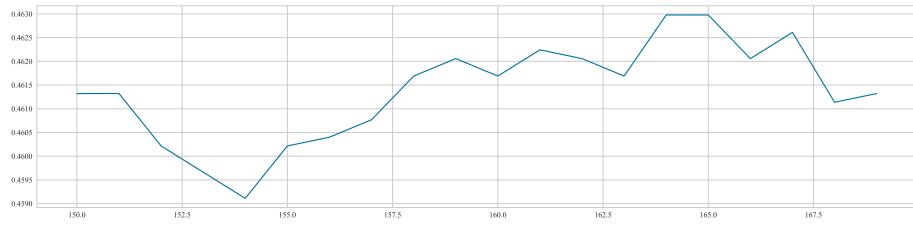


图 20 随机森林 n_estimators 调优图像，第二次

准确率最高，故我们选择 n_estimators=161 作为参数值。根据上述分析结果，进一步细化随机森林学习曲线，我们将在 150 ~ 170 之间寻找最佳值。我们绘制出 n_estimators 在 150 ~ 170 之间的学习曲线，见图 20。通过观察该图，我们发现对于该数据集，该模型的 n_estimators 最佳值为 164，故我们选择 n_estimators=164 作为参数值。

之后我们对参数 max_features, min_samples_leaf, criterion, max_depth 利用 Python 的 sklearn.model_selection 中的 GridSearchCV 模块进行调节，最终得到在该评分下，随机森林模型最优参数，结果见表 5。

表 5 语音业务-网络覆盖与信号强度评分下随机森林模型最优参数

参数	最优选取
n_estimators	164
max_features	log2
min_samples_leaf	8
criterion	gini
max_depth	19

通过计算，该模型调参前准确率为 45.67%，调节最优参数后，准确率上升至 51.01%，较原先提升了 11.69%，可以发现，调参效果良好。

- **XGBoost、KNN、SVM、LightGBM、多分类逻辑回归调优：**对于这些模型，我们采用网格搜索的方法进行调参，设置好需要调参的对象及范围，利用 Python 的 sklearn 库中的 model_selection.GridSearchCV 模块，进行调优。具体代码可以参看附录 [D]。通过分析，可以发现，各个模型在调参后，准确率均有一定提升，较原模型准确率的提升部分模型可以达到 14.25%，提升的百分点约为 6.28%，调参效果良好，符合预期效果。

5.3.5 特征选择

为了更好地预测用户评分，尽可能地提升模型的准确率、泛化能力，以及降低其预测的平均绝对误差、均方误差，我们在问题一的基础上，对特征进行了筛选，并且结合 XGBoost 模型的特牲重要性可视化进行综合分析。对语言及上网业务共计八项评分，每一个进行了综合特征筛选。这里我们以语音业务-语音通话整体满意度评分为例，进行特征选择的分析。根据 XGBoost 模型，我们绘制出该评分下影响因素的重要性可视化图，如图 21 所示。³

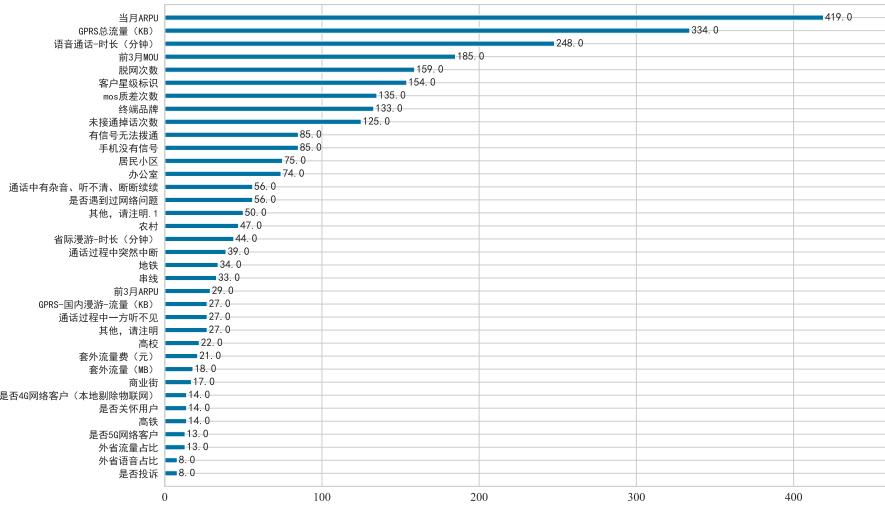


图 21 语音通话整体满意度各项指标重要程度，XGBoost

观察该图，我们可以发现，对于语音业务的语音通话整体满意度评分的影响因素，XGBoost 模型计算的结果与随机森林模型大致相似。我们综合表 2 以及图 21 结果，选择两个模型分析出的重要性高的特征，且剔除随机森林模型量化结果在 0.0010 以下，以及 XGBoost 模型量化结果在 10 以下的特征，它们为：“是否 4G 网络客户（本地剔除物联网）”、“外省语音占比”、“是否投诉”三项，并在这基础上建立后续模型，进行分析、预测。

而对于语音及上网业务的其余评分，思路与上述一致，由于篇幅原因，这里不再赘述。对于各评分的 XGBoost 特征重要性可视图，读者可以在附件中查看，见图 41~图 47。

5.3.6 模型的选择及集成

对八项评分，我们均建立随机森林、XGBoost、KNN、SVM、LightGBM、多分类逻辑回归模型，且进行参数调优，得到每一个最优模型。并且计算每一个模型的准确率 (Accuracy)、平均绝对误差 (Mean Absolute Error, MAE)、均方误差 (Mean Square Error, MSE)。对于多分类模型，该三项指标，计算公式如下：

- **准确率**

$$\text{Accuracy} = \frac{N_{\text{TruePredict}}}{N_{\text{Sample}}} \quad (27)$$

其中， $N_{\text{TruePredict}}$ 为预测正确的样本数， N_{Sample} 为被预测的样本总数；

³其中由于上网业务的特征较多，对于其下的四项评分，我们仅选取了 20 个特征进行了可视化，但对评分特征选择的分析，本文是对所有特征进行选择的。

- 平均绝对误差

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (28)$$

其中, y_i 为实际值, \hat{y}_i 为预测值;

- 均方误差

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (29)$$

利用上述公式, 我们可以计算出每个模型这三项指标, 结果见表 6、表 7 以及表 8。

表 6 随机森林、XGBoost 各评分三项指标结果

评分项目	随机森林			XGBoost		
	准确率	平均绝对误差	均方误差	准确率	平均绝对误差	均方误差
语音通话整体满意度	0.5829	1.2910	6.5378	0.5949	1.2320	6.0331
网络覆盖与信号强度	0.5101	1.5304	7.6077	0.4936	1.4926	7.0562
语音通话清晰度	0.5424	1.2864	6.0691	0.5331	1.3039	6.0976
语音通话稳定性	0.5184	1.3996	6.5433	0.5184	1.4282	6.7928
手机上网整体满意度	0.4459	1.7692	8.6752	0.4359	1.7222	8.2265
网络覆盖与信号强度	0.3903	1.7322	7.5214	0.3775	1.7863	7.8775
手机上网速度	0.3775	1.7892	7.7664	0.3533	1.7222	6.9587
手机上网稳定性	0.3803	1.8348	8.0912	0.3889	1.7792	7.8846

表 7 KNN、SVM 各评分三项指标结果

评分项目	KNN			SVM		
	准确率	平均绝对误差	均方误差	准确率	平均绝对误差	均方误差
语音通话整体满意度	0.5893	1.2680	6.5055	0.5884	1.3582	7.3103
网络覆盖与信号强度	0.5009	1.5506	7.6924	0.4991	1.5948	8.0571
语音通话清晰度	0.5543	1.3002	6.2063	0.5543	1.3932	7.1133
语音通话稳定性	0.5239	1.4162	6.7385	0.5267	1.4678	7.2652
手机上网整体满意度	0.4217	1.8148	8.6268	0.4330	1.8575	9.3846
网络覆盖与信号强度	0.3946	1.7792	8.0271	0.3960	1.8504	8.6966
手机上网速度	0.3818	1.7806	7.7094	0.3732	1.8519	8.5071
手机上网稳定性	0.3832	1.7977	8.1054	0.3818	1.9259	9.0057

表 8 LightGBM、多分类逻辑回归各评分三项指标结果

评分项目	LightGBM			多分类逻辑回归		
	准确率	平均绝对误差	均方误差	准确率	平均绝对误差	均方误差
语音通话整体满意度	0.5737	1.2366	5.8131	0.5829	1.2330	6.0230
网络覆盖与信号强度	0.4899	1.5037	7.1298	0.5000	1.4797	7.0101
语音通话清晰度	0.5396	1.2606	5.6860	0.5451	1.2845	5.9843
语音通话稳定性	0.5110	1.3517	6.1455	0.5285	1.3923	6.5727
手机上网整体满意度	0.4444	1.7123	8.1852	0.4302	1.7821	8.7365
网络覆盖与信号强度	0.3846	1.7393	7.4772	0.3718	1.8547	8.4530
手机上网速度	0.3704	1.7222	7.2464	0.3761	1.8219	8.0840
手机上网稳定性	0.3675	1.8433	8.1140	0.3818	1.8305	8.2550

通过分析上述表格, 我们综合模型的准确率、平均绝对误差、均方误差的表现, 对每一项评分选择基模型以及第二层模型, 选择好的组合见表 9。

表 9 各个评分预测 Stacking 集成学习模型的建立

评分项目	基模型					第二层模型	
	XGBoost	KNN	SVM	LightGBM	LR	RF	
语音通话整体满意度	RF	XGBoost	KNN	SVM	LR	LightGBM	
网络覆盖与信号强度	XGBoost	KNN	SVM	LightGBM	LR	RF	
语音通话清晰度	RF	SVM	KNN	LightGBM	LR	XGBoost	
语音通话稳定性	RF	XGBoost	KNN	LightGBM	LR	SVM	
手机上网整体满意度	RF	XGBoost	KNN	LightGBM	LR	SVM	
网络覆盖与信号强度	RF	XGBoost	KNN	LightGBM	LR	SVM	
手机上网速度	RF	XGBoost	KNN	LightGBM	LR	SVM	
手机上网稳定性	RF	XGBoost	KNN	LightGBM	LR	SVM	

依据表 9 的融合组合，我们最终建立对于语音及上网业务每一项评分的用户评分预测模型，用于对附件 2 以及附件 4 中的用户评分进行预测。

5.3.7 客户评分预测

在“5.1 数据的准备”中，我们提到对学习数据与预测数据进行一致化，这是为了统一预测的自变量，避免不同的自变量的混乱，导致预测错误。经过对预测数据集的处理，利用上述已建立好的八个模型，对每一位用户的评分进行预测。这里我们需要注意的是：首先，要保证传入模型的变量要与训练时传入的指标一致；其次，在上文中我们提到我们选用多分类解决，而需要对评分进行标签编码，即将原评分 $y \in [1, 10]$ 映射至新评分标签 $y' \in [0, 9]$ ，即有关系式 $y' = y - 1$ ，而对于新数据的预测，我们要在模型的每个预测结果上加 1，避免预测结果出错。由于被预测用户过多，我们将不在论文中展示，而将以文件形式保存至“result.xlsx”。

5.3.8 模型预测结果合理性分析

本文对于数据集充分分析，多方面考虑，建立多模型调参融合的 Stacking 集成学习模型，且对模型训练采用五折交叉验证，保证模型的稳健性。对于语音业务数据的处理，我们将数据集划分训练集与测试集，比例为 8:2；对于上网业务数据的处理，我们将数据集划分训练集与测试集，比例为 9:1。对于两项业务这样处理有以下几点原因：

- 为验证模型的效果、分析模型的合理性、对模型参数进行调优、有监督地在数据上进行学习，更好地分析模型对于重要特征的选择，进行特征选择等，因此我们需要对数据集划分训练集与测试集；
- 对于语音业务，我们划分训练集与测试集比例为 8:2，这是由于我们观察到，语音业务的数据分布较优，且需要学习的特征相对于上网业务较少，若过分提高该比例，模型可能会产生过拟合的情况，无法对未知数据进行高效分析，泛化能力差；
- 对于上网业务，我们划分训练集与测试集比例为 9:1，这是由于我们观察到，上网业务需要学习的特征较多，若训练集样本过少，可能导致训练的模型发生欠拟合的情况，未能更好地学习到数据的内在规律，导致模型的多项指标未达到期望值。

此外，由于用户评分性质，我们选择多分类模型解决，为了更好地学习、预测，我们建立多个分类模型，且对各模型进行超参数的调节，在一定程度上提高模型的预测精度，分析

各个影响因素的特征重要性。此外利用 Stacking，对多模型进行集成学习，使得最终模型可以学习到各个模型的特性，且在一定程度上提升模型的泛化能力。

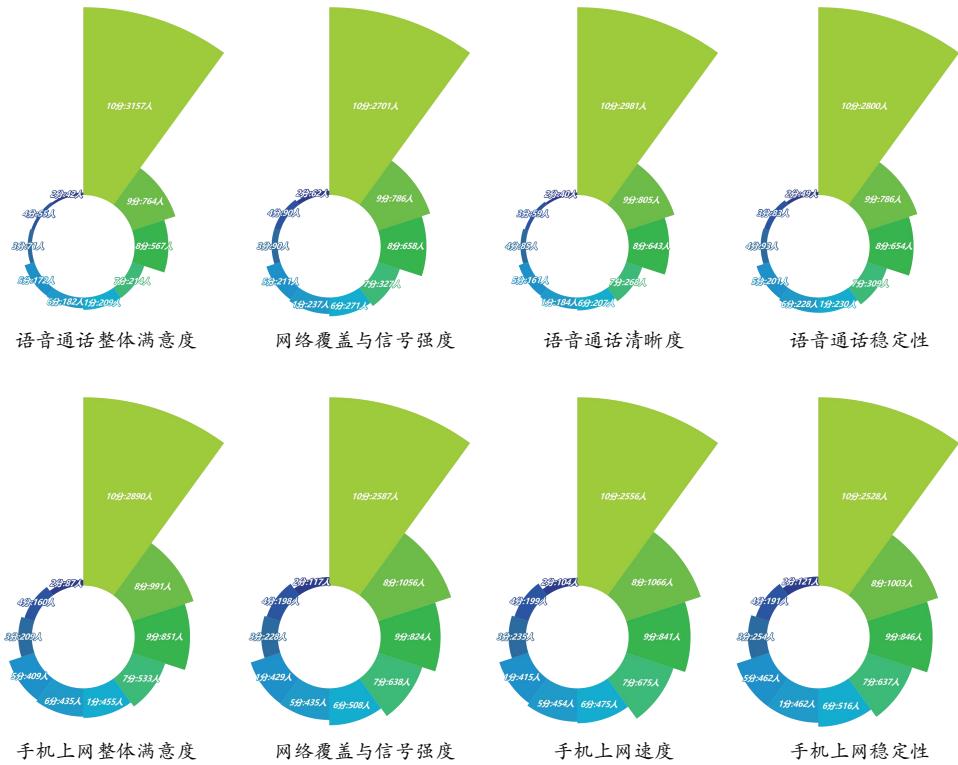


图 22 [附件 1、附件 2] 各类评分人数南丁格尔玫瑰图

同时我们计算出各个最终模型（即八个 Stacking 集成学习后的模型）的准确率。然而观察到，各评分的分类不平衡，即某些评分的样本量要远远大于某些评分，见图 22。因此，仅由准确率评估模型效果过于片面，且可能得到错误结论，因此我们还计算出各模型的平均绝对误差、均方误差，且准确率取五折交叉验证平均值，结果见表 10。

表 10 各评分预测模型效果

模型	五折交叉验证平均准确率	平均绝对误差	均方误差
模型一 [预测语音业务, 语音通话整体满意度]	0.5773	1.2937	6.3877
模型二 [预测语音业务, 网络覆盖与信号强度]	0.4880	1.5387	7.3416
模型三 [预测语音业务, 语音通话清晰度]	0.5405	1.3527	6.4540
模型四 [预测语音业务, 语音通话稳定性]	0.5212	1.3913	6.3748
模型五 [预测上网业务, 手机上网整体满意度]	0.4359	1.7094	8.0684
模型六 [预测上网业务, 网络覆盖与信号强度]	0.3803	1.7650	7.7764
模型七 [预测上网业务, 手机上网速度]	0.3761	1.7208	7.3134
模型八 [预测上网业务, 手机上网稳定性]	0.3875	1.8276	8.0897

我们将表 6、表 7、表 8 与表 10 进行对比分析，可以发现，Stacking 模型较其组成的模型各结果均有一定优化，且预测结果有良好的稳健性。Stacking 学习其组成模型的优势，更大范围地高效利用数据集间隐藏的关系，具有良好的预测效果。

为了更好地评估模型，对预测结果的合理性进行分析，我们绘制出各个模型的混淆矩阵热力图、分类报告、ROC/AUC 曲线。这里由于篇幅原因，我们仅展示“预测语音业务-语音通话整体满意度”三幅模型效果可视化图形，其余模型的分析与其一致。对于其余模型的可视化图形，读者可在附录中查看。其余七个模型的混淆矩阵热力图见图 48~图 54；分类报告见图 55~图 61；ROC/AUC 曲线见图 62~图 68。

- **混淆矩阵热力图**。该可视化图形的每一行表示样本标签的实际类别，在本题中表示用户评分的实际值⁴，而每一行表示样本标签的预测类别，在本题中表示用户评分的预测值。因此该图示的主对角线数据之和即为模型预测准确的样本数。对于多分类模型，我们可以随机指定一类为正类，而其余就为对应的负类。这里我们需要引入四项值，分别为 TP 、 FN 、 FP 、 TN ，其中 T 为 True，F 为 False，这两个字母表示预测值与实际值是否相同；P 为 Positive，N 为 Negative，这两个字母表示预测出的是属于正类（阳性）还是负类（阴性）。而混淆矩阵热力图即为这些值组成，该图示可以直观地观察到预测准确与错误的情况，以及模型对于每一类别的区分程度。模型一的混淆矩阵热力图见图 23。

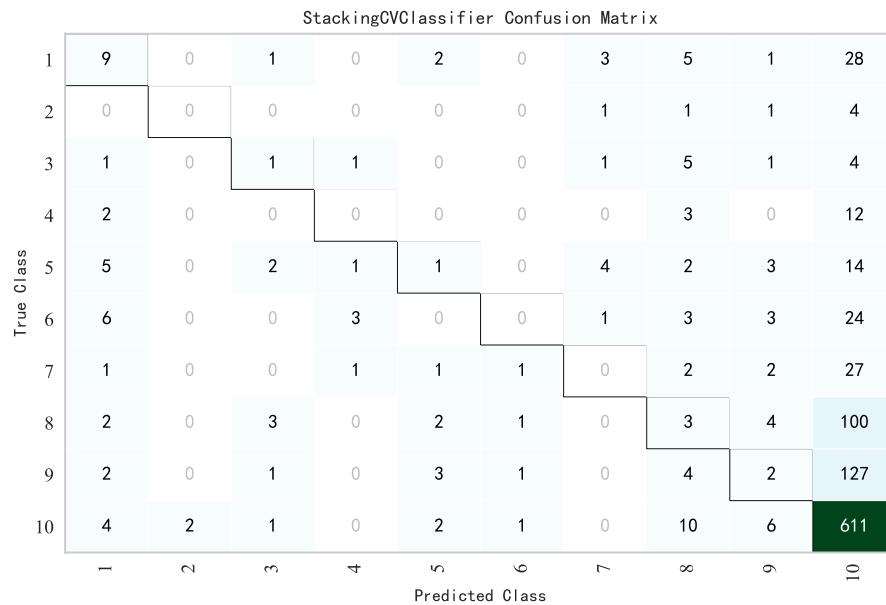


图 23 模型一混淆矩阵热力图 [语音业务-语音通话整体满意度]

观察该图，我们可以发现，该模型对于预测用户评分具有较好的效果，主对角线附近元素较多，说明模型预测正确的误差较小，预测得分与用户实际评分比较接近，可以较好预测用户评分。

- **分类报告**。分类报告图示可以直观得到模型各项参数，包括每一类别的精确率（Precision），召回率（Recall），F1 分数值（F1-Score）。对于这三项值，其计算公式如下：

– 精确率

$$\text{Precision} = \frac{TP}{TP + FP} \quad (30)$$

– 召回率

$$\text{Recall} = \frac{TP}{TP + FN} \quad (31)$$

– F1 分数值

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (32)$$

⁴上文中提到我们对用户评分进行标签编码，从原来的 $[1, 10]$ 映射至新评分标签 $[0, 9]$ ，即在原评分基础上减 1，而在混淆矩阵热力图及分类报告图示中我们将标签编码已映射回原评分，对于模型的 ROC/AUC 曲线，我们未映射回原评分。

根据上述(30)式、(31)式、(32)式，我们可以计算出每一个模型对于每一类别的三项指标值，并绘制分类报告图，对于模型一的分类报告，见图 24。此外我们还利用宏观(Macro)、微观(Micro)、权重(Weighted)三种方法计算模型的精确率、召回率以及F1 分数值。宏观方法即取所有类别得出的值的平均值，不考虑样本的不平衡性；微观方法将所有样本视为整体；权重方法计算的是加权平均值。结果见表 11。

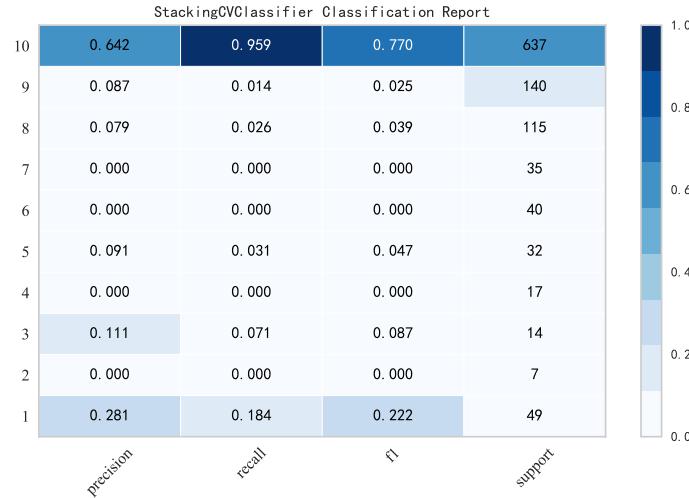


图 24 模型一分类报告 [语音业务-语音通话整体满意度]

表 11 模型一宏观、微观、权重计算准确率、召回率、F1 分数值

Method	Precision	Recall	F1-Score
Macro	0.1291	0.1285	0.1190
Micro	0.5773	0.5773	0.5773
Weighted	0.4373	0.5855	0.4841

根据上述图表，我们可以发现，对于不平衡的样本数据，若仅仅计算模型的准确率，显然不能很好体现模型的效果，同时我们也应该根据实际业务需求，对 Macro 及 Micro 方法进行综合分析。在样本类别的数目分布不均衡时，Macro 方法会同等对待每一种分类；而 Micro 方法较 Macro 方法更加趋向于客观结果的分析。此外，对于模型的精确率、召回率，我们可以根据定义发现，这两项值显然较大，模型效果较好。同时根据定义，我们可以发现模型的精确率、召回率在理想情况下是相差较小的，我们可以根据图表结果验证，符合预期效果。对于模型的 F1 分数值，其为精确率与召回率的调和平均数，因此当精确率与召回率均有较好表现时，F1 分数值会有较优秀表现。我们也可对(32)式进行一定变换，可以得到

$$F1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \quad (33)$$

根据该式，我们可以得出上述结论。

- **ROC/AUC 曲线。**在分析特征曲线及曲线下面积(Receiver Operating Characteristic/Area Under the Curve, ROC/AUC)图之前，我们需要了解模型的相关参数，定义如下：

– 灵敏度 (Sensitivity)。灵敏度又被称为真阳性率, 即 TP 率, 定义为:

$$\text{Sensitivity} = TPR = \frac{TP}{TP + FN} \quad (34)$$

– 特异性 (Specificity)。特异性又被称为真阴性率, 即 TN 率, 定义为:

$$\text{Specificity} = TNR = \frac{TN}{TN + FP} \quad (35)$$

– 1-Specificity。称为假阳性率 (False Positive Rate, FPR), 定义为:

$$FPR = 1 - \text{Specificity} = \frac{FP}{FP + TN} \quad (36)$$

– 1-Sensitivity。称为假阴性率 (False Negative Rate, FNR), 定义为:

$$FNR = 1 - \text{Sensitivity} = \frac{FN}{FN + TP} \quad (37)$$

FPR 和 FNR 均对数据分布的变化不敏感^[17], 因此这两个指标可以用于在不平衡的数据上建立的模型效果的评价。

对于 ROC/AUC 曲线, 其以每一类别的 1-Specificity 即 FPR 为横坐标, 以 Sensitivity 即 TPR 为纵坐标, 其可体现出模型的灵敏度与特异性之间的关系与差异。因此, 该图的理想点位于左上角, 即 $FPR = 0$ 且 $TPR = 1$, 换言之, 当曲线越靠近左上角, 模型效果就越优。从而, 我们可以得到另一项指标, 即曲线下面积 (Area Under the Curve, AUC), 由上述分析可知, AUC 值越高, 模型的整体效果也就越优。对于模型一的 ROC/AUC 曲线, 见图 25。

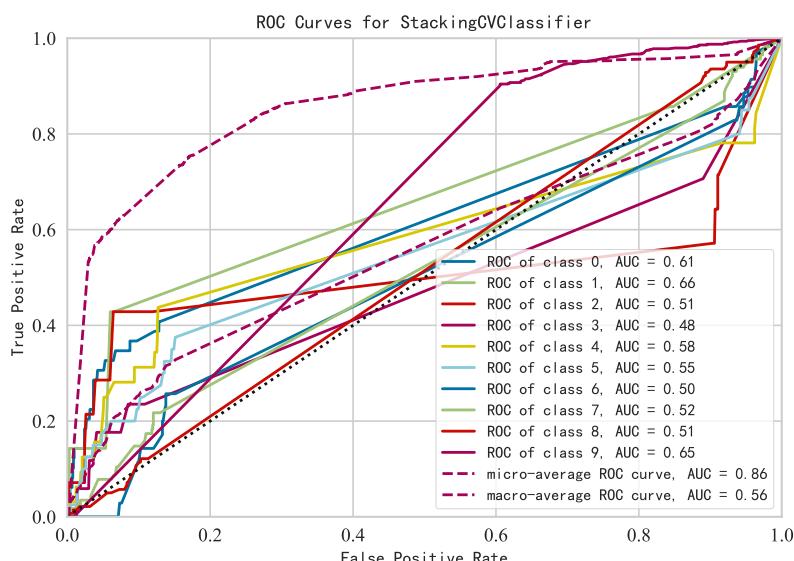


图 25 模型一 ROC/AUC 曲线 [语音业务-语音通话整体满意度]

根据上图结果, 我们可以发现, 模型一预测用户对于“语音业务-语音通话整体满意度”

的评分结果中，除对于类别 3（即实际评分为 4 分）的 $AUC < 0.50$ 之外，其余的 $AUC \geq 0.50$ ，即以 $y = x$ 为分界线。同时，我们可以发现“macro-average ROC curve”指标，其是通过 Macro 方法求得，在上文中我们提到，该数据样本的标签分类是严重不平衡的，该方法能够平等对待每一项分类，在此方法下，我们可以对于小样本类别的准确率有一定把握，其曲线下面积 $AUC = 0.56$ ，位于分界线左上，预测效果良好；而对于“micro-average ROC curve”指标，其 $AUC = 0.86$ ，这指的是利用 Micro 方法求得曲线，曲线下面积为 0.86，曲线下面积较大，且曲线有向左上角最优点靠近的趋势，可以说明模型整体能力较优。

此外，为观察预测出的评分分布是否大体与已知评分数据集评分分布大致一致，我们再次绘制南丁格尔玫瑰图，见图 26。观察该图及图 22，我们可以发现预测出的评分分布与已知评分数据集评分分布大体一致，符合数据的一般性及内在规律。

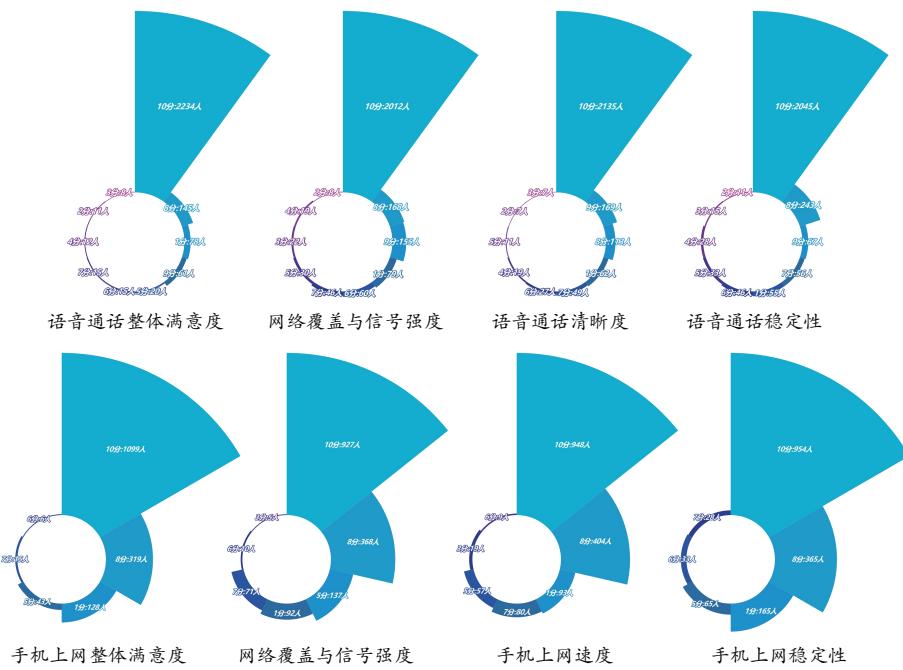


图 26 [附件 3、附件 4] 各类评分人数南丁格尔玫瑰图

对于模型一 [预测语音业务，语音通话整体满意度]，根据表 10，我们可以发现，其五折交叉验证平均准确率为 57.73%，平均绝对误差为 1.2937，均方误差为 6.3877，即平均每位用户评分预测误差控制在 1 分左右，易得均方根误差为 $\sqrt{6.3877} \approx 2.5274$ ，可以发现，预测值与准确值误差程度在可接受范围内，预测效果较好。而此外，我们知道，对于用户评分，其属于主观性评分，而该评分难以准确量化，给模型带来一定复杂性，即在相近的分数处，较难于区分。此外，对于部分用户，可能存在未详细阅读问卷，导致乱评、错评现象，这一类属于明显异常值，显然，该预测模型可以在一定程度上识别出异常值，且效果良好。同时，我们绘制模型的混淆矩阵热力图、分类报告、ROC/AUC 曲线，从上述分析中，我们有理由认为模型的预测效果较好，但同时也存在一定缺陷，如对小样本的预测有一定难以区分的现象，而若我们要更多地分析研究小样本用户的情况，这会带来一定的局限性，我们也将提及改进措施。

六、模型的评价与推广

6.1 模型的评价

- **模型的优点:**

1. 对数据进行综合处理，层次清晰，模型具有一定解释性；
2. 数据标准化，避免量纲不一造成的影响学习影响的情况；
3. 特征筛选，减少不重要性因素占比，减少数据维度，提升模型学习效率，一定程度上避免数据噪声，适当降低模型复杂度，使模型高效化，防止过拟合；
4. 特征构造，由原数据构造出新数据特征，适当增多数据维度，防止欠拟合；
5. 综合熵权法、灰色关联度分析及随机森林量化影响程度，避免局部最优；
6. 对各模型进行参数调优，尽可能提高模型的多方面能力；
7. 加入正则化方法，一定程度上也可防止过拟合；
8. 交叉验证，更好地利用数据集，减少数据浪费，提高模型的泛化能力，验证模型的稳健性，防止过拟合情况的发生；
9. 模型设置任意随机种子，在保证划分训练集及测试集的一般性、随机性的同时，确保可重复性的结果，方便后续处理；
10. 通过主成分分析及随机森林进行特征选择，保证客观性；
11. 多模型 Stacking 集成学习，更好地利用已有数据，多方面学习数据中内在联系，结合多个模型优良方面，避免陷入局部最优，对数据有更好的把控能力，提升模型的泛化能力、提高预测准确率、提高模型稳健性、鲁棒性，同时减小预测误差，且对异常值有一定识别能力。

- **模型的缺点:**

1. 模型对于小样本分类的识别能力较差，难以对这些用户进行深入分析；
2. 模型对于预测主观性评分，难以提供完全一致的评分结果；
3. Stacking 在构造时，有一定复杂度，对基模型的要求较高；
4. 对于部分评分，特征构造出的因素有一定局限性；
5. 用户评分为主观性结果，本文大多模型选用客观性较强的模型进行解决，对数据利用有一定失真。

- 模型的改进:

1. 在收集数据时，问卷设计需要更加合理化，多方面考虑其余未考虑到的影响因素对用户评分的影响；
2. 在允许条件下获得更多训练样本；
3. 对各模型可以选用非完全一致的特征，提升各模型的独特性，有目的地进行选择，减少学习的数据维度，加快模型收敛速度，使得模型学习高效化，结果准确化；
4. 适当增加或减少数据维度，建立复杂度适中的模型；
5. 对不平衡的多分类，可以采用“下采样”或“上采样”方法，使得分类平衡，但需要更多的数据集；
6. 可适当增加基模型个数，并提高对基模型的筛选要求；
7. 对主观性评分，可以建立主客观相结合的模型，从而优化模型各项指标。

6.2 模型的推广

机器学习可利用现有的数据集进行有目的的训练，在此基础上预测分类标签下人为难以确定的结果，极大方便了当今对复杂数据的处理；多种机器学习相互结合，利用 Stacking 集成学习的方法，可以有效提高模型各方面能力，减少判断错误的情况。针对小部分样本的学习，需要更容易区分类别的特征进行学习，以及利用特征工程等方法进行解决。对于机器学习模型，我们可以作出其可视化图像，观察到模型的各项指标不易发现的问题，如欠拟合、过拟合等情况，我们可以依据模型效果评估可视化来对模型进行一定的调优。本文是以移动用户对业务的评分为基础，我们运用了多种机器学习的模型，再结合 Stacking 进行集成学习，可以发现模型的效果较优，对主观性评分模型有较好把控能力。利用该模型，可以根据用户对某些影响因素的情况，预测用户对于这项业务的满意程度，再结合相关描述性信息，有的放矢地解决用户遇到的问题，提升客户的满意程度，提升产品的服务质量，从而为业务创造更多价值。该模型在一定程度上虽有一定欠缺，但不仅仅可用于该领域的评分，也可用于其余领域，如用户对于某一产品的评价预测，根据用户评价，改善产品质量，提升经济效益，实现双赢。

参考文献

- [1] CSDN. 【数据预处理】sklearn 实现数据预处理（归一化、标准化）[EB/OL]. https://blog.csdn.net/weixin_44109827/article/details/124786873.
- [2] 肖杨, 李亚, 王海瑞, 常梦容. 基于皮尔逊相关系数的滚动轴承混合域特征选择方法 [J]. 化工自动化及仪表, 2022, 49(03):308-315. DOI:10.20030/j.cnki.1000-3932.202203009.
- [3] 王殿武, 赵云斌, 尚丽英, 王凤刚, 张震. 皮尔逊相关系数算法在 B 油田优选化学防砂措施井的应用 [J]. 精细与专用化学品, 2022, 30(07):26-28. DOI:10.19482/j.cn11-3237.2022.07.07.
- [4] 姚文字, 李杰, 李岩峰, 高娜, 王涛. 基于熵权法的呼吸机质量综合评价研究 [C]//. 中国医学装备大会暨 2022 医学装备展览会论文汇编 (下册) . [出版者不详], 2022:162-167. DOI:10.26914/c.cnkihy.2022.042155.
- [5] 谢赤, 钟赞. 熵权法在银行经营绩效综合评价中的应用 [J]. 中国软科学, 2002(09):109-111+108.
- [6] 张广海, 董跃蕾. 青岛市旅游业影响因素灰色关联度分析及其发展路径 [J]. 德州学院学报, 2022, 38(04):54-58.
- [7] 饶雷, 冉军, 陶建权, 胡号朋, 吴沁, 熊圣新. 基于随机森林的海上风电机组发电机轴承异常状态监测方法 [J]. 船舶工程, 2022, 44(S2):27-31. DOI:10.13788/j.cnki.cbgc.2022.S2.06.
- [8] 陈振宇, 刘金波, 李晨, 季晓慧, 李大鹏, 黄运豪, 狄方春, 高兴宇, 徐立中. 基于 LSTM 与 XGBoost 组合模型的超短期电力负荷预测 [J]. 电网技术, 2020, 44(02):614-620. DOI:10.13335/j.1000-3673.pst.2019.1566.
- [9] 杨贵军, 徐雪, 赵富强. 基于 XGBoost 算法的用户评分预测模型及应用 [J]. 数据分析与知识发现, 2019, 3(01):118-126.
- [10] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785-794. <https://doi.org/10.1145/2939672.2939785>.
- [11] 傅湘, 纪昌明. 区域水资源承载能力综合评价——主成分分析法的应用 [J]. 长江流域资源与环境, 1999(02):168-173.
- [12] 张著英, 黄玉龙, 王翰虎. 一个高效的 KNN 分类算法 [J]. 计算机科学, 2008(03):170-172.
- [13] 汪海燕, 黎建辉, 杨风雷. 支持向量机理论及算法研究综述 [J]. 计算机应用研究, 2014, 31(05):1281-1286.
- [14] 马晓君, 沙靖嵒, 牛雪琪. 基于 LightGBM 算法的 P2P 项目信用评级模型的设计及应用 [J]. 数量经济技术经济研究, 2018, 35(05):144-160. DOI:10.13653/j.cnki.jqte.20180503.001.
- [15] 唐敏, 张宇浩, 邓国强. 高效的非交互式隐私保护逻辑回归模型 [J/OL]. 计算机工程:1-11[2023-01-04]. DOI:10.19678/j.issn.1000-3428.0065549.
- [16] 史佳琪, 张建华. 基于多模型融合 Stacking 集成学习方式的负荷预测方法 [J]. 中国电机工程学报, 2019, 39(14):4032-4042. DOI:10.13334/j.0258-8013.pcsee.181510.
- [17] A.Tharwat, Applied Computing and Informatics (2018). <https://doi.org/10.1016/j.aci.2018.08.003>.

附录

[A] 图表

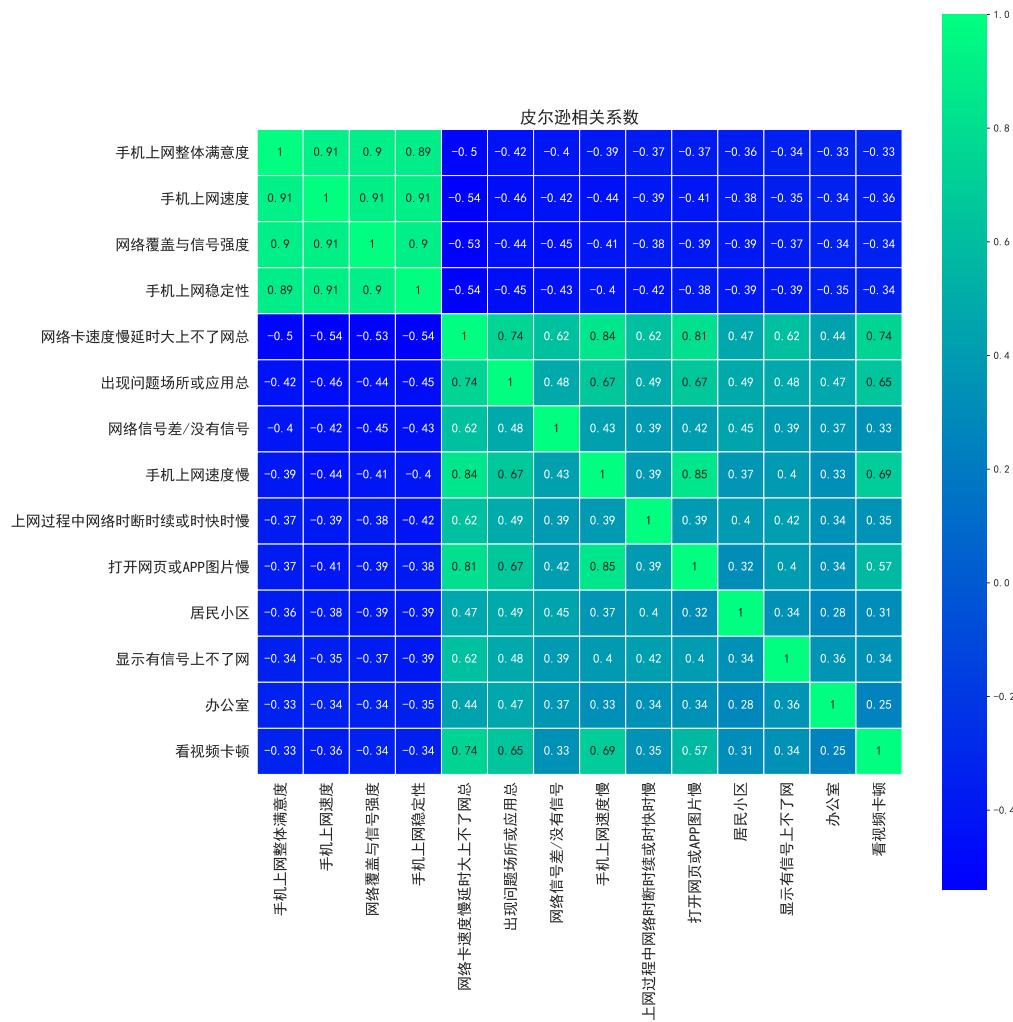


图 27 上网业务评分与其影响因素皮尔逊相关系数热力图

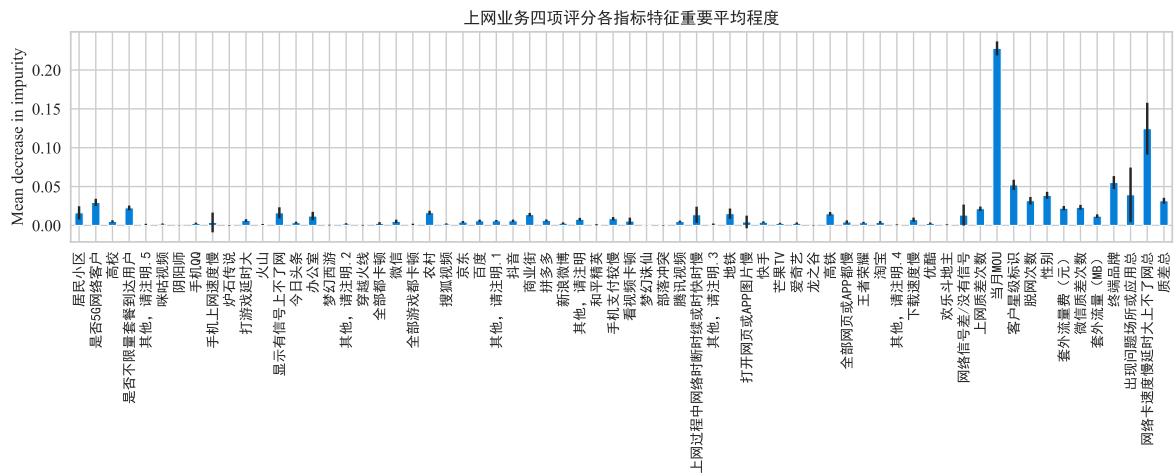


图 28 上网业务四项评分各指标特征重要平均程度

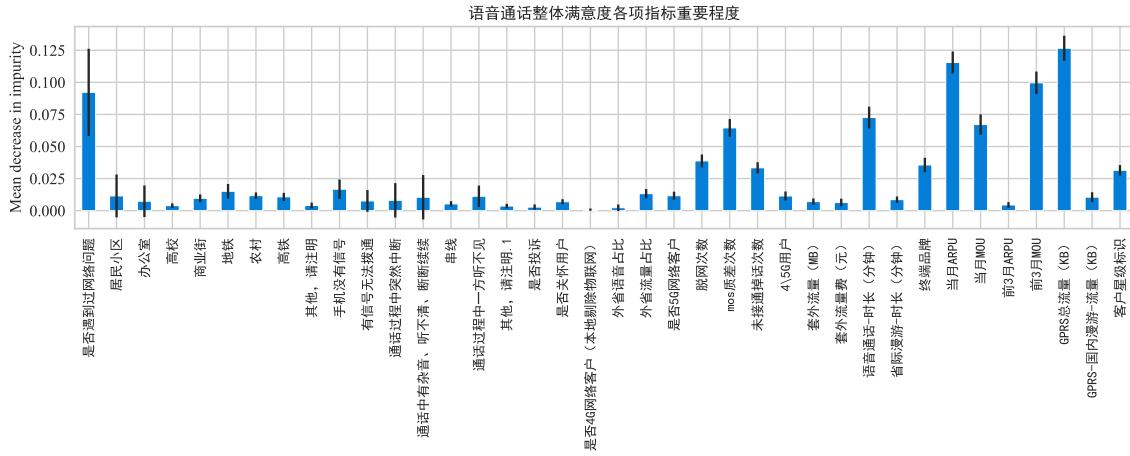


图 29 语音业务-语音通话整体满意度各项指标重要程度

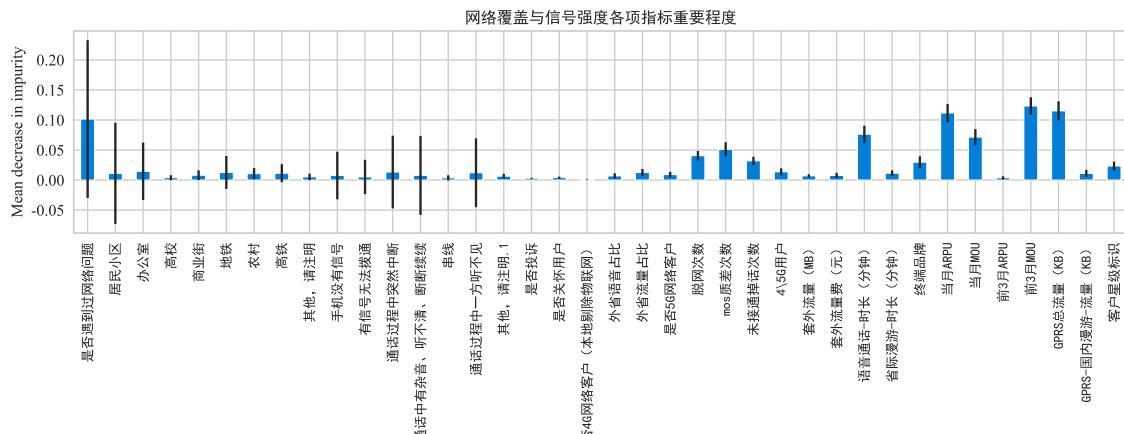


图 30 语音业务-网络覆盖与信号强度各项指标重要程度

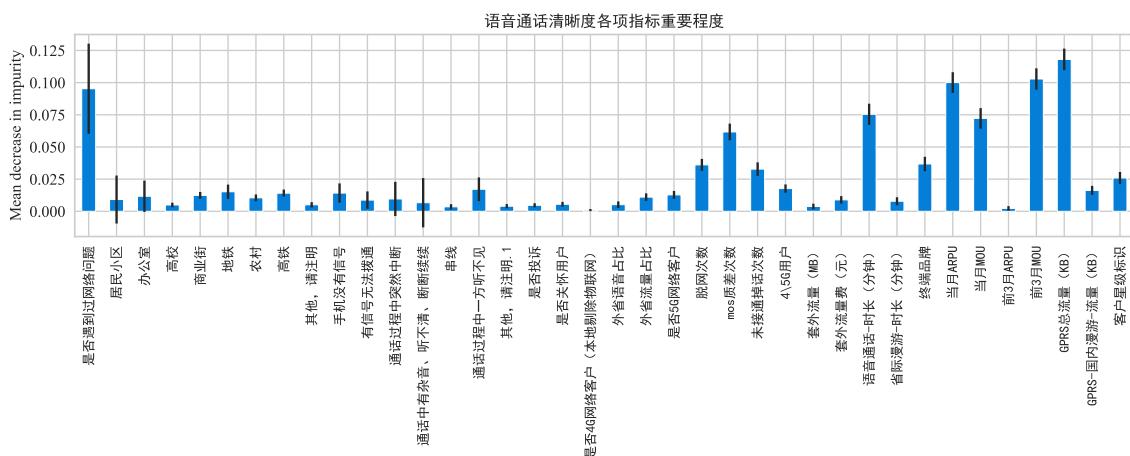


图 31 语音业务-语音通话清晰度各项指标重要程度

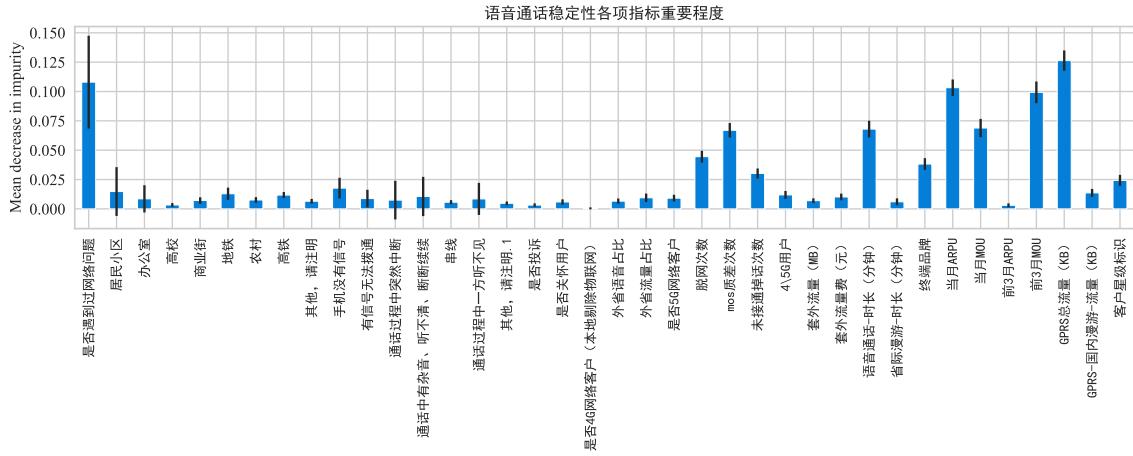


图 32 语音业务-语音通话稳定性各项指标重要程度

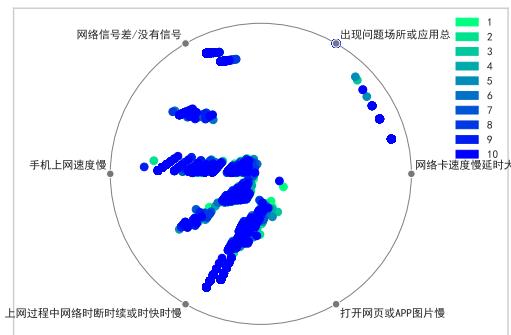


图 33 手机上网整体满意度 RidViz

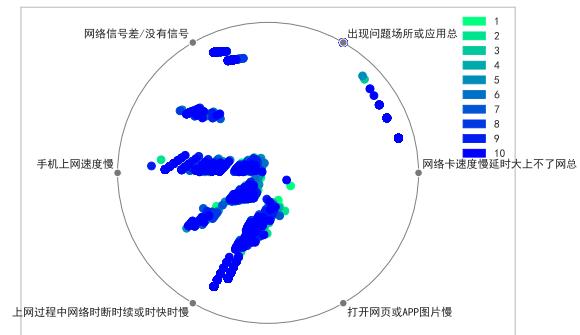


图 34 网络覆盖与信号强度与指标 RidViz

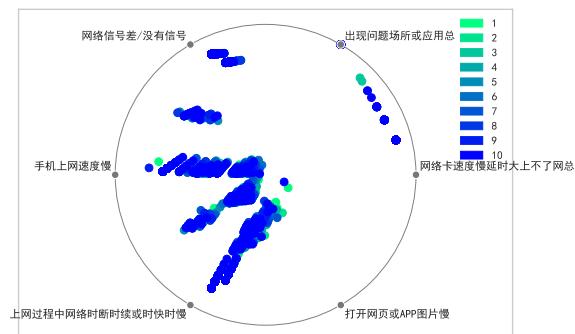


图 35 手机上网速度 RidViz

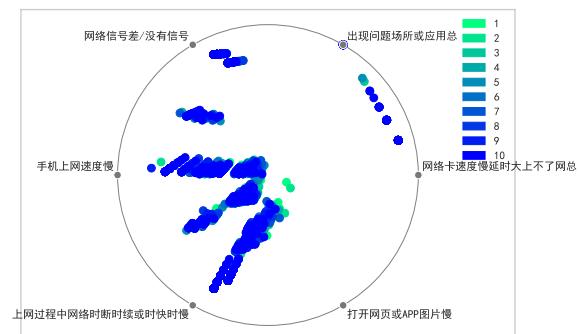


图 36 手机上网稳定性 RidViz

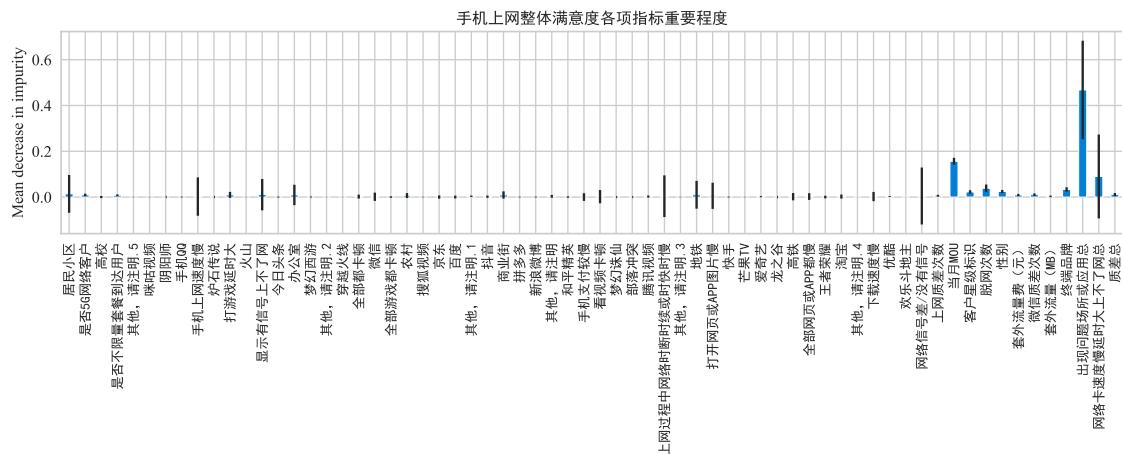


图 37 上网业务-手机上网整体满意度各项指标重要程度

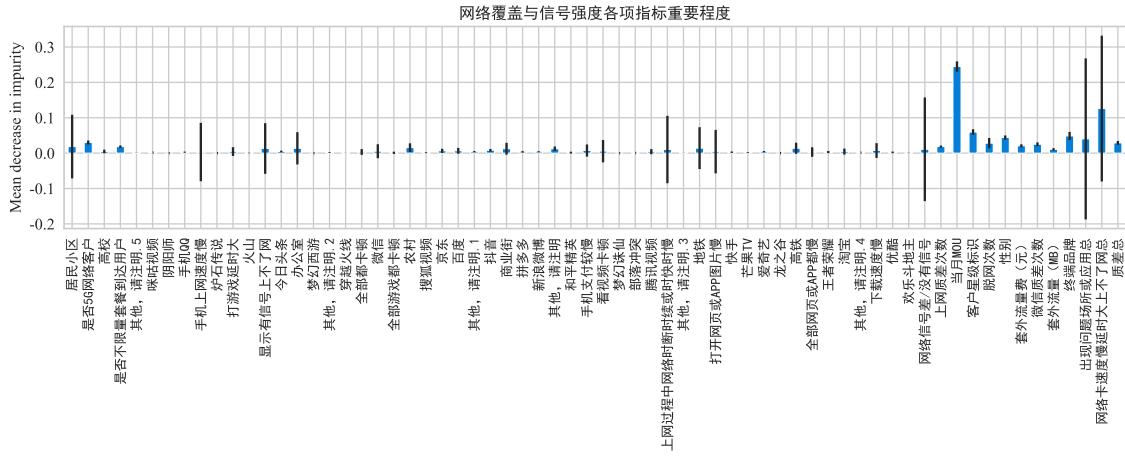


图 38 上网业务-网络覆盖与信号强度各项指标重要程度

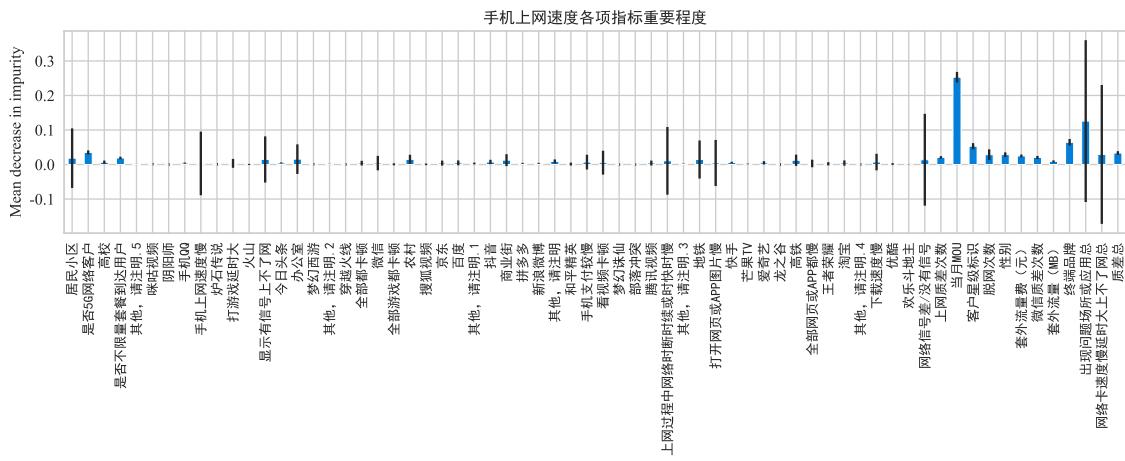


图 39 上网业务-手机上网速度各项指标重要程度

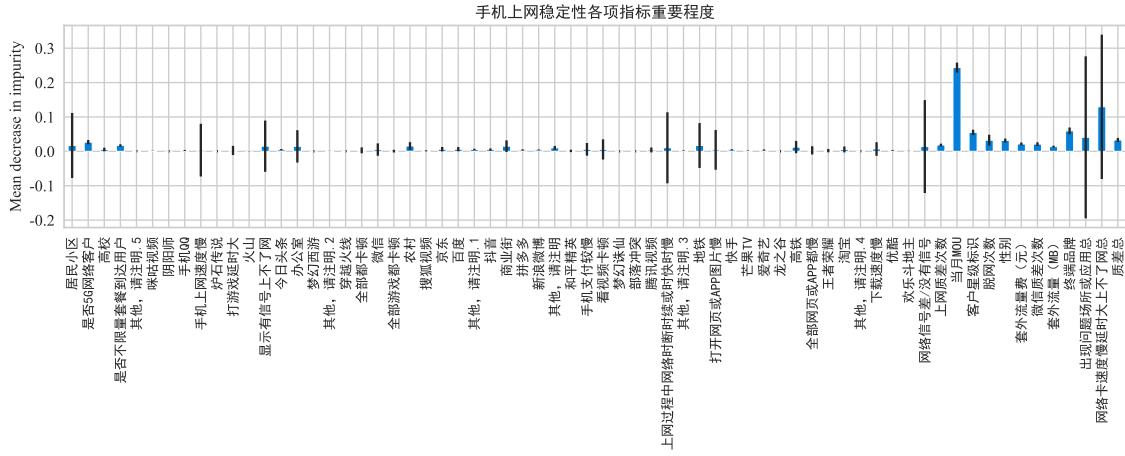


图 40 上网业务-手机上网稳定性各项指标重要程度

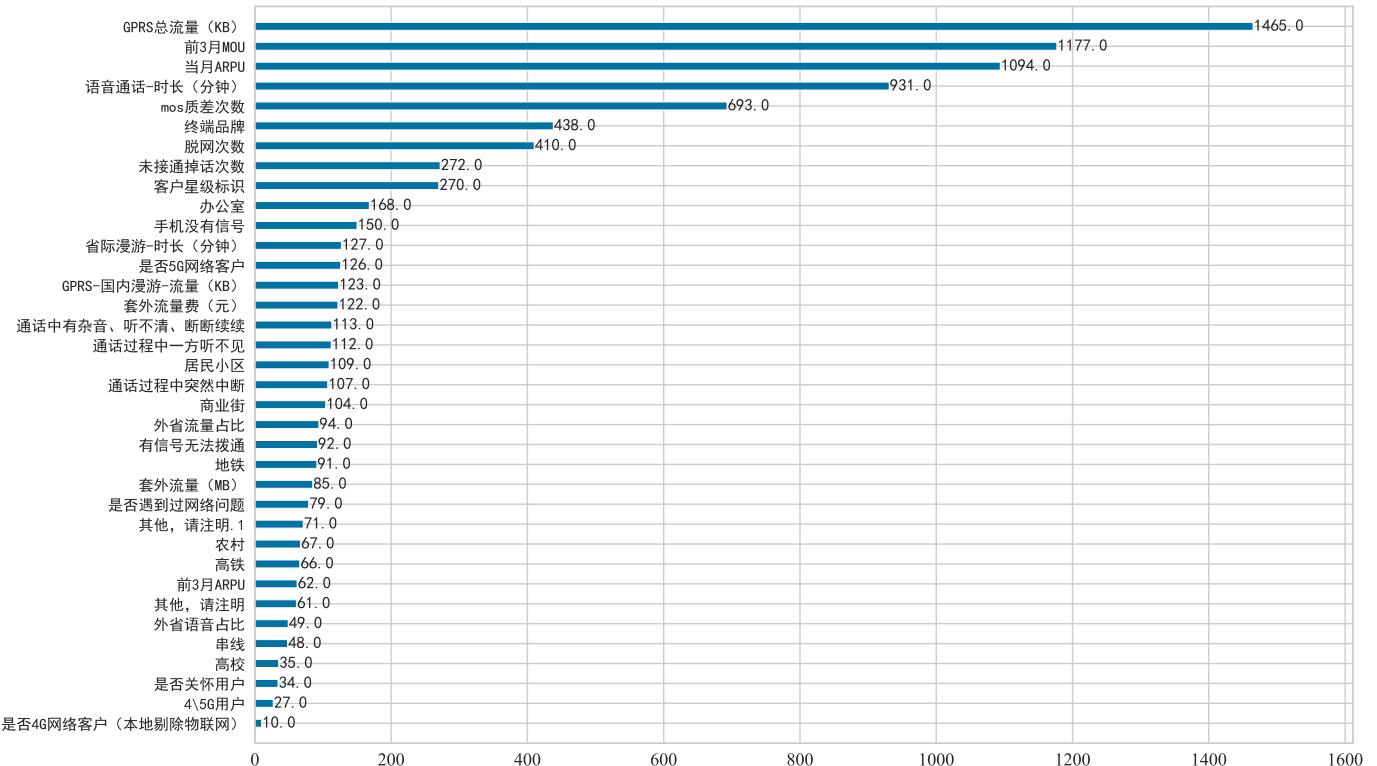


图 41 语音业务-网络覆盖与信号强度各项指标重要程度, XGBoost

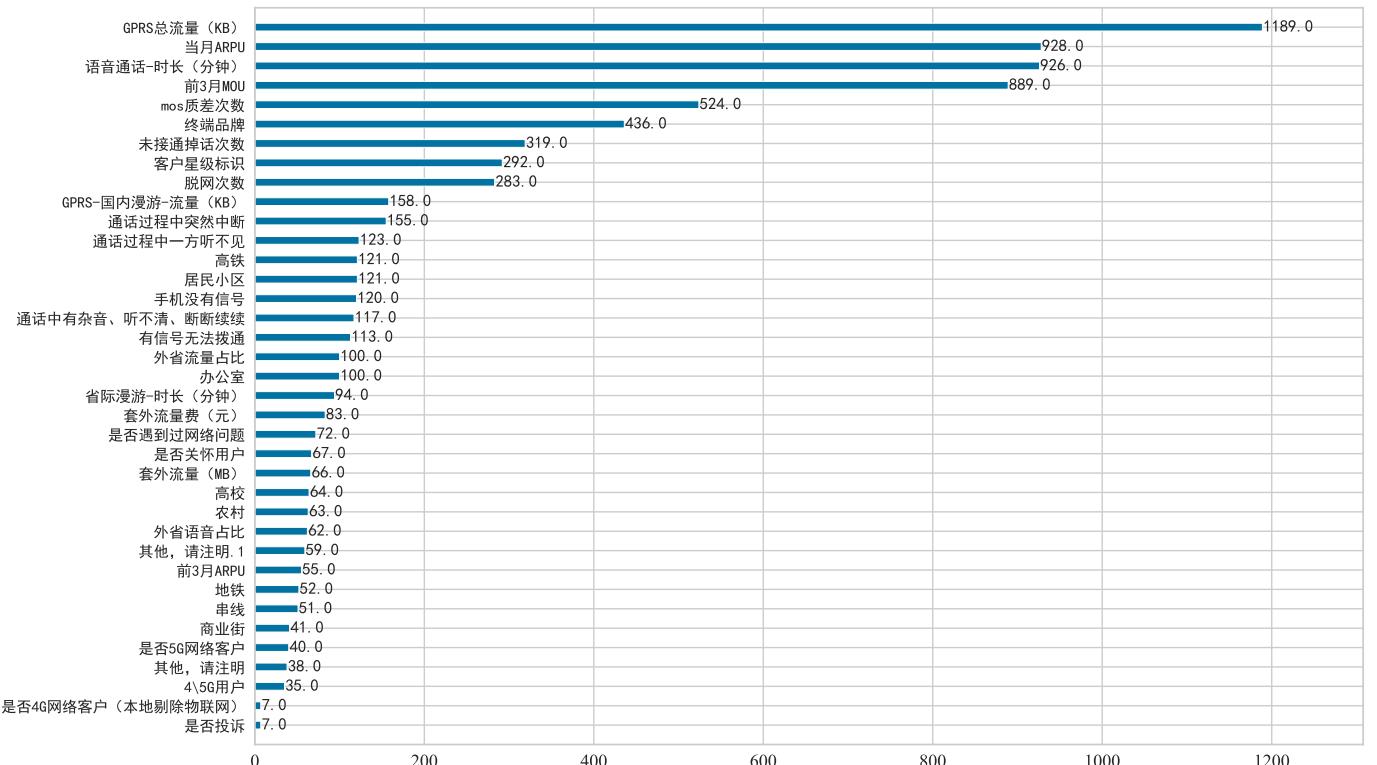


图 42 语音业务-语音通话清晰度各项指标重要程度, XGBoost

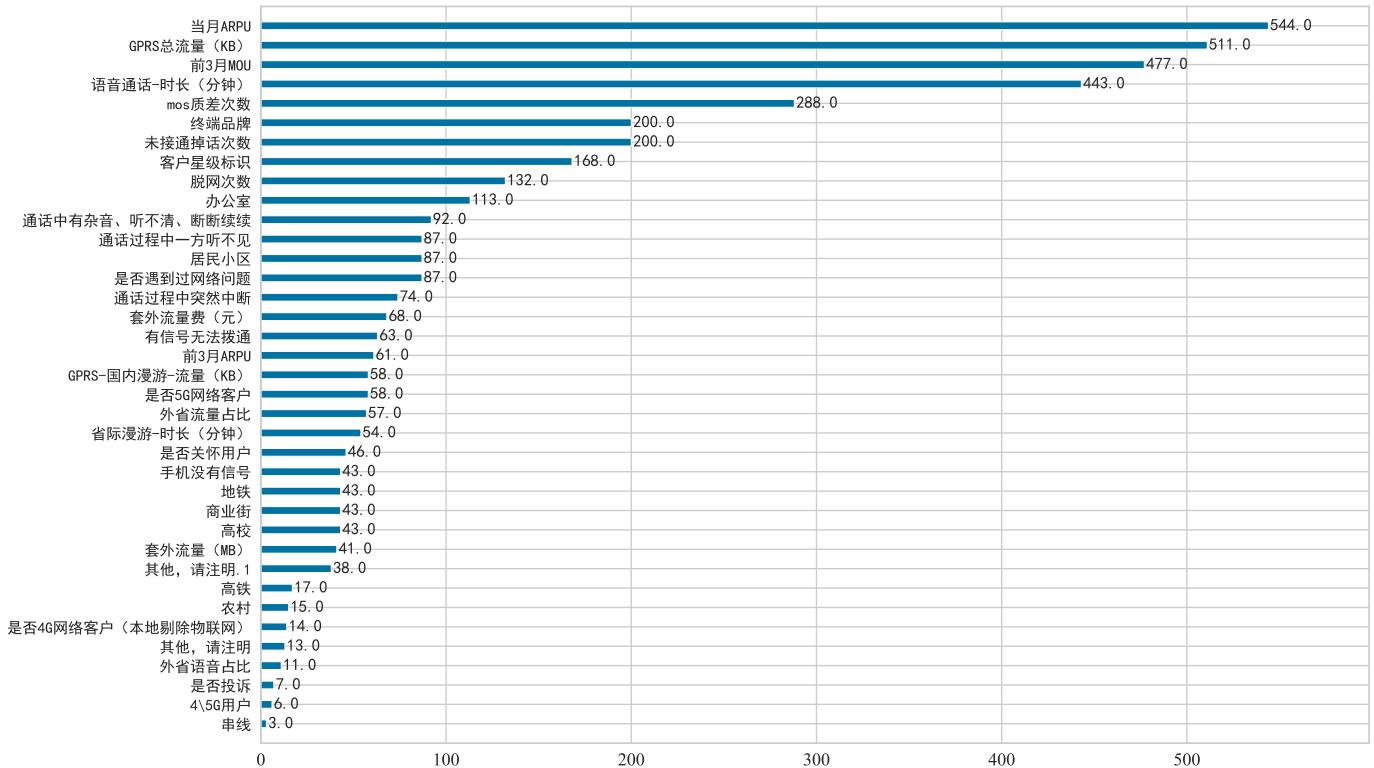


图 43 语音业务-语音通话稳定性各项指标重要程度, XGBoost

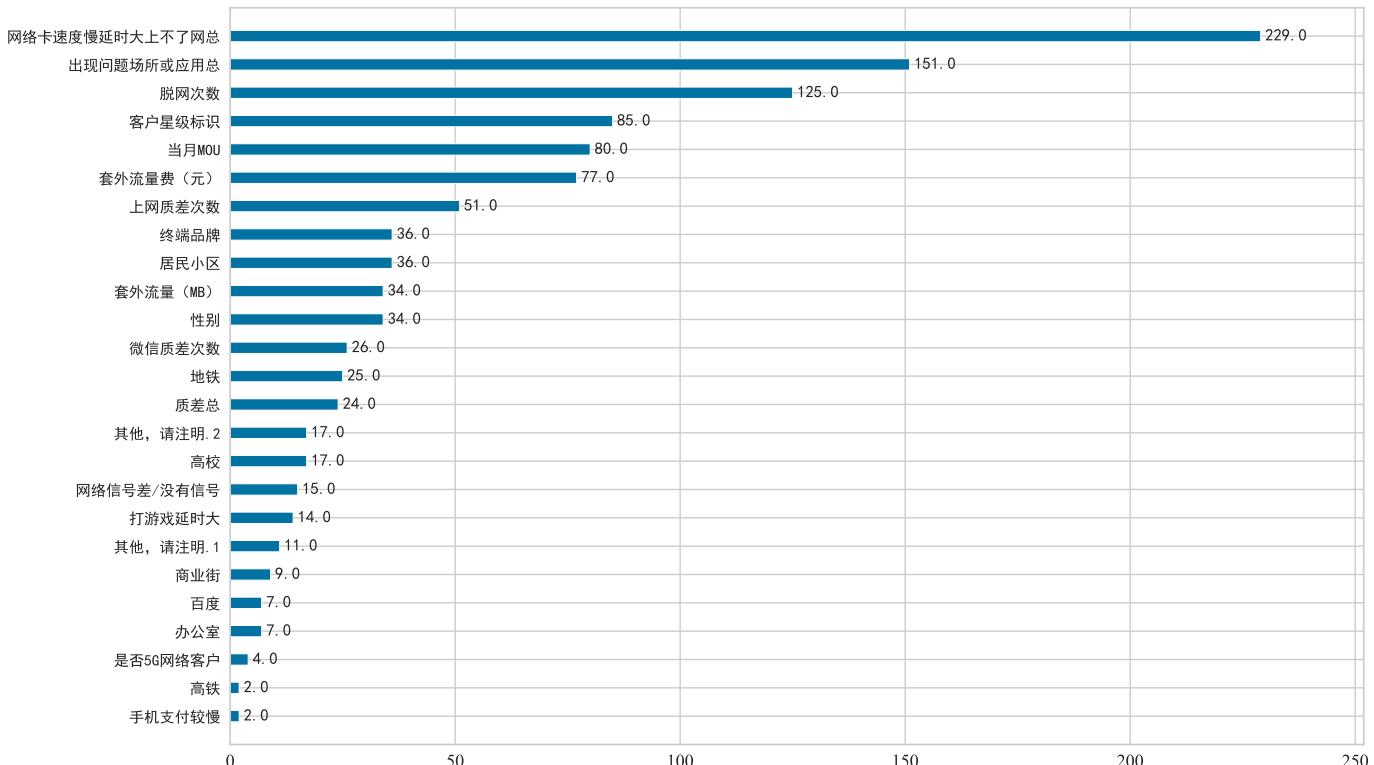


图 44 上网业务-手机上网整体满意度各项指标重要程度, XGBoost

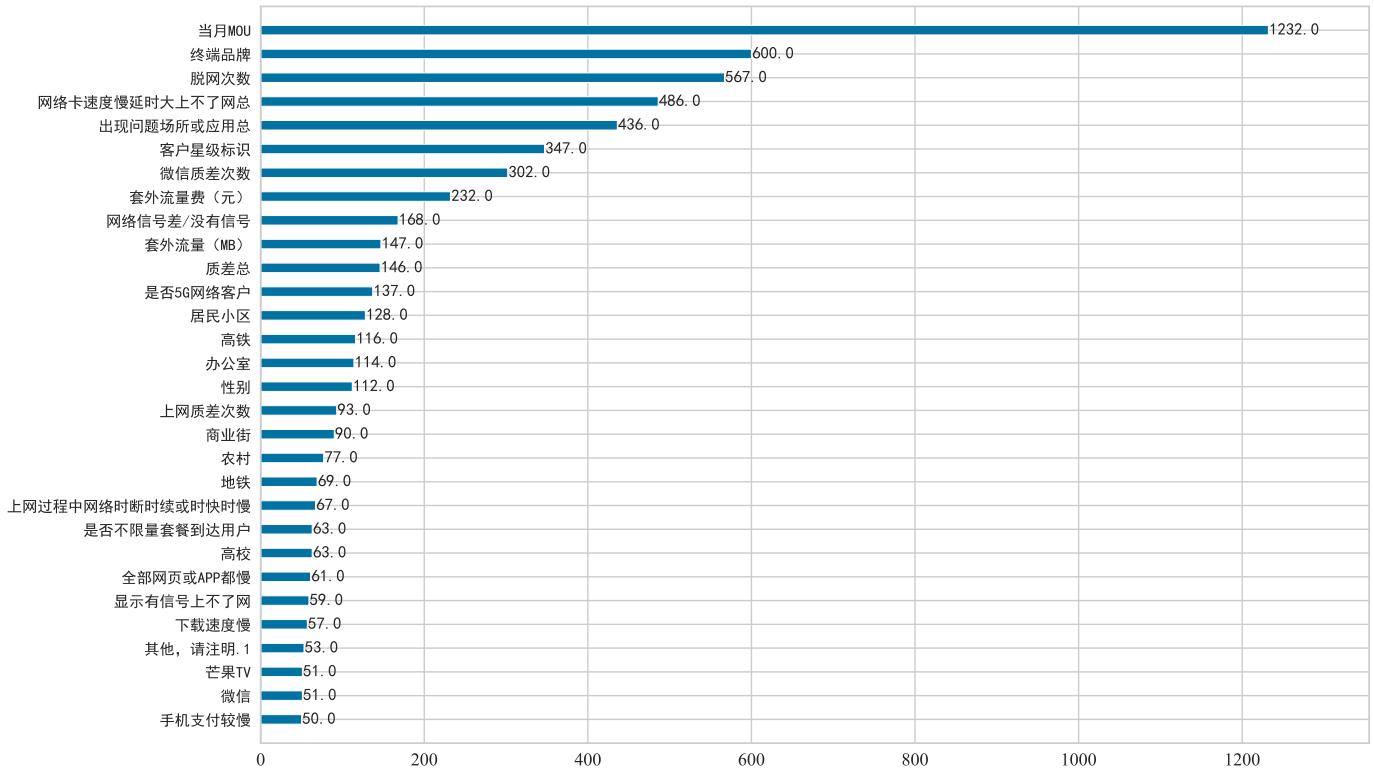


图 45 上网业务-网络覆盖与信号强度各项指标重要程度, XGBoost

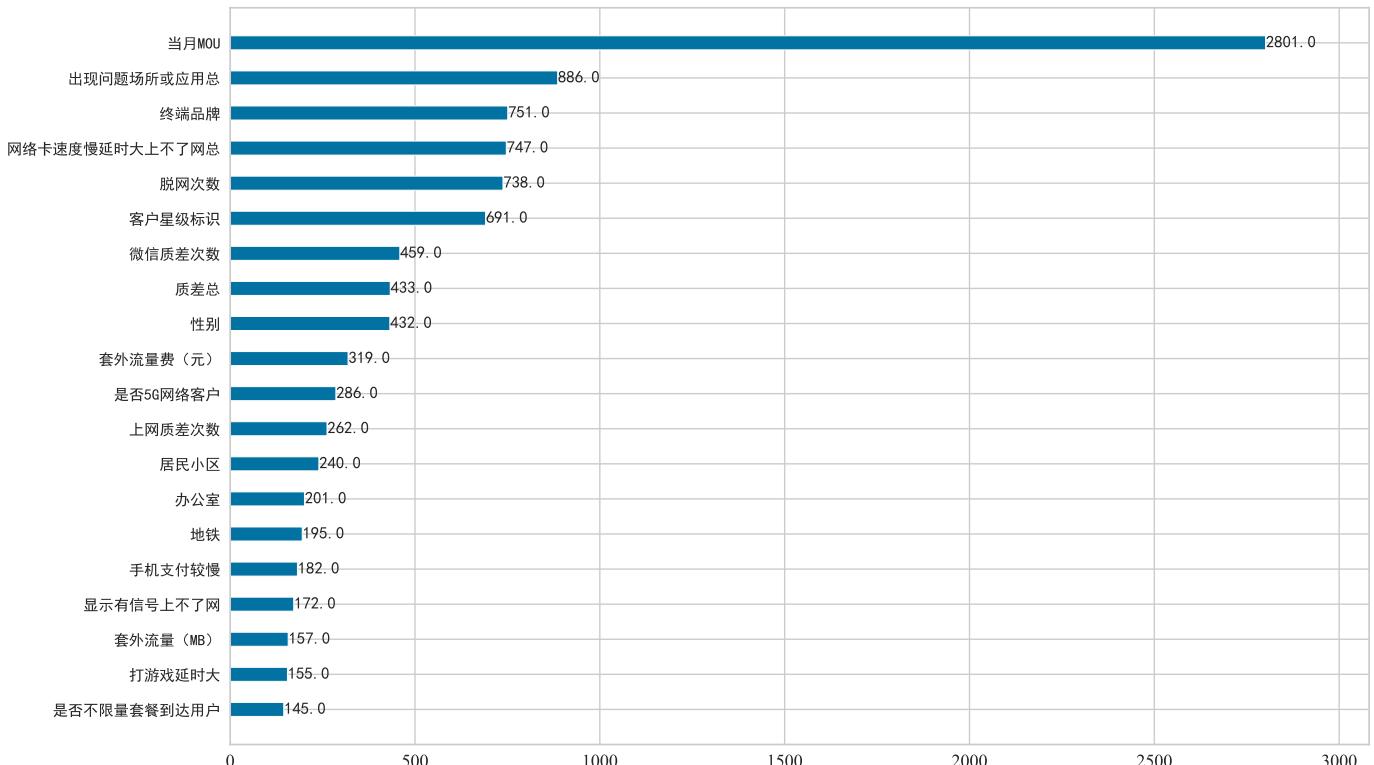


图 46 上网业务-手机上网速度各项指标重要程度, XGBoost

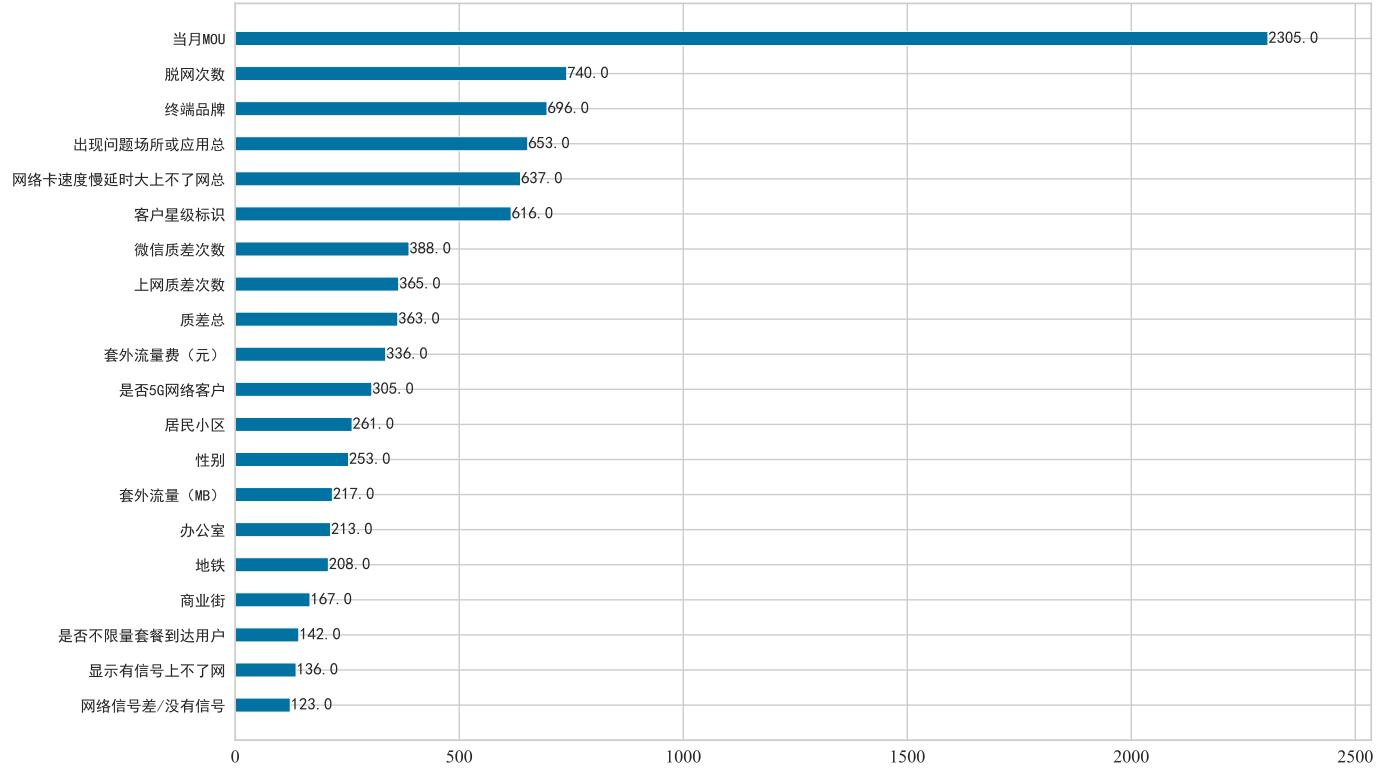


图 47 上网业务-手机上网稳定性各项指标重要程度，XGBoost

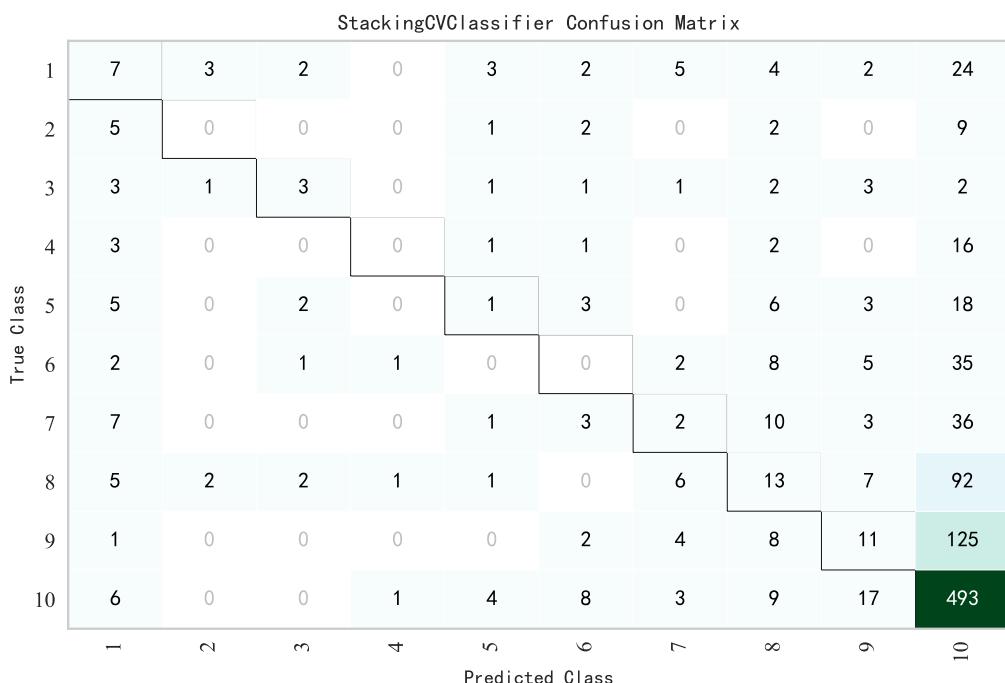


图 48 模型二混淆矩阵热力图 [语音业务-网络覆盖与信号强度]

StackingCVClassifier Confusion Matrix										
True Class	1	2	3	4	5	6	7	8	9	24
1	6	1	1	0	1	2	2	4	5	24
2	0	0	0	0	0	0	1	2	2	4
3	0	0	1	0	0	0	0	2	2	6
4	3	0	0	0	1	0	3	0	0	12
5	3	0	1	1	1	1	0	7	2	16
6	2	0	0	0	1	4	0	1	4	22
7	2	0	0	0	2	1	2	5	2	37
8	3	1	1	1	3	3	3	5	8	107
9	0	0	0	0	1	0	1	8	8	132
10	5	1	0	1	3	7	3	11	8	560
Predicted Class										

图 49 模型三混淆矩阵热力图 [语音业务-语音通话清晰度]

StackingCVClassifier Confusion Matrix										
True Class	1	2	3	4	5	6	7	8	9	10
1	17	0	0	0	5	3	0	11	0	12
2	2	0	0	0	0	1	1	4	0	3
3	13	0	1	0	4	0	2	9	0	1
4	5	0	0	0	2	0	0	5	0	4
5	8	0	0	0	3	1	1	16	0	14
6	7	0	0	0	2	2	1	19	0	17
7	7	0	0	0	5	1	0	25	0	23
8	9	0	0	1	5	1	1	32	0	67
9	2	0	0	0	1	1	0	17	0	56
10	2	0	0	0	0	3	0	30	0	217
Predicted Class										

图 50 模型四混淆矩阵热力图 [语音业务-语音通话稳定性]

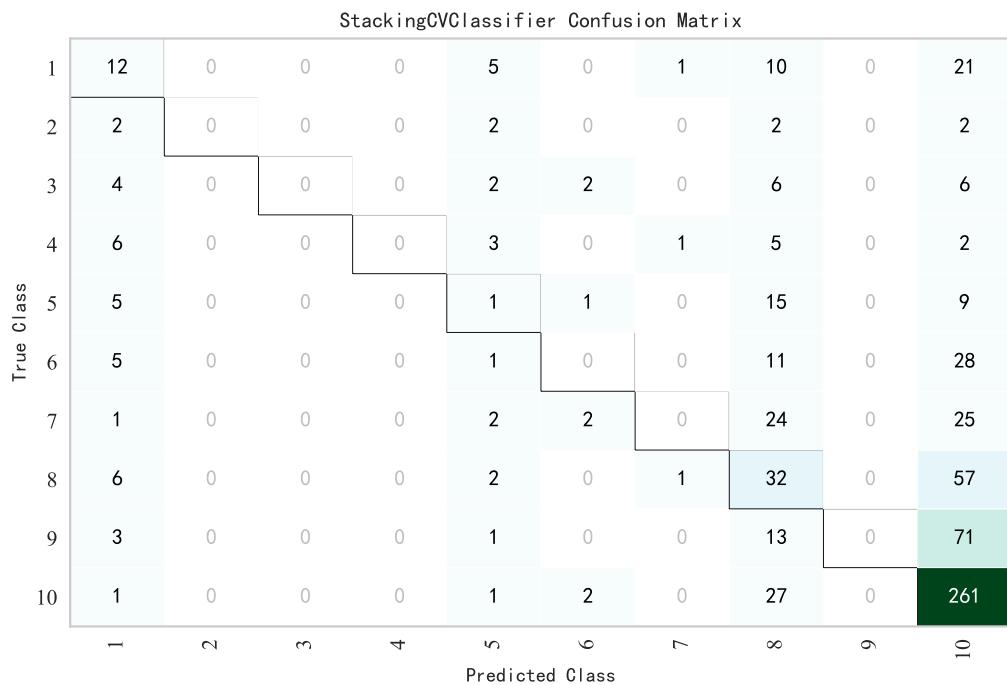


图 51 模型五混淆矩阵热力图 [上网业务-手机上网整体满意度]

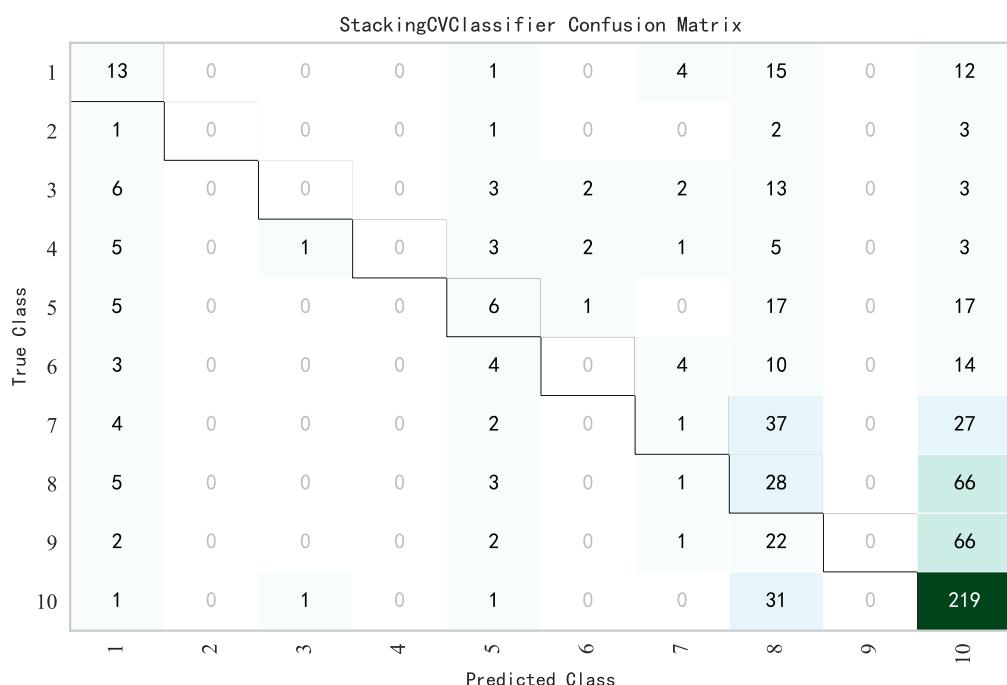


图 52 模型六混淆矩阵热力图 [上网业务-网络覆盖与信号强度]

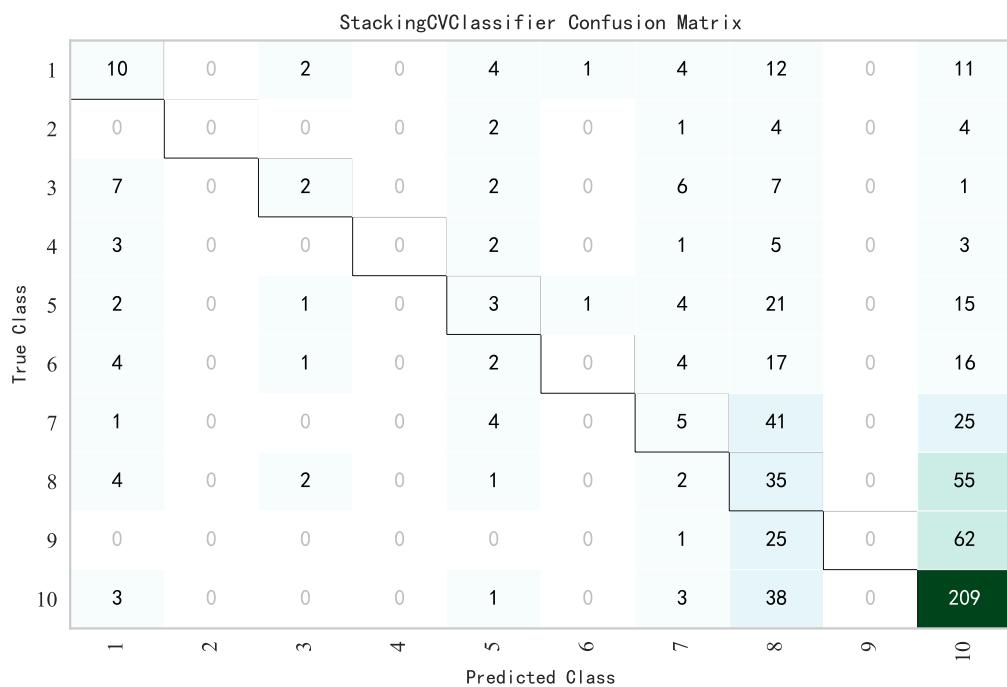


图 53 模型七混淆矩阵热力图 [上网业务-手机上网速度]

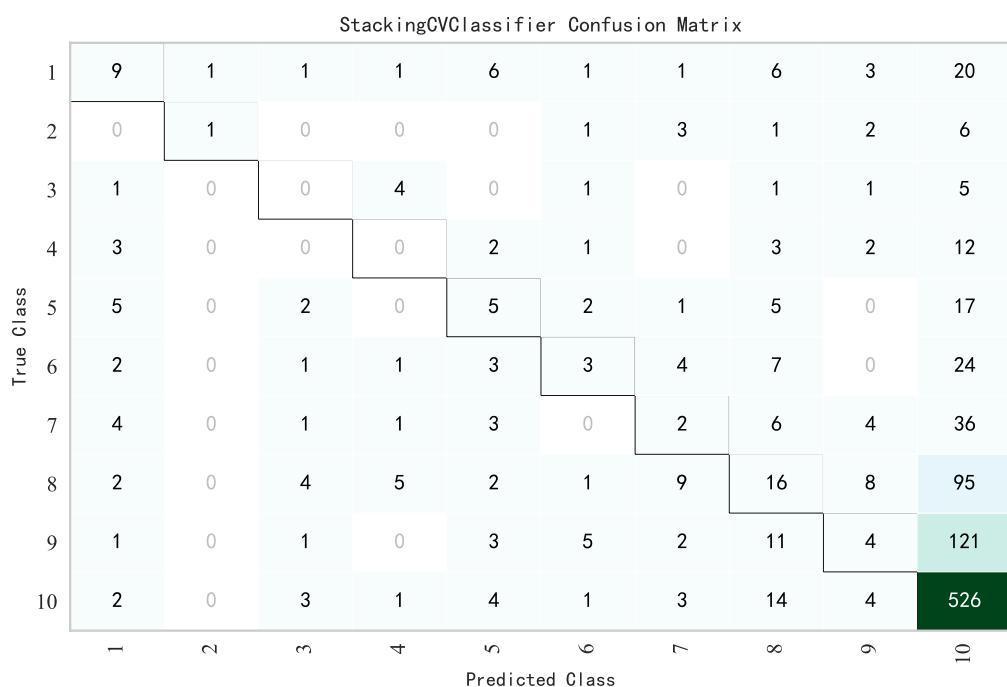


图 54 模型八混淆矩阵热力图 [上网业务-手机上网稳定性]

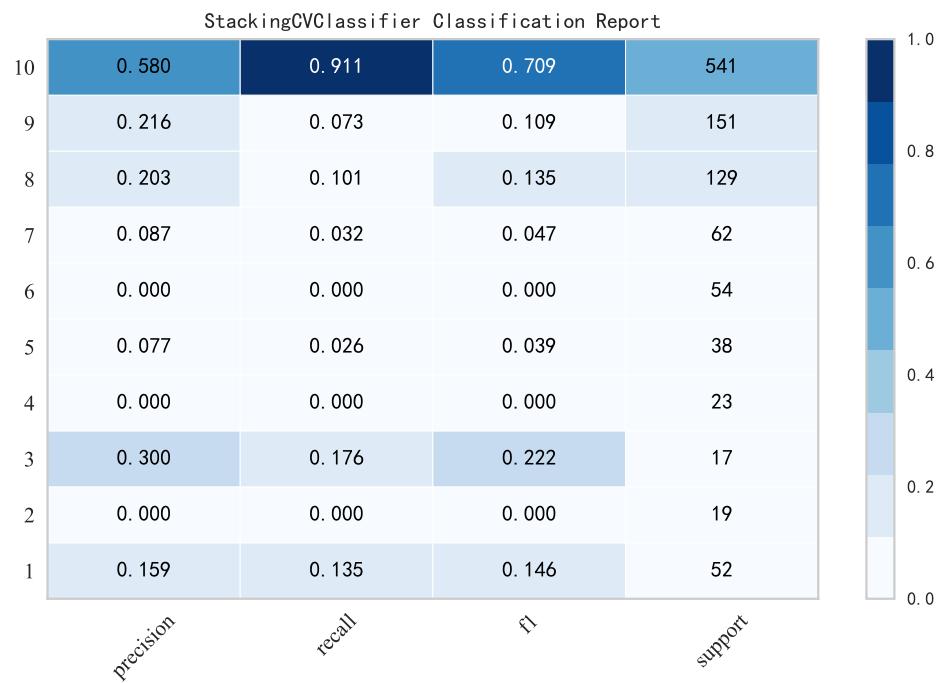


图 55 模型二分类报告 [语音业务-网络覆盖与信号强度]

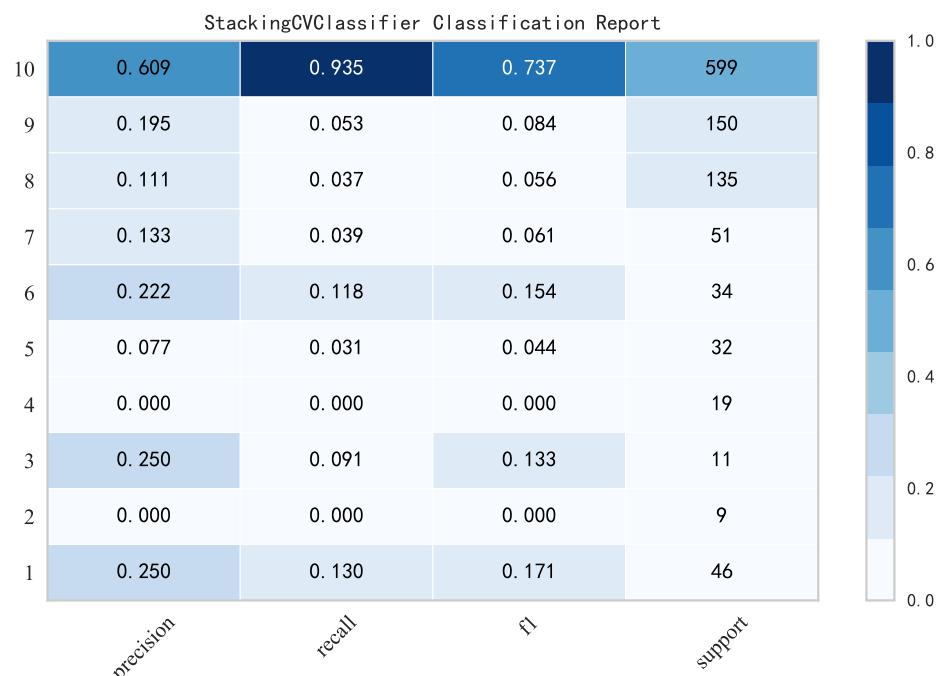


图 56 模型三分类报告 [语音业务-语音通话清晰度]

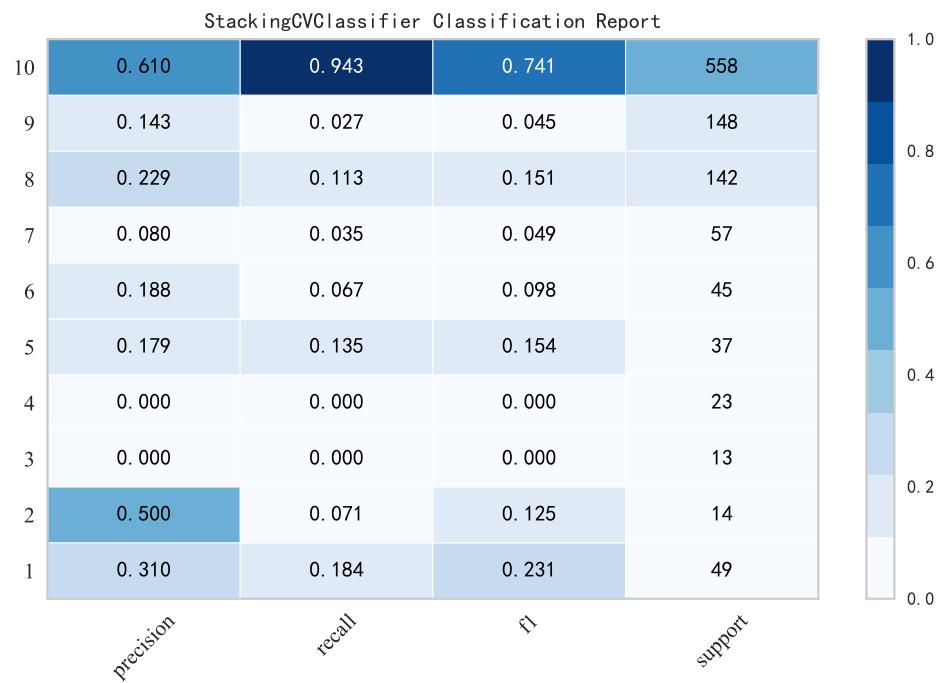


图 57 模型四分类报告 [语音业务-语音通话稳定性]

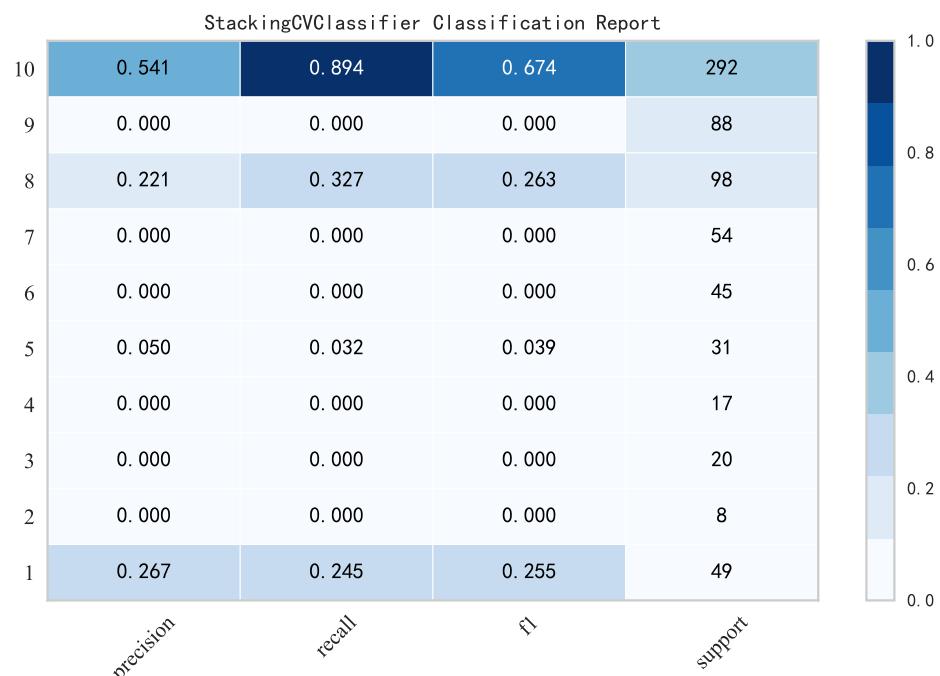


图 58 模型五分类报告 [上网业务-手机上网整体满意度]

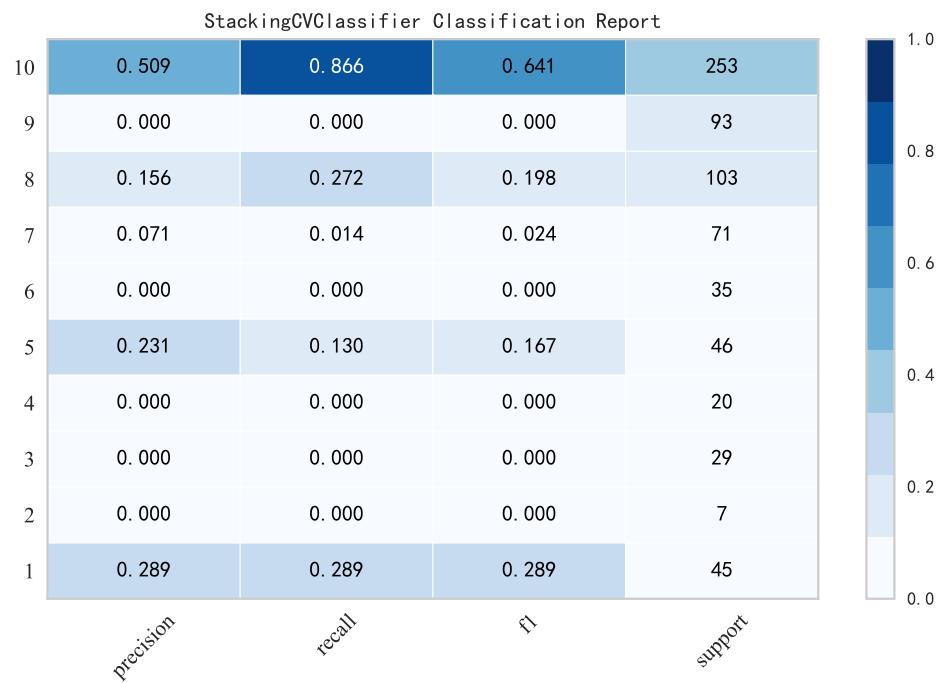


图 59 模型六分类报告 [上网业务-网络覆盖与信号强度]

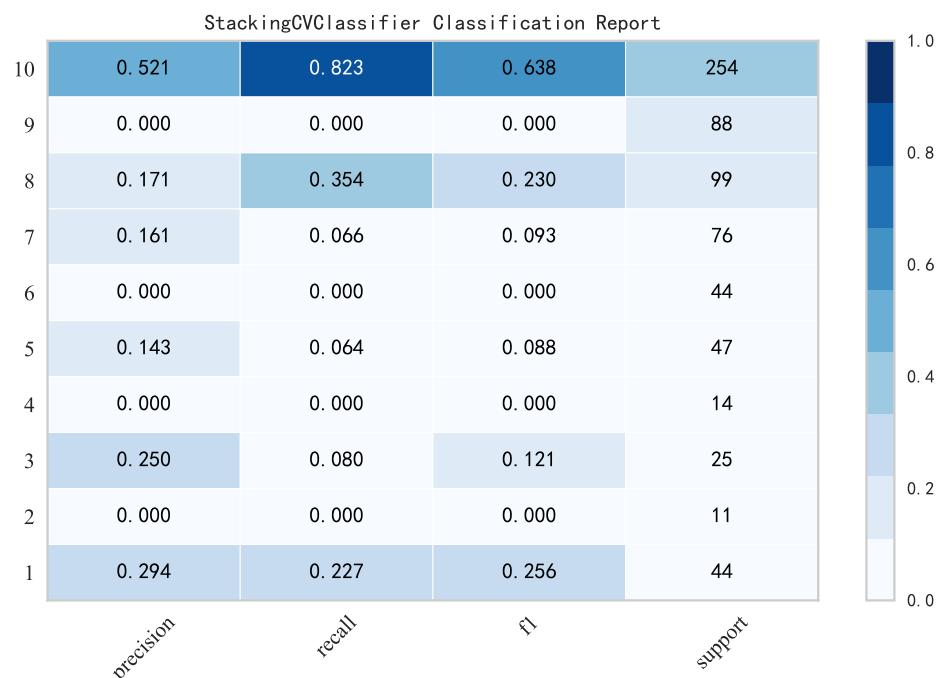


图 60 模型七分类报告 [上网业务-手机上网速度]

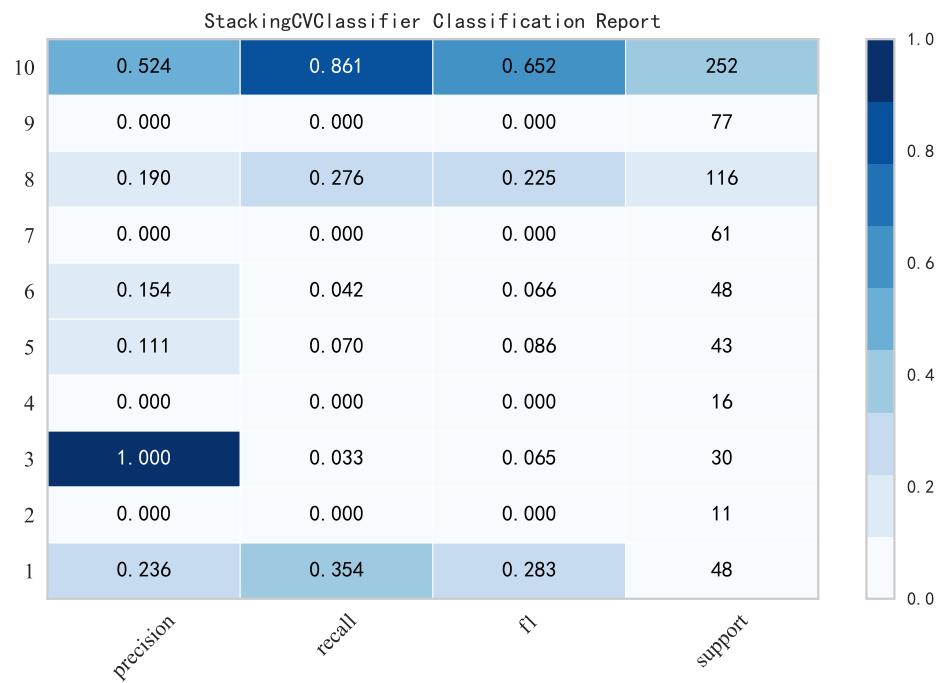


图 61 模型八分类报告 [上网业务-手机上网稳定性]

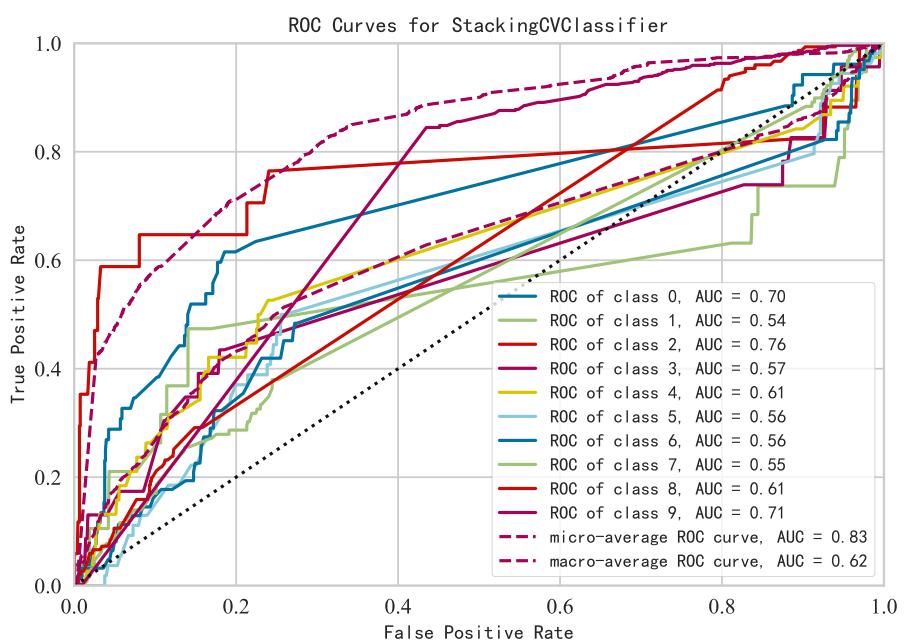


图 62 模型二 ROC/AUC 曲线 [语音业务-网络覆盖与信号强度]

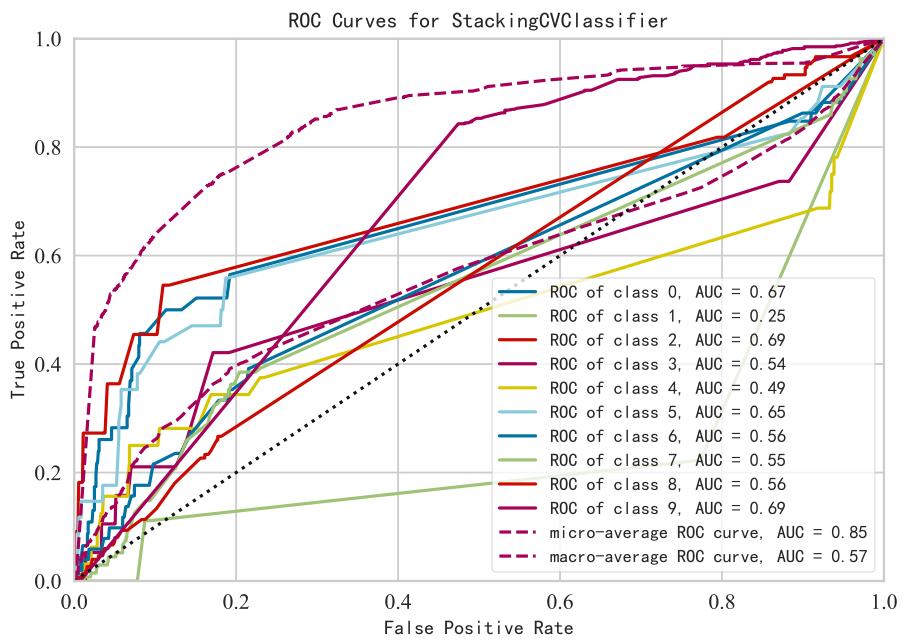


图 63 模型三 ROC/AUC 曲线 [语音业务-语音通话清晰度]

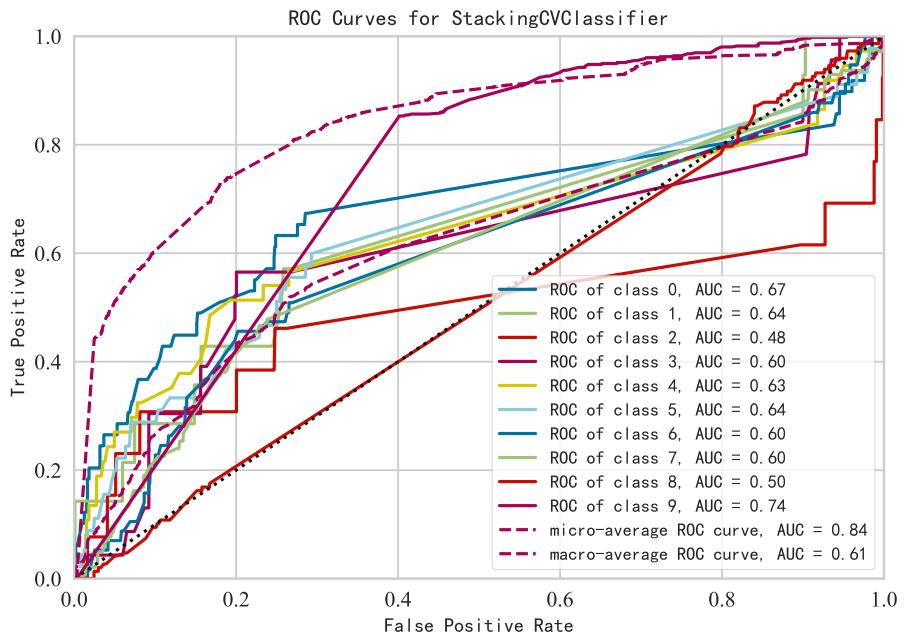


图 64 模型四 ROC/AUC 曲线 [语音业务-语音通话稳定性]

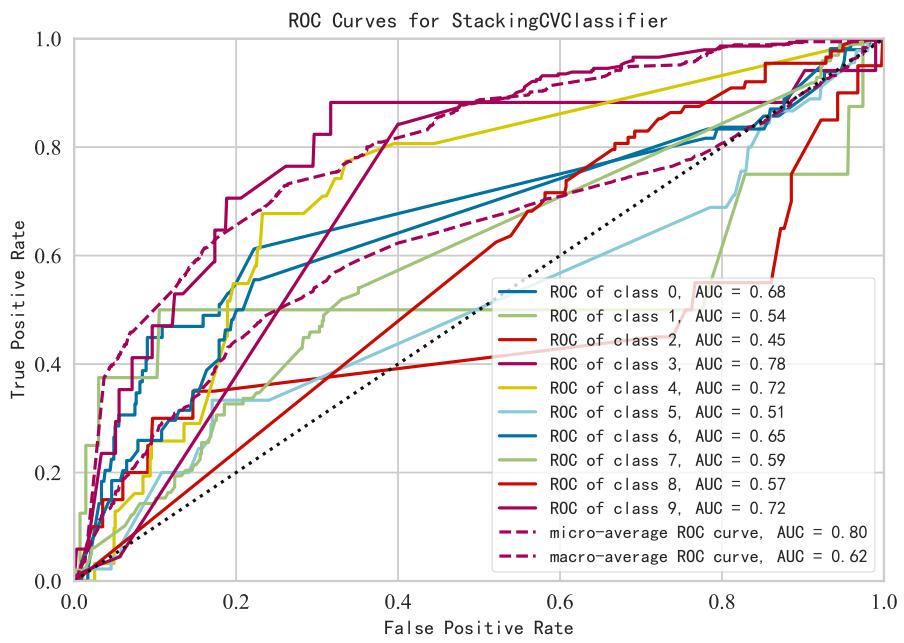


图 65 模型五 ROC/AUC 曲线 [上网业务-手机上网整体满意度]

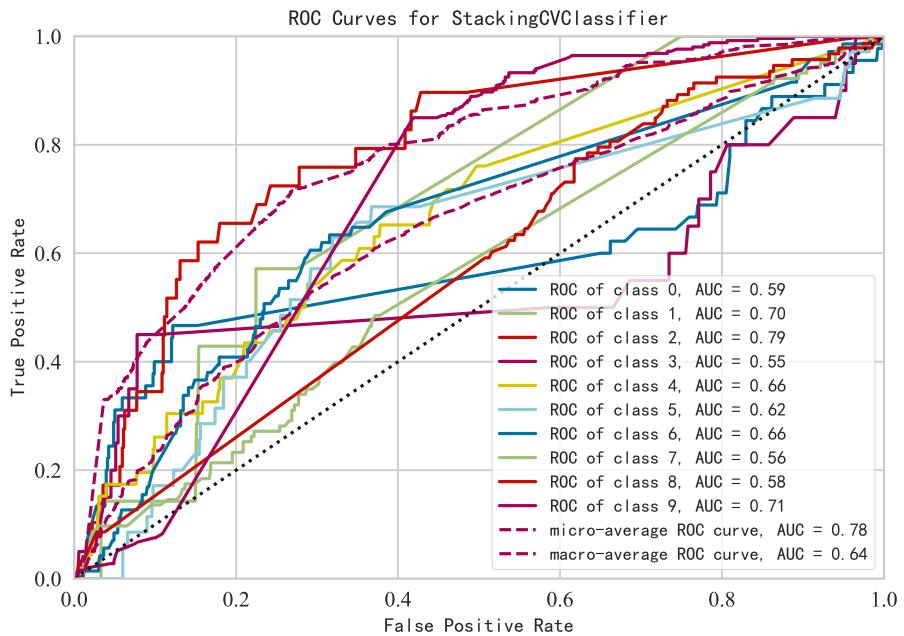


图 66 模型六 ROC/AUC 曲线 [上网业务-网络覆盖与信号强度]

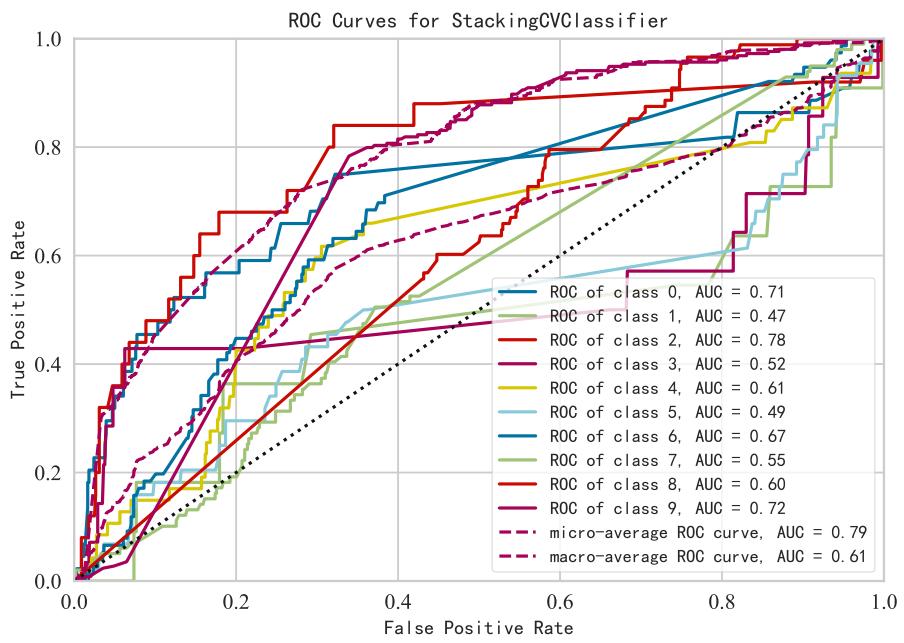


图 67 模型七 ROC/AUC 曲线 [上网业务-手机上网速度]

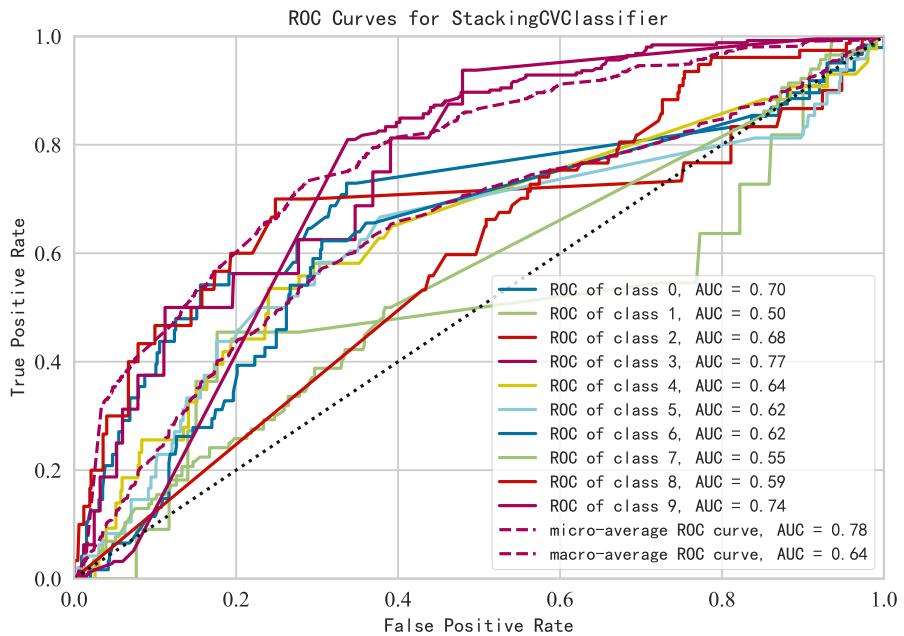


图 68 模型八 ROC/AUC 曲线 [上网业务-手机上网稳定性]

表 12 语音业务熵权法关联分析影响程度量化结果

因素	量化值	因素	量化值
省际漫游-时长 (分钟)	0.1404	有信号无法拨通	0.0398
前 3 月 ARPU	0.1360	办公室	0.0364
套外流量 (MB)	0.1330	地铁	0.0345
GPRS-国内漫游-流量 (KB)	0.1288	通话过程中突然中断	0.0312
套外流量费 (元)	0.1232	手机没有信号	0.0306
脱网次数	0.0848	通话过程中一方听不见	0.0305
其他, 请注明.1	0.0777	通话中有杂音、听不清、断断续续	0.0279
未接通掉话次数	0.0773	GPRS 总流量 (KB)	0.0255
高校	0.0770	居民小区	0.0251
是否投诉	0.0741	语音通话-时长 (分钟)	0.0246
是否关怀用户	0.0700	当月 MOU	0.0246
串线	0.0690	是否 5G 网络客户	0.0222
其他, 请注明	0.0648	前 3 月 MOU	0.0220
外省语音占比	0.0591	当月 ARPU	0.0180
外省流量占比	0.0586	是否遇到过网络问题	0.0150
商业街	0.0560	4\5G 用户	0.0045
农村	0.0535	终端品牌	0.0039
mos 质差次数	0.0512	客户星级标识	0.0021
高铁	0.0469	是否 4G 网络客户 (本地剔除物联网)	0.0001

表 13 语音业务灰度关联分析影响程度量化结果

因素	量化值	因素	量化值
前 3 月 ARPU	0.9920	商业街	0.9424
套外流量 (MB)	0.9918	农村	0.9351
套外流量费 (元)	0.9890	mos 质差次数	0.9275
GPRS-国内漫游-流量 (KB)	0.9878	高铁	0.9123
脱网次数	0.9858	有信号无法拨通	0.8797
省际漫游-时长 (分钟)	0.9856	办公室	0.8603
其他, 请注明.1	0.9827	当月 ARPU	0.8487
高校	0.9819	地铁	0.8482
未接通掉话次数	0.9790	通话过程中突然中断	0.8244
GPRS 总流量 (KB)	0.9781	手机没有信号	0.8200
是否投诉	0.9777	通话过程中一方听不见	0.8192
是否关怀用户	0.9716	通话中有杂音、听不清、断断续续	0.7973
串线	0.9699	居民小区	0.7711
前 3 月 MOU	0.9683	是否 5G 网络客户	0.7416
语音通话-时长 (分钟)	0.9671	是否遇到过网络问题	0.6471
当月 MOU	0.9671	客户星级标识	0.4986
其他, 请注明	0.9626	终端品牌	0.4616
外省语音占比	0.9556	4\5G 用户	0.4386
外省流量占比	0.9546	是否 4G 网络客户 (本地剔除物联网)	0.3375

表 14 上网业务主要因素及其量化结果, EWM 及灰色关联度分析

因素	EWM 量化	灰色关联度量化	因素	EWM 量化	灰色关联度量化
套外流量 (MB)	0.0238	0.9891	新浪微博	0.0164	0.9415
其他, 请注明.5	0.0238	0.9893	爱奇艺	0.0164	0.9948
火山	0.0238	0.9895	梦幻诛仙	0.0160	0.9370
全部都卡顿	0.0238	0.9897	脱网次数	0.0155	0.9323
拼多多	0.0236	0.9863	打开网页或 APP 图片慢	0.0139	0.9129
王者荣耀	0.0235	0.9856	部落冲突	0.0136	0.9087
优酷	0.0234	0.9852	是否不限量套餐到达用户	0.0132	0.9040
咪咕视频	0.0234	0.9851	商业街	0.0132	0.9039
全部游戏都卡顿	0.0232	0.9918	套外流量费 (元)	0.0132	0.9036
京东	0.0231	0.9839	上网质差次数	0.0129	0.8989
性别	0.0231	0.9920	搜狐视频	0.0127	0.8960
看视频卡顿	0.0229	0.9829	抖音	0.0118	0.9865
地铁	0.0228	0.9824	快手	0.0112	0.8690
其他, 请注明.2	0.0220	0.9930	手机支付较慢	0.0104	0.9929
上网过程中网络时断时续或时快时慢	0.0207	0.9721	梦幻西游	0.0103	0.9893
	0.0207	0.9718	办公室	0.0097	0.8380
客户星级标识	0.0207	0.9718	其他, 请注明.4	0.0091	0.8232
手机 QQ	0.0206	0.9717	百度	0.0087	0.9909
下载速度慢	0.0206	0.9715	淘宝	0.0081	0.7952
腾讯视频	0.0205	0.9709	显示有信号上不了网	0.0080	0.7898
龙之谷	0.0203	0.9938	质差总	0.0077	0.9864
高校	0.0202	0.9689	手机上网速度慢	0.0076	0.7778
微信质差次数	0.0201	0.9685	其他, 请注明	0.0072	0.9899
炉石传说	0.0200	0.9939	居民小区	0.0066	0.7422
农村	0.0196	0.9654	全部网页或 APP 都慢	0.0065	0.7389
其他, 请注明.3	0.0189	0.9612	终端品牌	0.0056	0.7044
今日头条	0.0187	0.9601	是否 5G 网络客户	0.0054	0.6964
网络信号差/没有信号	0.0183	0.9944	出现问题场所或应用总	0.0049	0.8886
	0.0182	0.9561	网络卡速度慢延时大上不了网总	0.0047	0.7801
欢乐斗地主	0.0179	0.9945	穿越火线	0.0041	0.9102
阴阳师	0.0175	0.9507	其他, 请注明.1	0.0028	0.5579
高铁	0.0175	0.9507	芒果 TV	0.0015	0.8814
当月 MOU	0.0175	0.9507	微信	0.0003	0.5094
打游戏延时大	0.0172	0.9485			
和平精英	0.0166	0.9431			

[B] 支撑文件列表

支撑文件列表如下（列表中不包含原始数据集）：

文件（夹）名	描述
result.xlsx	用户评分预测结果
所有量化结果.xlsx	问题一量化结果
模型参数.xlsx	各个模型评估参数以及模型选择依据
语音业务词云.txt	语音业务词云图文本内容
上网业务词云.txt	上网业务词云图文本内容
语音业务数据分析.ipynb	语音业务分析 Jupyter 文件
上网业务数据分析.ipynb	上网业务分析 Jupyter 文件
语音业务数据分析.html	语音业务分析运行结果
上网业务数据分析.html	上网业务分析运行结果
bg.jpg	词语底图
figuresNightingaleRoseDiagramF.py	原始数据用户评分南丁格尔玫瑰图程序
figuresNightingaleRoseDiagramP.py	预测数据用户评分南丁格尔玫瑰图程序
figuresOne	语音业务所有图示文件夹
figuresTwo	上网业务所有图示文件夹
figuresNightingaleRoseDiagramF	原始数据用户评分南丁格尔玫瑰图示（八项评分）
figuresNightingaleRoseDiagramP	预测数据用户评分南丁格尔玫瑰图示（八项评分）

[C] 使用的软件、环境

为解决该问题，我们所使用的主要软件有：

- TeX Live 2022
- Visual Studio Code 1.74.2
- WPS Office 2022 冬季更新 (13703)
- Python 3.10.4
- Pycharm Professional 2022.3

Python 环境下所用使用到的库及其版本如下：

库	版本	库	版本
copy	内置库	missingno	0.5.1
jieba	0.42.1	mlxtend	0.20.2
jupyter	1.0.0	numpy	1.22.4+mkl
jupyter-client	7.3.1	openpyxl	3.0.10
jupyter-console	6.4.3	pandas	1.4.2
jupyter-contrib-core	0.4.0	pyecharts	1.9.1
jupyter-contrib-nbextensions	0.5.1	scikit-learn	0.22.2.post1
jupyter-core	4.10.0	seaborn	0.11.2
jupyter-highlight-selected-word	0.2.0	sklearn	0.0
jupyterlab-pygments	0.2.2	snapshot_phantomjs	0.0.3
jupyterlab-widgets	1.1.0	warnings	内置库
jupyter-latex-envs	1.4.6	wordcloud	1.8.1
jupyter-nbextensions-configuration	0.5.0	xgboost	1.6.1
matplotlib	3.5.2	yellowbrick	1.4

[D] 问题解决源程序

D.1 语音业务分析代码 [针对附件 1 与附件 3]

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # 2022 MathorCup 大数据 IssueB
5
6 # # 语音业务数据分析
7
8 # ## 初步导入相关第三方库
9
10 # In[1]:
11
12
13 import pandas as pd
14 import numpy as np
15 import sklearn.preprocessing as sp
16 import warnings
17
18 warnings.filterwarnings("ignore")
19
20 # ## 读取附件1与附件3
21
22 # In[2]:
23
24
25 dataOne = pd.read_excel("附件1语音业务用户满意度数据.xlsx", sheet_name='Sheet1')
26 dataThree = pd.read_excel("附件3语音业务用户满意度预测数据.xlsx", sheet_name='语音')
27
28 # In[3]:
29
30
31 dataOne
32
33 # In[4]:
34
35
36 dataThree
37
38 # ## 处理附件1与附件3
39
40 # ### 查看附件1与附件3表头
41
42 # In[5]:
```

```
43
44
45 dataOneColumnsList = list(dataOne.columns)
46 dataThreeColumnsList = list(dataThree.columns)
47
48 # In[6]:
49
50
51 dataOneColumnsList
52
53 # In[7]:
54
55
56 dataThreeColumnsList
57
58 # In[8]:
59
60
61 set(dataOneColumnsList) & set(dataThreeColumnsList)
62
63 # In[9]:
64
65
66 set(dataOneColumnsList) - set(dataThreeColumnsList)
67
68 # ### 对附件1增加一项指标[是否投诉]，来源于[家宽投诉 与 资费投诉]，并删除[家宽投诉 与
    资费投诉]
69
70 # In[10]:
71
72
73 dataOne['资费投诉'] = dataOne.loc[:, ['家宽投诉', '资费投诉']].apply(lambda x1: x1.
    sum(), axis=1)
74 dataOne.drop(['家宽投诉'], axis=1, inplace=True)
75 dataOne.rename(columns={'资费投诉': '是否投诉'}, inplace=True)
76 dataOne
77
78 # In[11]:
79
80
81 dataOneColumnsList = list(dataOne.columns)
82 dataOneColumnsList
83
84 # In[12]:
```

```

85
86
87     dataThreeColumnsList = list(dataThree.columns)
88     dataThreeColumnsList
89
90     # In[13]:
91
92
93     set(dataOneColumnsList) - set(dataThreeColumnsList)
94
95     # ### 剔除附件1中在附件3中没有的列指标，以及剔除四项不重要列
96
97     # In[14]:
98
99
100    dataOne.drop(['用户id', '用户描述', '用户描述.1', '重定向次数', '重定向驻留时长', '语音方式', '是否去过营业厅', 'ARPU(家庭宽带)', '是否实名登记用户', '当月欠费金额', '前第3个月欠费金额', '终端品牌类型'], axis=1, inplace=True)
101    dataOne
102
103    # ### 填补空缺值、数据利于理解化、清洗处理
104
105    # In[15]:
106
107
108    dataOne.info()
109
110    # In[16]:
111
112
113    dataOne.isnull().sum()
114
115    # In[17]:
116
117
118    dataOne['外省流量占比'] = dataOne['外省流量占比'].fillna(0)
119    dataOne["是否关怀用户"] = dataOne["是否关怀用户"].fillna(0)
120    dataOne["外省流量占比"] = dataOne["外省流量占比"].astype(str).replace('%', '')
121    dataOne["外省语音占比"] = dataOne["外省语音占比"].astype(str).replace('%', '')
122    dataOne
123
124    # In[18]:
125
126

```

```

127 dataOne.replace({"是否遇到过网络问题": {2: 0}, "居民小区": {-1: 0}, "办公室": {-1: 0,
128     2: 1}, "高校": {-1: 0, 3: 1}, "商业街": {-1: 0, 4: 1}, "地铁": {-1: 0, 5: 1}, "农村": {-1: 0, 6: 1}, "高铁": {-1: 0, 7: 1}, "其他, 请注明": {-1: 0, 98: 1}, "手机没有信号": {-1: 0}, "有信号无法拨通": {-1: 0, 2: 1}, "通话过程中突然中断": {-1: 0, 3: 1}, "通话中有杂音、听不清、断断续续": {-1: 0, 4: 1}, "串线": {-1: 0, 5: 1}, "通话过程中一方听不见": {-1: 0, 6: 1}, "其他, 请注明.1": {-1: 0, 98: 1}, "是否关怀用户": {"是": 1}, "是否4G网络客户(本地剔除物联网)": {"是": 1, "否": 0}, "是否5G网络客户": {"是": 1, "否": 0}, "客户星级标识": {"未评级": 0, "准星": 1, "一星": 2, "二星": 3, "三星": 4, "银卡": 5, "金卡": 6, "白金卡": 7, "钻石卡": 8}}, inplace=True)
129
130 dataOne
131
132
133 # In[19]:
134
135 # ### 空缺值可视化
136
137 # In[20]:
138
139
140 import missingno
141 import matplotlib.pyplot as plt
142
143 plt.rcParams['font.sans-serif'] = ['SimHei']
144 plt.rcParams['axes.unicode_minus'] = False
145 missingno.bar(dataOne, color='blue')
146 plt.tight_layout()
147
148 # In[21]:
149
150
151 missingno.matrix(dataOne, color=(190 / 255, 190 / 255, 190 / 255))
152 plt.savefig('figuresOne\\[附件1]附件1空缺值可视化.pdf', bbox_inches='tight')
153
154 # ### 空缺值处理
155
156 # In[22]:
157
158
159 dataOneMiss = dataOne.isnull()
160 dataOne[dataOneMiss.any(axis=1) == True]
161

```

```

162 # In[23]:
163
164
165 dataOne.dropna(inplace=True)
166 dataOne = dataOne.reset_index(drop=True)
167 dataOne
168
169 # In[24]:
170
171
172 dataOne.dtypes
173
174 # ### 格式转化
175
176 # In[25]:
177
178
179 dataOne['外省语音占比'] = dataOne['外省语音占比'].astype('float64')
180 dataOne['外省流量占比'] = dataOne['外省流量占比'].astype('float64')
181 dataOne['是否4G网络客户（本地剔除物联网）'] = dataOne['是否4G网络客户（本地剔除物联网）'].astype('int64')
182 dataOne['4\5G用户'] = dataOne['4\5G用户'].astype(str)
183 dataOne['终端品牌'] = dataOne['终端品牌'].astype(str)
184 dataOne
185
186 # ### 标签编码，包括四项评分，视为分类问题
187
188 # In[26]:
189
190
191 le = sp.LabelEncoder()
192
193 OverallSatisfactionVoiceCalls = le.fit_transform(dataOne["语音通话整体满意度"])
194 NetworkCoverageSignalStrength = le.fit_transform(dataOne["网络覆盖与信号强度"])
195 VoiceCallDefinition = le.fit_transform(dataOne["语音通话清晰度"])
196 VoiceCallStability = le.fit_transform(dataOne["语音通话稳定性"])
197
198 FourFiveUser = le.fit_transform(dataOne["4\5G用户"])
199 TerminalBrand = le.fit_transform(dataOne["终端品牌"])
200
201 dataOne["语音通话整体满意度"] = pd.DataFrame(OverallSatisfactionVoiceCalls)
202 dataOne["网络覆盖与信号强度"] = pd.DataFrame(NetworkCoverageSignalStrength)
203 dataOne["语音通话清晰度"] = pd.DataFrame(VoiceCallDefinition)
204 dataOne["语音通话稳定性"] = pd.DataFrame(VoiceCallStability)

```

```

205
206 dataOne["4\\5G用户"] = pd.DataFrame(FourFiveUser)
207 dataOne["终端品牌"] = pd.DataFrame(TerminalBrand)
208 dataOne
209
210
211 # ### 处理"是否投诉"指标
212
213 # In[27]:
214
215
216 def complain(x):
217     if x != 0:
218         return 1
219     else:
220         return 0
221
222
223 for i in range(len(dataOne)):
224     dataOne.loc[i, '是否投诉'] = complain(dataOne.loc[i, '是否投诉'])
225
226 dataOne
227
228 # In[28]:
229
230
231 dataOne.dtypes
232
233 # ### 格式转化
234
235 # In[29]:
236
237
238 dataOne['是否5G网络客户'] = dataOne['是否5G网络客户'].astype('int64')
239 dataOne['客户星级标识'] = dataOne['客户星级标识'].astype('int64')
240 dataOne
241
242 # In[30]:
243
244
245 dataOne.describe()
246
247 # ### 数据可视化
248

```

```

249 # In[31]:
250
251
252 import matplotlib.pyplot as plt
253
254 plt.rcParams['font.sans-serif'] = ['SimHei']
255 plt.rcParams['axes.unicode_minus'] = False
256
257 box_data = dataOne[['语音通话整体满意度', '网络覆盖与信号强度', '语音通话清晰度', '语音
    通话稳定性', ]]
258 plt.grid(True)
259 plt.boxplot(box_data,
260             notch=True,
261             sym="b+",
262             vert=False,
263             showmeans=True,
264             labels=['语音通话整体满意度', '网络覆盖与信号强度', '语音通话清晰度', '语音
    通话稳定性', ])
265 plt.yticks(size=14)
266 plt.xticks(size=14, font='Times New Roman')
267 plt.tight_layout()
268 plt.savefig('figuresOne\\[附件1] [语音通话整体满意度、网络覆盖与信号强度、语音通话清晰
    度、语音通话稳定性] 评分箱线图.pdf')
269
270 # In[32]:
271
272
273 import seaborn as sns
274
275 plt.style.use('ggplot')
276 sns.set_style('whitegrid')
277 plt.rcParams['font.sans-serif'] = ['SimHei']
278 plt.rcParams['axes.unicode_minus'] = False
279 CorrDataOneAll = dataOne.corr().abs()
280 N = 14
281 ColDataOneRange = CorrDataOneAll.nlargest(N, '语音通话整体满意度')[['语音通话整体满意
    度']].index
282 plt.subplots(figsize=(N, N))
283 plt.title('皮尔逊相关系数', size=16)
284 sns.heatmap(dataOne[ColDataOneRange].corr(),
285               linewidths=0.1,
286               vmax=1.0,
287               square=True,
288               cmap=plt.cm.winter,

```

```

289         linecolor='white',
290         annot=True,
291         annot_kws={"size": 12})
292 plt.xticks(fontsize=14)
293 plt.yticks(fontsize=14)
294 plt.tight_layout()
295 plt.savefig('figuresOne\\[附件1]皮尔逊相关系数（14个）.pdf')
296
297 # In[33]:
298
299
300 from yellowbrick.features.radviz import RadViz
301
302 plt.rcParams['font.sans-serif'] = ['SimHei']
303 plt.rcParams['axes.unicode_minus'] = False
304
305 x = dataOne[['是否遇到过网络问题', '居民小区', '手机没有信号', '有信号无法拨通', '通话过程中突然中断', '办公室', '通话过程中一方听不见', '通话中有杂音、听不清、断断续续', '商业街', '地铁']]
306 y = dataOne['语音通话整体满意度']
307
308 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
309 visualizer = RadViz(classes=classes, colormap='winter_r')
310 visualizer.fit(x, y)
311 visualizer.transform(x)
312 visualizer.show(outpath='figuresOne\\[附件1]语音通话整体满意度RidViz.pdf')
313
314 # In[34]:
315
316
317 from yellowbrick.features.radviz import RadViz
318
319 plt.rcParams['font.sans-serif'] = ['SimHei']
320 plt.rcParams['axes.unicode_minus'] = False
321
322 x = dataOne[['是否遇到过网络问题', '居民小区', '手机没有信号', '有信号无法拨通', '通话过程中突然中断', '办公室', '通话过程中一方听不见', '通话中有杂音、听不清、断断续续', '商业街', '地铁']]
323 y = dataOne['网络覆盖与信号强度']
324
325 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
326 visualizer = RadViz(classes=classes, colormap='winter_r')
327 visualizer.fit(x, y)
328 visualizer.transform(x)

```

```

329     visualizer.show(outpath='figuresOne\\[附件1] 网络覆盖与信号强度RidViz.pdf')
330
331 # In[35]:
332
333
334 from yellowbrick.features.radviz import RadViz
335
336 plt.rcParams['font.sans-serif'] = ['SimHei']
337 plt.rcParams['axes.unicode_minus'] = False
338
339 x = dataOne[['是否遇到过网络问题', '居民小区', '手机没有信号', '有信号无法拨通', '通话过程中突然中断', '办公室', '通话过程中一方听不见', '通话中有杂音、听不清、断断续续', '商业街', '地铁']]
340 y = dataOne['语音通话清晰度']
341
342 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
343 visualizer = RadViz(classes=classes, colormap='winter_r')
344 visualizer.fit(x, y)
345 visualizer.transform(x)
346 visualizer.show(outpath='figuresOne\\[附件1] 语音通话清晰度RidViz.pdf')
347
348 # In[36]:
349
350
351 from yellowbrick.features.radviz import RadViz
352
353 plt.rcParams['font.sans-serif'] = ['SimHei']
354 plt.rcParams['axes.unicode_minus'] = False
355
356 x = dataOne[['是否遇到过网络问题', '居民小区', '手机没有信号', '有信号无法拨通', '通话过程中突然中断', '办公室', '通话过程中一方听不见', '通话中有杂音、听不清、断断续续', '商业街', '地铁']]
357 y = dataOne['语音通话稳定性']
358
359 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
360 visualizer = RadViz(classes=classes, colormap='winter_r')
361 visualizer.fit(x, y)
362 visualizer.transform(x)
363 visualizer.show(outpath='figuresOne\\[附件1] 语音通话稳定性RidViz.pdf')
364
365 # ### 数据标准化
366
367 # In[37]:
368

```

```

369
370 StandardTransform = dataOne[['脱网次数', 'mos质差次数', '未接通掉话次数', '4\5G用户',
   , '套外流量 (MB)', '套外流量费 (元)', '语音通话-时长 (分钟)', '省际漫游-时长 (分钟)',
   , '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU', '前3月MOU', 'GPRS总流量 (KB)', 'GPRS-国内漫游-流量 (KB)', '客户星级标识']]
371 StandardTransformScaler = sp.StandardScaler()
372 StandardTransformScaler = StandardTransformScaler.fit(StandardTransform)
373 StandardTransform = StandardTransformScaler.transform(StandardTransform)
374 StandardTransform = pd.DataFrame(StandardTransform)
375 StandardTransform.columns = ['脱网次数', 'mos质差次数', '未接通掉话次数', '4\5G用户',
   , '套外流量 (MB)', '套外流量费 (元)', '语音通话-时长 (分钟)', '省际漫游-时长 (分钟)',
   , '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU', '前3月MOU', 'GPRS总流量 (KB)', 'GPRS-国内漫游-流量 (KB)', '客户星级标识']
376 StandardTransform
377
378 # In[38]:
379
380
381 dataOneLeave = dataOne.loc[:, ~dataOne.columns.isin(['脱网次数', 'mos质差次数', '未
   接通掉话次数', '4\5G用户', '套外流量 (MB)', '套外流量费 (元)', '语音通话-时长
   (分钟)', '省际漫游-时长 (分钟)', '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU',
   , '前3月MOU', 'GPRS总流量 (KB)', 'GPRS-国内漫游-流量 (KB)', '客户星级标识'])]
382
383 # In[39]:
384
385
386 dataOneNewStandard = pd.concat([dataOneLeave, StandardTransform], axis=1)
387 dataOneNewStandard
388
389 # In[40]:
390
391
392 dataOneNewStandard.columns = ['语音通话整体满意度', '网络覆盖与信号强度', '语音通话清
   晰度', '语音通话稳定性', '是否遇到过网络问题', '居民小区', '办公室', '高校', '商业
   街', '地铁', '农村', '高铁', '其他, 请注明', '手机没有信号', '有信号无法拨通', '通
   话过程中突然中断', '通话中有杂音、听不清、断断续续', '串线', '通话过程中一方听不见',
   , '其他, 请注明.1', '是否投诉', '是否关怀用户', '是否4G网络客户 (本地剔除物联网)', ,
   '外省语音占比', '外省流量占比', '是否5G网络客户', '脱网次数', 'mos质差次数', '未
   接通掉话次数', '4\5G用户', '套外流量 (MB)', '套外流量费 (元)', '语音通话-时长
   (分钟)', '省际漫游-时长 (分钟)', '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU',
   , '前3月MOU', 'GPRS总流量 (KB)', 'GPRS-国内漫游-流量 (KB)', '客户星级标识']
393 dataOneNewStandard
394 # ### 数据归一化

```

```

396
397 # In[41]:
398
399
400 MinMaxTransform = dataOne[['脱网次数', 'mos质差次数', '未接通掉话次数', '4\5G用户',
401     '套外流量(MB)', '套外流量费(元)', '语音通话-时长(分钟)', '省际漫游-时长(分
402     钟)', '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU', '前3月MOU', 'GPRS总流量(
403     KB)', 'GPRS-国内漫游-流量(KB)', '客户星级标识']]
404 MinMaxTransformScaler = spMinMaxScaler()
405 MinMaxTransformScaler = MinMaxTransformScaler.fitMinMaxTransform)
406 MinMaxTransform = MinMaxTransformScaler.transformMinMaxTransform)
407 MinMaxTransform = pd.DataFrameMinMaxTransform)
408 MinMaxTransform.columns = ['脱网次数', 'mos质差次数', '未接通掉话次数', '4\5G用户',
409     '套外流量(MB)', '套外流量费(元)', '语音通话-时长(分钟)', '省际漫游-时长(分
410     钟)', '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU', '前3月MOU', 'GPRS总流量(
411     KB)', 'GPRS-国内漫游-流量(KB)', '客户星级标识']
412 MinMaxTransform
413
414 # In[42]:
415
416
417 dataOneNewMinMax = pd.concat([dataOneLeave, MinMaxTransform], axis=1)
418 dataOneNewMinMax.columns = ['语音通话整体满意度', '网络覆盖与信号强度', '语音通话清晰
419     度', '语音通话稳定性', '是否遇到过网络问题', '居民小区', '办公室', '高校', '商业街
420     ', '地铁', '农村', '高铁', '其他, 请注明', '手机没有信号', '有信号无法拨通', '通话过程中突然中断',
421     '通话中有杂音、听不清、断断续续', '串线', '通话过程中一方听不见',
422     '其他, 请注明.1', '是否投诉', '是否关怀用户', '是否4G网络客户(本地剔除物联网)', '外省语音占比',
423     '外省流量占比', '是否5G网络客户', '脱网次数', 'mos质差次数', '未接
424     通掉话次数', '4\5G用户', '套外流量(MB)', '套外流量费(元)', '语音通话-时长(分
425     钟)', '省际漫游-时长(分钟)', '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU',
426     '前3月MOU', 'GPRS总流量(KB)', 'GPRS-国内漫游-流量(KB)', '客户星级标识']
427 dataOneNewMinMax
428
429 # ## 熵权法
430
431 # In[43]:
432
433
434 import copy
435
436
437 def ewm(data):
438     label_need = data.keys()[:]
439     data1 = data[label_need].values

```

```

426     data2 = data1
427     [m, n] = data2.shape
428     data3 = copy.deepcopy(data2)
429     y_min = 0.002
430     y_max = 1
431     for j in range(0, n):
432         d_max = max(data2[:, j])
433         d_min = min(data2[:, j])
434         data3[:, j] = (y_max - y_min) * (data2[:, j] - d_min) / (d_max - d_min) +
435             y_min
436     p = copy.deepcopy(data3)
437     for j in range(0, n):
438         p[:, j] = data3[:, j] / sum(data3[:, j])
439     e = copy.deepcopy(data3[0, :])
440     for j in range(0, n):
441         e[j] = -1 / np.log(m) * sum(p[:, j] * np.log(p[:, j]))
442     w = (1 - e) / sum(1 - e)
443     total = 0
444     for sum_w in range(0, len(w)):
445         total = total + w[sum_w]
446     print(f'权重为: {w}, 权重之和为: {total}')
447
448 # In[44]:
449
450
451     ewm(dataOneLeave.iloc[:, 4:])
452
453 # In[45]:
454
455
456     dataOneTransform = dataOne[['脱网次数', 'mos质差次数', '未接通掉话次数', '4\\5G用户',
457         '套外流量 (MB)', '套外流量费 (元)', '语音通话-时长 (分钟)', '省际漫游-时长 (分钟)',
458         '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU', '前3月MOU', 'GPRS总流量 (KB)',
459         'GPRS-国内漫游-流量 (KB)', '客户星级标识']]
460
461     dataOneTransform
462
463
464 # In[46]:
465
466
467     ewm(dataOneTransform)
468
469
470 # ## 灰色关联

```

```

466
467 # In[47]:
468
469
470 def grey(data):
471     label_need = data.keys() [:]
472     data1 = data[label_need].values
473     [m, n] = data1.shape
474     data2 = data1.astype('float')
475     data3 = data2
476     ymin = 0.002
477     ymax = 1
478     for j in range(0, n):
479         d_max = max(data2[:, j])
480         d_min = min(data2[:, j])
481         data3[:, j] = (ymax - ymin) * (data2[:, j] - d_min) / (d_max - d_min) + ymin
482
483     for i in range(0, n):
484         data3[:, i] = np.abs(data3[:, i] - data3[:, 0])
485     data4 = data3
486     d_max = np.max(data4)
487     d_min = np.min(data4)
488     a = 0.5
489     data4 = (d_min + a * d_max) / (data4 + a * d_max)
490     xs = np.mean(data4, axis=0)
491     print(xs)
492
493
494 # In[48]:
495
496
497 grey(dataOne.loc[:, ~dataOne.columns.isin(['网络覆盖与信号强度', '语音通话清晰度', '语音通话稳定性'])])
498
499 # In[49]:
500
501
502 grey(dataOne.loc[:, ~dataOne.columns.isin(['语音通话整体满意度', '语音通话清晰度', '语音通话稳定性'])])
503
504 # In[50]:
505
506
507 grey(dataOne.loc[:, ~dataOne.columns.isin(['语音通话整体满意度', '网络覆盖与信号强度',

```

```

    , '语音通话稳定性'])])
508
509 # In[51]:
510
511
512 grey(dataOne.loc[:, ~dataOne.columns.isin(['语音通话整体满意度', '网络覆盖与信号强度',
513     , '语音通话清晰度'])])
514
515 # ## 特征工程与机器学习
516
517 # ### 多输出多类别分类
518
519
520
521 XdataOneMulti = dataOneNewStandard.loc[:, ~dataOneNewStandard.columns.isin(['语音通
522    话整体满意度', '网络覆盖与信号强度', '语音通话清晰度', '语音通话稳定性'])]
523 ydataOneMulti = dataOneNewStandard[['语音通话整体满意度', '网络覆盖与信号强度', '语音
524    通话清晰度', '语音通话稳定性']]]
525
526
527 from sklearn.model_selection import train_test_split
528
529 XdataOneMulti_train, XdataOneMulti_test, ydataOneMulti_train, ydataOneMulti_test =
530     train_test_split(XdataOneMulti, ydataOneMulti, test_size=0.2, random_state=2022)
531
532 # In[54]:
533
534
535 from sklearn.tree import DecisionTreeClassifier
536 from sklearn.ensemble import RandomForestClassifier
537
538 DecisionTreeMulti = DecisionTreeClassifier(random_state=2022)
539 RandomForestMulti = RandomForestClassifier(random_state=2022)
540 DecisionTreeMulti = DecisionTreeMulti.fit(XdataOneMulti_train, ydataOneMulti_train)
541 RandomForestMulti = RandomForestMulti.fit(XdataOneMulti_train, ydataOneMulti_train)
542
543 # In[55]:
544
545 from sklearn.metrics import mean_absolute_error
546 from sklearn.metrics import mean_squared_error

```

```

547
548 print(f'决策树平均绝对误差: ,
549     f'{mean_absolute_error(ydataOneMulti_test, DecisionTreeMulti.predict(
550         XdataOneMulti_test), sample_weight=None, multioutput="uniform_average")}\n
551         ,
552         f'决策树均方误差: ,
553         f'{mean_squared_error(ydataOneMulti_test, DecisionTreeMulti.predict(
554             XdataOneMulti_test), sample_weight=None, multioutput="uniform_average")})')
555 print(f'随机森林平均绝对误差: ,
556     f'{mean_absolute_error(ydataOneMulti_test, RandomForestMulti.predict(
557         XdataOneMulti_test), sample_weight=None, multioutput="uniform_average")}\n
558         ,
559         f'随机森林均方误差: ,
560         f'{mean_squared_error(ydataOneMulti_test, RandomForestMulti.predict(
561             XdataOneMulti_test), sample_weight=None, multioutput="uniform_average")})')
562
563 # In[56]:
564
565 std = np.std([i.feature_importances_ for i in RandomForestMulti.estimators_], axis
566 =0)
567 importances = DecisionTreeMulti.feature_importances_
568 feat_with_importance = pd.Series(importances, XdataOneMulti.columns)
569 fig, ax = plt.subplots(figsize=(12, 5))
570 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 127 / 255, 215 /
571 255))
572 ax.set_title("语音业务四项评分各指标特征重要平均程度")
573 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
574 plt.yticks(font='Times New Roman')
575 plt.tight_layout()
576 plt.savefig('figuresOne\\[附件1]语音业务四项评分各指标特征重要平均程度.pdf')
577
578 # In[57]:
579
580 feat_with_importance
581
582 # In[58]:
583
584 from sklearn.decomposition import PCA
585
586 pca = PCA()
587 pca.fit(dataOneNewStandard)

```

```

583 evr = pca.explained_variance_ratio_
584
585 plt.figure(figsize=(12, 5))
586 plt.plot(range(0, len(evr)), evr.cumsum(), marker="d", linestyle="--")
587 plt.xlabel("Number of components", font='Times New Roman')
588 plt.ylabel("Cumulative explained variance", font='Times New Roman')
589 plt.xticks(font='Times New Roman')
590 plt.yticks(font='Times New Roman')
591 plt.tight_layout()
592 plt.savefig("figuresOne\\[附件1]PCA累计解释方差图.pdf")
593
594 # In[59]:
595
596
597 from sklearn.multioutput import MultiOutputClassifier
598
599 RandomForestMulti = MultiOutputClassifier(RandomForestClassifier(random_state=2022)
600 )
600 RandomForestMulti = RandomForestMulti.fit(XdataOneMulti_train, ydataOneMulti_train)
601 RandomForestMulti_score = RandomForestMulti.score(XdataOneMulti_test,
602 ydataOneMulti_test)
602 print(f'随机森林平均绝对误差: ,
603 f'{mean_absolute_error(ydataOneMulti_test, RandomForestMulti.predict(
604 XdataOneMulti_test), sample_weight=None, multioutput="uniform_average")}\n
604 ,
605 f'随机森林均方误差: ,
606 f'{mean_squared_error(ydataOneMulti_test, RandomForestMulti.predict(
607 XdataOneMulti_test), sample_weight=None, multioutput="uniform_average")}'')
608 RandomForestMulti_score
609
610 # ### "语音通话整体满意度"学习
611
612
613 XdataOneFirst = dataOneNewStandard.loc[:, ~dataOneNewStandard.columns.isin(['语音通
614 话整体满意度', '网络覆盖与信号强度', '语音通话清晰度', '语音通话稳定性'])]
614 ydataOneFirst = dataOneNewStandard['语音通话整体满意度']
615 XdataOneFirst_train, XdataOneFirst_test, ydataOneFirst_train, ydataOneFirst_test =
616 train_test_split(XdataOneFirst, ydataOneFirst, test_size=0.2, random_state=2022)
617
618 # ##### 决策树, 随机森林
619
620 # In[61]:

```

```

620
621
622 DecisionTreeFirst = DecisionTreeClassifier(random_state=2022)
623 RandomForestFirst = RandomForestClassifier(random_state=2022)
624 DecisionTreeFirst = DecisionTreeFirst.fit(XdataOneFirst_train, ydataOneFirst_train)
625 RandomForestFirst = RandomForestFirst.fit(XdataOneFirst_train, ydataOneFirst_train)
626 RandomForestFirst_score = RandomForestFirst.score(XdataOneFirst_test,
627     ydataOneFirst_test)
628 RandomForestFirst_score
629 # In[62]:
630
631
632 std = np.std([i.feature_importances_ for i in RandomForestFirst.estimators_], axis
633 =0)
634 importances = DecisionTreeFirst.feature_importances_
635 feat_with_importance = pd.Series(importances, XdataOneFirst.columns)
636 fig, ax = plt.subplots(figsize=(12, 5))
637 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
638 255))
639 ax.set_title("语音通话整体满意度各项指标重要程度")
640 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
641 plt.yticks(font='Times New Roman')
642 plt.tight_layout()
643 plt.savefig("figuresOne\\[附件1]语音通话整体满意度各项指标重要程度.pdf")
644
645 # In[63]:
646
647
648 feat_with_importance
649 # #### XGBoost
650
651
652 # In[64]:
653
654
655 from xgboost import XGBClassifier
656
657 XGBFirst = XGBClassifier(learning_rate=0.01,
658     n_estimators=14,
659     max_depth=5,
660     min_child_weight=1,
661     gamma=0.,
662     subsample=1,

```

```

661             colsample_btree=1,
662             scale_pos_weight=1,
663             random_state=2022,
664             silent=0)
665 XGBFirst.fit(XdataOneFirst_train, ydataOneFirst_train)
666 XGBFirst_score = XGBFirst.score(XdataOneFirst_test, ydataOneFirst_test)
667 XGBFirst_score
668
669 # In[65]:
670
671
672 from xgboost import plot_importance
673
674 fig, ax = plt.subplots(figsize=(14, 8))
675 plot_importance(XGBFirst, height=0.4, ax=ax)
676 plt.xticks(fontsize=13, font='Times New Roman')
677 plt.yticks(fontsize=11)
678 ax.set_title("")
679 ax.set_ylabel("")
680 ax.set_xlabel("")
681 plt.tight_layout()
682 plt.savefig('figuresOne\\[附件1]语音通话整体满意度各项指标重要程度 (XGBoost,F-score)
683 .pdf')
684
685 # #### KNN
686
687
688
689 from sklearn.neighbors import KNeighborsClassifier
690
691 KNNFirst = KNeighborsClassifier()
692 KNNFirst.fit(XdataOneFirst_train, ydataOneFirst_train)
693 KNNFirst_score = KNNFirst.score(XdataOneFirst_test, ydataOneFirst_test)
694 KNNFirst_score
695
696 # In[67]:
697
698
699 from sklearn.neighbors import KNeighborsClassifier
700 from sklearn.model_selection import GridSearchCV
701
702 KNN_turing_param_grid = [{weights: ['uniform'],
703                           'n_neighbors': [k for k in range(2, 20)]}],

```

```

704             {'weights': ['distance'],
705              'n_neighbors': [k for k in range(2, 20)],
706              'p': [p for p in range(1, 5)]}]
707
708 KNN_turing = KNeighborsClassifier()
709
710 KNN_turing_grid_search = GridSearchCV(KNN_turing,
711                                         param_grid=KNN_turing_param_grid,
712                                         n_jobs=-1,
713                                         verbose=2)
714
715 KNN_turing_grid_search.fit(XdataOneFirst_train, ydataOneFirst_train)
716
717 # In[68]:
718
719 KNN_turing_grid_search.best_score_
720
721 # In[69]:
722
723 KNN_turing_grid_search.best_params_
724
725 # In[70]:
726
727 KNNFirst_new = KNeighborsClassifier(n_neighbors=23, p=1, weights='distance')
728 KNNFirst_new.fit(XdataOneFirst_train, ydataOneFirst_train)
729 KNNFirst_new_score = KNNFirst_new.score(XdataOneFirst_test, ydataOneFirst_test)
730 KNNFirst_new_score
731
732 # #### 支持向量机
733
734 # In[71]:
735
736
737 from sklearn.svm import SVC
738
739 SVMFirst = SVC(random_state=2022)
740 SVMFirst.fit(XdataOneFirst_train, ydataOneFirst_train)
741 SVMFirst_score = SVMFirst.score(XdataOneFirst_test, ydataOneFirst_test)
742 SVMFirst_score
743
744 # #### lightgbm
745
746 # In[72]:
747

```

```

748
749     from lightgbm import LGBMClassifier
750
751     LightgbmFirst = LGBMClassifier(learning_rate=0.1,
752                                     lambda_l1=0.1,
753                                     lambda_l2=0.2,
754                                     max_depth=4,
755                                     objective='multiclass',
756                                     num_class=3,
757                                     random_state=2022)
758
759     LightgbmFirst.fit(XdataOneFirst_train, ydataOneFirst_train)
760     LightgbmFirst_score = LightgbmFirst.score(XdataOneFirst_test, ydataOneFirst_test)
761     LightgbmFirst_score
762
763     # ##### 逻辑回归
764
765
766
767     from sklearn.linear_model import LogisticRegression
768
769     LogisticRegressionFirst = LogisticRegression(multi_class="multinomial", solver="newton-cg", max_iter=1000)
770     LogisticRegressionFirst = LogisticRegressionFirst.fit(XdataOneFirst_train,
771                                                        ydataOneFirst_train)
772     LogisticRegressionFirst_score = LogisticRegressionFirst.score(XdataOneFirst_test,
773                                                                ydataOneFirst_test)
774     LogisticRegressionFirst_score
775
776
777     # In[74]:
778
779     print(f'模型一中RF平均绝对误差: ',
780           f'{mean_absolute_error(ydataOneFirst_test, RandomForestFirst.predict(
781               XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n',
782           ,
783     f'模型一中RF均方误差: ',
784     f'{mean_squared_error(ydataOneFirst_test, RandomForestFirst.predict(
785         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average"))}')
786     print(f'模型一中XGBoost平均绝对误差: ',
787           f'{mean_absolute_error(ydataOneFirst_test, XGBFirst.predict(XdataOneFirst_test
788                               ), sample_weight=None, multioutput="uniform_average")}\n',
789     f'模型一中XGBoost均方误差: ',
790     f'{mean_squared_error(ydataOneFirst_test, XGBFirst.predict(XdataOneFirst_test)

```

```

        , sample_weight=None, multioutput="uniform_average")}'})

785 print(f'模型一中KNN平均绝对误差: ,
786     f'{mean_absolute_error(ydataOneFirst_test, KNNFirst_new.predict(
787         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
788         ,
789         f'模型一中KNN均方误差: ,
790         f'{mean_squared_error(ydataOneFirst_test, KNNFirst_new.predict(
791             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}'')

792 print(f'模型一中SVM平均绝对误差: ,
793     f'{mean_absolute_error(ydataOneFirst_test, SVMFirst.predict(XdataOneFirst_test
794         ), sample_weight=None, multioutput="uniform_average")}\n'
795         f'模型一中SVM均方误差: ,
796         f'{mean_squared_error(ydataOneFirst_test, SVMFirst.predict(XdataOneFirst_test
797             ), sample_weight=None, multioutput="uniform_average")}'')

798 print(f'模型一中LightGBM平均绝对误差: ,
799     f'{mean_absolute_error(ydataOneFirst_test, LightgbmFirst.predict(
800         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
801         ,
802         f'模型一中LightGBM均方误差: ,
803         f'{mean_squared_error(ydataOneFirst_test, LightgbmFirst.predict(
804             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}'')

805 print(f'模型一中LR平均绝对误差: ,
806     f'{mean_absolute_error(ydataOneFirst_test, LogisticRegressionFirst.predict(
807         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
808         ,
809         f'模型一中LR均方误差: ,
810         f'{mean_squared_error(ydataOneFirst_test, LogisticRegressionFirst.predict(
811             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}'')

812 # ##### 集成学习
813
814 # In[75]:
815
816
817 from mlxtend.classifier import StackingCVClassifier
818
819 FirstModel = StackingCVClassifier(
820     classifiers=[LogisticRegressionFirst, XGBFirst, KNNFirst_new, SVMFirst,
821                 LightgbmFirst],
822     meta_classifier=RandomForestClassifier(random_state=2022), random_state=2022, cv
823     =5)
824
825 FirstModel.fit(XdataOneFirst_train, ydataOneFirst_train)
826 FirstModel_score = FirstModel.score(XdataOneFirst_test, ydataOneFirst_test)
827
828 FirstModel_score

```

```

815
816 # In[76]:
817
818
819 print(f'模型一平均绝对误差: ',
820     f'{mean_absolute_error(ydataOneFirst_test, FirstModel.predict(
821         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n',
822     ,
823     f'模型一均方误差: ',
824     f'{mean_squared_error(ydataOneFirst_test, FirstModel.predict(
825         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}')
826
827 # ### "网络覆盖与信号强度"学习
828
829
830 # In[77]:
831
832
833 XdataOneSecond = dataOneNewStandard.loc[:, ~dataOneNewStandard.columns.isin(['语音通话整体满意度', '网络覆盖与信号强度', '语音通话清晰度', '语音通话稳定性'])]
834 ydataOneSecond = dataOneNewStandard['网络覆盖与信号强度']
835 XdataOneSecond_train, XdataOneSecond_test, ydataOneSecond_train,
836     ydataOneSecond_test = train_test_split(XdataOneSecond, ydataOneSecond, test_size
837     =0.2, random_state=2022)
838
839 # ##### 决策树、随机森林
840
841 # In[78]:
842
843 DecisionTreeSecond = DecisionTreeClassifier(random_state=2022)
844 RandomForestSecond = RandomForestClassifier(random_state=2022)
845 DecisionTreeSecond = DecisionTreeSecond.fit(XdataOneSecond_train,
846     ydataOneSecond_train)
847 RandomForestSecond = RandomForestSecond.fit(XdataOneSecond_train,
848     ydataOneSecond_train)
849 RandomForestSecond_score = RandomForestSecond.score(XdataOneSecond_test,
850     ydataOneSecond_test)
851 RandomForestSecond_score
852
853 # In[79]:
854
855 from sklearn.model_selection import cross_val_score

```

```

850     scorel = []
851     for i in range(0, 200, 10):
852         RFC = RandomForestClassifier(n_estimators=i + 1,
853                                     n_jobs=-1,
854                                     random_state=2022)
855         score = cross_val_score(RFC, XdataOneSecond, ydataOneSecond, cv=10).mean()
856         scorel.append(score)
857
858     print(max(scorel), (scorel.index(max(scorel)) * 10) + 1)
859     plt.figure(figsize=[20, 5])
860     plt.plot(range(1, 201, 10), scorel)
861     plt.xticks(fontsize=12, font='Times New Roman')
862     plt.yticks(fontsize=12, font='Times New Roman')
863     plt.tight_layout()
864     plt.savefig("figuresOne\\[附件1]随机森林调参F.pdf")
865
866 # In[80]:
867
868
869     scorel = []
870     for i in range(150, 170):
871         RFC = RandomForestClassifier(n_estimators=i,
872                                     n_jobs=-1,
873                                     random_state=2022)
874         score = cross_val_score(RFC, XdataOneSecond, ydataOneSecond, cv=10).mean()
875         scorel.append(score)
876
877     print(max(scorel), ([*range(150, 170)][scorel.index(max(scorel))]))
878     plt.figure(figsize=[20, 5])
879     plt.plot(range(150, 170), scorel)
880     plt.xticks(fontsize=12, font='Times New Roman')
881     plt.yticks(fontsize=12, font='Times New Roman')
882     plt.tight_layout()
883     plt.savefig("figuresOne\\[附件1]随机森林调参S.pdf")
884
885 # In[81]:
886
887
888 import numpy as np
889
890 param_grid = {'max_features': ['auto', 'sqrt', 'log2']}
891 RFC = RandomForestClassifier(n_estimators=164, random_state=2022)
892 GS = GridSearchCV(RFC, param_grid, cv=10)
893 GS.fit(XdataOneSecond, ydataOneSecond)

```

```

894 GS.best_params_
895
896 # In[82]:
897
898
899 param_grid = {'min_samples_leaf': np.arange(1, 11, 1)}
900 RFC = RandomForestClassifier(n_estimators=164, random_state=2022, max_features='
901 log2')
902 GS = GridSearchCV(RFC, param_grid, cv=10)
903 GS.fit(XdataOneSecond, ydataOneSecond)
904 GS.best_params_
905
906 # In[83]:
907
908
909 param_grid = {'criterion': ['gini', 'entropy']}
910 RFC = RandomForestClassifier(n_estimators=164, random_state=2022, max_features='
911 log2', min_samples_leaf=8)
912 GS = GridSearchCV(RFC, param_grid, cv=10)
913 GS.fit(XdataOneSecond, ydataOneSecond)
914 GS.best_params_
915
916 # In[84]:
917
918
919 from sklearn.model_selection import GridSearchCV
920
921 param_grid = {'max_depth': np.arange(1, 20, 1)}
922 RFC = RandomForestClassifier(n_estimators=164, random_state=2022, max_features='
923 log2', min_samples_leaf=8)
924 GS = GridSearchCV(RFC, param_grid, cv=10)
925 GS.fit(XdataOneSecond, ydataOneSecond)
926 GS.best_params_
927
928 # In[85]:
929
930
931 RandomForestSecond = RandomForestClassifier(n_estimators=164, random_state=2022,
932 min_samples_leaf=8, max_depth=19)
933 RandomForestSecond = RandomForestSecond.fit(XdataOneSecond_train,
934 ydataOneSecond_train)
935 RandomForestSecond_score = RandomForestSecond.score(XdataOneSecond_test,
936 ydataOneSecond_test)
937 RandomForestSecond_score

```

```

932
933 # In[86]:
934
935
936 std = np.std([i.feature_importances_ for i in RandomForestSecond.estimators_], axis
   =0)
937 importances = DecisionTreeSecond.feature_importances_
938 feat_with_importance = pd.Series(importances, XdataOneSecond.columns)
939 fig, ax = plt.subplots(figsize=(12, 5))
940 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
   255))
941 ax.set_title("网络覆盖与信号强度各项指标重要程度")
942 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
943 plt.yticks(font='Times New Roman')
944 plt.tight_layout()
945 plt.savefig("figuresOne\\[附件1] 网络覆盖与信号强度各项指标重要程度.pdf")
946
947 # In[87]:
948
949
950 feat_with_importance
951
952 # #### XGBoost
953
954 # In[88]:
955
956
957 from xgboost import XGBClassifier
958
959 XGBSecond = XGBClassifier(learning_rate=0.02,
   n_estimators=13,
   max_depth=8,
   min_child_weight=1,
   gamma=0.05,
   subsample=1,
   colsample_btree=1,
   scale_pos_weight=1,
   random_state=2022,
   silent=0)
960 XGBSecond.fit(XdataOneSecond_train, ydataOneSecond_train)
961 XGBSecond_score = XGBSecond.score(XdataOneSecond_test, ydataOneSecond_test)
962 XGBSecond_score
963
964
965
966
967
968
969
970
971
972
973 # In[89]:

```

```

974
975
976     from xgboost import plot_importance
977
978     fig, ax = plt.subplots(figsize=(14, 8))
979     plot_importance(XGBSecond, height=0.4, ax=ax)
980     plt.xticks(fontsize=13, font='Times New Roman')
981     plt.yticks(fontsize=11)
982     ax.set_title("")
983     ax.set_ylabel("")
984     ax.set_xlabel("")
985     plt.tight_layout()
986     plt.savefig('figuresOne\\[附件1]网络覆盖与信号强度各项指标重要程度 (XGBoost,F-score)
987                 .pdf')
988
989     # ##### KNN
990
991
992
993     from sklearn.neighbors import KNeighborsClassifier
994
995     KNNSecond = KNeighborsClassifier()
996     KNNSecond.fit(XdataOneSecond_train, ydataOneSecond_train)
997     KNNSecond_score = KNNSecond.score(XdataOneSecond_test, ydataOneSecond_test)
998     KNNSecond_score
999
1000    # In[91]:
1001
1002
1003    from sklearn.neighbors import KNeighborsClassifier
1004    from sklearn.model_selection import GridSearchCV
1005
1006    KNN_turing_param_grid = [{'weights': ['uniform'],
1007                             'n_neighbors': [k for k in range(40, 50)]},
1008                             {'weights': ['distance'],
1009                             'n_neighbors': [k for k in range(40, 50)],
1010                             'p': [p for p in range(1, 5)]}]
1011
1012    KNN_turing = KNeighborsClassifier()
1013    KNN_turing_grid_search = GridSearchCV(KNN_turing,
1014                                           param_grid=KNN_turing_param_grid,
1015                                           n_jobs=-1,
1016                                           verbose=2)
1017
1018    KNN_turing_grid_search.fit(XdataOneSecond_train, ydataOneSecond_train)

```

```

1017
1018 # In[92]:
1019
1020
1021 KNN_turing_grid_search.best_score_
1022
1023 # In[93]:
1024
1025
1026 KNN_turing_grid_search.best_params_
1027
1028 # In[94]:
1029
1030
1031 KNNSecond_new = KNeighborsClassifier(algorithm='auto', leaf_size=30,
1032                                         metric='minkowski',
1033                                         n_jobs=-1,
1034                                         n_neighbors=46, p=1,
1035                                         weights='distance')
1036 KNNSecond_new.fit(XdataOneSecond_train, ydataOneSecond_train)
1037 KNNSecond_new_score = KNNSecond_new.score(XdataOneSecond_test, ydataOneSecond_test)
1038 KNNSecond_new_score
1039
1040 # #### 支持向量机
1041
1042 # In[95]:
1043
1044
1045 from sklearn.svm import SVC
1046
1047 SVMSecond = SVC(random_state=2022)
1048 SVMSecond.fit(XdataOneSecond_train, ydataOneSecond_train)
1049 SVMSecond_score = SVMSecond.score(XdataOneSecond_test, ydataOneSecond_test)
1050 SVMSecond_score
1051
1052 # #### lightgbm
1053
1054 # In[96]:
1055
1056
1057 from lightgbm import LGBMClassifier
1058
1059 LightgbmSecond = LGBMClassifier(learning_rate=0.1,
1060                                 lambda_l1=0.1,

```

```

1061                         lambda_l2=0.2,
1062                         max_depth=3,
1063                         objective='multiclass',
1064                         num_class=3,
1065                         random_state=2022)
1066 LightgbmSecond.fit(XdataOneSecond_train, ydataOneSecond_train)
1067 LightgbmSecond_score = LightgbmSecond.score(XdataOneSecond_test,
1068                                         ydataOneSecond_test)
1069 LightgbmSecond_score
1070
1071 # ##### 逻辑回归
1072 # In[97]:
1073
1074
1075 from sklearn.linear_model import LogisticRegression
1076
1077 LogisticRegressionSecond = LogisticRegression(multi_class="multinomial", solver=
1078                                               "newton-cg", max_iter=2000)
1079 LogisticRegressionSecond = LogisticRegressionSecond.fit(XdataOneSecond_train,
1080                                         ydataOneSecond_train)
1081 LogisticRegressionSecond_score = LogisticRegressionSecond.score(XdataOneSecond_test
1082                                         , ydataOneSecond_test)
1083 LogisticRegressionSecond_score
1084
1085 # In[98]:
1086
1087
1088 print(f'模型二中RF平均绝对误差: ',
1089       f'{mean_absolute_error(ydataOneSecond_test, RandomForestSecond.predict(
1090                               XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\\
1091       n'
1092       f'模型二中RF均方误差: ',
1093       f'{mean_squared_error(ydataOneSecond_test, RandomForestSecond.predict(
1094                               XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\',
1095       )
1096 print(f'模型二中XGBoost平均绝对误差: ',
1097       f'{mean_absolute_error(ydataOneSecond_test, XGBSecond.predict(
1098                               XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\\
1099       n'
1100       f'模型二中XGBoost均方误差: ',
1101       f'{mean_squared_error(ydataOneSecond_test, XGBSecond.predict(
1102                               XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\',
1103       )

```

```

1093     print(f'模型二中KNN平均绝对误差: ,
1094         f'{mean_absolute_error(ydataOneSecond_test, KNNSecond_new.predict(
1095             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\\
1096             n',
1097             f'模型二中KNN均方误差: ,
1098                 f'{mean_squared_error(ydataOneSecond_test, KNNSecond_new.predict(
1099                     XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\',
1100                     )
1101         print(f'模型二中SVM平均绝对误差: ,
1102             f'{mean_absolute_error(ydataOneSecond_test, SVMSecond.predict(
1103                 XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\\
1104                 n',
1105             f'模型二中SVM均方误差: ,
1106                 f'{mean_squared_error(ydataOneSecond_test, SVMSecond.predict(
1107                     XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\',
1108                     )
1109         print(f'模型二中LightGBM平均绝对误差: ,
1110             f'{mean_absolute_error(ydataOneSecond_test, LightgbmSecond.predict(
1111                 XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\\
1112                 n',
1113             f'模型二中LightGBM均方误差: ,
1114                 f'{mean_squared_error(ydataOneSecond_test, LightgbmSecond.predict(
1115                     XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\',
1116                     )
1117         print(f'模型二中LR平均绝对误差: ,
1118             f'{mean_absolute_error(ydataOneSecond_test, LogisticRegressionSecond.predict(
1119                 XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\\
# ##### 集成学习
# In[99]:
from mlxtend.classifier import StackingCVClassifier
SecondModel = StackingCVClassifier(
    classifiers=[RandomForestSecond, XGBSecond, KNNSecond_new, SVMSecond,
        LogisticRegressionSecond],
    meta_classifier=LGBMClassifier(random_state=2022), random_state=2022, cv=5)

```

```

1120 SecondModel.fit(XdataOneSecond_train, ydataOneSecond_train)
1121 SecondModel_score = SecondModel.score(XdataOneSecond_test, ydataOneSecond_test)
1122 SecondModel_score
1123
1124 # In[100]:
1125
1126
1127 print(f'模型二平均绝对误差: ,
1128     f'{mean_absolute_error(ydataOneSecond_test, SecondModel.predict(
1129         XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\ \
1130     n',
1131     f'模型二均方误差: ,
1132     f'{mean_squared_error(ydataOneSecond_test, SecondModel.predict(
1133         XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}',
1134     )
1135
1136
1137 # ### "语音通话清晰度"学习
1138
1139
1140 # In[101]:
1141
1142
1143 XdataOneThird = dataOneNewStandard.loc[:, ~dataOneNewStandard.columns.isin(['语音通
1144    话整体满意度', '网络覆盖与信号强度', '语音通话清晰度', '语音通话稳定性'])]
1145 ydataOneThird = dataOneNewStandard['语音通话清晰度']
1146 XdataOneThird_train, XdataOneThird_test, ydataOneThird_train, ydataOneThird_test =
1147     train_test_split(XdataOneThird, ydataOneThird, test_size=0.2, random_state=2022)
1148
1149
1150 # ##### 决策树、随机森林
1151
1152
1153 # In[102]:
1154
1155
1156 DecisionTreeThird = DecisionTreeClassifier(random_state=2022)
1157 RandomForestThird = RandomForestClassifier(random_state=2022)
1158 DecisionTreeThird = DecisionTreeThird.fit(XdataOneThird_train, ydataOneThird_train)
1159 RandomForestThird = RandomForestThird.fit(XdataOneThird_train, ydataOneThird_train)
1160 RandomForestThird_score = RandomForestThird.score(XdataOneThird_test,
1161     ydataOneThird_test)
1162 RandomForestThird_score
1163
1164
1165 # In[103]:
1166
1167
1168 std = np.std([i.feature_importances_ for i in RandomForestThird.estimators_], axis

```

```

=0)

1157 importances = DecisionTreeThird.feature_importances_
1158 feat_with_importance = pd.Series(importances, XdataOneThird.columns)
1159 fig, ax = plt.subplots(figsize=(12, 5))
1160 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
1161                                         255))
1162 ax.set_title("语音通话清晰度各项指标重要程度")
1163 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
1164 plt.yticks(font='Times New Roman')
1165 plt.tight_layout()
1166 plt.savefig("figuresOne\\[附件1]语音通话清晰度各项指标重要程度.pdf")
1167
1168 # In[104]:
1169
1170 feat_with_importance
1171
1172 # ##### XGBoost
1173
1174 # In[105]:
1175
1176
1177 from xgboost import XGBClassifier
1178
1179 XGBThird = XGBClassifier(learning_rate=0.02,
1180                           n_estimators=14,
1181                           max_depth=8,
1182                           min_child_weight=1,
1183                           gamma=0.05,
1184                           subsample=1,
1185                           colsample_btree=1,
1186                           scale_pos_weight=1,
1187                           random_state=2022,
1188                           silent=0)
1189 XGBThird.fit(XdataOneThird_train, ydataOneThird_train)
1190 XGBThird_score = XGBThird.score(XdataOneThird_test, ydataOneThird_test)
1191 XGBThird_score
1192
1193 # In[106]:
1194
1195
1196 from xgboost import plot_importance
1197 fig, ax = plt.subplots(figsize=(14, 8))

```

```

1199 plot_importance(XGBThird, height=0.4, ax=ax)
1200 plt.xticks(fontsize=13, font='Times New Roman')
1201 plt.yticks(fontsize=11)
1202 ax.set_title(""))
1203 ax.set_ylabel("")
1204 ax.set_xlabel("")
1205 plt.tight_layout()
1206 plt.savefig('figuresOne\\[附件1]语音通话清晰度各项指标重要程度 (XGBoost,F-score) .pdf
1207 ')
1208 # ##### KNN
1209
1210 # In[107]:
1211
1212
1213 from sklearn.neighbors import KNeighborsClassifier
1214
1215 KNNThird = KNeighborsClassifier()
1216 KNNThird.fit(XdataOneThird_train, ydataOneThird_train)
1217 KNNThird_score = KNNThird.score(XdataOneThird_test, ydataOneThird_test)
1218 KNNThird_score
1219
1220 # In[108]:
1221
1222
1223 from sklearn.neighbors import KNeighborsClassifier
1224 from sklearn.model_selection import GridSearchCV
1225
1226 KNN_turing_param_grid = [{'weights': ['uniform'],
1227                             'n_neighbors': [k for k in range(30, 40)]},
1228                             {'weights': ['distance'],
1229                             'n_neighbors': [k for k in range(30, 40)],
1230                             'p': [p for p in range(1, 5)]}]
1231 KNN_turing = KNeighborsClassifier()
1232 KNN_turing_grid_search = GridSearchCV(KNN_turing,
1233                                         param_grid=KNN_turing_param_grid,
1234                                         n_jobs=-1,
1235                                         verbose=2)
1236 KNN_turing_grid_search.fit(XdataOneThird_train, ydataOneThird_train)
1237
1238 # In[109]:
1239
1240
1241 KNN_turing_grid_search.best_score_

```

```

1242
1243 # In[110]:
1244
1245
1246 KNN_turing_grid_search.best_params_
1247
1248 # In[111]:
1249
1250
1251 KNNThird_new = KNeighborsClassifier(algorithm='auto', leaf_size=30,
1252                                         metric='minkowski',
1253                                         n_jobs=-1,
1254                                         n_neighbors=40, p=1,
1255                                         weights='uniform')
1256 KNNThird_new.fit(XdataOneThird_train, ydataOneThird_train)
1257 KNNThird_new_score = KNNThird_new.score(XdataOneThird_test, ydataOneThird_test)
1258 KNNThird_new_score
1259
1260 # #### 支持向量机
1261
1262 # In[112]:
1263
1264
1265 from sklearn.svm import SVC
1266
1267 SVMThird = SVC(random_state=2022)
1268 SVMThird.fit(XdataOneThird_train, ydataOneThird_train)
1269 SVMThird_score = SVMThird.score(XdataOneThird_test, ydataOneThird_test)
1270 SVMThird_score
1271
1272 # #### lightgbm
1273
1274 # In[113]:
1275
1276
1277 from lightgbm import LGBMClassifier
1278
1279 LightgbmThird = LGBMClassifier(learning_rate=0.1,
1280                                 lambda_l1=0.1,
1281                                 lambda_l2=0.2,
1282                                 max_depth=9,
1283                                 objective='multiclass',
1284                                 num_class=4,
1285                                 random_state=2022)

```

```

1286 LightgbmThird.fit(XdataOneThird_train, ydataOneThird_train)
1287 LightgbmThird_score = LightgbmThird.score(XdataOneThird_test, ydataOneThird_test)
1288 LightgbmThird_score
1289
1290 # ##### 逻辑回归
1291
1292 # In[114]:
1293
1294
1295 from sklearn.linear_model import LogisticRegression
1296
1297 LogisticRegressionThird = LogisticRegression(multi_class="multinomial", solver="newton-cg", max_iter=2000)
1298 LogisticRegressionThird = LogisticRegressionThird.fit(XdataOneThird_train,
1299             ydataOneThird_train)
1300 LogisticRegressionThird_score = LogisticRegressionThird.score(XdataOneThird_test,
1301             ydataOneThird_test)
1302 LogisticRegressionThird_score
1303
1304
1305 print(f'模型三中RF平均绝对误差: ,
1306     f'{mean_absolute_error(ydataOneThird_test, RandomForestThird.predict(
1307         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n
1308     ,
1309     f'模型三中RF均方误差: ,
1310     f'{mean_squared_error(ydataOneThird_test, RandomForestThird.predict(
1311         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}')
1312 print(f'模型三中XGBoost平均绝对误差: ,
1313     f'{mean_absolute_error(ydataOneThird_test, XGBThird.predict(XdataOneThird_test
1314         ), sample_weight=None, multioutput="uniform_average")}\n'
1315     f'模型三中XGBoost均方误差: ,
1316     f'{mean_squared_error(ydataOneThird_test, XGBThird.predict(XdataOneThird_test
1317         ), sample_weight=None, multioutput="uniform_average")}')
1318 print(f'模型三中KNN平均绝对误差: ,
1319     f'{mean_absolute_error(ydataOneThird_test, KNNThird_new.predict(
1320         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n
1321     ,
1322     f'模型三中KNN均方误差: ,
1323     f'{mean_squared_error(ydataOneThird_test, KNNThird_new.predict(
1324         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}')
1325 print(f'模型三中SVM平均绝对误差: ,
1326     f'{mean_absolute_error(ydataOneThird_test, SVMThird.predict(XdataOneThird_test

```

```

        ), sample_weight=None, multioutput="uniform_average")}\n'
1319 f'模型三中SVM均方误差: '\n
1320 f'{mean_squared_error(ydataOneThird_test, SVMThird.predict(XdataOneThird_test),
1321                         sample_weight=None, multioutput="uniform_average"))}'
1322 print(f'模型三中LightGBM平均绝对误差: '\n
1323     f'{mean_absolute_error(ydataOneThird_test, LightgbmThird.predict(
1324         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n'
1325     ,\n
1326     f'模型三中LightGBM均方误差: '\n
1327     f'{mean_squared_error(ydataOneThird_test, LightgbmThird.predict(
1328         XdataOneThird_test), sample_weight=None, multioutput="uniform_average"))}')
1329 print(f'模型三中LR平均绝对误差: '\n
1330     f'{mean_absolute_error(ydataOneThird_test, LogisticRegressionThird.predict(
1331         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n'
1332     ,\n
1333     f'模型三中LR均方误差: '\n
1334     f'{mean_squared_error(ydataOneThird_test, LogisticRegressionThird.predict(
1335         XdataOneThird_test), sample_weight=None, multioutput="uniform_average"))}')
1336
1337 # ##### 集成学习
1338
1339 # In[116]:
1340
1341
1342
1343
1344 from mlxtend.classifier import StackingCVClassifier
1345
1346 ThirdModel = StackingCVClassifier(
1347     classifiers=[XGBThird, LightgbmThird, KNNThird_new, SVMThird,
1348                  LogisticRegressionThird],
1349     meta_classifier=RandomForestClassifier(random_state=2022), random_state=2022, cv
1350     =5)
1351
1352 ThirdModel.fit(XdataOneThird_train, ydataOneThird_train)
1353 ThirdModel_score = ThirdModel.score(XdataOneThird_test, ydataOneThird_test)
1354 ThirdModel_score
1355
1356
1357 # In[117]:
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367 print(f'模型三平均绝对误差: '\n
1368     f'{mean_absolute_error(ydataOneThird_test, ThirdModel.predict(
1369         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n'
1370     ,\n
1371     f'模型三均方误差: '\n
1372     f'{mean_squared_error(ydataOneThird_test, ThirdModel.predict(

```

```

1351 XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}]')
1352 # ### "语音通话稳定性"学习
1353
1354 # In[118]:
1355
1356
1357 XdataOneFourth = dataOneNewStandard.loc[:, ~dataOneNewStandard.columns.isin(['语音通
    话整体满意度', '网络覆盖与信号强度', '语音通话清晰度', '语音通话稳定性'])]
1358 ydataOneFourth = dataOneNewStandard['语音通话稳定性']
1359 XdataOneFourth_train, XdataOneFourth_test, ydataOneFourth_train,
    ydataOneFourth_test = train_test_split(XdataOneFourth, ydataOneFourth, test_size
=0.2, random_state=2022)
1360
1361 # ##### 决策树、随机森林
1362
1363 # In[119]:
1364
1365
1366 DecisionTreeFourth = DecisionTreeClassifier(random_state=2022)
1367 RandomForestFourth = RandomForestClassifier(random_state=2022)
1368 DecisionTreeFourth = DecisionTreeFourth.fit(XdataOneFourth_train,
    ydataOneFourth_train)
1369 RandomForestFourth = RandomForestFourth.fit(XdataOneFourth_train,
    ydataOneFourth_train)
1370 RandomForestFourth_score = RandomForestFourth.score(XdataOneFourth_test,
    ydataOneFourth_test)
1371 RandomForestFourth_score
1372
1373 # In[120]:
1374
1375
1376 std = np.std([i.feature_importances_ for i in RandomForestFourth.estimators_], axis
=0)
1377 importances = DecisionTreeFourth.feature_importances_
1378 feat_with_importance = pd.Series(importances, XdataOneFourth.columns)
1379 fig, ax = plt.subplots(figsize=(12, 5))
1380 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
255))
1381 ax.set_title("语音通话稳定性各项指标重要程度")
1382 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
1383 plt.yticks(font='Times New Roman')
1384 plt.tight_layout()
1385 plt.savefig("figuresOne\\[附件1]语音通话稳定性各项指标重要程度.pdf")

```

```

1386
1387 # In[121]:
1388
1389
1390 feat_with_importance
1391
1392 # #### XGBoost
1393
1394 # In[122]:
1395
1396
1397 from xgboost import XGBClassifier
1398
1399 XGBFourth = XGBClassifier(learning_rate=0.02,
1400                         n_estimators=14,
1401                         max_depth=6,
1402                         min_child_weight=1,
1403                         gamma=0.05,
1404                         subsample=1,
1405                         colsample_btree=1,
1406                         scale_pos_weight=1,
1407                         random_state=2022,
1408                         silent=0)
1409 XGBFourth.fit(XdataOneFourth_train, ydataOneFourth_train)
1410 XGBFourth_score = XGBFourth.score(XdataOneFourth_test, ydataOneFourth_test)
1411 XGBFourth_score
1412
1413 # In[123]:
1414
1415
1416 from xgboost import plot_importance
1417
1418 fig, ax = plt.subplots(figsize=(14, 8))
1419 plot_importance(XGBFourth, height=0.4, ax=ax)
1420 plt.xticks(fontsize=13, font='Times New Roman')
1421 plt.yticks(fontsize=11)
1422 ax.set_title('')
1423 ax.set_ylabel('')
1424 ax.set_xlabel('')
1425 plt.tight_layout()
1426 plt.savefig('figuresOne\\[附件1]语音通话稳定性各项指标重要程度 (XGBoost,F-score) .pdf
1427 ')
1428 # #### KNN

```

```

1429
1430 # In[124]:
1431
1432
1433 from sklearn.neighbors import KNeighborsClassifier
1434
1435 KNNFourth = KNeighborsClassifier()
1436 KNNFourth.fit(XdataOneFourth_train, ydataOneFourth_train)
1437 KNNFourth_score = KNNFourth.score(XdataOneFourth_test, ydataOneFourth_test)
1438 KNNFourth_score
1439
1440 # In[125]:
1441
1442
1443 from sklearn.neighbors import KNeighborsClassifier
1444 from sklearn.model_selection import GridSearchCV
1445
1446 KNN_turing_param_grid = [{'weights': ['uniform'],
1447                             'n_neighbors': [k for k in range(35, 45)]},
1448                             {'weights': ['distance'],
1449                             'n_neighbors': [k for k in range(35, 45)] ,
1450                             'p': [p for p in range(1, 5)]}]
1451 KNN_turing = KNeighborsClassifier()
1452 KNN_turing_grid_search = GridSearchCV(KNN_turing,
1453                                         param_grid=KNN_turing_param_grid,
1454                                         n_jobs=-1,
1455                                         verbose=2)
1456 KNN_turing_grid_search.fit(XdataOneFourth_train, ydataOneFourth_train)
1457
1458 # In[126]:
1459
1460
1461 KNN_turing_grid_search.best_score_
1462
1463 # In[127]:
1464
1465
1466 KNN_turing_grid_search.best_params_
1467
1468 # In[128]:
1469
1470
1471 KNNFourth_new = KNeighborsClassifier(algorithm='auto', leaf_size=30,
1472                                         metric='minkowski',

```

```

1473                         n_jobs=-1,
1474                         n_neighbors=43, p=1,
1475                         weights='distance')
1476 KNNFourth_new.fit(XdataOneFourth_train, ydataOneFourth_train)
1477 KNNFourth_new_score = KNNFourth_new.score(XdataOneFourth_test, ydataOneFourth_test)
1478 KNNFourth_new_score
1479
1480 # ##### 支持向量机
1481
1482 # In[129]:
1483
1484
1485 from sklearn.svm import SVC
1486
1487 SVMFourth = SVC(random_state=2022)
1488 SVMFourth.fit(XdataOneFourth_train, ydataOneFourth_train)
1489 SVMFourth_score = SVMFourth.score(XdataOneFourth_test, ydataOneFourth_test)
1490 SVMFourth_score
1491
1492 # ##### lightgbm
1493
1494 # In[130]:
1495
1496
1497 from lightgbm import LGBMClassifier
1498
1499 LightgbmFourth = LGBMClassifier(learning_rate=0.1,
1500                         lambda_l1=0.1,
1501                         lambda_l2=0.2,
1502                         max_depth=10,
1503                         objective='multiclass',
1504                         num_class=4,
1505                         random_state=2022)
1506 LightgbmFourth.fit(XdataOneFourth_train, ydataOneFourth_train)
1507 LightgbmFourth_score = LightgbmFourth.score(XdataOneFourth_test,
1508                                         ydataOneFourth_test)
1509 LightgbmFourth_score
1510
1511 # ##### 逻辑回归
1512
1513
1514 # In[131]:
1515
1516
1517 from sklearn.linear_model import LogisticRegression

```

```

1516
1517 LogisticRegressionFourth = LogisticRegression(multi_class="multinomial", solver="
1518     newton-cg", max_iter=2000)
1519 LogisticRegressionFourth = LogisticRegressionFourth.fit(XdataOneFourth_train,
1520     ydataOneFourth_train)
1521 LogisticRegressionFourth_score = LogisticRegressionFourth.score(XdataOneFourth_test
1522     , ydataOneFourth_test)
1523 LogisticRegressionFourth_score
1524
1525 # In[132]:
1526
1527 print(f'模型四中RF平均绝对误差: ,
1528       f'{mean_absolute_error(ydataOneFourth_test, RandomForestFourth.predict(
1529           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
1530       n',
1531       f'模型四中RF均方误差: ,
1532       f'{mean_squared_error(ydataOneFourth_test, RandomForestFourth.predict(
1533           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
1534       )
1535 print(f'模型四中XGBoost平均绝对误差: ,
1536       f'{mean_absolute_error(ydataOneFourth_test, XGBFourth.predict(
1537           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
1538       n',
1539       f'模型四中XGBoost均方误差: ,
1540       f'{mean_squared_error(ydataOneFourth_test, XGBFourth.predict(
1541           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
1542       )
1543 print(f'模型四中KNN平均绝对误差: ,
1544       f'{mean_absolute_error(ydataOneFourth_test, KNNFourth_new.predict(
1545           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
1546       n',
1547       f'模型四中KNN均方误差: ,
1548       f'{mean_squared_error(ydataOneFourth_test, KNNFourth_new.predict(
1549           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
1550       )
1551 print(f'模型四中SVM平均绝对误差: ,
1552       f'{mean_absolute_error(ydataOneFourth_test, SVMFourth.predict(
1553           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
1554       n',
1555       f'模型四中SVM均方误差: ,
1556       f'{mean_squared_error(ydataOneFourth_test, SVMFourth.predict(
1557           XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
1558       )

```

```

1541 print(f'模型四中LightGBM平均绝对误差: ,  

1542     f'{mean_absolute_error(ydataOneFourth_test, LightgbmFourth.predict(  

1543         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\n  

1544  

1545 f'模型四中LightGBM均方误差: ,  

1546     f'{mean_squared_error(ydataOneFourth_test, LightgbmFourth.predict(  

1547         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}'  

1548 )  

1549 print(f'模型四中LR平均绝对误差: ,  

1550     f'{mean_absolute_error(ydataOneFourth_test, LogisticRegressionFourth.predict(  

1551         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\n  

1552 f'模型四中LR均方误差: ,  

1553     f'{mean_squared_error(ydataOneFourth_test, LogisticRegressionFourth.predict(  

1554         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}',  

1555 )  

1556  

1557 # ##### 集成学习  

1558  

1559 # In[133]:  

1560  

1561  

1562 from mlxtend.classifier import StackingCVClassifier  

1563  

1564 FourthModel = StackingCVClassifier(  

1565     classifiers=[RandomForestFourth, LightgbmFourth, KNNFourth_new,  

1566         LogisticRegressionFourth, SVMFourth],  

1567     meta_classifier=XGBClassifier(random_state=2022), random_state=2022, cv=5)  

1568 FourthModel.fit(XdataOneFourth_train, ydataOneFourth_train)  

1569 FourthModel_score = FourthModel.score(XdataOneFourth_test, ydataOneFourth_test)  

1570 FourthModel_score  

1571  

1572 # In[134]:  

1573  

1574  

1575 print(f'模型四平均绝对误差: ,  

1576     f'{mean_absolute_error(ydataOneFourth_test, FourthModel.predict(  

1577         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\n  

1578 f'模型四均方误差: ,  

1579     f'{mean_squared_error(ydataOneFourth_test, FourthModel.predict(  

1580         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}',  

1581 )

```

```
1572 # ## 预测附件3四项评分
1573
1574 # In[135]:
1575
1576
1577 dataThree = pd.read_excel("附件3语音业务用户满意度预测数据.xlsx", sheet_name='语音')
1578 dataThree
1579
1580 # ### 附件格式统一
1581
1582 # In[136]:
1583
1584
1585 dataThree.drop(['用户id', '用户描述', '用户描述.1', '性别', '终端品牌类型', '是否不限量套餐到达用户'], axis=1, inplace=True)
1586
1587 # In[137]:
1588
1589
1590 dataThree
1591
1592 # In[138]:
1593
1594
1595 dataThree.isnull().sum()
1596
1597 # In[139]:
1598
1599
1600 dataThree["外省流量占比"] = dataThree["外省流量占比"].astype(str).replace('%', '')
1601 dataThree["外省语音占比"] = dataThree["外省语音占比"].astype(str).replace('%', '')
1602 dataThree
1603
1604 # In[140]:
1605
1606
1607 dataThree.replace({"是否遇到过网络问题": {2: 0}, "居民小区": {-1: 0}, "办公室": {-1: 0, 2: 1}, "高校": {-1: 0, 3: 1}, "商业街": {-1: 0, 4: 1}, "地铁": {-1: 0, 5: 1}, "农村": {-1: 0, 6: 1}, "高铁": {-1: 0, 7: 1}, "其他, 请注明": {-1: 0, 98: 1}, "手机没有信号": {-1: 0}, "有信号无法拨通": {-1: 0, 2: 1}, "通话过程中突然中断": {-1: 0, 3: 1}, "通话中有杂音、听不清、断断续续": {-1: 0, 4: 1}, "串线": {-1: 0, 5: 1}, "通话过程中一方听不见": {-1: 0, 6: 1}, "其他, 请注明.1": {-1: 0, 98: 1}, "是否投诉": {"是": 1, "否": 0}, "是否关怀用户": {"是": 1, "否": 0}, "是否4G网络客户(本地剔除物联网)": {"是": 1, "否": 0}, "是否5G网络客户": {"是": 1, "否": 0}, "客
```

```

户星级标识": {'未评级': 0, '准星': 1, '一星': 2, '二星': 3, '三星': 4, '银卡': 5,
    '金卡': 6, '白金卡': 7, '钻石卡': 8}, "终端品牌": {"苹果": 22, '华为': 11, '小米
科技': 14, '步步高': 18, '欧珀': 17, '三星': 4, 'realme': 1, 'O': 0, '万普拉斯':
3, '锤子': 24, '万普': 8, '中邮通信': 21, '索尼爱立信': 6, '亿城': 6, '宇龙': 6,
'中国移动': 7, '中兴': 10, '黑鲨': 25, '海信': 16, '摩托罗拉': 9, '诺基亚': 12,
'奇酷': 13}}, inplace=True)
1608 dataThree
1609
1610 # In[141]:
1611
1612
1613 dataThree['外省语音占比'] = dataThree['外省语音占比'].astype('float64')
1614 dataThree['外省流量占比'] = dataThree['外省流量占比'].astype('float64')
1615 dataThree['是否4G网络客户(本地剔除物联网)'] = dataThree['是否4G网络客户(本地剔除物
联网)'].astype('int64')
1616 dataThree['4\5G用户'] = dataThree['4\5G用户'].astype(str)
1617 dataThree
1618
1619 # In[142]:
1620
1621
1622 le = sp.LabelEncoder()
1623
1624 FourFiveUser = le.fit_transform(dataThree["4\5G用户"])
1625 dataThree["4\5G用户"] = pd.DataFrame(FourFiveUser)
1626 dataThree
1627
1628 # In[143]:
1629
1630
1631 dataThree['是否5G网络客户'] = dataThree['是否5G网络客户'].astype('int64')
1632 dataThree['客户星级标识'] = dataThree['客户星级标识'].astype('int64')
1633 dataThree['终端品牌'] = dataThree['终端品牌'].astype('int32')
1634 dataThree
1635
1636 # In[144]:
1637
1638
1639 dataThreeStandardTransform = dataThree[['脱网次数', 'mos质差次数', '未接通掉话次数',
    '4\5G用户', '套外流量(MB)', '套外流量费(元)', '语音通话-时长(分钟)', '省际
    漫游-时长(分钟)', '终端品牌', '当月ARPU', '当月MOU', '前3月ARPU', '前3月MOU',
    'GPRS总流量(KB)', 'GPRS-国内漫游-流量(KB)', '客户星级标识']]
1640 dataThreeStandardTransformScaler = sp.StandardScaler()
1641 dataThreeStandardTransformScaler = dataThreeStandardTransformScaler.fit(

```

```

        dataThreeStandardTransform)
1642 dataThreeStandardTransform = dataThreeStandardTransformScaler.transform(
        dataThreeStandardTransform)
1643 dataThreeStandardTransform = pd.DataFrame(dataThreeStandardTransform)
1644 dataThreeStandardTransform.columns = [‘脱网次数’, ‘mos质差次数’, ‘未接通掉话次数’, ,
4\\5G用户’, , 套外流量 (MB) , , 套外流量费 (元) , , 语音通话-时长 (分钟) , , 省际漫游-时长 (分钟) , , 终端品牌’ , , 当月ARPU’ , , 当月MOU’ , , 前3月ARPU’ , , 前3月MOU’ , ,
GPRS总流量 (KB) , , GPRS-国内漫游-流量 (KB) , , 客户星级标识’]
1645 dataThreeStandardTransform
1646
1647 # In[145]:
1648
1649
1650 dataThreeLeave = dataThree.loc[:, ~dataThree.columns.isin([‘脱网次数’, ‘mos质差次数’,
, ‘未接通掉话次数’, ‘4\\5G用户’, , 套外流量 (MB) , , 套外流量费 (元) , , 语音通话-
时长 (分钟) , , 省际漫游-时长 (分钟) , , 终端品牌’ , , 当月ARPU’ , , 当月MOU’ , , 前3月
ARPU’ , , 前3月MOU’ , , GPRS总流量 (KB) , , GPRS-国内漫游-流量 (KB) , , 客户星级标识’
])]
1651 dataThreeNewStandard = pd.concat([dataThreeLeave, dataThreeStandardTransform], axis
=1)
1652 dataThreeNewStandard.columns = [‘是否遇到过网络问题’, , 居民小区’, , 办公室’, , 高校’, ,
商业街’, , 地铁’, , 农村’, , 高铁’, , 其他, 请注明’, , 手机没有信号’, , 有信号无法拨通’,
, 通话过程中突然中断’, , 通话中有杂音、听不清、断断续续’, , 串线’, , 通话过程中一方听
不见’, , 其他, 请注明.1’, , 是否投诉’, , 是否关怀用户’, , 是否4G网络客户 (本地剔除物联网)’,
, 外省语音占比’, , 外省流量占比’, , 是否5G网络客户’, , 脱网次数’, , mos质差次数’,
, 未接通掉话次数’, , 4\\5G用户’, , 套外流量 (MB) , , 套外流量费 (元) , , 语音通话-
时长 (分钟) , , 省际漫游-时长 (分钟) , , 终端品牌’ , , 当月ARPU’ , , 当月MOU’ , , 前3月
ARPU’ , , 前3月MOU’ , , GPRS总流量 (KB) , , GPRS-国内漫游-流量 (KB) , , 客户星级标识’]
1653 dataThreeNewStandard
1654
1655 # In[146]:
1656
1657
1658 dataOneNewStandard
1659
1660 # ### 预测语音业务评分
1661 # 需要注意到在所有预测结果上加上1, 由于之前将评分编码为[0,9], 这里需要再映射回[1,10]
1662
1663 # In[147]:
1664
1665
1666 Xpre = dataThreeNewStandard
1667 # #### 语音通话整体满意度

```

```
1669
1670 # In[148]:
1671
1672
1673 FirstPre = FirstModel.predict(Xpre)
1674 FirstPre
1675
1676 # ##### 网络覆盖与信号强度
1677
1678 # In[149]:
1679
1680
1681 SecondPre = SecondModel.predict(Xpre)
1682 SecondPre
1683
1684 # ##### 语音通话清晰度
1685
1686 # In[150]:
1687
1688
1689 ThirdPre = ThirdModel.predict(Xpre)
1690 ThirdPre
1691
1692 # ##### 语音通话稳定性
1693
1694 # In[151]:
1695
1696
1697 FourthPre = FourthModel.predict(Xpre)
1698 FourthPre
1699
1700 # ## 模型效果分析
1701
1702 # ### 混淆矩阵热力图
1703
1704 # ##### 模型一
1705
1706 # In[152]:
1707
1708
1709 from yellowbrick.classifier import ConfusionMatrix
1710
1711 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1712 confusion_matrix = ConfusionMatrix(FirstModel, classes=classes, cmap='BuGn')
```

```

1713 confusion_matrix.fit(XdataOneFirst_train, ydataOneFirst_train)
1714 confusion_matrix.score(XdataOneFirst_test, ydataOneFirst_test)
1715 plt.xticks(font='Times New Roman')
1716 plt.yticks(font='Times New Roman')
1717 confusion_matrix.show(outpath='figuresOne\\[附件1]模型一混淆矩阵热力图.pdf')
1718
1719 # ##### 模型二
1720
1721 # In[153]:
1722
1723
1724 from yellowbrick.classifier import ConfusionMatrix
1725
1726 classes = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’]
1727 confusion_matrix = ConfusionMatrix(SecondModel, classes=classes, cmap='BuGn')
1728 confusion_matrix.fit(XdataOneSecond_train, ydataOneSecond_train)
1729 confusion_matrix.score(XdataOneSecond_test, ydataOneSecond_test)
1730 plt.xticks(font='Times New Roman')
1731 plt.yticks(font='Times New Roman')
1732 confusion_matrix.show(outpath='figuresOne\\[附件1]模型二混淆矩阵热力图.pdf')
1733
1734 # ##### 模型三
1735
1736 # In[154]:
1737
1738
1739 from yellowbrick.classifier import ConfusionMatrix
1740
1741 classes = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’]
1742 confusion_matrix = ConfusionMatrix(ThirdModel, classes=classes, cmap='BuGn')
1743 confusion_matrix.fit(XdataOneThird_train, ydataOneThird_train)
1744 confusion_matrix.score(XdataOneThird_test, ydataOneThird_test)
1745 plt.xticks(font='Times New Roman')
1746 plt.yticks(font='Times New Roman')
1747 confusion_matrix.show(outpath='figuresOne\\[附件1]模型三混淆矩阵热力图.pdf')
1748
1749 # ##### 模型四
1750
1751 # In[155]:
1752
1753
1754 from yellowbrick.classifier import ConfusionMatrix
1755
1756 classes = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’]

```

```

1757 confusion_matrix = ConfusionMatrix(FourthModel, classes=classes, cmap='BuGn')
1758 confusion_matrix.fit(XdataOneFourth_train, ydataOneFourth_train)
1759 confusion_matrix.score(XdataOneFourth_test, ydataOneFourth_test)
1760 plt.xticks(font='Times New Roman')
1761 plt.yticks(font='Times New Roman')
1762 confusion_matrix.show(outpath='figuresOne\\[附件1]模型四混淆矩阵热力图.pdf')
1763
1764 # ### 分类报告
1765
1766 # #### 模型一
1767
1768 # In[156]:
1769
1770
1771 from yellowbrick.classifier import ClassificationReport
1772
1773 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1774 visualizer = ClassificationReport(FirstModel, classes=classes, support=True, cmap='Blues')
1775 visualizer.fit(XdataOneFirst_train, ydataOneFirst_train)
1776 visualizer.score(XdataOneFirst_test, ydataOneFirst_test)
1777 plt.xticks(font='Times New Roman')
1778 plt.yticks(font='Times New Roman')
1779 visualizer.show(outpath='figuresOne\\[附件1]模型一分类报告.pdf')
1780
1781 # #### 模型二
1782
1783 # In[157]:
1784
1785
1786 from yellowbrick.classifier import ClassificationReport
1787
1788 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1789 visualizer = ClassificationReport(SecondModel, classes=classes, support=True, cmap='Blues')
1790 visualizer.fit(XdataOneSecond_train, ydataOneSecond_train)
1791 visualizer.score(XdataOneSecond_test, ydataOneSecond_test)
1792 plt.xticks(font='Times New Roman')
1793 plt.yticks(font='Times New Roman')
1794 visualizer.show(outpath='figuresOne\\[附件1]模型二分类报告.pdf')
1795
1796 # #### 模型三
1797
1798 # In[158]:

```

```

1799
1800
1801     from yellowbrick.classifier import ClassificationReport
1802
1803     classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1804     visualizer = ClassificationReport(ThirdModel, classes=classes, support=True, cmap='
1805         Blues')
1806     visualizer.fit(XdataOneThird_train, ydataOneThird_train)
1807     visualizer.score(XdataOneThird_test, ydataOneThird_test)
1808     plt.xticks(font='Times New Roman')
1809     plt.yticks(font='Times New Roman')
1810     visualizer.show(outpath='figuresOne\\[附件1]模型三分类报告.pdf')
1811
1812 # ##### 模型四
1813
1814
1815
1816     from yellowbrick.classifier import ClassificationReport
1817
1818     classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1819     visualizer = ClassificationReport(FourthModel, classes=classes, support=True, cmap='
1820         Blues')
1821     visualizer.fit(XdataOneFourth_train, ydataOneFourth_train)
1822     visualizer.score(XdataOneFourth_test, ydataOneFourth_test)
1823     plt.xticks(font='Times New Roman')
1824     plt.yticks(font='Times New Roman')
1825     visualizer.show(outpath='figuresOne\\[附件1]模型四分类报告.pdf')
1826
1827 # ### ROC AUC曲线
1828
1829
1830 # In[160]:
1831
1832
1833     from yellowbrick.classifier import ROCAUC
1834
1835     visualizer = ROCAUC(FirstModel)
1836     visualizer.fit(XdataOneFirst_train, ydataOneFirst_train)
1837     visualizer.score(XdataOneFirst_test, ydataOneFirst_test)
1838     plt.xticks(font='Times New Roman')
1839     plt.yticks(font='Times New Roman')
1840     visualizer.show(outpath='figuresOne\\[附件1]模型一ROCAUC.pdf')

```

```

1841
1842 # ##### 模型二
1843
1844 # In[161]:
1845
1846
1847 from yellowbrick.classifier import ROCAUC
1848
1849 visualizer = ROCAUC(SecondModel)
1850 visualizer.fit(XdataOneSecond_train, ydataOneSecond_train)
1851 visualizer.score(XdataOneSecond_test, ydataOneSecond_test)
1852 plt.xticks(font='Times New Roman')
1853 plt.yticks(font='Times New Roman')
1854 visualizer.show(outpath='figuresOne\\[附件1] 模型二ROCAUC.pdf')
1855
1856 # ##### 模型三
1857
1858 # In[162]:
1859
1860
1861 from yellowbrick.classifier import ROCAUC
1862
1863 visualizer = ROCAUC(ThirdModel)
1864 visualizer.fit(XdataOneThird_train, ydataOneThird_train)
1865 visualizer.score(XdataOneThird_test, ydataOneThird_test)
1866 plt.xticks(font='Times New Roman')
1867 plt.yticks(font='Times New Roman')
1868 visualizer.show(outpath='figuresOne\\[附件1] 模型三ROCAUC.pdf')
1869
1870 # ##### 模型四
1871
1872 # In[163]:
1873
1874
1875 from yellowbrick.classifier import ROCAUC
1876
1877 visualizer = ROCAUC(FourthModel)
1878 visualizer.fit(XdataOneFourth_train, ydataOneFourth_train)
1879 visualizer.score(XdataOneFourth_test, ydataOneFourth_test)
1880 plt.xticks(font='Times New Roman')
1881 plt.yticks(font='Times New Roman')
1882 visualizer.show(outpath='figuresOne\\[附件1] 模型四ROCAUC.pdf')
1883
1884 # ### 平均绝对误差，均方误差

```

```

1885
1886 # ##### 模型一
1887
1888 # In[164]:
1889
1890
1891 print(f'模型一平均绝对误差: ,\n
1892     f'{mean_absolute_error(ydataOneFirst_test, FirstModel.predict(
1893         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
1894         ,\n
1895         f'模型一均方误差: ,\n
1896         f'{mean_squared_error(ydataOneFirst_test, FirstModel.predict(
1897             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")})')
1898 print(f'模型一中RF平均绝对误差: ,\n
1899     f'{mean_absolute_error(ydataOneFirst_test, RandomForestFirst.predict(
1900         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
1901         ,\n
1902         f'模型一中RF均方误差: ,\n
1903         f'{mean_squared_error(ydataOneFirst_test, RandomForestFirst.predict(
1904             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")})')
1905 print(f'模型一中XGBoost平均绝对误差: ,\n
1906     f'{mean_absolute_error(ydataOneFirst_test, XGBFirst.predict(XdataOneFirst_test
1907         ), sample_weight=None, multioutput="uniform_average")}\n'
1908         ,\n
1909         f'模型一中XGBoost均方误差: ,\n
1910         f'{mean_squared_error(ydataOneFirst_test, XGBFirst.predict(XdataOneFirst_test)
1911             , sample_weight=None, multioutput="uniform_average")})')
1912 print(f'模型一中KNN平均绝对误差: ,\n
1913     f'{mean_absolute_error(ydataOneFirst_test, KNNFirst_new.predict(
1914         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
1915         ,\n
1916         f'模型一中KNN均方误差: ,\n
1917         f'{mean_squared_error(ydataOneFirst_test, KNNFirst_new.predict(
1918             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")})')
1919 print(f'模型一中SVM平均绝对误差: ,\n
1920     f'{mean_absolute_error(ydataOneFirst_test, SVMFirst.predict(XdataOneFirst_test
1921         ), sample_weight=None, multioutput="uniform_average")}\n'
1922         ,\n
1923         f'模型一中SVM均方误差: ,\n
1924         f'{mean_squared_error(ydataOneFirst_test, SVMFirst.predict(XdataOneFirst_test)
1925             , sample_weight=None, multioutput="uniform_average")})')
1926 print(f'模型一中LightGBM平均绝对误差: ,\n
1927     f'{mean_absolute_error(ydataOneFirst_test, LightgbmFirst.predict(
1928         XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
1929         ,\n
1930         f'模型一中LightGBM均方误差: ,'

```

```

1914         f'{mean_squared_error(ydataOneFirst_test, LightgbmFirst.predict(
1915             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
1915     print(f'模型一中LR平均绝对误差: ,\n
1916         f'{mean_absolute_error(ydataOneFirst_test, LogisticRegressionFirst.predict(
1917             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n
1917         ,\n
1918         f'模型一中LR均方误差: ,\n
1919         f'{mean_squared_error(ydataOneFirst_test, LogisticRegressionFirst.predict(
1920             XdataOneFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
1921
1922     # ##### 模型二
1923
1924
1925     print(f'模型二平均绝对误差: ,\n
1926         f'{mean_absolute_error(ydataOneSecond_test, SecondModel.predict(
1927             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n
1927         n,\n
1928         f'模型二均方误差: ,\n
1929         f'{mean_squared_error(ydataOneSecond_test, SecondModel.predict(
1930             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n
1930         )
1931     print(f'模型二中RF平均绝对误差: ,\n
1932         f'{mean_absolute_error(ydataOneSecond_test, RandomForestSecond.predict(
1933             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n
1933         n,\n
1934         f'模型二中RF均方误差: ,\n
1935         f'{mean_squared_error(ydataOneSecond_test, RandomForestSecond.predict(
1936             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n
1936         )
1937     print(f'模型二中XGBoost平均绝对误差: ,\n
1938         f'{mean_absolute_error(ydataOneSecond_test, XGBSecond.predict(
1939             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n
1939         n,\n
1940         f'模型二中XGBoost均方误差: ,\n
1941         f'{mean_squared_error(ydataOneSecond_test, XGBSecond.predict(
1942             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n
1942         )
1943     print(f'模型二中KNN平均绝对误差: ,\n
1944         f'{mean_absolute_error(ydataOneSecond_test, KNNSecond_new.predict(
1945             XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n
1945         n,\n
1946         f'模型二中KNN均方误差: ,\n

```

```

1940     f'{mean_squared_error(ydataOneSecond_test, KNNSecond_new.predict(
1941         XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n'
1942 print(f'模型二中SVM平均绝对误差: ,\n
1943     f'{mean_absolute_error(ydataOneSecond_test, SVMSecond.predict(
1944         XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n'
1945 f'模型二中SVM均方误差: ,\n
1946     f'{mean_squared_error(ydataOneSecond_test, SVMSecond.predict(
1947         XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n'
1948 print(f'模型二中LightGBM平均绝对误差: ,\n
1949     f'{mean_absolute_error(ydataOneSecond_test, LightgbmSecond.predict(
1950         XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n'
1951 f'模型二中LightGBM均方误差: ,\n
1952     f'{mean_squared_error(ydataOneSecond_test, LightgbmSecond.predict(
1953         XdataOneSecond_test), sample_weight=None, multioutput="uniform_average")}\n'
1954 # ##### 模型三
1955
1956 # In[166]:
1957
1958
1959 print(f'模型三平均绝对误差: ,\n
1960     f'{mean_absolute_error(ydataOneThird_test, ThirdModel.predict(
1961         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n'
1962     ,\n
1963 f'模型三均方误差: ,\n
1964     f'{mean_squared_error(ydataOneThird_test, ThirdModel.predict(
1965         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")})')
1966 print(f'模型三中RF平均绝对误差: ,\n
1967     f'{mean_absolute_error(ydataOneThird_test, RandomForestThird.predict(
1968         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n'
1969     ,

```

```

1965     f'模型三中RF均方误差: ,
1966     f'{mean_squared_error(ydataOneThird_test, RandomForestThird.predict(
1967         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")})'
1967 print(f'模型三中XGBoost平均绝对误差: ,
1968     f'{mean_absolute_error(ydataOneThird_test, XGBThird.predict(XdataOneThird_test
1969         ), sample_weight=None, multioutput="uniform_average")}\n'
1969 f'模型三中XGBoost均方误差: ,
1970     f'{mean_squared_error(ydataOneThird_test, XGBThird.predict(XdataOneThird_test)
1971         , sample_weight=None, multioutput="uniform_average")})'
1971 print(f'模型三中KNN平均绝对误差: ,
1972     f'{mean_absolute_error(ydataOneThird_test, KNNThird_new.predict(
1973         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n
1973 ,
1973 f'模型三中KNN均方误差: ,
1974     f'{mean_squared_error(ydataOneThird_test, KNNThird_new.predict(
1975         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")})'
1975 print(f'模型三中SVM平均绝对误差: ,
1976     f'{mean_absolute_error(ydataOneThird_test, SVMThird.predict(XdataOneThird_test
1977         ), sample_weight=None, multioutput="uniform_average")}\n'
1977 f'模型三中SVM均方误差: ,
1978     f'{mean_squared_error(ydataOneThird_test, SVMThird.predict(XdataOneThird_test)
1979         , sample_weight=None, multioutput="uniform_average")})'
1979 print(f'模型三中LightGBM平均绝对误差: ,
1980     f'{mean_absolute_error(ydataOneThird_test, LightgbmThird.predict(
1981         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n
1981 ,
1981 f'模型三中LightGBM均方误差: ,
1982     f'{mean_squared_error(ydataOneThird_test, LightgbmThird.predict(
1983         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")})'
1983 print(f'模型三中LR平均绝对误差: ,
1984     f'{mean_absolute_error(ydataOneThird_test, LogisticRegressionThird.predict(
1985         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")}\n
1985 ,
1985 f'模型三中LR均方误差: ,
1986     f'{mean_squared_error(ydataOneThird_test, LogisticRegressionThird.predict(
1987         XdataOneThird_test), sample_weight=None, multioutput="uniform_average")})'

1988 # ##### 模型四
1989
1990 # In[167]:
1991
1992
1993 print(f'模型四平均绝对误差: ,
1994     f'{mean_absolute_error(ydataOneFourth_test, FourthModel.predict(

```

```

        XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
n'
f'模型四均方误差: ,
f'{mean_squared_error(ydataOneFourth_test, FourthModel.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型四中RF平均绝对误差: ,
f'{mean_absolute_error(ydataOneFourth_test, RandomForestFourth.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
n'
f'模型四中RF均方误差: ,
f'{mean_squared_error(ydataOneFourth_test, RandomForestFourth.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型四中XGBoost平均绝对误差: ,
f'{mean_absolute_error(ydataOneFourth_test, XGBFourth.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
n'
f'模型四中XGBoost均方误差: ,
f'{mean_squared_error(ydataOneFourth_test, XGBFourth.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型四中KNN平均绝对误差: ,
f'{mean_absolute_error(ydataOneFourth_test, KNNFourth_new.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
n'
f'模型四中KNN均方误差: ,
f'{mean_squared_error(ydataOneFourth_test, KNNFourth_new.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型四中SVM平均绝对误差: ,
f'{mean_absolute_error(ydataOneFourth_test, SVMFourth.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
n'
f'模型四中SVM均方误差: ,
f'{mean_squared_error(ydataOneFourth_test, SVMFourth.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型四中LightGBM平均绝对误差: ,
f'{mean_absolute_error(ydataOneFourth_test, LightgbmFourth.predict(
    XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
n'
f'模型四中LightGBM均方误差: ,
f'{mean_squared_error(ydataOneFourth_test, LightgbmFourth.predict(

```

```

        XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")\}
    )

2017 print(f'模型四中LR平均绝对误差: ,
2018     f'{mean_absolute_error(ydataOneFourth_test, LogisticRegressionFourth.predict(
2019         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\\
2020     n',
2021
2022     f'模型四中LR均方误差: ,
2023     f'{mean_squared_error(ydataOneFourth_test, LogisticRegressionFourth.predict(
2024         XdataOneFourth_test), sample_weight=None, multioutput="uniform_average")}\',
2025     )
2026
2027
2028 # ## 高频词汇云图
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044

```

D.2 上网业务分析代码 [针对附件 2 与附件 4]

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # 2022 MathorCup 大数据 IssueB
5
6 # # 上网业务数据分析
7
8 # ## 初步导入相关第三方库
9
10 # In[1]:
11
12
13 import pandas as pd
14 import numpy as np
15 import sklearn.preprocessing as sp
16 import warnings
17
18 warnings.filterwarnings("ignore")
19
20 # ## 读取附件2与附件4
21
22 # In[2]:
23
24
25 dataTwo = pd.read_excel("附件2上网业务用户满意度数据.xlsx", sheet_name='用后即评满意度分析0620(Q1655704201796)_P')
26 dataFour = pd.read_excel("附件4上网业务用户满意度预测数据.xlsx", sheet_name='上网')
27
28 # In[3]:
29
30
31 dataTwo
32
33 # In[4]:
34
35
36 dataFour
37
38 # ## 处理附件2与附件4
39
40 # ### 查看附件2与附件4表头交集
41
42 # In[5]:
```

```

43
44
45 list(set(list(dataTwo.columns)) & set(list(dataFour.columns)))
46
47 # ### 剔除不重合的列指标, 以及不重要列指标
48
49 # In[6]:
50
51
52 dataTwo = dataTwo[['手机上网整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性', '居民小区', '是否5G网络客户', '高校', '是否不限量套餐到达用户', '其他,请注明.5', '咪咕视频', '阴阳师', '手机QQ', '手机上网速度慢', '炉石传说', '打游戏延时大', '火山', '显示有信号上不了网', '今日头条', '办公室', '上网质差次数', '梦幻西游', '当月MOU', '其他,请注明.2', '客户星级标识', '穿越火线', '全部都卡顿', '微信', '全部游戏都卡顿', '脱网次数', '性别', '套外流量费(元)', '农村', '搜狐视频', '京东', '微信质差次数', '百度', '套外流量(MB)', '其他,请注明.1', '抖音', '商业街', '拼多多', '新浪微博', '其他,请注明', '和平精英', '手机支付较慢', '看视频卡顿', '终端品牌', '梦幻诛仙', '部落冲突', '腾讯视频', '上网过程中网络时断时续或时快时慢', '其他,请注明.3', '地铁', '打开网页或APP图片慢', '快手', '芒果TV', '爱奇艺', '龙之谷', '高铁', '全部网页或APP都慢', '王者荣耀', '淘宝', '其他,请注明.4', '下载速度慢', '优酷', '欢乐斗地主', '网络信号差/没有信号']]]

53 dataTwo
54
55 # In[7]:
56
57
58 dataFour = dataFour[['居民小区', '是否5G网络客户', '高校', '是否不限量套餐到达用户', '其他,请注明.5', '咪咕视频', '阴阳师', '手机QQ', '手机上网速度慢', '炉石传说', '打游戏延时大', '火山', '显示有信号上不了网', '今日头条', '办公室', '上网质差次数', '梦幻西游', '当月MOU', '其他,请注明.2', '客户星级标识', '穿越火线', '全部都卡顿', '微信', '全部游戏都卡顿', '脱网次数', '性别', '套外流量费(元)', '农村', '搜狐视频', '京东', '微信质差次数', '百度', '套外流量(MB)', '其他,请注明.1', '抖音', '商业街', '拼多多', '新浪微博', '其他,请注明', '和平精英', '手机支付较慢', '看视频卡顿', '终端品牌', '梦幻诛仙', '部落冲突', '腾讯视频', '上网过程中网络时断时续或时快时慢', '其他,请注明.3', '地铁', '打开网页或APP图片慢', '快手', '芒果TV', '爱奇艺', '龙之谷', '高铁', '全部网页或APP都慢', '王者荣耀', '淘宝', '其他,请注明.4', '下载速度慢', '优酷', '欢乐斗地主', '网络信号差/没有信号']]]

59 dataFour
60
61 # ### 填补空缺值、数据利于理解化、清洗处理
62
63 # In[8]:
64
65

```

```

66 dataTwo.isnull().sum()
67
68 # In[9]:
69
70
71 dataFour.isnull().sum()
72
73 # In[10]:
74
75
76 dataTwo = dataTwo.fillna(0)
77 dataTwo
78
79 # In[11]:
80
81
82 dataTwo.isnull().sum()
83
84 # In[12]:
85
86
87 dataTwo.replace({'居民小区': {-1: 0}, '是否5G网络客户': {'否': 0, '是': 1}, '高校': {-1: 0, 3: 1}, '是否不限量套餐到达用户': {'否': 0, '是': 1}, '其他, 请注明.5': {-1: 0, 98: 1}, '咪咕视频': {-1: 0, 9: 1}, '阴阳师': {-1: 0, 10: 1}, '手机QQ': {-1: 0, 2: 1}, '手机上网速度慢': {-1: 0, 4: 1}, '炉石传说': {-1: 0, 9: 1}, '打游戏延时大': {-1: 0, 2: 1}, '火山': {-1: 0, 8: 1}, '显示有信号上不了网': {-1: 0, 2: 1}, '今日头条': {-1: 0, 6: 1}, '办公室': {-1: 0, 2: 1}, '梦幻西游': {-1: 0, 4: 1}, '其他, 请注明.2': {-1: 0, 98: 1}, '客户星级标识': {'未评级': 0, '准星': 1, '一星': 2, '二星': 3, '三星': 4, '银卡': 5, '金卡': 6, '白金卡': 7, '钻石卡': 8}, '穿越火线': {-1: 0, 3: 1}, '全部都卡顿': {-1: 0, 99: 1}, '微信': {-1: 0}, '全部游戏都卡顿': {-1: 0, 99: 1}, '性别': {'男': 1, '女': -1, '性别不详': 0}, '农村': {-1: 0, 6: 1}, '搜狐视频': {-1: 0, 5: 1}, '京东': {-1: 0, 4: 1}, '百度': {-1: 0, 5: 1}, '其他, 请注明.1': {-1: 0, 98: 1}, '抖音': {-1: 0, 6: 1}, '商业街': {-1: 0, 4: 1}, '拼多多': {-1: 0, 8: 1}, '新浪微博': {-1: 0, 7: 1}, '其他, 请注明': {-1: 0, 98: 1}, '和平精英': {-1: 0}, '手机支付较慢': {-1: 0, 5: 1}, '看视频卡顿': {-1: 0}, '梦幻诛仙': {-1: 0, 6: 1}, '部落冲突': {-1: 0, 8: 1}, '腾讯视频': {-1: 0, 3: 1}, '上网过程中网络时断时续或时快时慢': {-1: 0, 3: 1}, '其他, 请注明.3': {-1: 0, 98: 1}, '地铁': {-1: 0, 5: 1}, '打开网页或APP图片慢': {-1: 0, 3: 1}, '快手': {-1: 0, 7: 1}, '芒果TV': {-1: 0, 4: 1}, '爱奇艺': {-1: 0}, '龙之谷': {-1: 0, 5: 1}, '高铁': {-1: 0, 7: 1}, '全部网页或APP都慢': {-1: 0, 99: 1}, '王者荣耀': {-1: 0, 2: 1}, '淘宝': {-1: 0, 3: 1}, '其他, 请注明.4': {-1: 0, 98: 1}, '下载速度慢': {-1: 0, 4: 1}, '优酷': {-1: 0, 2: 1}, '欢乐斗地主': {-1: 0, 7: 1}, '网络信号差/没有信号': {-1: 0}, '终端品牌': {'0': 0, '苹果': 1, '华为': 2, '小米科技': 3, '步步高': 4, '欧珀': 5, 'realme': 6, '三星': 7, '万普拉斯': 8, '黑鲨': 9, '锤子': 10}}
```

```

10, '摩托罗拉': 11, '中邮通信': 12, '万普': 13, '诺基亚': 14, '联通': 15, '中国移动': 16, '中兴': 17, '华硕': 18, '联想': 19, '魅族': 20, '奇酷': 21, 'TD': 22, '北京珠穆朗玛移动通信有限公司': 23, '飞利浦': 24, '捷开通讯科技': 25, '金立': 26, '酷比': 27, '欧博信': 28, '索尼爱立信': 29, '维图': 30, '甄十信息科技(上海)有限公司': 31, '中国电信': 32} }, inplace=True)

88 dataTwo

89

90 # ### 标签编码，四项评分视为分类问题

91

92 # In[13]: 

93

94

95 le = sp.LabelEncoder()

96

97 OverallSatisfactionMobileInternetAccess = le.fit_transform(dataTwo['手机上网整体满意度'])

98 NetworkCoverageSignalStrength = le.fit_transform(dataTwo['网络覆盖与信号强度'])

99 MobileInternetAccessSpeed = le.fit_transform(dataTwo['手机上网速度'])

100 MobileInternetAccessStability = le.fit_transform(dataTwo['手机上网稳定性'])

101

102 dataTwo["手机上网整体满意度"] = pd.DataFrame(OverallSatisfactionMobileInternetAccess)

103

104 dataTwo["网络覆盖与信号强度"] = pd.DataFrame(NetworkCoverageSignalStrength)

105 dataTwo["手机上网速度"] = pd.DataFrame(MobileInternetAccessSpeed)

106 dataTwo["手机上网稳定性"] = pd.DataFrame(MobileInternetAccessStability)

107

108 dataTwo

109 # ### 特征构造

110

111 # In[14]: 

112

113

114 dataTwo.columns

115

116 # In[15]: 

117

118

119 dataTwo['出现问题场所或应用总'] = dataTwo.loc[:, ~dataTwo.columns.isin(['手机上网整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性', '是否5G网络客户', '是否不限量套餐到达用户', '手机上网速度慢', '打游戏延时大', '显示有信号上不了网', '上网质差次数', '当月MOU', '客户星级标识', '全部都卡顿', '全部游戏都卡顿', '脱网次数', '性别', '套外流量费(元)', '微信质差次数', '百度', '套外流量(MB)', '手机支付较慢', '看视频卡顿', '终端品牌', '上网过程中网络时断时续或时快时慢', '打开网页或APP'])]

```

```

    图片慢'，'全部网页或APP都慢'，'下载速度慢'，'网络信号差/没有信号']]].apply(lambda
        x1: x1.sum(), axis=1)

120 dataTwo
121
122 # In[16]:
123
124
125 dataTwo[['网络卡速度慢延时大上不了网总']] = dataTwo.loc[:, ['手机上网速度慢'，'打游戏延
    时大'，'显示有信号上不了网'，'全部都卡顿'，'全部游戏都卡顿'，'手机支付较慢'，'看视
    频卡顿'，'上网过程中网络时断时续或时快时慢'，'打开网页或APP图片慢'，'全部网页或APP
    都慢'，'下载速度慢'，'网络信号差/没有信号']]].apply(lambda x1: x1.sum(), axis=1)

126 dataTwo
127
128 # In[17]:
129
130
131 dataTwo[['质差总']] = dataTwo.loc[:, ['微信质差次数'，'上网质差次数']].apply(lambda x1:
        x1.sum(), axis=1)
132 dataTwo
133
134 # ### 数据可视化
135
136 # In[18]:
137
138
139 import matplotlib.pyplot as plt
140
141 plt.rcParams['font.sans-serif'] = ['SimHei']
142 plt.rcParams['axes.unicode_minus'] = False
143
144 box_data = dataTwo[['手机上网整体满意度'，'网络覆盖与信号强度'，'手机上网速度'，'手机
    上网稳定性'，]]
145 plt.grid(True)
146 plt.boxplot(box_data,
147             notch=True,
148             sym="b+",
149             vert=False,
150             showmeans=True,
151             labels=['手机上网整体满意度'，'网络覆盖与信号强度'，'手机上网速度'，'手机上
    网稳定性'，])
152 plt.yticks(size=14)
153 plt.xticks(size=14, font='Times New Roman')
154 plt.tight_layout()
155 plt.savefig('figuresTwo\\[附件2] [手机上网整体满意度、网络覆盖与信号强度、手机上网速

```

度、手机上网稳定性]评分箱线图.pdf')

```
156
157 # In[19]:
158
159
160 import seaborn as sns
161
162 plt.style.use('ggplot')
163 sns.set_style('whitegrid')
164 plt.rcParams['font.sans-serif'] = ['SimHei']
165 plt.rcParams['axes.unicode_minus'] = False
166 CorrDataTwoAll = dataTwo.corr().abs()
167 N = 14
168 ColDataTwoRange = CorrDataTwoAll.nlargest(N, '手机上网整体满意度')[['手机上网整体满意度']].index
169 plt.subplots(figsize=(N, N))
170 plt.title('皮尔逊相关系数', size=16)
171 sns.heatmap(dataTwo[ColDataTwoRange].corr(),
172               linewidths=0.1,
173               vmax=1.0,
174               square=True,
175               cmap=plt.cm.winter,
176               linecolor='white',
177               annot=True,
178               annot_kws={"size": 12})
179 plt.xticks(fontsize=14)
180 plt.yticks(fontsize=14)
181 plt.tight_layout()
182 plt.savefig('figuresTwo\\[附件2]皮尔逊相关系数（14个）.pdf')
183
184 # In[20]:
185
186
187 from yellowbrick.features.radviz import RadViz
188
189 plt.rcParams['font.sans-serif'] = ['SimHei']
190 plt.rcParams['axes.unicode_minus'] = False
191
192 x = dataTwo[['网络卡速度慢延时大上不了网总', '出现问题场所或应用总', '网络信号差/没有信号', '手机上网速度慢', '上网过程中网络时断时续或时快时慢', '打开网页或APP图片慢']]
193 y = dataTwo['手机上网整体满意度']
194
195 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
```

```

196 visualizer = RadViz(classes=classes, colormap='winter_r')
197 visualizer.fit(x, y)
198 visualizer.transform(x)
199 visualizer.show(outpath='figuresTwo\\[附件2]手机上网整体满意度RidViz.pdf')
200
201 # In[21]:
202
203
204 from yellowbrick.features.radviz import RadViz
205
206 plt.rcParams['font.sans-serif'] = ['SimHei']
207 plt.rcParams['axes.unicode_minus'] = False
208
209 x = dataTwo[['网络卡速度慢延时大上不了网总', '出现问题场所或应用总', '网络信号差/没有
210     信号', '手机上网速度慢', '上网过程中网络时断时续或时快时慢', '打开网页或APP图片慢
211     ]]
212 y = dataTwo['网络覆盖与信号强度']
213
214 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
215 visualizer = RadViz(classes=classes, colormap='winter_r')
216 visualizer.fit(x, y)
217 visualizer.transform(x)
218 visualizer.show(outpath='figuresTwo\\[附件2]网络覆盖与信号强度RidViz.pdf')
219
220 # In[22]:
221
222
223 from yellowbrick.features.radviz import RadViz
224
225 plt.rcParams['font.sans-serif'] = ['SimHei']
226 plt.rcParams['axes.unicode_minus'] = False
227
228 x = dataTwo[['网络卡速度慢延时大上不了网总', '出现问题场所或应用总', '网络信号差/没有
229     信号', '手机上网速度慢', '上网过程中网络时断时续或时快时慢', '打开网页或APP图片慢
230     ]]
231 y = dataTwo['手机上网速度']
232
233 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
234 visualizer = RadViz(classes=classes, colormap='winter_r')
235 visualizer.fit(x, y)
236 visualizer.transform(x)
237 visualizer.show(outpath='figuresTwo\\[附件2]手机上网速度RidViz.pdf')
238
239 # In[23]:

```

```

236
237
238     from yellowbrick.features.radviz import RadViz
239
240     plt.rcParams['font.sans-serif'] = ['SimHei']
241     plt.rcParams['axes.unicode_minus'] = False
242
243     x = dataTwo[['网络卡速度慢延时大上不了网总', '出现问题场所或应用总', '网络信号差/没有
244         信号', '手机上网速度慢', '上网过程中网络时断时续或时快时慢', '打开网页或APP图片慢',
245         ]]
246
247     y = dataTwo['手机上网稳定性']
248
249     classes = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
250     visualizer = RadViz(classes=classes, colormap='winter_r')
251     visualizer.fit(x, y)
252     visualizer.transform(x)
253     visualizer.show(outpath='figuresTwo\\[附件2]手机上网稳定性RidViz.pdf')
254
255     # ### 数据标准化
256
257     # In[24]:
258
259
260     StandardTransform = dataTwo[['上网质差次数', '当月MOU', '客户星级标识', '脱网次数',
261         '性别', '套外流量费(元)', '微信质差次数', '套外流量(MB)', '终端品牌',
262         '出现问题场所或应用总', '网络卡速度慢延时大上不了网总', '质差总']]
263     StandardTransformScaler = sp.StandardScaler()
264     StandardTransformScaler = StandardTransformScaler.fit(StandardTransform)
265     StandardTransform = StandardTransformScaler.transform(StandardTransform)
266     StandardTransform = pd.DataFrame(StandardTransform)
267     StandardTransform.columns = ['上网质差次数', '当月MOU', '客户星级标识', '脱网次数',
268         '性别', '套外流量费(元)', '微信质差次数', '套外流量(MB)', '终端品牌',
269         '出现问题场所或应用总', '网络卡速度慢延时大上不了网总', '质差总']
270
271     StandardTransform
272
273     # In[25]:
274
275
276     dataTwoLeave = dataTwo.loc[:, ~dataTwo.columns.isin(['上网质差次数', '当月MOU', '客
277         户星级标识', '脱网次数', '性别', '套外流量费(元)', '微信质差次数', '套外流量
278         (MB)', '终端品牌', '出现问题场所或应用总', '网络卡速度慢延时大上不了网总', '质差总'])]
279
280     # In[26]:

```

```

271
272
273     dataTwoNewStandard = pd.concat([dataTwoLeave, StandardTransform], axis=1)
274     dataTwoNewStandard
275
276     # ## 熵权法
277
278     # In[27]:
279
280
281     import copy
282
283
284     def ewm(data):
285         label_need = data.keys() [:]
286         data1 = data[label_need].values
287         data2 = data1
288         [m, n] = data2.shape
289         data3 = copy.deepcopy(data2)
290         y_min = 0.002
291         y_max = 1
292         for j in range(0, n):
293             d_max = max(data2[:, j])
294             d_min = min(data2[:, j])
295             data3[:, j] = (y_max - y_min) * (data2[:, j] - d_min) / (d_max - d_min) +
296                         y_min
297         p = copy.deepcopy(data3)
298         for j in range(0, n):
299             p[:, j] = data3[:, j] / sum(data3[:, j])
300         e = copy.deepcopy(data3[0, :])
301         for j in range(0, n):
302             e[j] = -1 / np.log(m) * sum(p[:, j] * np.log(p[:, j]))
303         w = (1 - e) / sum(1 - e)
304         total = 0
305         for sum_w in range(0, len(w)):
306             total = total + w[sum_w]
307         print(f'权重为: {w}\n权重之和为: {total}')
308
309     # In[28]:
310
311
312     ewm(dataTwo.iloc[:, 4:])
313

```

```

314
315 # ## 灰色关联
316
317 # In[29]:
318
319
320 def grey(data):
321     label_need = data.keys()[:]
322     data1 = data[label_need].values
323     [m, n] = data1.shape
324     data2 = data1.astype('float')
325     data3 = data2
326     ymin = 0.002
327     ymax = 1
328     for j in range(0, n):
329         d_max = max(data2[:, j])
330         d_min = min(data2[:, j])
331         data3[:, j] = (ymax - ymin) * (data2[:, j] - d_min) / (d_max - d_min) + ymin
332
333     for i in range(0, n):
334         data3[:, i] = np.abs(data3[:, i] - data3[:, 0])
335     data4 = data3
336     d_max = np.max(data4)
337     d_min = np.min(data4)
338     a = 0.5
339     data4 = (d_min + a * d_max) / (data4 + a * d_max)
340     xs = np.mean(data4, axis=0)
341     print(xs)
342
343
344 # In[30]:
345
346
347 grey(dataTwo.loc[:, ~dataTwo.columns.isin(['网络覆盖与信号强度', '手机上网速度', '手机上网稳定性'])])
348 grey(dataTwo.loc[:, ~dataTwo.columns.isin(['手机上网整体满意度', '手机上网速度', '手机上网稳定性'])])
349 grey(dataTwo.loc[:, ~dataTwo.columns.isin(['手机上网整体满意度', '网络覆盖与信号强度', '手机上网速度'])])
350 grey(dataTwo.loc[:, ~dataTwo.columns.isin(['手机上网整体满意度', '网络覆盖与信号强度', '手机上网稳定性'])])
351
352 # ## 机器学习
353

```

```

354 # ### 多输出多类别分类
355
356 # In[31]:
357
358
359 XdataTwoMulti = dataTwoNewStandard.loc[:, ~dataTwoNewStandard.columns.isin(['手机上
网整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性'])]
360 ydataTwoMulti = dataTwoNewStandard[['手机上网整体满意度', '网络覆盖与信号强度', '手机
上网速度', '手机上网稳定性']]
361
362 # In[32]:
363
364
365 from sklearn.model_selection import train_test_split
366
367 XdataTwoMulti_train, XdataTwoMulti_test, ydataTwoMulti_train, ydataTwoMulti_test =
train_test_split(XdataTwoMulti, ydataTwoMulti, test_size=0.2, random_state=2022)
368
369 # In[33]:
370
371
372 from sklearn.tree import DecisionTreeClassifier
373 from sklearn.ensemble import RandomForestClassifier
374
375 DecisionTreeMulti = DecisionTreeClassifier(random_state=2022)
376 RandomForestMulti = RandomForestClassifier(random_state=2022)
377 DecisionTreeMulti = DecisionTreeMulti.fit(XdataTwoMulti_train, ydataTwoMulti_train)
378 RandomForestMulti = RandomForestMulti.fit(XdataTwoMulti_train, ydataTwoMulti_train)
379
380 # In[34]:
381
382
383 from sklearn.metrics import mean_absolute_error
384 from sklearn.metrics import mean_squared_error
385
386 print(f'决策树平均绝对误差: ',
387 f'{mean_absolute_error(ydataTwoMulti_test, DecisionTreeMulti.predict(
XdataTwoMulti_test), sample_weight=None, multioutput="uniform_average")}\n
',
388 f'决策树均方误差: ',
389 f'{mean_squared_error(ydataTwoMulti_test, DecisionTreeMulti.predict(
XdataTwoMulti_test), sample_weight=None, multioutput="uniform_average"))}')
390 print(f'随机森林平均绝对误差: ',
391 f'{mean_absolute_error(ydataTwoMulti_test, RandomForestMulti.predict(

```

```

        XdataTwoMulti_test), sample_weight=None, multioutput="uniform_average")}\n
        ,\n
        f'随机森林均方误差: ,\n
        f'{mean_squared_error(ydataTwoMulti_test, RandomForestMulti.predict(\n
            XdataTwoMulti_test), sample_weight=None, multioutput="uniform_average")})\n
\n
394\n
395 # In[35]:\n
396\n
397\n
398 std = np.std([i.feature_importances_ for i in RandomForestMulti.estimators_], axis\n
    =0)\n
399 importances = DecisionTreeMulti.feature_importances_\n
400 feat_with_importance = pd.Series(importances, XdataTwoMulti.columns)\n
401 fig, ax = plt.subplots(figsize=(12, 5))\n
402 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 127 / 255, 215 /\n
    255))\n
403 ax.set_title("上网业务四项评分各指标特征重要平均程度")\n
404 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')\n
405 plt.yticks(font='Times New Roman')\n
406 plt.tight_layout()\n
407 plt.savefig('figuresTwo\\[附件2]上网业务四项评分各指标特征重要平均程度.pdf')\n
408\n
409 # In[36]:\n
410\n
411\n
412 feat_with_importance\n
413\n
414 # In[37]:\n
415\n
416\n
417 from sklearn.decomposition import PCA\n
418\n
419 pca = PCA()\n
420 pca.fit(dataTwoNewStandard)\n
421 evr = pca.explained_variance_ratio_\n
422\n
423 plt.figure(figsize=(12, 5))\n
424 plt.plot(range(0, len(evr)), evr.cumsum(), marker="d", linestyle="--")\n
425 plt.xlabel("Number of components", font='Times New Roman')\n
426 plt.ylabel("Cumulative explained variance", font='Times New Roman')\n
427 plt.xticks(font='Times New Roman')\n
428 plt.yticks(font='Times New Roman')\n
429 plt.tight_layout()\n
430 plt.savefig("figuresTwo\\[附件2]PCA累计解释方差图.pdf")\n

```

```

431
432 # In[38]:
433
434
435 from sklearn.multioutput import MultiOutputClassifier
436
437 RandomForestMulti = MultiOutputClassifier(RandomForestClassifier(random_state=2022)
438 )
439 RandomForestMulti = RandomForestMulti.fit(XdataTwoMulti_train, ydataTwoMulti_train)
440 RandomForestMulti_score = RandomForestMulti.score(XdataTwoMulti_test,
441 ydataTwoMulti_test)
442 print(f'随机森林平均绝对误差: ',
443 f'{mean_absolute_error(ydataTwoMulti_test, RandomForestMulti.predict(
444 XdataTwoMulti_test), sample_weight=None, multioutput="uniform_average")}\n',
445
446 f'随机森林均方误差: ',
447 f'{mean_squared_error(ydataTwoMulti_test, RandomForestMulti.predict(
448 XdataTwoMulti_test), sample_weight=None, multioutput="uniform_average")}')
449 RandomForestMulti_score
450
451 # ### "手机上网整体满意度"学习
452
453 # In[39]:
454
455 XdataTwoFirst = dataTwoNewStandard.loc[:, ~dataTwoNewStandard.columns.isin(['手机上
456 网整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性'])]
457 ydataTwoFirst = dataTwoNewStandard['手机上网整体满意度']
458 XdataTwoFirst_train, XdataTwoFirst_test, ydataTwoFirst_train, ydataTwoFirst_test =
459 train_test_split(XdataTwoFirst, ydataTwoFirst, test_size=0.1, random_state=2022)
460
461 # ##### 决策树、随机森林
462
463 # In[40]:
464
465 DecisionTreeFirst = DecisionTreeClassifier(random_state=2022, min_samples_leaf=16)
466 RandomForestFirst = RandomForestClassifier(random_state=2022, n_estimators=166,
467 min_samples_leaf=16)
468 DecisionTreeFirst = DecisionTreeFirst.fit(XdataTwoFirst_train, ydataTwoFirst_train)
469 RandomForestFirst = RandomForestFirst.fit(XdataTwoFirst_train, ydataTwoFirst_train)
470 RandomForestFirst_score = RandomForestFirst.score(XdataTwoFirst_test,
471 ydataTwoFirst_test)
472 RandomForestFirst_score

```

```

466
467 # In[41]:
468
469
470 std = np.std([i.feature_importances_ for i in RandomForestFirst.estimators_], axis
   =0)
471 importances = DecisionTreeFirst.feature_importances_
472 feat_with_importance = pd.Series(importances, XdataTwoFirst.columns)
473 fig, ax = plt.subplots(figsize=(12, 5))
474 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
   255))
475 ax.set_title("手机上网整体满意度各项指标重要程度")
476 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
477 plt.yticks(font='Times New Roman')
478 plt.tight_layout()
479 plt.savefig("figuresTwo\\[附件2]手机上网整体满意度各项指标重要程度.pdf")
480
481 # In[42]:
482
483
484 feat_with_importance
485
486 # #### XGBoost
487
488 # In[43]:
489
490
491 from xgboost import XGBClassifier
492
493 XGBFirst = XGBClassifier(learning_rate=0.01,
   n_estimators=17,
   max_depth=3,
   min_child_weight=1,
   gamma=0.,
   subsample=1,
   colsample_btree=1,
   scale_pos_weight=1,
   random_state=2022,
   silent=0)
494 XGBFirst.fit(XdataTwoFirst_train, ydataTwoFirst_train)
495 XGBFirst_score = XGBFirst.score(XdataTwoFirst_test, ydataTwoFirst_test)
496 XGBFirst_score
497
498 # In[44]:

```

```

508
509
510     from xgboost import plot_importance
511
512     fig, ax = plt.subplots(figsize=(14, 8))
513     plot_importance(XGBFirst, height=0.4, ax=ax)
514     plt.xticks(fontsize=13, font='Times New Roman')
515     plt.yticks(fontsize=11)
516     ax.set_title("")
517     ax.set_ylabel("")
518     ax.set_xlabel("")
519     plt.tight_layout()
520     plt.savefig('figuresTwo\\[附件2]手机上网整体满意度各项指标重要程度 (XGBoost,F-score)
521             .pdf')
522
523     # ##### KNN
524
525
526
527     from sklearn.neighbors import KNeighborsClassifier
528
529     KNNFirst = KNeighborsClassifier(n_neighbors=36, p=1, weights='distance')
530     KNNFirst.fit(XdataTwoFirst_train, ydataTwoFirst_train)
531     KNNFirst_score = KNNFirst.score(XdataTwoFirst_test, ydataTwoFirst_test)
532     KNNFirst_score
533
534     # ##### 支持向量机
535
536     # In[46]:
537
538
539     from sklearn.svm import SVC
540
541     SVMFirst = SVC(random_state=2022)
542     SVMFirst.fit(XdataTwoFirst_train, ydataTwoFirst_train)
543     SVMFirst_score = SVMFirst.score(XdataTwoFirst_test, ydataTwoFirst_test)
544     SVMFirst_score
545
546     # ##### lightgbm
547
548     # In[47]:
549
550

```

```

551 from lightgbm import LGBMClassifier
552
553 LightgbmFirst = LGBMClassifier(learning_rate=0.1,
554                                 lambda_l1=0.1,
555                                 lambda_l2=0.2,
556                                 max_depth=1,
557                                 objective='multiclass',
558                                 num_class=3,
559                                 random_state=2022)
560 LightgbmFirst.fit(XdataTwoFirst_train, ydataTwoFirst_train)
561 LightgbmFirst_score = LightgbmFirst.score(XdataTwoFirst_test, ydataTwoFirst_test)
562 LightgbmFirst_score
563
564 # ##### 逻辑回归
565
566 # In[48]:
567
568
569 from sklearn.linear_model import LogisticRegression
570
571 LogisticRegressionFirst = LogisticRegression(multi_class="multinomial", solver=
572                                               "newton-cg", max_iter=2000)
573 LogisticRegressionFirst = LogisticRegressionFirst.fit(XdataTwoFirst_train,
574                                                       ydataTwoFirst_train)
575 LogisticRegressionFirst_score = LogisticRegressionFirst.score(XdataTwoFirst_test,
576                                                               ydataTwoFirst_test)
577 LogisticRegressionFirst_score
578
579 # In[49]:
580
581 print(f'模型一中RF平均绝对误差: ',
582       f'{mean_absolute_error(ydataTwoFirst_test, RandomForestFirst.predict(
583                               XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n',
584       ,
585       f'模型一中RF均方误差: ',
586       f'{mean_squared_error(ydataTwoFirst_test, RandomForestFirst.predict(
587                               XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")})')
588 print(f'模型一中XGBoost平均绝对误差: ',
589       f'{mean_absolute_error(ydataTwoFirst_test, XGBFirst.predict(XdataTwoFirst_test
590                               ), sample_weight=None, multioutput="uniform_average")}\n',
591       f'模型一中XGBoost均方误差: ',
592       f'{mean_squared_error(ydataTwoFirst_test, XGBFirst.predict(XdataTwoFirst_test
593                               ), sample_weight=None, multioutput="uniform_average")})')

```

```

587 print(f'模型一中KNN平均绝对误差: ,
588     f'{mean_absolute_error(ydataTwoFirst_test, KNNFirst.predict(XdataTwoFirst_test
589         ), sample_weight=None, multioutput="uniform_average")}\n'
590     f'模型一中KNN均方误差: ,
591     f'{mean_squared_error(ydataTwoFirst_test, KNNFirst.predict(XdataTwoFirst_test)
592         , sample_weight=None, multioutput="uniform_average")}')
593 print(f'模型一中SVM平均绝对误差: ,
594     f'{mean_absolute_error(ydataTwoFirst_test, SVMFirst.predict(XdataTwoFirst_test
595         ), sample_weight=None, multioutput="uniform_average")}\n'
596     f'模型一中SVM均方误差: ,
597     f'{mean_squared_error(ydataTwoFirst_test, SVMFirst.predict(XdataTwoFirst_test)
598         , sample_weight=None, multioutput="uniform_average")}')
599 print(f'模型一中LightGBM平均绝对误差: ,
600     f'{mean_absolute_error(ydataTwoFirst_test, LightgbmFirst.predict(
601         XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
602     ,
603     f'模型一中LightGBM均方误差: ,
604     f'{mean_squared_error(ydataTwoFirst_test, LightgbmFirst.predict(
605         XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}')
606 print(f'模型一中LR平均绝对误差: ,
607     f'{mean_absolute_error(ydataTwoFirst_test, LogisticRegressionFirst.predict(
608         XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
609     ,
610     f'模型一中LR均方误差: ,
611     f'{mean_squared_error(ydataTwoFirst_test, LogisticRegressionFirst.predict(
612         XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}')
613
614 # ##### 集成学习
615
616 # In[50]:
617
618
619 from mlxtend.classifier import StackingCVClassifier
620
621 FirstModel = StackingCVClassifier(
622     classifiers=[LogisticRegressionFirst, XGBFirst, KNNFirst, RandomForestFirst,
623                 LightgbmFirst],
624     meta_classifier=SVC(random_state=2022), random_state=2022, cv=5)
625 FirstModel.fit(XdataTwoFirst_train, ydataTwoFirst_train)
626 FirstModel_score = FirstModel.score(XdataTwoFirst_test, ydataTwoFirst_test)
627 FirstModel_score
628
629 # In[51]:

```

```

620
621     print(f'模型一平均绝对误差: ,
622           f'{mean_absolute_error(ydataTwoFirst_test, FirstModel.predict(
623             XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n
624             ,
625             f'模型一均方误差: ,
626             f'{mean_squared_error(ydataTwoFirst_test, FirstModel.predict(
627               XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}'')
628
629 # ### "网络覆盖与信号强度"学习
630
631 # In[52]:
632
633 XdataTwoSecond = dataTwoNewStandard.loc[:, ~dataTwoNewStandard.columns.isin(['手机上
634 网整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性'])]
635 ydataTwoSecond = dataTwoNewStandard['网络覆盖与信号强度']
636 XdataTwoSecond_train, XdataTwoSecond_test, ydataTwoSecond_train,
637     ydataTwoSecond_test = train_test_split(XdataTwoSecond, ydataTwoSecond, test_size
638     =0.1, random_state=2022)
639
640 # ##### 决策树、随机森林
641
642 # In[53]:
643
644 DecisionTreeSecond = DecisionTreeClassifier(random_state=2022)
645 RandomForestSecond = RandomForestClassifier(random_state=2022, n_estimators=159,
646     min_samples_leaf=20)
647 DecisionTreeSecond = DecisionTreeSecond.fit(XdataTwoSecond_train,
648     ydataTwoSecond_train)
649 RandomForestSecond = RandomForestSecond.fit(XdataTwoSecond_train,
650     ydataTwoSecond_train)
651 RandomForestSecond_score = RandomForestSecond.score(XdataTwoSecond_test,
652     ydataTwoSecond_test)
653 RandomForestSecond_score
654
655 # In[54]:
656
657 std = np.std([i.feature_importances_ for i in RandomForestSecond.estimators_], axis
658     =0)
659 importances = DecisionTreeSecond.feature_importances_
660 feat_with_importance = pd.Series(importances, XdataTwoSecond.columns)

```

```

653     fig, ax = plt.subplots(figsize=(12, 5))
654     feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
655         255))
656     ax.set_title("网络覆盖与信号强度各项指标重要程度")
657     ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
658     plt.yticks(font='Times New Roman')
659     plt.tight_layout()
660     plt.savefig("figuresTwo\\[附件2]网络覆盖与信号强度各项指标重要程度.pdf")
661
662 # In[55]:
663
664 feat_with_importance
665
666 # #### XGBoost
667
668 # In[56]:
669
670
671 from xgboost import XGBClassifier
672
673 XGBSecond = XGBClassifier(learning_rate=0.02,
674                             n_estimators=17,
675                             max_depth=6,
676                             min_child_weight=1,
677                             gamma=0.05,
678                             subsample=1,
679                             colsample_btree=1,
680                             scale_pos_weight=1,
681                             random_state=2022,
682                             silent=0)
683 XGBSecond.fit(XdataTwoSecond_train, ydataTwoSecond_train)
684 XGBSecond_score = XGBSecond.score(XdataTwoSecond_test, ydataTwoSecond_test)
685 XGBSecond_score
686
687 # In[57]:
688
689
690 from xgboost import plot_importance
691
692 fig, ax = plt.subplots(figsize=(14, 8))
693 plot_importance(XGBSecond, height=0.4, ax=ax, max_num_features=30)
694 plt.xticks(fontsize=13, font='Times New Roman')
695 plt.yticks(fontsize=11)

```

```

696     ax.set_title("")
697     ax.set_ylabel("")
698     ax.set_xlabel("")
699     plt.tight_layout()
700     plt.savefig('figuresTwo\\[附件2] 网络覆盖与信号强度各项指标重要程度 (XGBoost,F-score)
701         .pdf')
702
703     # ##### KNN
704
705
706
707     from sklearn.neighbors import KNeighborsClassifier
708
709     KNNSecond = KNeighborsClassifier(algorithm='auto', leaf_size=42,
710                                         metric='minkowski',
711                                         n_jobs=-1,
712                                         n_neighbors=46, p=1,
713                                         weights='distance')
714     KNNSecond.fit(XdataTwoSecond_train, ydataTwoSecond_train)
715     KNNSecond_score = KNNSecond.score(XdataTwoSecond_test, ydataTwoSecond_test)
716     KNNSecond_score
717
718     # ##### 支持向量机
719
720     # In[59]:
721
722
723     from sklearn.svm import SVC
724
725     SVMSecond = SVC(random_state=2022)
726     SVMSecond.fit(XdataTwoSecond_train, ydataTwoSecond_train)
727     SVMSecond_score = SVMSecond.score(XdataTwoSecond_test, ydataTwoSecond_test)
728     SVMSecond_score
729
730     # ##### lightgbm
731
732     # In[60]:
733
734
735     from lightgbm import LGBMClassifier
736
737     LightgbmSecond = LGBMClassifier(learning_rate=0.1,
738                                     lambda_l1=0.1,

```

```

739             lambda_l2=0.2,
740             max_depth=5,
741             objective='multiclass',
742             num_class=3,
743             random_state=2022)
744 LightgbmSecond.fit(XdataTwoSecond_train, ydataTwoSecond_train)
745 LightgbmSecond_score = LightgbmSecond.score(XdataTwoSecond_test,
746     ydataTwoSecond_test)
747 LightgbmSecond_score
748 # ##### 逻辑回归
749
750 # In[61]:
751
752
753 from sklearn.linear_model import LogisticRegression
754
755 LogisticRegressionSecond = LogisticRegression(multi_class="multinomial", solver=
756     "newton-cg", max_iter=3000)
757 LogisticRegressionSecond = LogisticRegressionSecond.fit(XdataTwoSecond_train,
758     ydataTwoSecond_train)
759 LogisticRegressionSecond_score = LogisticRegressionSecond.score(XdataTwoSecond_test
760     , ydataTwoSecond_test)
761 LogisticRegressionSecond_score
762
763 # In[62]:
764
765
766 print(f'模型二中RF平均绝对误差: ',
767     f'{mean_absolute_error(ydataTwoSecond_test, RandomForestSecond.predict(
768         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
769     n',
770     f'模型二中RF均方误差: ',
771     f'{mean_squared_error(ydataTwoSecond_test, RandomForestSecond.predict(
772         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
773     )
774
775 print(f'模型二中XGBoost平均绝对误差: ',
776     f'{mean_absolute_error(ydataTwoSecond_test, XGBSecond.predict(
777         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
778     n',
779     f'模型二中XGBoost均方误差: ',
780     f'{mean_squared_error(ydataTwoSecond_test, XGBSecond.predict(
781         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
782     )

```

```

771 print(f'模型二中KNN平均绝对误差: ,
772     f'{mean_absolute_error(ydataTwoSecond_test, KNNSecond.predict(
773         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
774         n',
775     f'模型二中KNN均方误差: ,
776     f'{mean_squared_error(ydataTwoSecond_test, KNNSecond.predict(
777         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
778         )
779 print(f'模型二中SVM平均绝对误差: ,
780     f'{mean_absolute_error(ydataTwoSecond_test, SVMSecond.predict(
781         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
782         n',
783     f'模型二中SVM均方误差: ,
784     f'{mean_squared_error(ydataTwoSecond_test, SVMSecond.predict(
785         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
786         )
787 print(f'模型二中LightGBM平均绝对误差: ,
788     f'{mean_absolute_error(ydataTwoSecond_test, LightgbmSecond.predict(
789         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
790         n',
791     f'模型二中LightGBM均方误差: ,
792     f'{mean_squared_error(ydataTwoSecond_test, LightgbmSecond.predict(
793         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
794         )
795 # ##### 集成学习
796 # In[63]:
797
798 from mlxtend.classifier import StackingCVClassifier
799
800 SecondModel = StackingCVClassifier(
801     classifiers=[RandomForestSecond, XGBSecond, KNNSecond, LogisticRegressionSecond,
802                 LightgbmSecond],
803     meta_classifier=SVC(random_state=2022), random_state=2022, cv=5)

```

```

798 SecondModel.fit(XdataTwoSecond_train, ydataTwoSecond_train)
799 SecondModel_score = SecondModel.score(XdataTwoSecond_test, ydataTwoSecond_test)
800 SecondModel_score
801
802 # In[64]:
803
804
805 print(f'模型二平均绝对误差: ,'
806     f'{mean_absolute_error(ydataTwoSecond_test, SecondModel.predict(
807         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
808         n'
809     f'模型二均方误差: ,'
810     f'{mean_squared_error(ydataTwoSecond_test, SecondModel.predict(
811         XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")},'
812         )
813
814
815 XdataTwoThird = dataTwoNewStandard.loc[:, ~dataTwoNewStandard.columns.isin(['手机上
816 网整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性'])]
817 ydataTwoThird = dataTwoNewStandard['手机上网速度']
818 XdataTwoThird_train, XdataTwoThird_test, ydataTwoThird_train, ydataTwoThird_test =
819     train_test_split(XdataTwoThird, ydataTwoThird, test_size=0.1, random_state=2022)
820
821 # ##### 决策树、随机森林
822
823
824 DecisionTreeThird = DecisionTreeClassifier(random_state=2022)
825 RandomForestThird = RandomForestClassifier(random_state=2022, n_estimators=162,
826     min_samples_leaf=20)
827 DecisionTreeThird = DecisionTreeThird.fit(XdataTwoThird_train, ydataTwoThird_train)
828 RandomForestThird = RandomForestThird.fit(XdataTwoThird_train, ydataTwoThird_train)
829 RandomForestThird_score = RandomForestThird.score(XdataTwoThird_test,
830     ydataTwoThird_test)
831 RandomForestThird_score
832
833 # In[67]:

```

```

834     std = np.std([i.feature_importances_ for i in RandomForestThird.estimators_], axis
835                 =0)
836
837     importances = DecisionTreeThird.feature_importances_
838     feat_with_importance = pd.Series(importances, XdataTwoThird.columns)
839     fig, ax = plt.subplots(figsize=(12, 5))
840     feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
841                                   255))
842     ax.set_title("手机上网速度各项指标重要程度")
843     ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
844     plt.yticks(font='Times New Roman')
845     plt.tight_layout()
846     plt.savefig("figuresTwo\\[附件2]手机上网速度各项指标重要程度.pdf")
847
848 # In[68]:
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
feat_with_importance
# #### XGBoost
# In[69]:
from xgboost import XGBClassifier
XGBThird = XGBClassifier(learning_rate=0.02,
                         n_estimators=16,
                         max_depth=8,
                         min_child_weight=1,
                         gamma=0.05,
                         subsample=1,
                         colsample_btree=1,
                         scale_pos_weight=1,
                         random_state=2022,
                         silent=0)
XGBThird.fit(XdataTwoThird_train, ydataTwoThird_train)
XGBThird_score = XGBThird.score(XdataTwoThird_test, ydataTwoThird_test)
XGBThird_score
# In[70]:
from xgboost import plot_importance

```

```

876     fig, ax = plt.subplots(figsize=(14, 8))
877     plot_importance(XGBThird, height=0.4, ax=ax, max_num_features=20)
878     plt.xticks(fontsize=13, font='Times New Roman')
879     plt.yticks(fontsize=11)
880     ax.set_title("")
881     ax.set_ylabel("")
882     ax.set_xlabel("")
883     plt.tight_layout()
884     plt.savefig('figuresTwo\\[附件2]手机上网速度各项指标重要程度 (XGBoost,F-score) .pdf')
885
886     # ##### KNN
887
888     # In[71]:
889
890
891     from sklearn.neighbors import KNeighborsClassifier
892
893     KNNThird = KNeighborsClassifier(n_neighbors=35, p=1)
894     KNNThird.fit(XdataTwoThird_train, ydataTwoThird_train)
895     KNNThird_score = KNNThird.score(XdataTwoThird_test, ydataTwoThird_test)
896     KNNThird_score
897
898     # ##### 支持向量机
899
900     # In[72]:
901
902
903     from sklearn.svm import SVC
904
905     SVMThird = SVC(random_state=2022)
906     SVMThird.fit(XdataTwoThird_train, ydataTwoThird_train)
907     SVMThird_score = SVMThird.score(XdataTwoThird_test, ydataTwoThird_test)
908     SVMThird_score
909
910     # ##### lightgbm
911
912     # In[73]:
913
914
915     LightgbmThird = LGBMClassifier(learning_rate=0.1,
916                                     lambda_l1=0.1,
917                                     lambda_l2=0.2,
918                                     max_depth=16,
919                                     objective='multiclass',

```

```

920                 num_class=4,
921                 random_state=2022)
922 LightgbmThird.fit(XdataTwoThird_train, ydataTwoThird_train)
923 LightgbmThird_score = LightgbmThird.score(XdataTwoThird_test, ydataTwoThird_test)
924 LightgbmThird_score
925
926 # ##### 逻辑回归
927
928 # In[74]:
929
930
931 from sklearn.linear_model import LogisticRegression
932
933 LogisticRegressionThird = LogisticRegression(multi_class="multinomial", solver=
934     "newton-cg", max_iter=2000)
935 LogisticRegressionThird = LogisticRegressionThird.fit(XdataTwoThird_train,
936     ydataTwoThird_train)
937 LogisticRegressionThird_score = LogisticRegressionThird.score(XdataTwoThird_test,
938     ydataTwoThird_test)
939 LogisticRegressionThird_score
940
941
942 print(f'模型三中RF平均绝对误差: ,
943       f'{mean_absolute_error(ydataTwoThird_test, RandomForestThird.predict(
944           XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n
945       ,
946       f'模型三中RF均方误差: ,
947       f'{mean_squared_error(ydataTwoThird_test, RandomForestThird.predict(
948           XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")})')
949 print(f'模型三中XGBoost平均绝对误差: ,
950       f'{mean_absolute_error(ydataTwoThird_test, XGBThird.predict(XdataTwoThird_test
951           ), sample_weight=None, multioutput="uniform_average")}\n'
952       f'模型三中XGBoost均方误差: ,
953       f'{mean_squared_error(ydataTwoThird_test, XGBThird.predict(XdataTwoThird_test
954           ), sample_weight=None, multioutput="uniform_average")})')
955 print(f'模型三中KNN平均绝对误差: ,
956       f'{mean_absolute_error(ydataTwoThird_test, KNNThird.predict(XdataTwoThird_test
957           ), sample_weight=None, multioutput="uniform_average")}\n'
958       f'模型三中KNN均方误差: ,
959       f'{mean_squared_error(ydataTwoThird_test, KNNThird.predict(XdataTwoThird_test
960           ), sample_weight=None, multioutput="uniform_average")})')
961 print(f'模型三中SVM平均绝对误差: ,

```

```

954         f'{mean_absolute_error(ydataTwoThird_test, SVMThird.predict(XdataTwoThird_test
955             ), sample_weight=None, multioutput="uniform_average")}\n'
956         f'模型三中SVM均方误差: '
957         f'{mean_squared_error(ydataTwoThird_test, SVMThird.predict(XdataTwoThird_test)
958             , sample_weight=None, multioutput="uniform_average")})'
959     print(f'模型三中LightGBM平均绝对误差: '
960           f'{mean_absolute_error(ydataTwoThird_test, LightgbmThird.predict(
961               XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n'
962           ,
963           f'模型三中LightGBM均方误差: '
964           f'{mean_squared_error(ydataTwoThird_test, LightgbmThird.predict(
965               XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")})')
966     print(f'模型三中LR平均绝对误差: '
967           f'{mean_absolute_error(ydataTwoThird_test, LogisticRegressionThird.predict(
968               XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n'
969           ,
970           f'模型三中LR均方误差: '
971           f'{mean_squared_error(ydataTwoThird_test, LogisticRegressionThird.predict(
972               XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")})')
973
974     # ##### 集成学习
975
976     # In[76]:
977
978
979     from mlxtend.classifier import StackingCVClassifier
980
981
982     ThirdModel = StackingCVClassifier(
983         classifiers=[XGBThird, LightgbmThird, KNNThird, RandomForestThird,
984             LogisticRegressionThird],
985         meta_classifier=SVC(random_state=2022), random_state=2022, cv=5)
986     ThirdModel.fit(XdataTwoThird_train, ydataTwoThird_train)
987     ThirdModel_score = ThirdModel.score(XdataTwoThird_test, ydataTwoThird_test)
988     ThirdModel_score
989
990
991     # In[77]:
992
993
994     print(f'模型三平均绝对误差: '
995           f'{mean_absolute_error(ydataTwoThird_test, ThirdModel.predict(
996               XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n'
997           ,
998           f'模型三均方误差: '
999           f'{mean_squared_error(ydataTwoThird_test, ThirdModel.predict(

```

```

987 XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}]')
988 # ### "手机上网稳定性"学习
989
990 # In[78]:
991
992
993 XdataTwoFourth = dataTwoNewStandard.loc[:, ~dataTwoNewStandard.columns.isin(['手机上
网整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性'])]
994 ydataTwoFourth = dataTwoNewStandard['手机上网稳定性']
995 XdataTwoFourth_train, XdataTwoFourth_test, ydataTwoFourth_train,
ydataTwoFourth_test = train_test_split(XdataTwoFourth, ydataTwoFourth, test_size
=0.1, random_state=2022)
996
997 # ##### 决策树、随机森林
998
999 # In[79]:
1000
1001
1002 DecisionTreeFourth = DecisionTreeClassifier(random_state=2022)
1003 RandomForestFourth = RandomForestClassifier(random_state=2022, n_estimators=166,
min_samples_leaf=20)
1004 DecisionTreeFourth = DecisionTreeFourth.fit(XdataTwoFourth_train,
ydataTwoFourth_train)
1005 RandomForestFourth = RandomForestFourth.fit(XdataTwoFourth_train,
ydataTwoFourth_train)
1006 RandomForestFourth_score = RandomForestFourth.score(XdataTwoFourth_test,
ydataTwoFourth_test)
1007 RandomForestFourth_score
1008
1009 # In[80]:
1010
1011
1012 std = np.std([i.feature_importances_ for i in RandomForestFourth.estimators_], axis
=0)
1013 importances = DecisionTreeFourth.feature_importances_
1014 feat_with_importance = pd.Series(importances, XdataTwoFourth.columns)
1015 fig, ax = plt.subplots(figsize=(12, 5))
1016 feat_with_importance.plot.bar(yerr=std, ax=ax, color=(5 / 255, 126 / 255, 215 /
255))
1017 ax.set_title("手机上网稳定性各项指标重要程度")
1018 ax.set_ylabel("Mean decrease in impurity", font='Times New Roman')
1019 plt.yticks(font='Times New Roman')
1020 plt.tight_layout()

```

```

1021 plt.savefig("figuresTwo\\[附件2]手机上网稳定性各项指标重要程度.pdf")
1022
1023 # In[81]:
1024
1025
1026 feat_with_importance
1027
1028 # ##### XGBoost
1029
1030 # In[82]:
1031
1032
1033 from xgboost import XGBClassifier
1034
1035 XGBFourth = XGBClassifier(learning_rate=0.02,
1036                             n_estimators=14,
1037                             max_depth=8,
1038                             min_child_weight=1,
1039                             gamma=0.05,
1040                             subsample=1,
1041                             colsample_btree=1,
1042                             scale_pos_weight=1,
1043                             random_state=2022,
1044                             silent=0)
1045 XGBFourth.fit(XdataTwoFourth_train, ydataTwoFourth_train)
1046 XGBFourth_score = XGBFourth.score(XdataTwoFourth_test, ydataTwoFourth_test)
1047 XGBFourth_score
1048
1049 # In[83]:
1050
1051
1052 from xgboost import plot_importance
1053
1054 fig, ax = plt.subplots(figsize=(14, 8))
1055 plot_importance(XGBFourth, height=0.4, ax=ax, max_num_features=20)
1056 plt.xticks(fontsize=13, font='Times New Roman')
1057 plt.yticks(fontsize=11)
1058 ax.set_title("")
1059 ax.set_ylabel("")
1060 ax.set_xlabel("")
1061 plt.tight_layout()
1062 plt.savefig('figuresTwo\\[附件2]手机上网稳定性各项指标重要程度 (XGBoost,F-score).pdf
1063 ')

```

```

1064 # ##### KNN
1065
1066 # In[84]:
1067
1068
1069 from sklearn.neighbors import KNeighborsClassifier
1070
1071 KNNFourth = KNeighborsClassifier(n_neighbors=36, p=1, )
1072 KNNFourth.fit(XdataTwoFourth_train, ydataTwoFourth_train)
1073 KNNFourth_score = KNNFourth.score(XdataTwoFourth_test, ydataTwoFourth_test)
1074 KNNFourth_score
1075
1076 # ##### 支持向量机
1077
1078 # In[85]:
1079
1080
1081 from sklearn.svm import SVC
1082
1083 SVMFourth = SVC(random_state=2022)
1084 SVMFourth.fit(XdataTwoFourth_train, ydataTwoFourth_train)
1085 SVMFourth_score = SVMFourth.score(XdataTwoFourth_test, ydataTwoFourth_test)
1086 SVMFourth_score
1087
1088 # ##### lightgbm
1089
1090 # In[86]:
1091
1092
1093 from lightgbm import LGBMClassifier
1094
1095 LightgbmFourth = LGBMClassifier(learning_rate=0.1,
1096                                 lambda_l1=0.1,
1097                                 lambda_l2=0.2,
1098                                 max_depth=14,
1099                                 objective='multiclass',
1100                                 num_class=4,
1101                                 random_state=2022)
1102 LightgbmFourth.fit(XdataTwoFourth_train, ydataTwoFourth_train)
1103 LightgbmFourth_score = LightgbmFourth.score(XdataTwoFourth_test,
1104                                              ydataTwoFourth_test)
1105 LightgbmFourth_score
1106 # ##### 逻辑回归

```

```

1107
1108 # In[87]:
1109
1110
1111 from sklearn.linear_model import LogisticRegression
1112
1113 LogisticRegressionFourth = LogisticRegression(multi_class="multinomial", solver="newton-cg", max_iter=2000)
1114 LogisticRegressionFourth = LogisticRegressionFourth.fit(XdataTwoFourth_train,
1115     ydataTwoFourth_train)
1116 LogisticRegressionFourth_score = LogisticRegressionFourth.score(XdataTwoFourth_test,
1117     , ydataTwoFourth_test)
1118 LogisticRegressionFourth_score
1119
1120
1121 # In[88]:
1122
1123 print(f'模型四中RF平均绝对误差: ',
1124     f'{mean_absolute_error(ydataTwoFourth_test, RandomForestFourth.predict(
1125         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n'
1126
1127 f'模型四中RF均方误差: ',
1128 f'{mean_squared_error(ydataTwoFourth_test, RandomForestFourth.predict(
1129         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n'
1130
1131 f'模型四中XGBoost平均绝对误差: ',
1132 f'{mean_absolute_error(ydataTwoFourth_test, XGBFourth.predict(
1133         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n'
1134
1135 f'模型四中XGBoost均方误差: ',
1136 f'{mean_squared_error(ydataTwoFourth_test, XGBFourth.predict(
1137         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n'
1138
1139 print(f'模型四中KNN平均绝对误差: ',
1140     f'{mean_absolute_error(ydataTwoFourth_test, KNNFourth.predict(
1141         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n'
1142
1143 f'模型四中KNN均方误差: ',
1144 f'{mean_squared_error(ydataTwoFourth_test, KNNFourth.predict(
1145         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n'
1146
1147 print(f'模型四中SVM平均绝对误差: ',
1148     f'{mean_absolute_error(ydataTwoFourth_test, SVMFourth.predict(
1149         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n'

```

```

        n'
1135     f'模型四中SVM均方误差: ,
1136     f'{mean_squared_error(ydataTwoFourth_test, SVMFourth.predict(
1137         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n
1138 print(f'模型四中LightGBM平均绝对误差: ,
1139     f'{mean_absolute_error(ydataTwoFourth_test, LightgbmFourth.predict(
1140         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n
1141     n'
1142     f'模型四中LightGBM均方误差: ,
1143     f'{mean_squared_error(ydataTwoFourth_test, LightgbmFourth.predict(
1144         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n
1145
1146 # ##### 集成学习
1147
1148 # In[89]:
1149
1150
1151 from mlxtend.classifier import StackingCVClassifier
1152
1153 FourthModel = StackingCVClassifier(
1154     classifiers=[RandomForestFourth, LightgbmFourth, KNNFourth,
1155                 LogisticRegressionFourth, XGBFourth],
1156     meta_classifier=SVC(random_state=2022), random_state=2022, cv=5)
1157 FourthModel.fit(XdataTwoFourth_train, ydataTwoFourth_train)
1158 FourthModel_score = FourthModel.score(XdataTwoFourth_test, ydataTwoFourth_test)
1159 FourthModel_score
1160
1161
1162
1163 print(f'模型四平均绝对误差: ,
1164     f'{mean_absolute_error(ydataTwoFourth_test, FourthModel.predict(
1165         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n

```

```

1165     f'模型四均方误差: ' ,
1166     f'{mean_squared_error(ydataTwoFourth_test, FourthModel.predict(
1167         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',
1168
1169 # ## 预测附件4四项评分
1170
1171
1172 # In[91]:
1173
1174
1175 dataFour
1176
1177 # In[92]:
1178
1179
1180 dataFour.replace({}{'居民小区': {-1: 0}, '是否5G网络客户': {'否': 0, '是': 1}, '高校': {-1: 0, 3: 1}, '是否不限量套餐到达用户': {'否': 0, '是': 1}, '其他, 请注明.5': {-1: 0, 98: 1}, '咪咕视频': {-1: 0, 9: 1}, '阴阳师': {-1: 0, 10: 1}, '手机QQ': {-1: 0, 2: 1}, '手机上网速度慢': {-1: 0, 4: 1}, '炉石传说': {-1: 0, 9: 1}, '打游戏延时大': {-1: 0, 2: 1}, '火山': {-1: 0, 8: 1}, '显示有信号上不了网': {-1: 0, 2: 1}, '今日头条': {-1: 0, 6: 1}, '办公室': {-1: 0, 2: 1}, '梦幻西游': {-1: 0, 4: 1}, '其他, 请注明.2': {-1: 0, 98: 1}, '客户星级标识': {'未评级': 0, '准星': 1, '一星': 2, '二星': 3, '三星': 4, '银卡': 5, '金卡': 6, '白金卡': 7, '钻石卡': 8}, '穿越火线': {-1: 0, 3: 1}, '全部都卡顿': {-1: 0, 99: 1}, '微信': {-1: 0}, '全部游戏都卡顿': {-1: 0, 99: 1}, '性别': {'男': 1, '女': -1, '性别不详': 0}, '农村': {-1: 0, 6: 1}, '搜狐视频': {-1: 0, 5: 1}, '京东': {-1: 0, 4: 1}, '百度': {-1: 0, 5: 1}, '其他, 请注明.1': {-1: 0, 98: 1}, '抖音': {-1: 0, 6: 1}, '商业街': {-1: 0, 4: 1}, '拼多多': {-1: 0, 8: 1}, '新浪微博': {-1: 0, 7: 1}, '其他, 请注明': {-1: 0, 98: 1}, '和平精英': {-1: 0}, '手机支付较慢': {-1: 0, 5: 1}, '看视频卡顿': {-1: 0}, '梦幻诛仙': {-1: 0, 6: 1}, '部落冲突': {-1: 0, 8: 1}, '腾讯视频': {-1: 0, 3: 1}, '上网过程中网络时断时续或时快时慢': {-1: 0, 3: 1}, '其他, 请注明.3': {-1: 0, 98: 1}, '地铁': {-1: 0, 5: 1}, '打开网页或APP图片慢': {-1: 0, 3: 1}, '快手': {-1: 0, 7: 1}, '芒果TV': {-1: 0, 4: 1}, '爱奇艺': {-1: 0}, '龙之谷': {-1: 0, 5: 1}, '高铁': {-1: 0, 7: 1}, '全部网页或APP都慢': {-1: 0, 99: 1}, '王者荣耀': {-1: 0, 2: 1}, '淘宝': {-1: 0, 3: 1}, '其他, 请注明.4': {-1: 0, 98: 1}, '下载速度慢': {-1: 0, 4: 1}, '优酷': {-1: 0, 2: 1}, '欢乐斗地主': {-1: 0, 7: 1}, '网络信号差/没有信号': {-1: 0}, '终端品牌': {'0': 0, '苹果': 1, '华为': 2, '小米科技': 3, '步步高': 4, '欧珀': 5, 'realme': 6, '三星': 7, '万普拉斯': 8, '黑鲨': 9, '锤子': 10, '摩托罗拉': 11, '中邮通信': 12, '万普': 13, '诺基亚': 14, '联通': 15, '中国移动': 16, '中兴': 17, '华硕': 18, '联想': 19, '魅族': 20, '奇酷': 21, 'TD': 22, '北京珠穆朗玛移动通信有限公司': 23, '飞利浦': 24, '捷开通讯科技': 25, '金立': 26, '酷比': 27, '欧博信': 28, '索尼爱立信': 29, '维图': 30, '甄十信息科技(上海)有限公司': 31}, '其他': {-1: 0, 98: 1}}

```

```

    公司': 31, '中国电信': 32, '天翼': 33, 'RealMe': 6}], inplace=True)
1181 dataFour
1182
1183 # In[93]:
1184
1185
1186 dataFour['出现问题场所或应用总'] = dataFour.loc[:, ~dataFour.columns.isin(['手机上网
整体满意度', '网络覆盖与信号强度', '手机上网速度', '手机上网稳定性', '是否5G网络客
户', '是否不限量套餐到达用户', '手机上网速度慢', '打游戏延时大', '显示有信号上不了
网', '上网质差次数', '当月MOU', '客户星级标识', '全部都卡顿', '全部游戏都卡顿',
'脱网次数', '性别', '套外流量费(元)', '微信质差次数', '百度', '套外流量(MB)', '手机
支付较慢', '看视频卡顿', '终端品牌', '上网过程中网络时断时续或时快时慢', '打开
网页或APP图片慢', '全部网页或APP都慢', '下载速度慢', '网络信号差/没有信号'])].
apply(lambda x1: x1.sum(), axis=1)
1187 dataFour['网络卡速度慢延时大上不了网总'] = dataFour.loc[:, ['手机上网速度慢', '打游戏
延时大', '显示有信号上不了网', '全部都卡顿', '全部游戏都卡顿', '手机支付较慢',
'看视频卡顿', '上网过程中网络时断时续或时快时慢', '打开网页或APP图片慢', '全部网页或
APP都慢', '下载速度慢', '网络信号差/没有信号']].apply(lambda x1: x1.sum(), axis
=1)
1188 dataFour['质差总'] = dataFour.loc[:, ['微信质差次数', '上网质差次数']].apply(lambda
x1: x1.sum(), axis=1)
1189 dataFour
1190
1191 # In[94]:
1192
1193
1194 dataFourStandardTransform = dataFour[['上网质差次数', '当月MOU', '客户星级标识',
'脱网次数', '性别', '套外流量费(元)', '微信质差次数', '套外流量(MB)', '终端品牌',
'出现问题场所或应用总', '网络卡速度慢延时大上不了网总', '质差总']]
1195 dataFourStandardTransformScaler = sp.StandardScaler()
1196 dataFourStandardTransformScaler = dataFourStandardTransformScaler.fit(
    dataFourStandardTransform)
1197 dataFourStandardTransform = dataFourStandardTransformScaler.transform(
    dataFourStandardTransform)
1198 dataFourStandardTransform = pd.DataFrame(dataFourStandardTransform)
1199 dataFourStandardTransform.columns = ['上网质差次数', '当月MOU', '客户星级标识',
'脱网
次数', '性别', '套外流量费(元)', '微信质差次数', '套外流量(MB)', '终端品牌',
'出现问题场所或应用总', '网络卡速度慢延时大上不了网总', '质差总']
1200 dataFourStandardTransform
1201
1202 # In[95]:
1203
1204
1205 dataFourLeave = dataFour.loc[:, ~dataFour.columns.isin(['上网质差次数', '当月MOU',
'
```

客户星级标识'，'脱网次数'，'性别'，'套外流量费(元)'，'微信质差次数'，'套外流量(MB)'，'终端品牌'，'出现问题场所或应用总'，'网络卡速度慢延时大上不了网总'，'质差总']])

1206 dataFourNewStandard = pd.concat([dataFourLeave, dataFourStandardTransform], axis=1)

1207 dataFourNewStandard

1208

1209 # In[96]:

1210

1211

1212 dataTwoNewStandard

1213

预测上网业务评分

1215 # 需要注意到在所有预测结果上加上1，由于之前将评分编码为[0,9]，这里需要再映射回[1,10]

1216

In[97]:

1218

1219

1220 Xpre = dataFourNewStandard

1221

手机上网整体满意度

1223

In[98]:

1225

1226

1227 FirstPre = FirstModel.predict(Xpre)

1228 FirstPre

1229

网络覆盖与信号强度

1231

In[99]:

1233

1234

1235 SecondPre = SecondModel.predict(Xpre)

1236 SecondPre

1237

手机上网速度

1239

In[100]:

1241

1242

1243 ThirdPre = ThirdModel.predict(Xpre)

1244 ThirdPre

1245

手机上网稳定性

```

1247
1248 # In[101]:
1249
1250
1251 FourthPre = FourthModel.predict(Xpre)
1252 FourthPre
1253
1254 # ## 模型效果分析
1255
1256 # ### 混淆矩阵热力图
1257
1258 # ##### 模型一
1259
1260 # In[102]:
1261
1262
1263 from yellowbrick.classifier import ConfusionMatrix
1264
1265 classes = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’]
1266 confusion_matrix = ConfusionMatrix(FirstModel, classes=classes, cmap=‘BuGn’)
1267 confusion_matrix.fit(XdataTwoFirst_train, ydataTwoFirst_train)
1268 confusion_matrix.score(XdataTwoFirst_test, ydataTwoFirst_test)
1269 plt.xticks(font=‘Times New Roman’)
1270 plt.yticks(font=‘Times New Roman’)
1271 confusion_matrix.show(outpath=‘figuresTwo\\[附件2]模型一混淆矩阵热力图.pdf’)
1272
1273 # ##### 模型二
1274
1275 # In[103]:
1276
1277
1278 from yellowbrick.classifier import ConfusionMatrix
1279
1280 classes = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’]
1281 confusion_matrix = ConfusionMatrix(SecondModel, classes=classes, cmap=‘BuGn’)
1282 confusion_matrix.fit(XdataTwoSecond_train, ydataTwoSecond_train)
1283 confusion_matrix.score(XdataTwoSecond_test, ydataTwoSecond_test)
1284 plt.xticks(font=‘Times New Roman’)
1285 plt.yticks(font=‘Times New Roman’)
1286 confusion_matrix.show(outpath=‘figuresTwo\\[附件2]模型二混淆矩阵热力图.pdf’)
1287
1288 # ##### 模型三
1289
1290 # In[104]:

```

```

1291
1292
1293     from yellowbrick.classifier import ConfusionMatrix
1294
1295     classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1296     confusion_matrix = ConfusionMatrix(ThirdModel, classes=classes, cmap='BuGn')
1297     confusion_matrix.fit(XdataTwoThird_train, ydataTwoThird_train)
1298     confusion_matrix.score(XdataTwoThird_test, ydataTwoThird_test)
1299     plt.xticks(font='Times New Roman')
1300     plt.yticks(font='Times New Roman')
1301     confusion_matrix.show(outpath='figuresTwo\\[附件2]模型三混淆矩阵热力图.pdf')
1302
1303 # ##### 模型四
1304
1305 # In[105]:
1306
1307
1308     from yellowbrick.classifier import ConfusionMatrix
1309
1310     classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1311     confusion_matrix = ConfusionMatrix(FourthModel, classes=classes, cmap='BuGn')
1312     confusion_matrix.fit(XdataTwoFourth_train, ydataTwoFourth_train)
1313     confusion_matrix.score(XdataTwoFourth_test, ydataTwoFourth_test)
1314     plt.xticks(font='Times New Roman')
1315     plt.yticks(font='Times New Roman')
1316     confusion_matrix.show(outpath='figuresTwo\\[附件2]模型四混淆矩阵热力图.pdf')
1317
1318 # ### 分类报告
1319
1320 # ##### 模型一
1321
1322 # In[106]:
1323
1324
1325     from yellowbrick.classifier import ClassificationReport
1326
1327     classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1328     visualizer = ClassificationReport(FirstModel, classes=classes, support=True, cmap='Blues')
1329     visualizer.fit(XdataTwoFirst_train, ydataTwoFirst_train)
1330     visualizer.score(XdataTwoFirst_test, ydataTwoFirst_test)
1331     plt.xticks(font='Times New Roman')
1332     plt.yticks(font='Times New Roman')
1333     visualizer.show(outpath='figuresTwo\\[附件2]模型一分类报告.pdf')

```

```

1334
1335 # ##### 模型二
1336
1337 # In[107]:
1338
1339
1340 from yellowbrick.classifier import ClassificationReport
1341
1342 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1343 visualizer = ClassificationReport(SecondModel, classes=classes, support=True, cmap=
    'Blues')
1344 visualizer.fit(XdataTwoSecond_train, ydataTwoSecond_train)
1345 visualizer.score(XdataTwoSecond_test, ydataTwoSecond_test)
1346 plt.xticks(font='Times New Roman')
1347 plt.yticks(font='Times New Roman')
1348 visualizer.show(outpath='figuresTwo\\[附件2]模型二分类报告.pdf')
1349
1350 # ##### 模型三
1351
1352 # In[108]:
1353
1354
1355 from yellowbrick.classifier import ClassificationReport
1356
1357 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1358 visualizer = ClassificationReport(ThirdModel, classes=classes, support=True, cmap='
    Blues')
1359 visualizer.fit(XdataTwoThird_train, ydataTwoThird_train)
1360 visualizer.score(XdataTwoThird_test, ydataTwoThird_test)
1361 plt.xticks(font='Times New Roman')
1362 plt.yticks(font='Times New Roman')
1363 visualizer.show(outpath='figuresTwo\\[附件2]模型三分类报告.pdf')
1364
1365 # ##### 模型四
1366
1367 # In[109]:
1368
1369
1370 from yellowbrick.classifier import ClassificationReport
1371
1372 classes = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
1373 visualizer = ClassificationReport(FourthModel, classes=classes, support=True, cmap=
    'Blues')
1374 visualizer.fit(XdataTwoFourth_train, ydataTwoFourth_train)

```

```

1375     visualizer.score(XdataTwoFourth_test, ydataTwoFourth_test)
1376     plt.xticks(font='Times New Roman')
1377     plt.yticks(font='Times New Roman')
1378     visualizer.show(outpath='figuresTwo\\[附件2] 模型四分类报告.pdf')
1379
1380     # ##### ROC AUC曲线
1381
1382     # ##### 模型一
1383
1384     # In[110]:
1385
1386
1387     from yellowbrick.classifier import ROCAUC
1388
1389     visualizer = ROCAUC(FirstModel)
1390     visualizer.fit(XdataTwoFirst_train, ydataTwoFirst_train)
1391     visualizer.score(XdataTwoFirst_test, ydataTwoFirst_test)
1392     plt.xticks(font='Times New Roman')
1393     plt.yticks(font='Times New Roman')
1394     visualizer.show(outpath='figuresTwo\\[附件2] 模型一ROCAUC.pdf')
1395
1396     # ##### 模型二
1397
1398     # In[111]:
1399
1400
1401     from yellowbrick.classifier import ROCAUC
1402
1403     visualizer = ROCAUC(SecondModel)
1404     visualizer.fit(XdataTwoSecond_train, ydataTwoSecond_train)
1405     visualizer.score(XdataTwoSecond_test, ydataTwoSecond_test)
1406     plt.xticks(font='Times New Roman')
1407     plt.yticks(font='Times New Roman')
1408     visualizer.show(outpath='figuresTwo\\[附件2] 模型二ROCAUC.pdf')
1409
1410     # ##### 模型三
1411
1412     # In[112]:
1413
1414
1415     from yellowbrick.classifier import ROCAUC
1416
1417     visualizer = ROCAUC(ThirdModel)
1418     visualizer.fit(XdataTwoThird_train, ydataTwoThird_train)

```

```

1419 visualizer.score(XdataTwoThird_test, ydataTwoThird_test)
1420 plt.xticks(font='Times New Roman')
1421 plt.yticks(font='Times New Roman')
1422 visualizer.show(outpath='figuresTwo\\[附件2] 模型三ROCAUC.pdf')
1423
1424 # ##### 模型四
1425
1426 # In[113]:
1427
1428
1429 from yellowbrick.classifier import ROCAUC
1430
1431 visualizer = ROCAUC(FourthModel)
1432 visualizer.fit(XdataTwoFourth_train, ydataTwoFourth_train)
1433 visualizer.score(XdataTwoFourth_test, ydataTwoFourth_test)
1434 plt.xticks(font='Times New Roman')
1435 plt.yticks(font='Times New Roman')
1436 visualizer.show(outpath='figuresTwo\\[附件2] 模型四ROCAUC.pdf')
1437
1438 # ### 平均绝对误差、均方误差
1439
1440 # ##### 模型一
1441
1442 # In[114]:
1443
1444
1445 print(f'模型一平均绝对误差: ,'
1446 f'{mean_absolute_error(ydataTwoFirst_test, FirstModel.predict(
1447 XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
1448 ,
1449 f'模型一均方误差: ,'
1450 f'{mean_squared_error(ydataTwoFirst_test, FirstModel.predict(
1451 XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")})')
1452 print(f'模型一中RF平均绝对误差: ,'
1453 f'{mean_absolute_error(ydataTwoFirst_test, RandomForestFirst.predict(
1454 XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
1455 ,
1456 f'模型一中RF均方误差: ,'
1457 f'{mean_squared_error(ydataTwoFirst_test, RandomForestFirst.predict(
1458 XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")})')
1459 print(f'模型一中XGBoost平均绝对误差: ,'
1460 f'{mean_absolute_error(ydataTwoFirst_test, XGBFirst.predict(XdataTwoFirst_test
1461 ), sample_weight=None, multioutput="uniform_average")}\n'
1462 f'模型一中XGBoost均方误差: ,'

```

```

1456         f'{mean_squared_error(ydataTwoFirst_test, XGBFirst.predict(XdataTwoFirst_test)
1457             , sample_weight=None, multioutput="uniform_average")}\n')
1458 print(f'模型一中KNN平均绝对误差: ,
1459       f'{mean_absolute_error(ydataTwoFirst_test, KNNFirst.predict(XdataTwoFirst_test)
1460           , sample_weight=None, multioutput="uniform_average")}\n'
1461       f'模型一中KNN均方误差: ,
1462       f'{mean_squared_error(ydataTwoFirst_test, KNNFirst.predict(XdataTwoFirst_test)
1463           , sample_weight=None, multioutput="uniform_average")}\n')
1464 print(f'模型一中SVM平均绝对误差: ,
1465       f'{mean_absolute_error(ydataTwoFirst_test, SVMFirst.predict(XdataTwoFirst_test)
1466           , sample_weight=None, multioutput="uniform_average")}\n'
1467       f'模型一中SVM均方误差: ,
1468       f'{mean_squared_error(ydataTwoFirst_test, SVMFirst.predict(XdataTwoFirst_test)
1469           , sample_weight=None, multioutput="uniform_average")}\n')
1470 print(f'模型一中LightGBM平均绝对误差: ,
1471       f'{mean_absolute_error(ydataTwoFirst_test, LightgbmFirst.predict(
1472           XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
1473       ,
1474       f'模型一中LightGBM均方误差: ,
1475       f'{mean_squared_error(ydataTwoFirst_test, LightgbmFirst.predict(
1476           XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n')
1477 print(f'模型一中LR平均绝对误差: ,
1478       f'{mean_absolute_error(ydataTwoFirst_test, LogisticRegressionFirst.predict(
1479           XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n'
1480       ,
1481       f'模型一中LR均方误差: ,
1482       f'{mean_squared_error(ydataTwoFirst_test, LogisticRegressionFirst.predict(
1483           XdataTwoFirst_test), sample_weight=None, multioutput="uniform_average")}\n')

1484 # ##### 模型二
1485
1486 # In[115]:
1487
1488
1489 print(f'模型二平均绝对误差: ,
1490       f'{mean_absolute_error(ydataTwoSecond_test, SecondModel.predict(
1491           XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\n'
1492       ,
1493       f'模型二均方误差: ,
1494       f'{mean_squared_error(ydataTwoSecond_test, SecondModel.predict(
1495           XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\n'
1496       )
1497 print(f'模型二中RF平均绝对误差: ,
1498       f'{mean_absolute_error(ydataTwoSecond_test, RandomForestSecond.predict(

```

```

        XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
    n'
f'模型二中RF均方误差: ,
f'{mean_squared_error(ydataTwoSecond_test, RandomForestSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型二中XGBoost平均绝对误差: ,
f'{mean_absolute_error(ydataTwoSecond_test, XGBSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
    n'
f'模型二中XGBoost均方误差: ,
f'{mean_squared_error(ydataTwoSecond_test, XGBSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型二中KNN平均绝对误差: ,
f'{mean_absolute_error(ydataTwoSecond_test, KNNSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
    n'
f'模型二中KNN均方误差: ,
f'{mean_squared_error(ydataTwoSecond_test, KNNSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型二中SVM平均绝对误差: ,
f'{mean_absolute_error(ydataTwoSecond_test, SVMSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
    n'
f'模型二中SVM均方误差: ,
f'{mean_squared_error(ydataTwoSecond_test, SVMSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型二中LightGBM平均绝对误差: ,
f'{mean_absolute_error(ydataTwoSecond_test, LightgbmSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
    n'
f'模型二中LightGBM均方误差: ,
f'{mean_squared_error(ydataTwoSecond_test, LightgbmSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\',
)
print(f'模型二中LR平均绝对误差: ,
f'{mean_absolute_error(ydataTwoSecond_test, LogisticRegressionSecond.predict(
    XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\\
    n'
f'模型二中LR均方误差: ,
f'{mean_squared_error(ydataTwoSecond_test, LogisticRegressionSecond.predict(

```

```

        XdataTwoSecond_test), sample_weight=None, multioutput="uniform_average")}\n
    )
1507
1508 # ##### 模型三
1509
1510 # In[116]:
1511
1512
1513 print(f'模型三平均绝对误差: ,\n
1514     f'{mean_absolute_error(ydataTwoThird_test, ThirdModel.predict(\n
1515         XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n
1516         ,\n
1517         f'模型三均方误差: ,\n
1518         f'{mean_squared_error(ydataTwoThird_test, ThirdModel.predict(\n
1519             XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")})')
1520
1521 print(f'模型三中RF平均绝对误差: ,\n
1522     f'{mean_absolute_error(ydataTwoThird_test, RandomForestThird.predict(\n
1523         XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n
1524         ,\n
1525         f'模型三中RF均方误差: ,\n
1526         f'{mean_squared_error(ydataTwoThird_test, RandomForestThird.predict(\n
1527             XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")})')
1528
1529 print(f'模型三中XGBoost平均绝对误差: ,\n
1530     f'{mean_absolute_error(ydataTwoThird_test, XGBThird.predict(XdataTwoThird_test\n
1531         ), sample_weight=None, multioutput="uniform_average")}\n'
1532         ,\n
1533         f'模型三中XGBoost均方误差: ,\n
1534         f'{mean_squared_error(ydataTwoThird_test, XGBThird.predict(XdataTwoThird_test)\n
1535             , sample_weight=None, multioutput="uniform_average")})')
1536
1537 print(f'模型三中KNN平均绝对误差: ,\n
1538     f'{mean_absolute_error(ydataTwoThird_test, KNNThird.predict(XdataTwoThird_test\n
1539         ), sample_weight=None, multioutput="uniform_average")}\n'
1540         ,\n
1541         f'模型三中KNN均方误差: ,\n
1542         f'{mean_squared_error(ydataTwoThird_test, KNNThird.predict(XdataTwoThird_test)\n
1543             , sample_weight=None, multioutput="uniform_average")})')
1544
1545 print(f'模型三中SVM平均绝对误差: ,\n
1546     f'{mean_absolute_error(ydataTwoThird_test, SVMThird.predict(XdataTwoThird_test\n
1547         ), sample_weight=None, multioutput="uniform_average")}\n'
1548         ,\n
1549         f'模型三中SVM均方误差: ,\n
1550         f'{mean_squared_error(ydataTwoThird_test, SVMThird.predict(XdataTwoThird_test)\n
1551             , sample_weight=None, multioutput="uniform_average")})')
1552
1553 print(f'模型三中LightGBM平均绝对误差: ,\n
1554     f'{mean_absolute_error(ydataTwoThird_test, LightgbmThird.predict(\n
1555         XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n
1556         ,

```

```

1535     f'模型三中LightGBM均方误差: ,  

1536     f'{mean_squared_error(ydataTwoThird_test, LightgbmThird.predict(  

1537         XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")})'  

1537 print(f'模型三中LR平均绝对误差: ,  

1538     f'{mean_absolute_error(ydataTwoThird_test, LogisticRegressionThird.predict(  

1539         XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")}\n  

1539 ,  

1539 f'模型三中LR均方误差: ,  

1540     f'{mean_squared_error(ydataTwoThird_test, LogisticRegressionThird.predict(  

1541         XdataTwoThird_test), sample_weight=None, multioutput="uniform_average")})'  

1541  

1542 # ##### 模型四  

1543  

1544 # In[117]:  

1545  

1546  

1547 print(f'模型四平均绝对误差: ,  

1548     f'{mean_absolute_error(ydataTwoFourth_test, FourthModel.predict(  

1549         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n  

1549 ,  

1549 f'模型四均方误差: ,  

1550     f'{mean_squared_error(ydataTwoFourth_test, FourthModel.predict(  

1551         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',  

1551 )  

1551 print(f'模型四中RF平均绝对误差: ,  

1552     f'{mean_absolute_error(ydataTwoFourth_test, RandomForestFourth.predict(  

1553         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n  

1553 ,  

1553 f'模型四中RF均方误差: ,  

1554     f'{mean_squared_error(ydataTwoFourth_test, RandomForestFourth.predict(  

1555         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',  

1555 )  

1555 print(f'模型四中XGBoost平均绝对误差: ,  

1556     f'{mean_absolute_error(ydataTwoFourth_test, XGBFourth.predict(  

1557         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n  

1557 ,  

1557 f'模型四中XGBoost均方误差: ,  

1558     f'{mean_squared_error(ydataTwoFourth_test, XGBFourth.predict(  

1559         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',  

1559 )  

1559 print(f'模型四中KNN平均绝对误差: ,  

1560     f'{mean_absolute_error(ydataTwoFourth_test, KNNFourth.predict(  

1561         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n  

1561 ,
```

```

1561     f'模型四中KNN均方误差: ' ,
1562     f'{mean_squared_error(ydataTwoFourth_test, KNNFourth.predict(
1563         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',
1564     )
1565
1566 print(f'模型四中SVM平均绝对误差: ' ,
1567       f'{mean_absolute_error(ydataTwoFourth_test, SVMFourth.predict(
1568           XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n',
1569       n'
1570     f'模型四中SVM均方误差: ' ,
1571     f'{mean_squared_error(ydataTwoFourth_test, SVMFourth.predict(
1572         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',
1573     )
1574
1575 print(f'模型四中LightGBM平均绝对误差: ' ,
1576       f'{mean_absolute_error(ydataTwoFourth_test, LightgbmFourth.predict(
1577           XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n',
1578       n'
1579     f'模型四中LightGBM均方误差: ' ,
1580     f'{mean_squared_error(ydataTwoFourth_test, LightgbmFourth.predict(
1581         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',
1582     )
1583
1584 print(f'模型四中LR平均绝对误差: ' ,
1585       f'{mean_absolute_error(ydataTwoFourth_test, LogisticRegressionFourth.predict(
1586           XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}\n',
1587       n'
1588     f'模型四中LR均方误差: ' ,
1589     f'{mean_squared_error(ydataTwoFourth_test, LogisticRegressionFourth.predict(
1590         XdataTwoFourth_test), sample_weight=None, multioutput="uniform_average")}',
1591     )
1592
1593 # ## 高频词汇云图
1594
1595 # In[118]:
1596
1597
1598
1599
1600
1601 import jieba
1602 import wordcloud
1603 from matplotlib.image import imread
1604
1605 jieba.setLogLevel(jieba.logging.INFO)
1606 report = open('上网业务词云.txt', 'r', encoding='utf-8').read()
1607 words = jieba.lcut(report)
1608 txt = []
1609 for word in words:
1610     if len(word) == 1:

```

```
1591         continue
1592     else:
1593         txt.append(word)
1594     a = ' '.join(txt)
1595     bg = imread("bg.jpg")
1596     w = wordcloud.WordCloud(background_color="white", font_path="msyh.ttc", mask=bg)
1597     w.generate(a)
1598     w.to_file("figuresTwo\\wordcloudS.png")
```

D.3 原始数据用户评分南丁格尔玫瑰图示

```
1 import pandas as pd
2 from pyecharts.charts import Pie
3 from pyecharts import options as opts
4 from snapshot_phantomjs import snapshot
5 from pyecharts.render import make_snapshot
6
7
8 def draw_pie(grade, num, n):
9     color_series = ['#9ECB3C', '#6DBC49', '#37B44E', '#3DBA78', '#14ADCF',
10                  '#209AC9', '#1E91CA', '#2C6BA0', '#2B55A1', '#2D3D8E']
11
12     df = pd.DataFrame({'grade': grade, 'num': num})
13     df.sort_values(by='num', ascending=False, inplace=True)
14     v = df['grade'].values.tolist()
15     d = df['num'].values.tolist()
16
17     pie1 = Pie(init_opts=opts.InitOpts(width='1000px', height='1000px'))
18     pie1.set_colors(color_series)
19     pie1.add("", [list(z) for z in zip(v, d)],
20               radius=["15%", "70%"],
21               center=["50%", "60%"],
22               rosetype="area")
23
24     pie1.set_global_opts(legend_opts=opts.LegendOpts(is_show=False), toolbox_opts=
25                           opts.ToolboxOpts())
26
27     pie1.set_series_opts(label_opts=opts.LabelOpts(is_show=True, position="inside",
28                           font_size=12, formatter="{b}分:{c}人", font_style="italic", font_weight="bold",
29                           font_family="Microsoft YaHei"),)
30
31     make_snapshot(snapshot, pie1.render(), f'figuresNightingaleRoseDiagramF\\{n}.png'
32                   , is_remove_html=True)
33
34
35     draw_pie([10, 9, 8, 7, 1, 6, 5, 3, 4, 2], [3157, 764, 567, 214, 209, 182, 172, 71,
36              55, 42], 1)
37
38     draw_pie([10, 9, 8, 7, 6, 1, 5, 3, 4, 2], [2701, 786, 658, 327, 271, 237, 211, 90,
39              90, 62], 2)
40
41     draw_pie([10, 9, 8, 7, 6, 1, 5, 4, 3, 2], [2981, 805, 643, 268, 207, 184, 161, 85,
42              59, 40], 3)
43
44     draw_pie([10, 9, 8, 7, 1, 6, 5, 4, 3, 2], [2800, 786, 654, 309, 230, 228, 201, 93,
45              83, 49], 4)
46
47
48     draw_pie([10, 8, 9, 7, 1, 6, 5, 3, 4, 2], [2890, 991, 851, 533, 455, 435, 409, 209,
49              160, 87], 5)
50
51     draw_pie([10, 8, 9, 7, 6, 5, 1, 3, 4, 2], [2587, 1056, 824, 638, 508, 435, 429,
52              228, 198, 117], 6)
```

```
34     draw_pie([10, 8, 9, 7, 6, 5, 1, 3, 4, 2], [2556, 1066, 841, 675, 475, 454, 415,
35         235, 199, 104], 7)
      draw_pie([10, 8, 9, 7, 6, 1, 5, 3, 4, 2], [2528, 1003, 846, 637, 516, 462, 462,
          254, 191, 121], 8)
```

D.4 预测数据用户评分南丁格尔玫瑰图示

```
1 import pandas as pd
2 from pyecharts.charts import Pie
3 from pyecharts import options as opts
4 from snapshot_phantomjs import snapshot
5 from pyecharts.render import make_snapshot
6
7
8 def draw_pie(grade, num, n):
9     color_series = ['#14ADCF', '#209AC9', '#1E91CA', '#2C6BA0', '#2B55A1',
10                  '#2D3D8E', '#44388E', '#6A368B', '#7D3990', '#A63F98']
11
12     df = pd.DataFrame({'grade': grade, 'num': num})
13     df.sort_values(by='num', ascending=False, inplace=True)
14     v = df['grade'].values.tolist()
15     d = df['num'].values.tolist()
16
17     pie1 = Pie(init_opts=opts.InitOpts(width='1000px', height='1000px'))
18     pie1.set_colors(color_series)
19     pie1.add("", [list(z) for z in zip(v, d)],
20               radius=["15%", "70%"],
21               center=["50%", "60%"],
22               rosetype="area")
23
24     pie1.set_global_opts(legend_opts=opts.LegendOpts(is_show=False), toolbox_opts=
25                           opts.ToolboxOpts())
26
27     pie1.set_series_opts(label_opts=opts.LabelOpts(is_show=True, position="inside",
28                           font_size=12, formatter="{b}分:{c}人", font_style="italic", font_weight="bold",
29                           font_family="Microsoft YaHei"),)
30
31     make_snapshot(snapshot, pie1.render(), f'figuresNightingaleRoseDiagramP\\{n}.png'
32                   , is_remove_html=True)
33
34
35     draw_pie([10, 8, 1, 9, 5, 6, 7, 4, 2, 3], [2234, 145, 78, 61, 20, 15, 15, 12, 11,
36                 8], 1)
37
38     draw_pie([10, 8, 9, 1, 6, 7, 5, 3, 4, 2], [2012, 168, 155, 79, 60, 46, 30, 22, 19,
39                 8], 2)
40
41     draw_pie([10, 9, 8, 1, 7, 6, 4, 5, 2, 3], [2135, 169, 113, 62, 49, 27, 19, 11, 7,
42                 7], 3)
43
44     draw_pie([10, 8, 9, 7, 1, 6, 5, 4, 3, 2], [2045, 243, 67, 56, 55, 46, 33, 28, 15,
45                 11], 4)
46
47     draw_pie([10, 8, 1, 5, 7, 6], [1099, 319, 128, 43, 15, 6], 5)
48
49     draw_pie([10, 8, 5, 1, 7, 6, 3], [927, 368, 137, 92, 71, 10, 5], 6)
50
51     draw_pie([10, 8, 1, 7, 5, 3, 6], [948, 404, 93, 80, 57, 19, 9], 7)
52
53     draw_pie([10, 8, 1, 5, 6, 7], [954, 365, 165, 65, 33, 28], 8)
```