队伍编号	MCB2200122
赛道	B

基于 XGB、逻辑回归、支持向量机、随机森林、GBDT、ABC-Boost 的 Stacking 语音上网满意度预测模型

摘　　要

随着生活水平的提高，用户对通话、上网的质量、服务有更大的期待与需求。用户对网络运营商的评价是运营商优化服务、提升质量的重要参考。本文通对中国移动用户语音和上网满意度数据进行 EDA、特征工程、建立模型、模型调参、模型融合等步骤，建立了基于 XGB、逻辑回归、支持向量机、随机森林、GBDT、ABC-Boost 的 Stacking 语音上网满意度预测模型。

模型需要预测的指标共有 8 个，本文将 8 个指标分开，分别预测，考虑到评价具有主观随意性，本文使用 **RMSE** 作为模型的评价指标，**准确率**作为辅助的评价指标。本文针对问题总体上分为特征选择和模型建立和预测两个部分。

针对问题一，**第一步**，对**数据**进行**预处理**。对数据进行**分箱**、对类别属性进行 **Label Encoder**、对缺失值进行填充、删除无用属性。**第二步**，进行**特征重要性分析**。求出语音和上网满意度数据的 **Kendall 相关性系数**；使用随机森林求出属性的特征重要性；在随机森林基础上求出 **Permutation importance**。通过研究发现：对于语音通话，最影响用户体验的是通话遇到的问题，如**手机没有信号**、**无法拨通**、**突然中断**、**有杂音**、**听不清**等问题。对于手机上网，最影响用户体验的是信号和网速问题，如**网络信号差**、**没有信号**、**网速慢**、**网速不稳定**、**热门 APP 卡顿**等问题。

针对问题二，**第一步**，**数据预处理**及**数据集划分**。根据问题一的结果，剔除连续型变量以及测试集中没有的指标，对类别数据进行 **One-hot 编码**；将数据集划分为训练集和测试集，比例为 8:2。**第二步**，模型筛选。选取 XGB、逻辑回归、支持向量机、随机森林、K 近邻算法、朴素贝叶斯、感知机、SGD、决策树、GBDT、ABC-Boost、神经网络共 12 个模型在训练集上训练，得出模型在测试集上的**准确率**和 **RMSE**。**第三步**，**模型调参**。综合准确率和 RMSE，选取 **XGB**、**逻辑回归**、**SVC**、**随机森林**、**GBDT**、**ABC-Boost** 共 6 个模型使用**网格搜索**进行超参数调优。**第四步**，**模型融合**。使用 **Stacking** 方法对 6 个模型进行特征融合，8 个标签在测试集上的准确率在 **40.17%-59.24%** 之间，RMSE 值在之间 **2.28-2.98** 之间。**第五步**，模型预测。使用训练好的模型，对 8 个目标属性进行预测，并将预测的结果导出到文件中。

最后我们进行了总结，并对模型进行了评价与推广。

关键词：**Permutation importance**；**随机森林**；**Stacking 模型**；**RMSE**

目录

一、 问题重述

1.1 问题背景

通信技术飞速发展，人们依赖于通信网络带来的遍历，网络运营商也越来越重视用户的使用体验，从而进一步提高服务水平与质量。

中国移动通信集团北京公司统计了客户对语音通话和上网的满意度打分，并统计了影响用户上网和通话的因素。通过对提供的数据进行分析建模，能够分析影响用户满意度的各种因素，从而为决策提供依据，提供更优质的服务。

1.2 问题分析

附件一和附件二是用户对移动语音通话和上网满意度的评分和数据，附件三和附件四是测试数据，不包含满意度评分结果，语音通话满意度需要预测的指标有四个，分别是语音通话满意度、网络覆盖与信号强度、语音通话清晰度、语音通话稳定性。上网满意度需要预测的指标有四个，分别是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。

针对第一问，需要分析影响客户语音业务和上网业务满意度的主要因素，可以首先查看标签对变量的相关性系数，从而衡量主要的因素；树模型在建模过程中能够得出变量的特征重要性，因此可以考虑使用决策树，随机森林等对数据进行建模，从而得出特征重要性。在随机森林的基础上建立 Permutation 重要性评价指标，能够更精确的获取变量的特征重要性。

针对第二问, 可以在第一问的基础上进一步进行数据处理，然后从主要的机器学习算法中筛选出模型效果较优的模型，对筛选出来的模型调节参数，然后使用 Stacking 对调节参数后的模型进行特征融合，最后使用 Stacking 模型对数据进行预测。

1.3 研究思路

本文整体的研究思路如图1.

首先导入包后对读取到的数据进行 EDA，为方便后续建模和计算，首先对数据进行处理，而数据包中属性量庞大，不方便后续建模，因此首先删除与模型准确性几乎无关的属性，并对剩余属性值中有缺失值的进行用 0 填充剩余有缺失值属性的缺失值，但此时数据量仍然较为庞大，因此将经过处理的数据进行分箱处理，实现数据的离散化。由于部分数据不方便归类，使用 LabelEncoder 对不连续的数字或者文本进行编号，至此数据处理完毕，可以进行下一步处理。

为得出影响客户语言业务和上网业务满意度的主要因素需进行特征重要性分析。首先求出语音和上网满意度数据的 Kendall 相关性系数，进行相关属性筛选；使用随机森

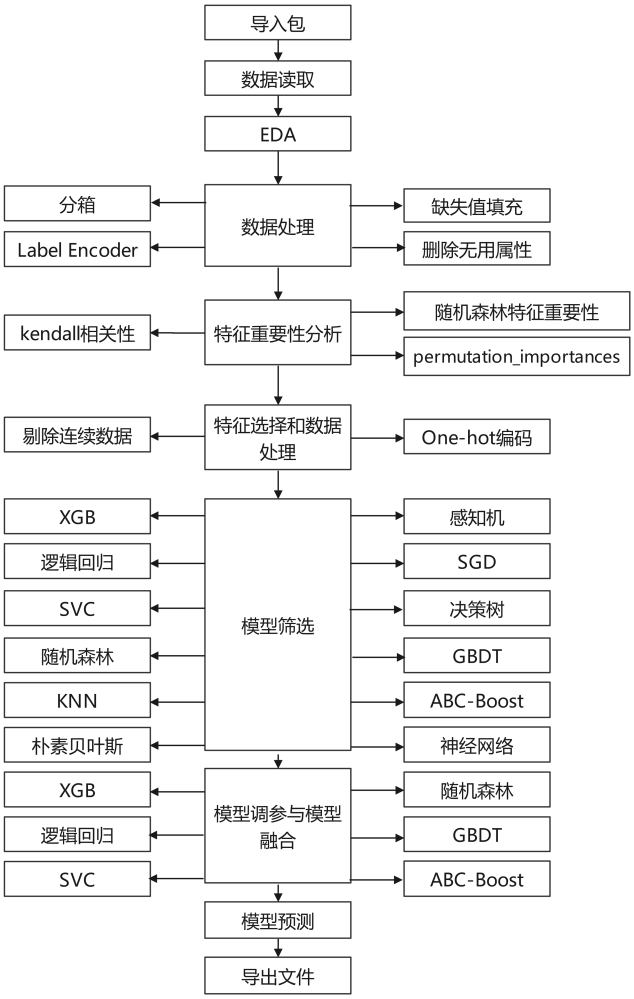


图 1　论文研究思路

林求出属性的特征重要性；在随机森林基础上求出 Permutation importance，进行变量筛选。至此问题一得到解决。

针对问题二，在问题一的基础上进行特征选择和处理，根据问题一的结果，首先将连续型变量以及测试集中没有的指标剔除，方便后续处理，对类别数据进行 One-hot 编码以此降低模型预测的误差；将数据集按 8：2 划分为训练集和测试集，至此数据处理完毕。

模型筛选。选取 XGB、逻辑回归、支持向量机、随机森林、K 近邻算法、朴素贝叶斯、感知机、SGD、决策树、GBDT、ABC-Boost、神经网络共 12 个模型在训练集上训练，得出模型在测试集上的准确率和 RMSE。

根据准确率和 RMSE 选取模型调参。综合准确率和 RMSE，最终选择出 XGB、逻辑回归、SVC、随机森林、GBDT、ABC-Boost 共 6 个模型，并用这六个模型使用网格搜索进行超参数调优。

模型融合与模型预测。使用 Stacking 方法对 6 个模型进行特征融合，使用融合后好的模型，对 8 个目标属性进行预测，并将预测的结果导出到文件中。

二、 特征选择

2.1 数据预处理

2.1.1 分箱

在语音通话中通过对数据进行分组统计，发现 [' 脱网次数', 'mos 质差次数', ' 未接通掉话次数', ' 重定向次数', ' 重定向驻留时长', ' 家宽投诉', ' 资费投诉', 'ARPU（家庭宽带）', ' 套外流量（MB）', ' 套外流量费（元）', ' 外省语音占比', ' 省际漫游-时长（分钟）', ' 前 3 月 ARPU', ' 外省流量占比', 'GPRS-国内漫游-流量（KB）', ' 当月欠费金额', ' 前第 3 个月欠费金额'] 这些特征的分布都是偏态分布，因此对这些特征进行分箱处理，分箱的方法使用的是

$$x = \begin{cases} 0, x <= 0 \\ 1, x > 0. \end{cases}$$

这样分箱的原因是，这些指标中含 0 的特征较多，分箱之后可以将 0 和非 0 的特征分开，从而指标的含义也发生了变化，代表有没有发生这个事件，这样分箱之后可以更好的进行特征选择。以' 脱网次数' 为例，等于 1 的数据代表有脱网事件，等于 0 的数据代表没有脱网事件。

按照同样的方法，对上网业务满意度的数据进行分箱处理，处理的变量及详细过程参照附录一。

2.1.2 Label Encoding

Label Encoding 是一种将类别特征转换为数值特征的方法，将类别特征的每个类别映射为一个整数，从而使得类别特征可以被模型使用。在本文中，对于类别特征，使用 Label Encoding 的方法进行处理，在语音满意度的数据中，进行 Label Encoding 的属性包括 ['4or5G 用户', ' 语音方式', ' 是否关怀用户', ' 是否去过营业厅', ' 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', ' 是否实名登记用户', ' 客户星级标识'] 以'4or5G 用户' 为例，将'4G' 映射为 0，将'5G' 映射为 1，这样就将类别特征转换为数值特征。

使用同样的方法，对上网业务满意度的数据进行 Label Encoding 处理，处理的变量及详细过程参照附录一。

2.1.3 缺失值和无用特征的处理

在语音通话中数据中，对于缺失值，使用 0 填充的方法进行处理。在语音通话中数据中，[' 用户 id',' 用户描述',' 终端品牌类型'] 是一些对建模无用的字符串类型数据，因此将这些特征删除。

使用同样的方法，对上网业务满意度的数据进行缺失值和无用特征的处理，处理的变量及详细过程参照附录一。

2.2 特征重要性分析

2.2.1 Kendall 相关性

相关性包括三种，分别是皮尔逊相关系数，斯皮尔曼相关系数和 Kendall 相关系数。在本文中，使用 Kendall 相关系数来衡量标签与变量的相关性，Kendall 相关系数的计算公式如下：

$$\tau = \frac{C - D}{\frac{n(n-1)}{2}}$$

其中，C 表示标签与变量的升序对数，D 表示标签与变量的降序对数，n 表示变量的取值个数。Kendall 相关系数的取值范围为 $[-1, 1]$，当 Kendall 相关系数接近 1 时，表示标签与变量的相关性较强，当 Kendall 相关系数接近-1 时，表示标签与变量的相关性较弱。

经过数据预处理之后，考虑到数据中的属性大部分为类别属性，因此使用 Kennall 相关系数来衡量标签与变量的相关性。

语音通话数据的 Kendall 相关系数部分计算结果如表1所示。完整的计算结果参照附录一。

	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	-0.444549	-0.486423	-0.459798	-0.499105
居民小区	-0.369733	-0.409377	-0.386846	-0.414010
办公室	-0.336016	-0.339024	-0.332816	-0.349356
高校	-0.128981	-0.128424	-0.125051	-0.139105
商业街	-0.209749	-0.208731	-0.203251	-0.205814
地铁	-0.242184	-0.270575	-0.254850	-0.265789
农村	-0.129456	-0.158894	-0.157536	-0.161360
高铁	-0.183300	-0.193037	-0.183382	-0.195884
其他，请注明	-0.108743	-0.113913	-0.096461	-0.133734
手机没有信号	-0.342434	-0.378955	-0.346209	-0.361691

表 1　语音通话的部分 Kendall 相关系数

在语音通话数据中，从计算结果中可以看出用户评分与用户的主观评价相关性最强，而与客观因素的相关性很弱，说明影响用户评分的是主观因素，客观因素对用户的评分影响较小。是否遇到过网络问题与语音通话稳定性的相关性最高，达到了 -0.499,

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、通话过程中一方听不见与评分的相关性均为 0.3-0.4，这些通话过程中出现的问题对用户的评分影响较大，影响用户的体验。

上网满意度数据的 Kendall 相关系数部分计算结果如表2所示。完整的计算结果参照附录一。

	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	-0.349356	-0.364682	-0.358926	-0.369619
办公室	-0.305906	-0.308342	-0.307713	-0.313720
高校	-0.143786	-0.147539	-0.150282	-0.154662
商业街	-0.211331	-0.211853	-0.215812	-0.213845
地铁	-0.287557	-0.296693	-0.285082	-0.303023
农村	-0.170501	-0.169520	-0.186651	-0.181086
高铁	-0.202367	-0.204478	-0.206837	-0.213856
其他，请注明	-0.102724	-0.095524	-0.097496	-0.104531
网络信号差/没有信号	-0.394298	-0.429587	-0.403823	-0.414503
显示有信号上不了网	-0.334929	-0.345201	-0.333677	-0.358332

表 2　上网满意度数据的部分 Kendall 相关系数

同样地，在上网数据中，从计算结果中可以得出以下结论：

- 用户评分与用户的主观评价相关性最强，而与客观因素的相关性很弱
- 在居民小区，办公室，高校，商业街，地铁，农村，高铁，手机没有信号等地区的上网满意度与用户评分的相关性均为 0.15-0.4，这些地区的上网满意度与用户评分的相关性较强。因此重点地区的信号质量会影响用户的上网满意度。
- 网络信号差/没有信号，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，手机上网速度慢，上网过程中出现网页打不开，图片加载不出来，视频卡顿等问题相关性平均在 3.5，对用户的评分影响较大，影响用户的体验。
- 在热门 APP 如抖音、快手、QQ、微信等以及热门游戏如王者荣耀、刺激战场等上网满意度与用户评分的相关性为 0.1-0.2，这些 APP 和游戏的上网满意度与用户评分的相关性较强。因此重点 APP 和游戏的信号质量会影响用户的上网满意度。

2.2.2 随机森林特征重要性

为了进一步分析语音通话和上网满意度数据中的特征对用户评分的影响，使用随机森林算法计算特征重要性，绘制语音特征重要性图如图2。

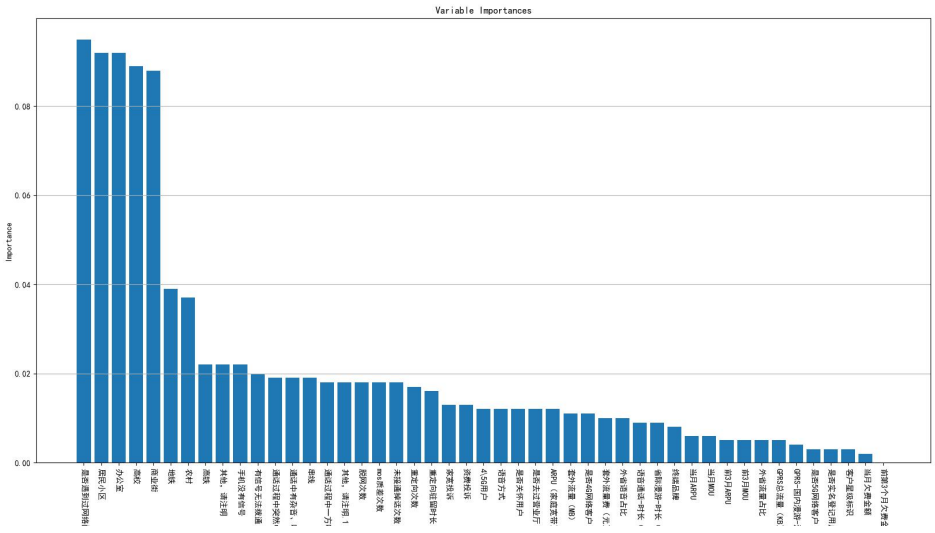


图 2　随机森林语音特征重要性图

由图可得，特征重要性主要分三个梯度，并且三个梯度断层明显，重要性由高到低依次降低。第一梯度为是否遇到过网络问题、居民小区、办公室、高校、商业街这五个因素。是否遇到过网络问题这一因素对客户打分影响程度最大，紧接着居民小区、办公室、高校、商业街对客户打分影响程度依次降低，但是否遇到过网络问题、居民小区、办公室、高校、商业街这五个因素的重要性相较于其他因素最为突出。第二梯度为地铁和农村这两个因素，地铁和农村相较于第三梯度的因素重要性也较为突出。第三梯度为剩余其他的因素，重要性不高。

绘制上网特征重要性图如图3。

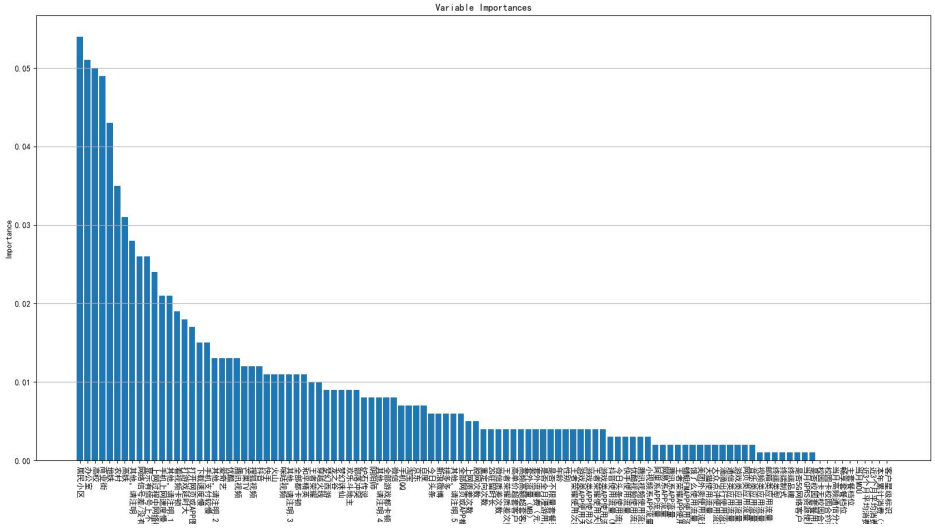


图 3　随机森林上网特征重要性图

由图可得，居民小区这一因素对客户打分影响程度最大，其次是办公室、高校、商业街、地铁和农村。办公室、高校和商业街重要性对比商业街、地铁和农村较高，且和居民小区重要性差别不大。总结，影响客户语音业务满意程度的因素主要是是否遇到过网络问题、居民小区、办公室、高校、商业街。影响客户上网业务满意度的主要因素是居民小区、办公室、高校、商业街。

2.2.3 Permutation importance

Permutation importance 是一种特征重要性的计算方法，它可以用来评估特征对模型的贡献。它的计算方法是将特征的值随机打乱，然后计算打乱后的模型的性能，如果模型的性能下降了，说明该特征对模型的贡献较大，如果模型的性能没有下降，说明该特征对模型的贡献较小。这个想法最早是由 Breiman（2001）[1] 提出，后来由 Fisher，Rudin，and Dominici（2018）改进 [2]。Permutation importance 的计算方法如下：

$$\text{Permutation importance} = \text{baseline performance} - \text{performance with feature permuted} \quad (1)$$

其中 baseline performance 是模型在原始数据上的性能，performance with feature permuted 是模型在特征值随机打乱后的性能。使用 Permutation importance 计算语音通话的特征对用户评分的影响，绘制语音特征重要性图如图4。

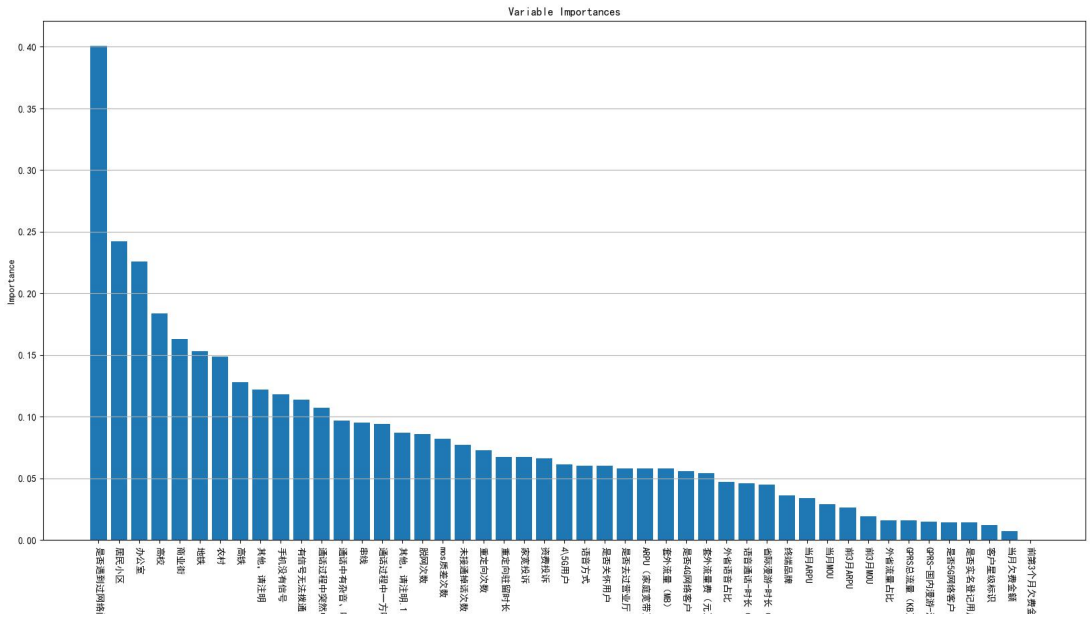


图 4　语音 Permutation importance

由图可得，是否遇到过网络问题这一因素对客户打分影响程度最大也最为突出，重要性远超其他因素。居民小区、办公室这两个因素对客户打分影响程度也较大，其次高

校、商业街、地铁和农村这几个因素重要性相对较高。使用 Permutation importance 计算上网的特征对用户评分的影响，绘制上网特征重要性图如图5。

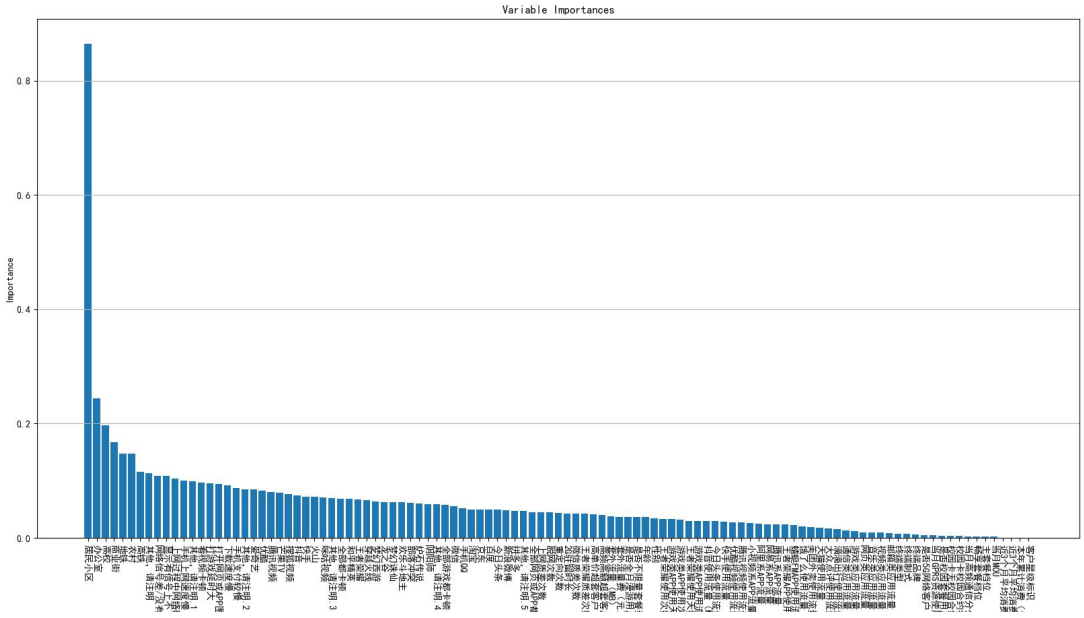


图 5　上网 Permutation importance

由图可以看出居民小区这一因素对客户打分影响程度最大也最为突出。办公室、高校、商业街、地铁和农村这几个因素对客户打分影响程度也较大，相较于之后的其他因素，这些因素的重要性都较高，其中地铁和农村的重要性相差不大。

三、 模型筛选

模型筛选是指从多个模型中选择一个最优的模型，模型筛选的目的是为了提高模型的准确性，减少模型的复杂度，提高模型的可解释性。

3.1 数据预处理

3.1.1 剔除连续变量

在特征选择中发现模型对连续变量的敏感性较强，然而连续值的特征往往是不稳定的，因此在模型筛选中剔除连续变量。

3.1.2 One-hot 编码

One-hot 编码是一种将分类变量转换为数值变量的方法，它将每个分类变量的每个取值转换为一个新的变量，新变量的取值为 0 或 1，0 表示该样本不属于该分类变量的

该取值，1 表示该样本属于该分类变量的该取值。One-hot 编码的计算方法如下：

$$\text{One-hot encoding} = \begin{cases} 1, & \text{if the sample belongs to the category} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

对语音通话和上网数据中的类别属性进行 One-hot 编码，得到语音通话数据和上网数据的 One-hot 编码数据。将数据划分为训练集和测试集，训练集占比为 80%，测试集占比为 20%。特别注意到，语音通话和上网数据中分别需要预测的变量有 4 个，因此需要对每个变量划分数据集。

3.2 模型的筛选

首先选择 XGB、逻辑回归、支持向量机、随机森林、K 近邻算法、朴素贝叶斯、感知机、SGD、决策树、GBDT、ABC-Boost、神经网络使用 12 个模型对语音通话和上网数据进行训练，得到 12 个模型的训练结果，然后使用测试集对 12 个模型进行测试，得到 12 个模型的测试结果。计算出 12 个模型在测试集上的准确率和 RMSE。语音通话数据的准确率和 RMSE 如表3所示。

	Model_voice	acc_voice	RMSE_voice
0	KNN	0.493790	2.567178
1	XGB	0.530589	2.306957
2	LR	0.554508	2.347573
3	SVC	0.561868	2.380458
4	LSVC	0.557958	2.392027
5	RFC	0.519779	2.425890
6	GNB	0.027599	6.498227
7	P	0.526909	2.356378
8	SGD	0.532659	2.327790
9	DTC	0.457222	2.670699
10	GBDT	0.547838	2.359656
11	ABC	0.540248	2.379188
12	MLP	0.521159	2.305453

表 3　12 个模型在语音通话数据的准确率和 RMSE

需要注意的是，无论是语音通话数据还是上网数据，需要预测的变量都有四个，这里对四个指标取了平均值。针对语音业务用户满意度数据使用 13 个模型进行拟合，得

到结果，可以发现大多数模型的准确率在 50% 左右，均方误差都在 2.5 附近，效果稳定。其中 SVC 的准确率最高，但均方误差不是最低的，本数据集使用均方误差来比较模型效果更合适，均方误差最低的模型是 MLP。上网数据的准确率和 RMSE 如表4所示。

	Model_network	acc_network	RMSE_network
0	KNN	0.493790	2.567178
1	XGB	0.530589	2.306957
2	LR	0.554508	2.347573
3	SVC	0.561868	2.380458
4	LSVC	0.557958	2.392027
5	RFC	0.522079	2.385903
6	GNB	0.027599	6.498227
7	P	0.526909	2.356378
8	SGD	0.518169	2.306762
9	DTC	0.456762	2.681358
10	GBDT	0.547608	2.357009
11	ABC	0.540248	2.379188
12	MLP	0.523919	2.351665

表 4　12 个模型在上网数据的准确率和 RMSE

针对上网业务用户满意度数据也使用 13 个模型进行拟合，得到结果，可以发现各个模型的效果与在语音数据集上的表现相差无几，准确率也在 50% 上下波动，均方误差在 2.3 附近，效果也是比较稳定。SVC 准确率仍然是最高的，但 SGD 是均方误差最小的模型。通过对两个数据集的拟合，可以发现，SVC 的拟合准确率是所有模型里最优秀的。可以通过准确率与均方误差结合筛选的方式筛选出模型

四、 模型调参与模型融合

4.1 模型调参

综合准确率和 RMSE，最终选择出 XGB、逻辑回归、SVC、随机森林、GBDT、ABC-Boost 共 6 个模型，并用这六个模型使用网格搜索进行超参数调优。

网格搜索是一种通过遍历给定的参数组合来优化模型表现的方法。网格搜索的优点是可以保证找到最优的参数组合。使用网格搜索对 6 个模型的参数进行调优，得到 6 个模型的最优参数。其中 XGB 的最优参数如表5所示。详细的调优过程和最优参数见附录

一。

	max_depth	n_estimators
xgb1_voice	5	25
xgb2_voice	5	25
xgb3_voice	5	40
xgb4_voice	5	40
xgb1_network	6	35
xgb2_network	6	35
xgb3_network	6	35
xgb4_network	5	40

表 5　12 个模型在语音通话数据的准确率和 RMSE

4.2 模型融合

模型融合是指将多个模型的预测结果进行融合，以提高预测的准确率。模型融合的方法有很多，这里使用的 Stacking 方法。Stacking 方法是将多个模型的预测结果作为新的特征，然后用新的特征训练一个新的模型，这个新的模型就是最终的模型。使用 Stacking 对调参后 6 个模型进行融合，得到最终的模型，并用这个模型对测试集进行预测，模型在测试集上的准确率和 RMSE 如表6所示。

	Model_stacking	acc_stacking	RMSE_stacking
0	语音通话整体满意度	0.592456	2.332326
1	网络覆盖与信号强度	0.521619	2.581692
2	语音通话清晰度	0.571297	2.286716
3	语音通话稳定性	0.539098	2.467638
4	手机上网整体满意度	0.442308	2.984051
5	网络覆盖与信号强度	0.415242	2.691524
6	手机上网速度	0.401709	2.667067
7	手机上网稳定性	0.412393	2.667067

表 6　Stacking 模型的准确率和 RMSE

对于语音数据集中的指标预测准确率均大于 50%，均方误差均小于 2.6，要普遍优于任意单一模型效果；上网数据集的指标预测准确率和均方误差更加稳定，可以得出结

论：模型融合后的效果更好且更稳定。

按照和训练集同样的方法，对附件三和附件四的数据进行预处理，然后使用融合后好的模型对处理后的数据进行预测，得到预测结果，并将预测结果写入到 result.xlsx 文件中。

五、 总结与模型的评价

5.1 总结

本文使用基于 XGB、逻辑回归、SVC、随机森林、GBDT、ABC-Boost 的 Stacking 模型对数据进行了拟合，虽然受到不同客户主观性评价的影响，但是模型的准确率在 50% 左右，RMSE 在 2.3 左右，反映了大部分客户的需求和特点。

对于语音通话，最影响用户体验的是在语音通话中遇到的问题，如遇到网络问题、手机没有信号、无法拨通、突然中断、有杂音、听不清等问题。

对于手机上网，最影响用户体验的是信号和网速问题，如网络信号差、没有信号、网速慢、网速不稳定等问题。同时热门 APP 卡顿时也会影响用户体验。

5.2 模型的改进与推广

- 特征较弱，应加强特征工程，提取更多的特征。
- 没有对异常值进行分析和处理。
- 更精细化的调优。
- 使用 optuna 对模型进行调优。

参考文献

[1] Breiman, Leo. “Random Forests.” Machine Learning 45 (1). Springer: 5-32 (2001).

[2] Fisher, Aaron, Cynthia Rudin, and Francesca Dominici. “All models are wrong, but many are useful: Learning a variable’s importance by studying an entire class of prediction models simultaneously.” http://arxiv.org/abs/1801.01489 (2018).

附录一　代码

2023 年 1 月 19 日

1　导入包

[1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'       # 显示多行

import matplotlib.pyplot as plt
plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
%matplotlib inline
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None) # 显示完整的列
pd.set_option('display.max_rows', None) # 显示完整的行
pd.set_option('display.expand_frame_repr', False) # 设置不折叠数据
pd.set_option('display.max_colwidth', 100)

import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

2　导入数据

[2]:
```
train_voice=pd.read_excel('附件 1 语音业务用户满意度数据.xlsx')
train_network=pd.read_excel('附件 2 上网业务用户满意度数据.xlsx')
test_voice=pd.read_excel('附件 3 语音业务用户满意度预测数据.xlsx')
test_network=pd.read_excel('附件 4 上网业务用户满意度预测数据.xlsx')
```

```
submission_voice = pd.read_excel('result.xlsx',sheet_name='语音')
submission_network = pd.read_excel('result.xlsx',sheet_name='上网')

columns_voice_y = train_voice.columns[1:5]
columns_voice_x = train_voice.columns[5:]

columns_network_y = train_network.columns[1:5]
columns_network_x = train_network.columns[5:]
```

3 EDA

[3]:
```
train_voice.shape,train_network.shape,test_voice.shape,test_network.
↪shape,submission_voice.shape,submission_network.shape
```

[3]: ((5433, 55), (7020, 125), (2599, 44), (1610, 87), (2599, 5), (1610, 5))

4 数据预处理

[4]:
```
train_voice['是否遇到过网络问题']=train_voice['是否遇到过网络问题'].map({2:0,1:1})

for i in range(1,8):
    train_voice[columns_voice_x[i]]=train_voice[columns_voice_x[i]].map({-1:0,i:
↪1})

train_voice[columns_voice_x[8]]=train_voice[columns_voice_x[8]].map({-1:0,98:1})

for i in range(1,7):
    train_voice[columns_voice_x[i+9]]=train_voice[columns_voice_x[i+9]].map({-1:
↪0,i:1})

train_voice[columns_voice_x[16]]=train_voice[columns_voice_x[16]].map({-1:0,98:
↪1})

for i in range(18,25):
    train_voice[columns_voice_x[i]]=pd.
↪cut(train_voice[columns_voice_x[i]],[-1,0.5,1000000],labels=[0,1])
```

```
train_voice[columns_voice_x[25]]=train_voice[columns_voice_x[25]].map({'4G':
↪0,'5G':1})

train_voice[columns_voice_x[26]]=train_voice[columns_voice_x[26]].map({'VOLTE':
↪1, 'EPSFB':2, 'CSFB':3, 'VONR':4, 'GSM':5, 'VoLTE':1})

train_voice[columns_voice_x[27]]=train_voice[columns_voice_x[27]].map({'是':1})
train_voice[columns_voice_x[28]]=train_voice[columns_voice_x[28]].map({'是':1})

train_voice[columns_voice_x[29]]=pd.cut(train_voice[columns_voice_x[29]],[-1,0.
↪5,1000000],labels=[0,1])
train_voice[columns_voice_x[30]]=pd.cut(train_voice[columns_voice_x[30]],[-1,0.
↪5,1000000],labels=[0,1])

train_voice[columns_voice_x[31]]=train_voice[columns_voice_x[31]].map({'是':
↪1,'否':0})

train_voice[columns_voice_x[32]]=pd.cut(train_voice[columns_voice_x[32]],[-1,0.
↪5,1000000],labels=[0,1])
train_voice[columns_voice_x[33]]=pd.cut(train_voice[columns_voice_x[33]],[-1,0.
↪5,1000000],labels=[0,1])
train_voice[columns_voice_x[35]]=pd.cut(train_voice[columns_voice_x[35]],[-1,0.
↪5,1000000],labels=[0,1])

train_voice[columns_voice_x[36]]=train_voice[columns_voice_x[36]].map({'苹果':
↪1,'华为':2,'小米科技':3,'步步高':4,'欧珀':5,'三星':6,'realme':6,0:6,
                                    '万普拉斯':6,'锤子':6,'万普':6,'联通':6,
                                    '中邮通信':6,'其他':6,'中国移动':6,'中兴':6,
                                    '魅族':6,'黑鲨':6,'联想':6,'智🄵互联':6,
                                    '海信':6,'摩托罗拉':6,'乐视移动':6,'天宇朗
通':6,
                                    '奇酷':6,'上海中兴易联通讯股份有限公司':6,'诺
基亚':6})
```

```
train_voice[columns_voice_x[40]]=pd.cut(train_voice[columns_voice_x[40]],[-1,0.
↪5,1000000],labels=[0,1])
train_voice[columns_voice_x[42]]=pd.cut(train_voice[columns_voice_x[42]],[-1,0.
↪5,1000000],labels=[0,1])
train_voice[columns_voice_x[44]]=pd.cut(train_voice[columns_voice_x[44]],[-1,0.
↪5,1000000],labels=[0,1])

train_voice[columns_voice_x[45]]=train_voice[columns_voice_x[45]].map({'否':
↪0,'是':1})
train_voice[columns_voice_x[46]]=train_voice[columns_voice_x[46]].map({'否':
↪0,'是':1})
train_voice[columns_voice_x[47]]=train_voice[columns_voice_x[47]].map({'三星':
↪1, '银卡':2, '二星':3, '一星':4, '金卡':5, '白金卡':6, '未评级':7, '准星':8,␣
↪'钻石卡':9})

train_voice[columns_voice_x[48]]=pd.cut(train_voice[columns_voice_x[48]],[-1,0.
↪5,1000000],labels=[0,1])
train_voice[columns_voice_x[49]]=pd.cut(train_voice[columns_voice_x[49]],[-1,0.
↪5,1000000],labels=[0,1])

for i in columns_voice_x:
    train_voice[columns_voice_x]=train_voice[columns_voice_x].fillna(0)

train_voice=train_voice.drop(columns=['用户 id','用户描述','用户描述.1','终端品牌
类型'],axis=1)
```

[5]:
```
def apply_colour(value):
    if abs(value) > 0.1:
        colour = '#FF0000' #Red
    else:
        colour = '#008000' #Green
    return 'color: %s' % colour

corr_voice= train_voice.corr(method='kendall').iloc[4:,[0,1,2,3]]
train_voice.corr(method='kendall').iloc[4:,[0,1,2,3]].style.
↪applymap(apply_colour)
```

[5]: <pandas.io.formats.style.Styler at 0x7f5a9a5dba10>

[6]:
```
for i in columns_network_x:
    train_network[i]=train_network[i].fillna(0)

for i in range(7):
    train_network[columns_network_x[i]]=train_network[columns_network_x[i]].
 ↪map({-1:0,i+1:1})

train_network[columns_network_x[7]]=train_network[columns_network_x[7]].map({-1:
 ↪0,98:1})

for i in range(9,13):
    train_network[columns_network_x[i]]=train_network[columns_network_x[i]].
 ↪map({-1:0,i+1-9:1})

train_network[columns_network_x[13]]=train_network[columns_network_x[13]].
 ↪map({-1:0,98:1})

for i in range(15,20):
    train_network[columns_network_x[i]]=train_network[columns_network_x[i]].
 ↪map({-1:0,i+1-15:1})

train_network[columns_network_x[20]]=train_network[columns_network_x[20]].
 ↪map({-1:0,98:1})

for i in range(22,31):
    train_network[columns_network_x[i]]=train_network[columns_network_x[i]].
 ↪map({-1:0,i+1-22:1})

train_network[columns_network_x[31]]=train_network[columns_network_x[31]].
 ↪map({-1:0,98:1})
train_network[columns_network_x[33]]=train_network[columns_network_x[33]].
 ↪map({-1:0,99:1})

for i in range(34,44):
```

```
    train_network[columns_network_x[i]]=train_network[columns_network_x[i]].
 ↪map({-1:0,i+1-34:1})

train_network[columns_network_x[44]]=train_network[columns_network_x[44]].
 ↪map({-1:0,98:1})
train_network[columns_network_x[46]]=train_network[columns_network_x[46]].
 ↪map({-1:0,99:1})

for i in range(47,55):
    train_network[columns_network_x[i]]=train_network[columns_network_x[i]].
 ↪map({-1:0,i+1-47:1})
train_network[columns_network_x[55]]=train_network[columns_network_x[55]].
 ↪map({-1:0,98:1})
train_network[columns_network_x[57]]=train_network[columns_network_x[57]].
 ↪map({-1:0,99:1})


# 按 0 分箱
columns_0=[58,59,60,61,62,63,66,67,72,73,74,75,76,77,
           78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,110]
for i in columns_0:
    train_network[columns_network_x[i]]=pd.
 ↪cut(train_network[columns_network_x[i]],[-1,0.5,1000000],labels=[0,1])

# 按是否类别
columns_is_or_not=[64,65,68,69,104,106,107,108]
for i in columns_is_or_not:
    train_network[columns_network_x[i]]=train_network[columns_network_x[i]].
 ↪map({'否':0, '是':1})

train_network[columns_network_x[70]]=pd.
 ↪cut(train_network[columns_network_x[70]],[0,20,30,40,50,60,120],
                                           labels=[1,2,3,4,5,6])
train_network[columns_network_x[71]]=train_network[columns_network_x[71]].
 ↪map({'男':1, '女':2, '性别不详':3})
```

```
train_network[columns_network_x[99]]=train_network[columns_network_x[99]].
 ↪map({'手机':1, '其它':2, '其他可穿戴':2, '平板电脑':2})
train_network[columns_network_x[101]]=train_network[columns_network_x[101]].
 ↪map({'5G 终端':3, '4G 终端':2, '2/3G 终端':1})
train_network[columns_network_x[102]]=train_network[columns_network_x[102]].
 ↪map({'苹果':1,'华为':2,'小米科技':3,'步步高':4,'欧珀':5,'三星':6,'realme':6,0:
 ↪6,
                                    '万普拉斯':6,'锤子':6,'万普':6,'联通':6,
                                    '中邮通信':6,'其他':6,'中国移动':6,'中兴':6,
                                    '魅族':6,'黑鲨':6,'联想':6,'智🄵互联':6,
                                    '海信':6,'摩托罗拉':6,'乐视移动':6,'天宇朗
通':6,
                                    '奇酷':6,'上海中兴易联通讯股份有限公司':6,'诺
基亚':6,'华硕':6,
                                    '甄十信息科技（上海）有限公司':6,'维图':6, '捷
开通讯科技':6, 'TD':6,
                                    '中国电信':6, '北京珠穆朗玛移动通信有限公司':
 ↪6, '欧博信':6,'飞利浦':6, '酷比':6, '索尼爱立信':6, '金立':6})
train_network[columns_network_x[105]]=pd.
 ↪cut(train_network[columns_network_x[105]],[0,10,20,30,50,100,10000],
                                            labels=[1,2,3,4,5,6])
train_network[columns_network_x[109]]=train_network[columns_network_x[109]].
 ↪map({'城区二分公司':1, '城区三分公司':2, '城区一分公司':3, '不详':4, '通州分公
司':5,
                                                                    '大
兴分公司':6, '昌平分公司':7, '顺义分公司':8,'房山分公司':9, '密云分公司':10,
                                                                    '怀
柔分公司':11, '平谷分公司':12, '延庆分公司':13})
train_network[columns_network_x[119]]=train_network[columns_network_x[119]].
 ↪map({'三星':1, '银卡':2, '二星':3, '一星':4, '金卡':5, '白金卡':6, '未评级':7,␣
 ↪'准星':8, '钻石卡':9})

train_network=train_network.drop(columns=['用户','场景备注数据','现象备注数
据','APP 大类备注','APP 小类视频备注','APP 小类游戏备注','APP 小类上网备注',
```

```
                                                    '操作系统','终端品牌类型','畅享套餐名
称','码号资源-激活时间','码号资源-发卡时间'],axis=1)
```

[7]:
```
corr_network = train_network.corr(method='kendall').iloc[4:,[0,1,2,3]]
train_network.corr(method='kendall').iloc[4:,[0,1,2,3]].style.
↪applymap(apply_colour)
```

[7]:
```
<pandas.io.formats.style.Styler at 0x7f5a9a23b1d0>
```

5 特征选择

5.0.1 语音

[8]:
```
from sklearn.model_selection import train_test_split
from sklearn import tree # 导入需要的模块
import numpy as np
from imblearn.over_sampling import RandomOverSampler

from sklearn.metrics import mean_squared_error
import math

def RMSE(model,x,y):
    return math.sqrt(mean_squared_error(y, model.predict(x)))
```

[9]:
```
train_voice = train_voice.astype('float32')
train_network = train_network.astype('float32')
train_network.fillna(0, inplace=True)
```

[10]:
```
y1 = train_voice.iloc[:,:4]
x1 = train_voice.iloc[:,4:]

y2 = train_network.iloc[:,:4]
x2 = train_network.iloc[:,4:]

ros = RandomOverSampler(random_state=420)
```

```
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.
↪2,random_state=420)
y1_1_train, y1_2_train, y1_3_train, y1_4_train = y1_train.iloc[:, 0], y1_train.
↪iloc[:, 1], y1_train.iloc[:, 2], y1_train.iloc[:, 3]
y1_1_test, y1_2_test, y1_3_test, y1_4_test = y1_test.iloc[:, 0], y1_test.iloc[:
↪, 1], y1_test.iloc[:, 2], y1_test.iloc[:, 3]

x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y2, test_size=0.
↪2,random_state=420)
y2_1_train, y2_2_train, y2_3_train, y2_4_train = y2_train.iloc[:, 0], y2_train.
↪iloc[:, 1], y2_train.iloc[:, 2], y2_train.iloc[:, 3]
y2_1_test, y2_2_test, y2_3_test, y2_4_test = y2_test.iloc[:, 0], y2_test.iloc[:
↪, 1], y2_test.iloc[:, 2], y2_test.iloc[:, 3]

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in [1,2]:
    for j in [1,2,3,4]:
        for s in ['train','test']:
            exec('y%s_%s_%s=le.fit_transform(y%s_%s_%s)'%(i,j,s,i,j,s))

for i in [1,2,3,4]:
    exec('x1_%s_train_balance, y1_%s_train_balance = x1_train,␣
↪y1_%s_train'%(i,i,i))



for i in [1,2,3,4]:
    exec('x2_%s_train_balance, y2_%s_train_balance = x2_train,␣
↪y2_%s_train'%(i,i,i))
```

[11]:
```
from xgboost.sklearn import XGBClassifier
from sklearn.ensemble import RandomForestClassifier

clf11 = RandomForestClassifier()
clf11 = clf11.fit(x1_1_train_balance,y1_1_train_balance) # 用训练集数据训练模型
#result11 = clf11.score(x1_test,y1_1_test) # 对我们训练的模型精度进行打分
```

```
result11 = RMSE(clf11,x1_test,y1_1_test)# 对我们训练的模型精度进行打分
importance11 = list(clf11.feature_importances_)

clf12 = RandomForestClassifier()
clf12 = clf12.fit(x1_2_train_balance,y1_2_train_balance) # 用训练集数据训练模型
#result12 = clf12.score(x1_test,y1_2_test) # 对我们训练的模型精度进行打分
result12 = RMSE(clf12,x1_test,y1_2_test) # 对我们训练的模型精度进行打分
importance12 = list(clf12.feature_importances_)

clf13 = RandomForestClassifier()
clf13 = clf13.fit(x1_3_train_balance,y1_3_train_balance)# 用训练集数据训练模型
#result13 = clf13.score(x1_test,y1_3_test) # 对我们训练的模型精度进行打分
result13 = RMSE(clf13,x1_test,y1_3_test) # 对我们训练的模型精度进行打分
importance13 = list(clf13.feature_importances_)

clf14 = RandomForestClassifier()
clf14 = clf14.fit(x1_4_train_balance,y1_4_train_balance) # 用训练集数据训练模型
#result14 = clf14.score(x1_test,y1_4_test) # 对我们训练的模型精度进行打分
result14 = RMSE(clf14,x1_test,y1_4_test) # 对我们训练的模型精度进行打分
importance14 = list(clf14.feature_importances_)
result11, result12, result13, result14
```

[11]:
```
(2.3192724805435088, 2.474525220925671, 2.2589900129404863, 2.3770599452770775)
```

[12]:
```
feature_list1 = list(train_voice.columns)[4:]
importance1 = [i for i in (importance11 + importance12 + importance13 +␣
 ↪importance14)]
feature_importance1 = [(feature, round(importance, 3)) for feature, importance␣
 ↪in zip(feature_list1, importance1)]
feature_importance1= sorted(feature_importance1, key=lambda x: x[1],␣
 ↪reverse=True)
feature_importance1
```

[12]:
```
[('GPRS 总流量 (KB) ', 0.092),
 ('当月 ARPU', 0.091),
 ('前 3 月 MOU', 0.09),
 ('当月 MOU', 0.087),
```

('语音通话-时长（分钟）', 0.086),
('终端品牌', 0.04),
('客户星级标识', 0.035),
('是否遇到过网络问题', 0.028),
('通话中有杂音、听不清、断断续续', 0.027),
('通话过程中突然中断', 0.023),
('居民小区', 0.022),
('办公室', 0.022),
('是否去过营业厅', 0.022),
('语音方式', 0.021),
('手机没有信号', 0.02),
('脱网次数', 0.02),
('有信号无法拨通', 0.019),
('通话过程中一方听不见', 0.019),
('未接通掉话次数', 0.018),
('mos 质差次数', 0.017),
('地铁', 0.016),
('高铁', 0.013),
('4or5G 用户', 0.012),
('商业街', 0.011),
('重定向次数', 0.011),
('重定向驻留时长', 0.011),
('是否 5G 网络客户', 0.011),
('是否实名登记用户', 0.011),
('农村', 0.01),
('GPRS-国内漫游-流量（KB）', 0.01),
('套外流量费（元）', 0.009),
('省际漫游-时长（分钟）', 0.009),
('其他，请注明', 0.008),
('串线', 0.007),
('套外流量（MB）', 0.006),
('前第 3 个月欠费金额', 0.006),
('高校', 0.005),
('其他，请注明.1', 0.005),
('是否关怀用户', 0.005),
('外省语音占比', 0.005),

```
('当月欠费金额', 0.005),
('家宽投诉', 0.004),
('外省流量占比', 0.004),
('ARPU（家庭宽带）', 0.003),
('前 3 月 ARPU', 0.003),
('是否 4G 网络客户（本地剔除物联网）', 0.001),
('资费投诉', 0.0)]
```

[13]:
```
x1_values = list(range(len(feature_importance1)))
plt.figure(figsize=(20, 10), dpi=100)
plt.bar(range(len(x1_values)), pd.DataFrame(feature_importance1).iloc[:,1],␣
↪orientation='vertical')
plt.xticks(x1_values, feature_list1, rotation=270)
plt.ylabel('Importance')
plt.xlabel('Variable')
plt.title('Variable Importances')
plt.grid(axis='y')
plt.savefig('2.jpg')
plt.show()
```

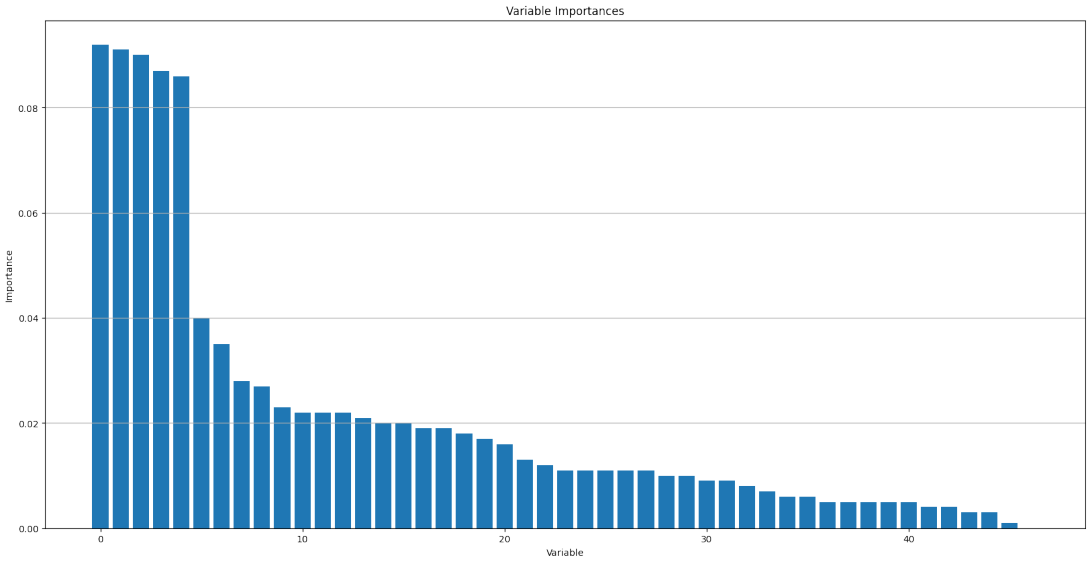
[13]:
```
<Figure size 2000x1000 with 0 Axes>
```

[13]:
```
<BarContainer object of 47 artists>
```

[13]:
```
Text(0, 0.5, 'Importance')
```

[13]:
```
Text(0.5, 0, 'Variable')
```

[13]:
```
Text(0.5, 1.0, 'Variable Importances')
```



[14]:
```
def permutation_importances(model, X, y, metric):
    baseline = metric(model,X, y)
    imp = []
    for col in X.columns:
        save = X[col].copy()
        X[col] = np.random.permutation(X[col])
        m = metric(model,X, y)
        X[col] = save
        imp.append(baseline - m)
    return np.array(imp)
```

[15]:
```
from sklearn.metrics import accuracy_score
im1=abs(permutation_importances(clf11,x1_test,y1_1_test,RMSE))
im2=abs(permutation_importances(clf12,x1_test,y1_2_test,RMSE))
im3=abs(permutation_importances(clf13,x1_test,y1_3_test,RMSE))
im4=abs(permutation_importances(clf14,x1_test,y1_4_test,RMSE))
feature_list1 = list(train_voice.columns)[4:]
importance2 = [i for i in (im1+im2+im3+im4)]
feature_importance = [(feature, round(importance, 3)) for feature, importance␣
 ↪in zip(feature_list1, (im1+im2+im3+im4))]
feature_importance=sorted(feature_importance, key=lambda x: x[1], reverse=True)
```

```
feature_importance
```

[15]:
```
[('是否遇到过网络问题', 0.372),
 ('通话中有杂音、听不清、断断续续', 0.229),
 ('办公室', 0.186),
 ('终端品牌', 0.174),
 ('有信号无法拨通', 0.139),
 ('语音通话-时长（分钟）', 0.132),
 ('通话过程中突然中断', 0.131),
 ('居民小区', 0.122),
 ('手机没有信号', 0.122),
 ('客户星级标识', 0.122),
 ('当月 MOU', 0.12),
 ('当月 ARPU', 0.109),
 ('农村', 0.104),
 ('未接通掉话次数', 0.098),
 ('通话过程中一方听不见', 0.096),
 ('4or5G 用户', 0.093),
 ('重定向驻留时长', 0.083),
 ('语音方式', 0.082),
 ('重定向次数', 0.077),
 ('地铁', 0.075),
 ('是否去过营业厅', 0.074),
 ('商业街', 0.068),
 ('前 3 月 MOU', 0.065),
 ('串线', 0.062),
 ('GPRS 总流量（KB）', 0.062),
 ('脱网次数', 0.059),
 ('外省语音占比', 0.059),
 ('是否 5G 网络客户', 0.059),
 ('高铁', 0.055),
 ('是否实名登记用户', 0.051),
 ('高校', 0.048),
 ('mos 质差次数', 0.048),
 ('GPRS-国内漫游-流量（KB）', 0.047),
 ('其他，请注明', 0.041),
```

```
('其他，请注明.1', 0.04),
('是否关怀用户', 0.037),
('套外流量费（元）', 0.033),
('省际漫游-时长（分钟）', 0.031),
('当月欠费金额', 0.031),
('前第 3 个月欠费金额', 0.03),
('家宽投诉', 0.027),
('ARPU（家庭宽带）', 0.027),
('套外流量（MB）', 0.023),
('是否 4G 网络客户（本地剔除物联网）', 0.017),
('前 3 月 ARPU', 0.017),
('外省流量占比', 0.017),
('资费投诉', 0.0)]
```

[16]:
```
x1_values = list(range(len(feature_importance)))
plt.figure(figsize=(20, 10), dpi=100)
plt.bar(range(len(x1_values)), pd.DataFrame(feature_importance).iloc[:,1],␣
↪orientation='vertical')
plt.xticks(x1_values, feature_list1, rotation=270)
plt.ylabel('Importance')
plt.xlabel('Variable')
plt.title('Variable Importances')
plt.grid(axis='y')
plt.savefig('3.jpg')
plt.show()
```

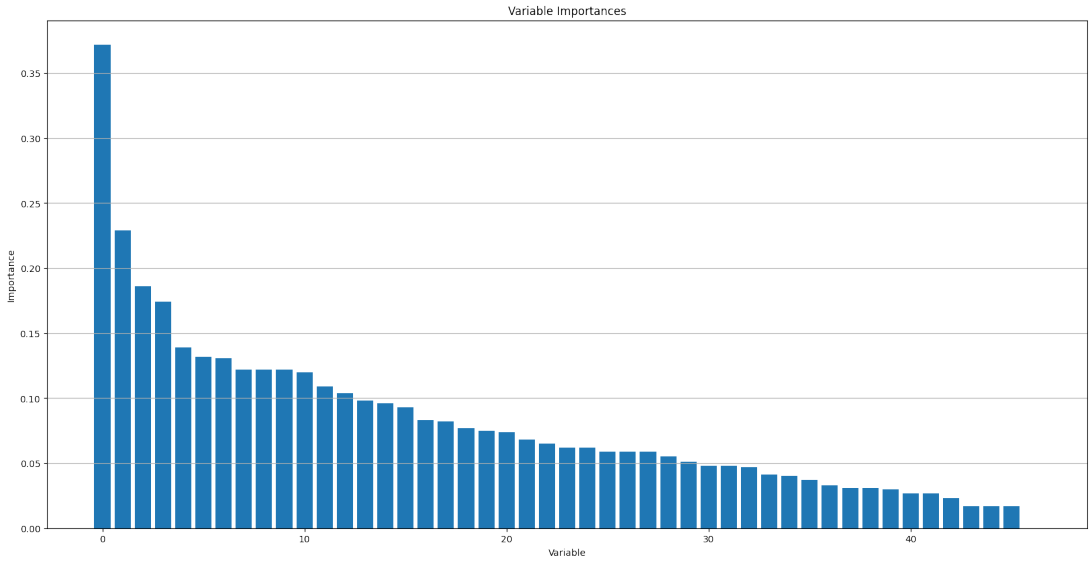
[16]:
```
<Figure size 2000x1000 with 0 Axes>
```

[16]:
```
<BarContainer object of 47 artists>
```

[16]:
```
Text(0, 0.5, 'Importance')
```

[16]:
```
Text(0.5, 0, 'Variable')
```

[16]:
```
Text(0.5, 1.0, 'Variable Importances')
```



[17]:
```
voice_list = []
for i in corr_voice.T.columns:
    if(abs(corr_voice.T[i].mean())>0.1 or (importance1[list(corr_voice.T.
 ↪columns).index(i)] > 0.015 and  importance2[list(corr_voice.T.columns).
 ↪index(i)] > 0.076)):i
```

[17]: '是否遇到过网络问题'

[17]: '居民小区'

[17]: '办公室'

[17]: '高校'

[17]: '商业街'

[17]: '地铁'

[17]: '农村'

[17]: '高铁'

[17]: '其他，请注明'

[17]: '手机没有信号'

[17]: '有信号无法拨通'

[17]: '通话过程中突然中断'

[17]: '通话中有杂音、听不清、断断续续'

[17]: '串线'

[17]: '通话过程中一方听不见'

[17]: '其他，请注明.1'

[17]: '是否关怀用户'

[17]: '终端品牌'

[17]: '当月 MOU'

[17]: '客户星级标识'

5.0.2 上网

[18]:
```
clf21 = RandomForestClassifier()
clf21 = clf21.fit(x2_train,y2_1_train) # 用训练集数据训练模型
result21 = clf21.score(x2_test,y2_1_test) # 对我们训练的模型精度进行打分
importance21 = list(clf21.feature_importances_)

clf22 = RandomForestClassifier()
clf22 = clf22.fit(x2_train,y2_2_train) # 用训练集数据训练模型
result22 = clf22.score(x2_test,y2_2_test) # 对我们训练的模型精度进行打分
importance22 = list(clf22.feature_importances_)

clf23 = RandomForestClassifier()
clf23 = clf23.fit(x2_train,y2_3_train) # 用训练集数据训练模型
result23 = clf23.score(x2_test,y2_3_test) # 对我们训练的模型精度进行打分
importance23 = list(clf23.feature_importances_)

clf24 =RandomForestClassifier()
clf24 = clf24.fit(x2_train,y2_4_train) # 用训练集数据训练模型
```

```
result24 = clf24.score(x2_test,y2_4_test) # 对我们训练的模型精度进行打分
importance24 = list(clf24.feature_importances_)
result21, result22, result23, result24
```

[18]:
```
(0.43660968660968663,
 0.3995726495726496,
 0.4024216524216524,
 0.4074074074074074)
```

[19]:
```
feature_list2 = list(train_network.columns)[4:]
importance2 = [i for i in (importance21 + importance22 + importance23 +␣
 ↪importance24)]
feature_importance2 = [(feature, round(importance, 3)) for feature, importance␣
 ↪in zip(feature_list2, importance2)]
feature_importance2 = sorted(feature_importance2, key=lambda x: x[1],␣
 ↪reverse=True)
feature_importance2
```

[19]:
```
[('当月 MOU', 0.056),
 ('本年累计消费（元）', 0.051),
 ('近 3 个月平均消费（剔除通信账户支付）', 0.049),
 ('近 3 个月平均消费（元）', 0.049),
 ('主套餐档位', 0.044),
 ('当月高频通信分公司', 0.035),
 ('年龄', 0.03),
 ('当月 GPRS 资源使用量（GB）', 0.028),
 ('网络信号差/没有信号', 0.027),
 ('终端品牌', 0.026),
 ('上网过程中网络时断时续或时快时慢', 0.023),
 ('重定向次数', 0.023),
 ('客户星级标识', 0.02),
 ('居民小区', 0.019),
 ('性别', 0.017),
 ('显示有信号上不了网', 0.016),
 ('办公室', 0.015),
 ('地铁', 0.015),
 ('快手使用流量', 0.013),
```

('是否 5G 网络客户', 0.013),
('畅享套餐档位', 0.013),
('手机上网速度慢', 0.012),
('2G 驻留时长', 0.012),
('优酷视频使用流量', 0.012),
('滴滴出行使用流量', 0.012),
('终端制式', 0.012),
('微信质差次数', 0.011),
('抖音使用流量 (MB) ', 0.011),
('大众点评使用流量', 0.011),
('游戏类应用流量', 0.011),
('高铁', 0.01),
('网易系 APP 流量', 0.01),
('音乐类应用流量', 0.01),
('农村', 0.009),
('打开网页或 APP 图片慢', 0.009),
('脱网次数', 0.009),
('小视频系 APP 流量', 0.009),
('饿了么使用流量', 0.009),
('商业街', 0.008),
('看视频卡顿', 0.008),
('上网质差次数', 0.008),
('是否不限量套餐到达用户', 0.008),
('游戏类 APP 使用流量', 0.008),
('今日头条使用流量', 0.008),
('腾讯视频使用流量', 0.008),
('游戏类 APP 使用天数', 0.007),
('游戏类 APP 使用次数', 0.007),
('其他, 请注明', 0.006),
('下载速度慢', 0.006),
('手机支付较慢', 0.006),
('微信', 0.006),
('套外流量费 (元) ', 0.006),
('淘宝', 0.005),
('全部网页或 APP 都慢', 0.005),
('天猫使用流量', 0.005),

('邮箱类应用流量', 0.005),
('高校', 0.004),
('打游戏延时大', 0.004),
('抖音', 0.004),
('全部都卡顿', 0.004),
('京东', 0.004),
('百度', 0.004),
('王者荣耀质差次数', 0.004),
('高单价超套客户（集团）', 0.004),
('套外流量（MB）', 0.004),
('王者荣耀使用次数', 0.004),
('王者荣耀使用天数', 0.004),
('阿里系 APP 流量', 0.004),
('腾讯系 APP 流量', 0.004),
('王者荣耀 APP 使用流量', 0.004),
('视频类应用流量', 0.004),
('其他，请注明.1', 0.003),
('爱奇艺', 0.003),
('腾讯视频', 0.003),
('今日头条', 0.003),
('蜻蜓 FMAPP 使用流量', 0.003),
('通信类应用流量', 0.003),
('是否校园套餐用户', 0.003),
('校园卡校园合约捆绑用户', 0.003),
('优酷', 0.002),
('快手', 0.002),
('其他，请注明.3', 0.002),
('王者荣耀', 0.002),
('全部游戏都卡顿', 0.002),
('手机 QQ', 0.002),
('新浪微博', 0.002),
('拼多多', 0.002),
('高频高额超套客户（集团）', 0.002),
('是否全月漫游用户', 0.002),
('网页类应用流量', 0.002),
('其他，请注明.2', 0.001),

```
('芒果 TV', 0.001),
('搜狐视频', 0.001),
('火山', 0.001),
('咪咕视频', 0.001),
('和平精英', 0.001),
('欢乐斗地主', 0.001),
('其他，请注明.4', 0.001),
('其他，请注明.5', 0.001),
('穿越火线', 0.0),
('梦幻西游', 0.0),
('龙之谷', 0.0),
('梦幻诛仙', 0.0),
('部落冲突', 0.0),
('炉石传说', 0.0),
('阴阳师', 0.0),
('美团外卖使用流量', 0.0),
('终端类型', 0.0),
('校园卡无校园合约用户', 0.0)]
```

[20]:
```
x2_values = list(range(len(feature_importance2)))
plt.figure(figsize=(20, 10), dpi=100)
plt.bar(range(len(x2_values)), pd.DataFrame(feature_importance2).iloc[:,1],␣
↪orientation='vertical')
plt.xticks(x2_values, feature_list2, rotation=270)
plt.ylabel('Importance')
plt.xlabel('Variable')
plt.title('Variable Importances')
plt.grid(axis='y')
plt.savefig('4.jpg')
plt.show()
```

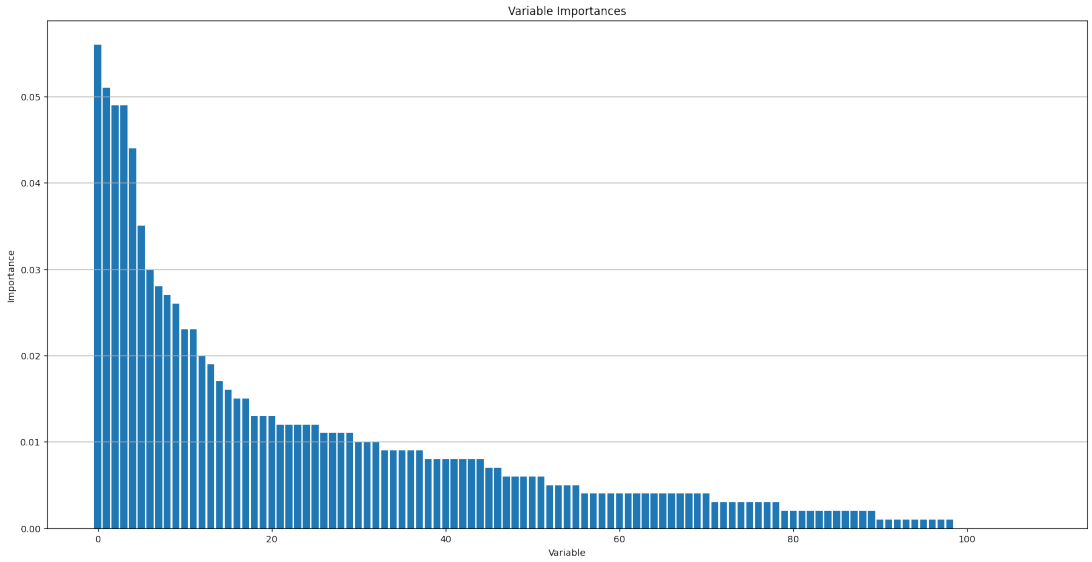
[20]:
```
<Figure size 2000x1000 with 0 Axes>
```

[20]:
```
<BarContainer object of 109 artists>
```

[20]:
```
Text(0, 0.5, 'Importance')
```

[20]:
```
Text(0.5, 0, 'Variable')
```

[20]: Text(0.5, 1.0, 'Variable Importances')



[21]:
```
from sklearn.metrics import accuracy_score
im1=abs(permutation_importances(clf21,x2_test,y2_1_test,RMSE))
im2=abs(permutation_importances(clf22,x2_test,y2_2_test,RMSE))
im3=abs(permutation_importances(clf23,x2_test,y2_3_test,RMSE))
im4=abs(permutation_importances(clf24,x2_test,y2_4_test,RMSE))
feature_list2 = list(train_network.columns)[4:]
importance2 = [i for i in (im1+im2+im3+im4)]
feature_importance = [(feature, round(importance, 3)) for feature, importance␣
↪in zip(feature_list2, (im1+im2+im3+im4))]
feature_importance = sorted(feature_importance, key=lambda x: x[1],␣
↪reverse=True)
feature_importance
```

[21]:
```
[('重定向次数', 1.033),
 ('近 3 个月平均消费（元）', 0.198),
 ('居民小区', 0.193),
 ('网络信号差/没有信号', 0.179),
 ('2G 驻留时长', 0.159),
 ('大众点评使用流量', 0.148),
```

('显示有信号上不了网', 0.141),
('本年累计消费（元）', 0.126),
('是否 5G 网络客户', 0.121),
('上网过程中网络时断时续或时快时慢', 0.119),
('脱网次数', 0.119),
('年龄', 0.119),
('当月高频通信分公司', 0.115),
('小视频系 APP 流量', 0.11),
('当月 MOU', 0.108),
('打开网页或 APP 图片慢', 0.106),
('客户星级标识', 0.105),
('微信', 0.103),
('快手使用流量', 0.101),
('地铁', 0.097),
('商业街', 0.096),
('游戏类 APP 使用天数', 0.089),
('手机上网速度慢', 0.088),
('主套餐档位', 0.088),
('游戏类应用流量', 0.087),
('当月 GPRS 资源使用量（GB）', 0.083),
('办公室', 0.077),
('抖音', 0.074),
('王者荣耀使用次数', 0.074),
('抖音使用流量（MB）', 0.073),
('看视频卡顿', 0.071),
('上网质差次数', 0.07),
('性别', 0.068),
('高铁', 0.067),
('游戏类 APP 使用次数', 0.065),
('网页类应用流量', 0.065),
('其他，请注明', 0.063),
('终端品牌', 0.061),
('打游戏延时大', 0.057),
('手机 QQ', 0.057),
('其他，请注明.1', 0.056),
('微信质差次数', 0.056),

('通信类应用流量', 0.056),
('腾讯视频使用流量', 0.054),
('阿里系 APP 流量', 0.054),
('腾讯系 APP 流量', 0.054),
('近 3 个月平均消费（剔除通信账户支付）', 0.054),
('今日头条使用流量', 0.053),
('全部都卡顿', 0.052),
('终端制式', 0.051),
('淘宝', 0.05),
('饿了么使用流量', 0.05),
('音乐类应用流量', 0.05),
('王者荣耀 APP 使用流量', 0.047),
('拼多多', 0.045),
('视频类应用流量', 0.045),
('高校', 0.041),
('手机支付较慢', 0.041),
('农村', 0.04),
('网易系 APP 流量', 0.04),
('畅享套餐档位', 0.039),
('新浪微博', 0.038),
('王者荣耀使用天数', 0.037),
('百度', 0.036),
('京东', 0.035),
('今日头条', 0.035),
('优酷视频使用流量', 0.035),
('天猫使用流量', 0.033),
('是否校园套餐用户', 0.033),
('下载速度慢', 0.032),
('和平精英', 0.032),
('王者荣耀', 0.031),
('游戏类 APP 使用流量', 0.031),
('滴滴出行使用流量', 0.031),
('邮箱类应用流量', 0.03),
('芒果 TV', 0.029),
('校园卡校园合约捆绑用户', 0.029),
('快手', 0.028),

```
('全部游戏都卡顿', 0.027),
('全部网页或 APP 都慢', 0.027),
('腾讯视频', 0.025),
('高单价超套客户（集团）', 0.025),
('套外流量（MB）', 0.021),
('是否不限量套餐到达用户', 0.019),
('优酷', 0.018),
('爱奇艺', 0.017),
('王者荣耀质差次数', 0.017),
('蜻蜓 FMAPP 使用流量', 0.017),
('是否全月漫游用户', 0.016),
('梦幻诛仙', 0.015),
('火山', 0.014),
('套外流量费（元）', 0.012),
('其他，请注明.3', 0.011),
('搜狐视频', 0.01),
('咪咕视频', 0.009),
('其他，请注明.4', 0.008),
('其他，请注明.2', 0.006),
('欢乐斗地主', 0.006),
('其他，请注明.5', 0.005),
('校园卡无校园合约用户', 0.004),
('部落冲突', 0.002),
('炉石传说', 0.001),
('高频高额超套客户（集团）', 0.001),
('穿越火线', 0.0),
('梦幻西游', 0.0),
('龙之谷', 0.0),
('阴阳师', 0.0),
('美团外卖使用流量', 0.0),
('终端类型', 0.0)]
```

[22]:
```
x2_values = list(range(len(feature_importance)))
plt.figure(figsize=(20, 10), dpi=100)
plt.bar(range(len(x2_values)), pd.DataFrame(feature_importance).iloc[:,1],␣
↪orientation='vertical')
```

```
plt.xticks(x2_values, feature_list2, rotation=270)
plt.ylabel('Importance')
plt.xlabel('Variable')
plt.title('Variable Importances')
plt.grid(axis='y')
plt.savefig('5.jpg')
plt.show()
```

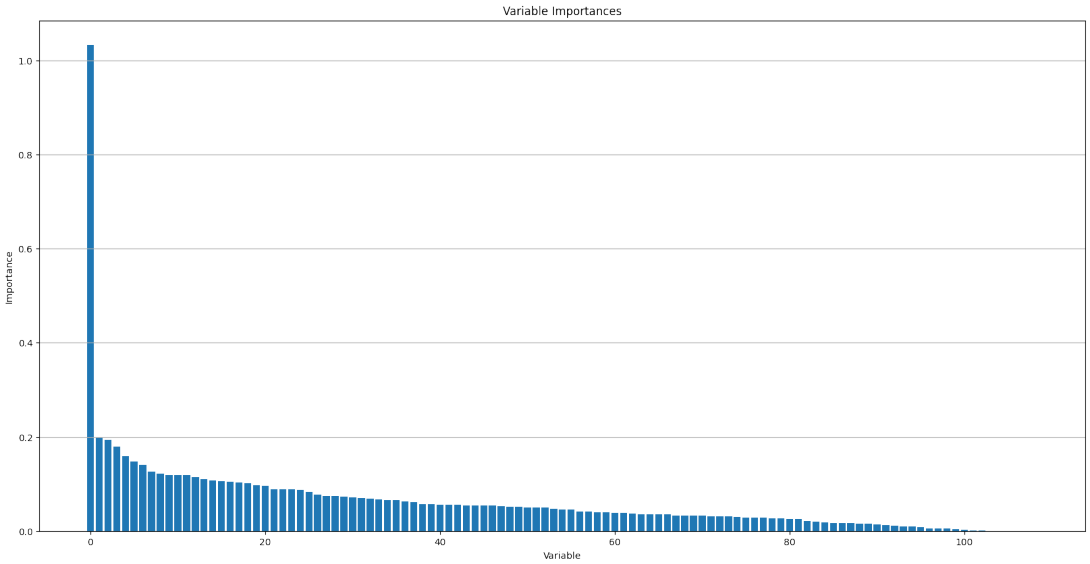
[22]: <Figure size 2000x1000 with 0 Axes>

[22]: <BarContainer object of 109 artists>

[22]: Text(0, 0.5, 'Importance')

[22]: Text(0.5, 0, 'Variable')

[22]: Text(0.5, 1.0, 'Variable Importances')



[23]:
```
network_list = []
for i in corr_network.T.columns:
```

```
    if(abs(corr_network.T[i].mean())>0.1 or (importance1[list(corr_network.T.
↪columns).index(i)] > 0.015 and  importance2[list(corr_network.T.columns).
↪index(i)] > 0.076)):i
```

[23]: '居民小区'

[23]: '办公室'

[23]: '高校'

[23]: '商业街'

[23]: '地铁'

[23]: '农村'

[23]: '高铁'

[23]: '其他，请注明'

[23]: '网络信号差/没有信号'

[23]: '显示有信号上不了网'

[23]: '上网过程中网络时断时续或时快时慢'

[23]: '手机上网速度慢'

[23]: '看视频卡顿'

[23]: '打游戏延时大'

[23]: '打开网页或 APP 图片慢'

[23]: '下载速度慢'

[23]: '手机支付较慢'

[23]: '爱奇艺'

[23]: '优酷'

[23]: '腾讯视频'

[23]: '抖音'

[23]: '快手'

[23]: '全部都卡顿'

[23]: '和平精英'

[23]: '王者荣耀'

[23]: '全部游戏都卡顿'

[23]: '微信'

[23]: '手机 QQ'

[23]: '淘宝'

[23]: '京东'

[23]: '百度'

[23]: '今日头条'

[23]: '新浪微博'

[23]: '拼多多'

[23]: '全部网页或 APP 都慢'

[23]: '终端品牌'

[23]: '当月高频通信分公司'

6 模型筛选

[24]:
```
train_voice.shape,train_network.shape
```

[24]: ((5433, 51), (7020, 113))

[25]:
```
try:
    train_voice=train_voice.drop(columns=['语音通话-时长（分钟）','当月 ARPU','当月 MOU','前 3 月 MOU','GPRS 总流量（KB）'],axis=1)
```

```
    for i in train_voice.columns:
        if(i not in test_voice.columns and i not in train_voice.iloc[:,:5].
 ↪columns):
            train_voice=train_voice.drop(i,axis=1)

    train_network=train_network.drop(columns=['主套餐档位','当月 MOU','近 3 个月
平均消费（剔除通信账户支付）','近 3 个月平均消费（元）','本年累计消费（元）'],axis=1)
    for i in train_network.columns:
        if(i not in test_network.columns and i not in train_network.iloc[:,:5].
 ↪columns and i not in ['是否遇到网络问题']):
            train_network=train_network.drop(i,axis=1)
except:
    print('并没有完整执行 try 的内容')
```

[26]:
```
train_voice.shape,train_network.shape
test_voice.shape,test_network.shape
```

[26]: ((5433, 35), (7020, 66))

[26]: ((2599, 44), (1610, 87))

[27]:
```
y1 = train_voice.iloc[:,:4]
x1_temp = train_voice.iloc[:,4:]

y2 = train_network.iloc[:,:4]
x2_temp = train_network.iloc[:,4:]

from sklearn.preprocessing import OneHotEncoder
x1 = pd.DataFrame([])
for i in x1_temp.columns:
    temp=pd.get_dummies(x1_temp[i])
    for j in range(len(temp.columns)):
        temp1 = pd.DataFrame(temp.iloc[:,[j]].values,columns=['%s_%s'%(i,j)])
        x1=pd.concat([x1,temp1],axis =1)

x2 = pd.DataFrame([])
for i in x2_temp.columns:
```

```
    temp=pd.get_dummies(x2_temp[i])
    for j in range(len(temp.columns)):
        temp1 = pd.DataFrame(temp.iloc[:,[j]].values,columns=['%s_%s'%(i,j)])
        x2=pd.concat([x2,temp1],axis =1)


ros = RandomOverSampler(random_state=420)

x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.
 ↪2,random_state=20)
y1_1_train, y1_2_train, y1_3_train, y1_4_train = y1_train.iloc[:, 0], y1_train.
 ↪iloc[:, 1], y1_train.iloc[:, 2], y1_train.iloc[:, 3]
y1_1_test, y1_2_test, y1_3_test, y1_4_test = y1_test.iloc[:, 0], y1_test.iloc[:
 ↪, 1], y1_test.iloc[:, 2], y1_test.iloc[:, 3]

x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y2, test_size=0.
 ↪2,random_state=20)
y2_1_train, y2_2_train, y2_3_train, y2_4_train = y2_train.iloc[:, 0], y2_train.
 ↪iloc[:, 1], y2_train.iloc[:, 2], y2_train.iloc[:, 3]
y2_1_test, y2_2_test, y2_3_test, y2_4_test = y2_test.iloc[:, 0], y2_test.iloc[:
 ↪, 1], y2_test.iloc[:, 2], y2_test.iloc[:, 3]

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in [1,2]:
    for j in [1,2,3,4]:
        for s in ['train','test']:
            exec('y%s_%s_%s=le.fit_transform(y%s_%s_%s)'%(i,j,s,i,j,s))

for i in [1,2,3,4]:
    exec('x1_%s_train_balance, y1_%s_train_balance = x1_train,␣
 ↪y1_%s_train'%(i,i,i))


for i in [1,2,3,4]:
```

```
    exec('x2_%s_train_balance, y2_%s_train_balance = x2_train,␣
 ↪y2_%s_train'%(i,i,i))
```

[28]:
```
from xgboost.sklearn import XGBClassifier as XGB
from sklearn.linear_model import LogisticRegression as LR
from sklearn.svm import SVC
from sklearn.svm import LinearSVC as LSVC
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.neighbors import KNeighborsClassifier  as KNN
from sklearn.naive_bayes import GaussianNB  as GNB
from sklearn.linear_model import Perceptron  as P
from sklearn.linear_model import SGDClassifier as SGD
from sklearn.tree import DecisionTreeClassifier  as DTC
from sklearn.ensemble import GradientBoostingClassifier as GBDT
from sklearn.ensemble import  AdaBoostClassifier as ABC
from sklearn.neural_network import MLPClassifier as MLP

models =␣
 ↪['KNN','XGB','LR','SVC','LSVC','RFC','GNB','P','SGD','DTC','GBDT','ABC','MLP']
#models = ['ABC','RFC','LR','XGB','SVC','GBDT']

for i in models:
    for j in [1,2,3,4]:
        exec('''%s%s_voice = %s()\n%s%s_voice = %s%s_voice.
 ↪fit(x1_%s_train_balance,y1_%s_train_balance)\nacc_%s_%s_voice = %s%s_voice.
 ↪score(x1_test,y1_%s_test)\nRMSE_%s_%s_voice =␣
 ↪RMSE(%s%s_voice,x1_test,y1_%s_test)'''
             %(i,j,i,i,j,i,j,j,j,i,j,i,j,j,i,j,i,j,j),globals())

    exec('''acc_%s_voice=(acc_%s_1_voice+acc_%s_2_voice+
                          acc_%s_3_voice+acc_%s_4_voice)/4\n
            RMSE_%s_voice=(RMSE_%s_1_voice+RMSE_%s_2_voice+
                           RMSE_%s_3_voice+RMSE_%s_4_voice)/4\n
            print('voice',i,',acc:',acc_%s_voice,'RMSE:',RMSE_%s_voice)'''
         %(i,i,i,i,i,i,i,i,i,i,i),globals())
```

```
for i in models:
    for j in [1,2,3,4]:
        exec('''%s%s_network = %s()\n%s%s_network = %s%s_network.
 ↪fit(x1_%s_train_balance,y1_%s_train_balance)\nacc_%s_%s_network =␣
 ↪%s%s_network.score(x1_test,y1_%s_test)\nRMSE_%s_%s_network =␣
 ↪RMSE(%s%s_network,x1_test,y1_%s_test)'''
             %(i,j,i,i,j,i,j,j,j,i,j,i,j,j,i,j,i,j,j),globals())

    exec('''acc_%s_network=(acc_%s_1_network+acc_%s_2_network+
                            acc_%s_3_network+acc_%s_4_network)/4\n
            RMSE_%s_network=(RMSE_%s_1_network+RMSE_%s_2_network+
                             RMSE_%s_3_network+RMSE_%s_4_network)/4\n
            print('network',i,',acc:',acc_%s_network,'RMSE:',RMSE_%s_network)'''
         %(i,i,i,i,i,i,i,i,i,i,i),globals())
```

```
voice KNN ,acc: 0.4937902483900644 RMSE: 2.567178353722658
voice XGB ,acc: 0.5305887764489421 RMSE: 2.3069573420811325
voice LR ,acc: 0.5545078196872125 RMSE: 2.3475730959531393
voice SVC ,acc: 0.561867525298988 RMSE: 2.3804584224601575
voice LSVC ,acc: 0.5579576816927323 RMSE: 2.3920266574031235
voice RFC ,acc: 0.5197792088316467 RMSE: 2.425890017114177
voice GNB ,acc: 0.027598896044158234 RMSE: 6.498227465239129
voice P ,acc: 0.5269089236430543 RMSE: 2.3563782626914955
voice SGD ,acc: 0.5326586936522539 RMSE: 2.3277895972083202
voice DTC ,acc: 0.45722171113155474 RMSE: 2.670698547753191
voice GBDT ,acc: 0.5478380864765409 RMSE: 2.359655976656089
voice ABC ,acc: 0.5402483900643974 RMSE: 2.379187851876166
voice MLP ,acc: 0.5211591536338547 RMSE: 2.3054529697948283
network KNN ,acc: 0.4937902483900644 RMSE: 2.567178353722658
network XGB ,acc: 0.5305887764489421 RMSE: 2.3069573420811325
network LR ,acc: 0.5545078196872125 RMSE: 2.3475730959531393
network SVC ,acc: 0.561867525298988 RMSE: 2.3804584224601575
network LSVC ,acc: 0.5579576816927323 RMSE: 2.3920266574031235
network RFC ,acc: 0.5220791168353266 RMSE: 2.385902812167356
network GNB ,acc: 0.027598896044158234 RMSE: 6.498227465239129
network P ,acc: 0.5269089236430543 RMSE: 2.3563782626914955
network SGD ,acc: 0.5181692732290708 RMSE: 2.3067617841377834
```

```
network DTC ,acc: 0.4567617295308188 RMSE: 2.6813579544462693
network GBDT ,acc: 0.547608095676173 RMSE: 2.357009361991244
network ABC ,acc: 0.5402483900643974 RMSE: 2.379187851876166
network MLP ,acc: 0.5239190432382704 RMSE: 2.351665402802151
```

[29]:
```
acc_voice=[]
RMSE_voice=[]
for i in models:
    acc_voice.append(eval('acc_%s_voice'%i))
    RMSE_voice.append(eval('RMSE_%s_voice'%i))

accs = pd.DataFrame({
    'Model_voice': models,
    'acc_voice': acc_voice,
    'RMSE_voice': RMSE_voice})
accs.sort_values(by='acc_voice', ascending=False)
accs.to_latex()
```

[29]:

	Model_voice	acc_voice	RMSE_voice
3	SVC	0.561868	2.380458
4	LSVC	0.557958	2.392027
2	LR	0.554508	2.347573
10	GBDT	0.547838	2.359656
11	ABC	0.540248	2.379188
8	SGD	0.532659	2.327790
1	XGB	0.530589	2.306957
7	P	0.526909	2.356378
12	MLP	0.521159	2.305453
5	RFC	0.519779	2.425890
0	KNN	0.493790	2.567178
9	DTC	0.457222	2.670699
6	GNB	0.027599	6.498227

[29]:
```
'\\begin{tabular}{llrr}\n\\toprule\n{} & Model\\_voice &  acc\\_voice &
RMSE\\_voice \\\\\n\\midrule\n0  &         KNN &    0.493790 &    2.567178
\\\\\n1  &         XGB &    0.530589 &    2.306957 \\\\\n2  &          LR &
0.554508 &    2.347573 \\\\\n3  &         SVC &    0.561868 &    2.380458 \\\\\n4
```

```
&         LSVC &    0.557958 &     2.392027 \\\\\n5  &          RFC &    0.519779 &
2.425890 \\\\\n6  &          GNB &    0.027599 &     6.498227 \\\\\n7  &
P &    0.526909 &     2.356378 \\\\\n8  &          SGD &    0.532659 &     2.327790
\\\\\n9  &          DTC &    0.457222 &     2.670699 \\\\\n10 &         GBDT &
0.547838 &     2.359656 \\\\\n11 &          ABC &    0.540248 &     2.379188
\\\\\n12 &          MLP &    0.521159 &     2.305453
\\\\\n\\bottomrule\n\\end{tabular}\n'
```

[30]:
```
acc_network=[]
RMSE_network=[]
for i in models:
    acc_network.append(eval('acc_%s_network'%i))
    RMSE_network.append(eval('RMSE_%s_network'%i))

accs = pd.DataFrame({
    'Model_network': models,
    'acc_network': acc_network,
    'RMSE_network': RMSE_network})
accs.sort_values(by='acc_network', ascending=False)
accs.to_latex()
```

[30]:

	Model_network	acc_network	RMSE_network
3	SVC	0.561868	2.380458
4	LSVC	0.557958	2.392027
2	LR	0.554508	2.347573
10	GBDT	0.547608	2.357009
11	ABC	0.540248	2.379188
1	XGB	0.530589	2.306957
7	P	0.526909	2.356378
12	MLP	0.523919	2.351665
5	RFC	0.522079	2.385903
8	SGD	0.518169	2.306762
0	KNN	0.493790	2.567178
9	DTC	0.456762	2.681358
6	GNB	0.027599	6.498227

[30]:
```
'\\begin{tabular}{llrr}\n\\toprule\n{} & Model\\_network &  acc\\_network &
RMSE\\_network \\\\\n\\midrule\n0  &           KNN &     0.493790 &
2.567178 \\\\\n1  &           XGB &     0.530589 &      2.306957 \\\\\n2  &
LR &     0.554508 &      2.347573 \\\\\n3  &           SVC &     0.561868 &
2.380458 \\\\\n4  &          LSVC &     0.557958 &      2.392027 \\\\\n5  &
RFC &     0.522079 &      2.385903 \\\\\n6  &           GNB &     0.027599 &
6.498227 \\\\\n7  &             P &     0.526909 &      2.356378 \\\\\n8  &
SGD &     0.518169 &      2.306762 \\\\\n9  &           DTC &     0.456762 &
2.681358 \\\\\n10 &          GBDT &     0.547608 &      2.357009 \\\\\n11 &
ABC &     0.540248 &      2.379188 \\\\\n12 &           MLP &     0.523919 &
2.351665 \\\\\n\\bottomrule\n\\end{tabular}\n'
```

7 模型调参

[31]:
```
xgb_param_list = []
from sklearn.model_selection import GridSearchCV
max_depth = [3, 4, 5, 6, 7]
n_estimators = [25,30,35,40]
p = {         'max_depth': max_depth,
              'n_estimators': n_estimators}

clf = XGBClassifier()



for i in [1, 2, 3, 4]:
    model = GridSearchCV(clf, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x1_%s_train_balance,y1_%s_train_balance)\nXGB%s_voice =␣
 ↪model.best_estimator_\nxgb_param_list.append(model.best_params_)'''%(i, i,␣
 ↪i),globals())
for j in [1, 2, 3, 4]:
    model = GridSearchCV(clf, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
```

```
    exec('''model.fit(x2_%s_train_balance,y2_%s_train_balance)\nXGB%s_network =␣
 ↪model.best_estimator_\nxgb_param_list.append(model.best_params_)'''%(j, j,␣
 ↪j),globals())
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Fitting 3 folds for each of 20 candidates, totalling 60 fits
```

[ ]:
```
lr_param_list = []
solver = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
multi_class = ['ovr', 'multinomial']
p = {
    'solver': solver,
    'multi_class': multi_class,
    'C': [0.01, 0.1, 1, 10]
}
lr = LR(penalty='l2')

for i in [1, 2, 3, 4]:
    model = GridSearchCV(lr, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x1_%s_train_balance,y1_%s_train_balance)\nLR%s_voice =␣
 ↪model.best_estimator_\nlr_param_list.append(model.best_params_)'''%(i, i,␣
 ↪i),globals())

for j in [1, 2, 3, 4]:
    model = GridSearchCV(lr, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x2_%s_train_balance,y2_%s_train_balance)\nLR%s_network =␣
 ↪model.best_estimator_\nlr_param_list.append(model.best_params_)'''%(j, j,␣
 ↪j),globals())
```

[33]:
```
rfc_param_list = []
n_estimators = range(10, 71, 10)
max_depth = [6, 7, 8]
p = {
    'n_estimators': n_estimators,
    'max_depth': max_depth
}
rfc = RFC()

for i in [1, 2, 3, 4]:
    model = GridSearchCV(rfc, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x1_%s_train_balance,y1_%s_train_balance)\nRFC%s_voice =␣
 ↪model.best_estimator_\nrfc_param_list.append(model.best_params_)'''%(i, i,␣
 ↪i),globals())

for j in [1, 2, 3, 4]:
    model = GridSearchCV(rfc, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x2_%s_train_balance,y2_%s_train_balance)\nRFC%s_network =␣
 ↪model.best_estimator_\nrfc_param_list.append(model.best_params_)'''%(j, j,␣
 ↪j),globals())
```

```
Fitting 3 folds for each of 21 candidates, totalling 63 fits
Fitting 3 folds for each of 21 candidates, totalling 63 fits
Fitting 3 folds for each of 21 candidates, totalling 63 fits
Fitting 3 folds for each of 21 candidates, totalling 63 fits
Fitting 3 folds for each of 21 candidates, totalling 63 fits
Fitting 3 folds for each of 21 candidates, totalling 63 fits
Fitting 3 folds for each of 21 candidates, totalling 63 fits
Fitting 3 folds for each of 21 candidates, totalling 63 fits
```

[34]:
```
svc_param_list = []
c = [0.01, 0.1, 1, 10]
gamma = [0.01, 0.1, 1, 10]
p ={
    'C':c,
```

```
    'gamma': gamma,
}
svc = SVC()

for i in [1, 2, 3, 4]:
    model = GridSearchCV(svc, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x1_%s_train_balance,y1_%s_train_balance)\nSVC%s_voice =␣
 ↪model.best_estimator_\nsvc_param_list.append(model.best_params_)'''%(i, i,␣
 ↪i),globals())

for j in [1, 2, 3, 4]:
    model = GridSearchCV(svc, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x2_%s_train_balance,y2_%s_train_balance)\nSVC%s_network =␣
 ↪model.best_estimator_\nsvc_param_list.append(model.best_params_)'''%(j, j,␣
 ↪j),globals())
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

[35]:
```
gbdt_param_list = []
n_estimators = [30, 40, 50]
max_depth = [5, 6, 7, 8]
p = {
    'n_estimators': n_estimators,
    'max_depth': max_depth,
}
gbdt = GBDT()

for i in [1, 2, 3, 4]:
```

```
    model = GridSearchCV(gbdt, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x1_%s_train_balance,y1_%s_train_balance)\nGBDT%s_voice =␣
 ↪model.best_estimator_\ngbdt_param_list.append(model.best_params_)'''%(i, i,␣
 ↪i),globals())

for j in [1, 2, 3, 4]:
    model = GridSearchCV(gbdt, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x2_%s_train_balance,y2_%s_train_balance)\nGBDT%s_network␣
 ↪= model.best_estimator_\ngbdt_param_list.append(model.best_params_)'''%(j,␣
 ↪j, j),globals())
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

[36]:
```
abc_param_list = []
learning_rate = [0.1, 0.2, 0.3]
algorithm = ['SAMME', 'SAMME.R']
n_estimators = [30, 40, 50, 60]
p = {
    'learning_rate': learning_rate,
    'algorithm': algorithm,
    'n_estimators': n_estimators
}
abc = ABC()

for i in [1, 2, 3, 4]:
    model = GridSearchCV(abc, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
```

```
    exec('''model.fit(x1_%s_train_balance,y1_%s_train_balance)\nABC%s_voice =␣
 ↪model.best_estimator_\nabc_param_list.append(model.best_params_)'''%(i, i,␣
 ↪i),globals())

for j in [1, 2, 3, 4]:
    model = GridSearchCV(abc, p, cv=3,␣
 ↪scoring='neg_mean_squared_error',verbose=1, n_jobs=-1)
    exec('''model.fit(x2_%s_train_balance,y2_%s_train_balance)\nABC%s_network =␣
 ↪model.best_estimator_\nabc_param_list.append(model.best_params_)'''%(j, j,␣
 ↪j),globals())
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

[37]:
```
xgb_param_frame = pd.DataFrame(xgb_param_list)
lr_param_frame = pd.DataFrame(lr_param_list)
rfc_param_frame = pd.DataFrame(rfc_param_list)
svc_param_frame = pd.DataFrame(svc_param_list)
gbdt_param_frame = pd.DataFrame(gbdt_param_list)
abc_param_frame = pd.DataFrame(abc_param_list)

xgb_param_frame.index = ['xgb1_voice', 'xgb2_voice', 'xgb3_voice',␣
 ↪'xgb4_voice', 'xgb1_network', 'xgb2_network', 'xgb3_network', 'xgb4_network']
lr_param_frame.index = ['lr1_voice', 'lr2_voice', 'lr3_voice', 'lr4_voice',␣
 ↪'lr1_network', 'lr2_network', 'lr3_network', 'lr4_network']
rfc_param_frame.index = ['rfc1_voice', 'rfc2_voice', 'rfc3_voice',␣
 ↪'rfc4_voice', 'rfc1_network', 'rfc2_network', 'rfc3_network', 'rfc4_network']
svc_param_frame.index = ['svc1_voice', 'svc2_voice', 'svc3_voice',␣
 ↪'svc4_voice', 'svc1_network', 'svc2_network', 'svc3_network', 'svc4_network']
```

```
gbdt_param_frame.index = ['gbdt1_voice', 'gbdt2_voice', 'gbdt3_voice',␣
 ↪'gbdt4_voice', 'gbdt1_network', 'gbdt2_network', 'gbdt3_network',␣
 ↪'gbdt4_network']
abc_param_frame.index = ['abc1_voice', 'abc2_voice', 'abc3_voice',␣
 ↪'abc4_voice', 'abc1_network', 'abc2_network', 'abc3_network', 'abc4_network']
```

[38]:
```
xgb_param_frame
xgb_param_frame.to_latex()
```

[38]:
```
              max_depth  n_estimators
xgb1_voice            5            25
xgb2_voice            5            25
xgb3_voice            5            40
xgb4_voice            5            40
xgb1_network          6            35
xgb2_network          6            35
xgb3_network          6            35
xgb4_network          5            40
```

[38]:
```
'\\begin{tabular}{lrr}\n\\toprule\n{} &  max\\_depth &  n\\_estimators
\\\\\n\\midrule\nxgb1\\_voice   &          5 &            25 \\\\\nxgb2\\_voice
&          5 &            25 \\\\\nxgb3\\_voice   &          5 &            40
\\\\\nxgb4\\_voice   &          5 &            40 \\\\\nxgb1\\_network &
6 &            35 \\\\\nxgb2\\_network &          6 &            35
\\\\\nxgb3\\_network &          6 &            35 \\\\\nxgb4\\_network &
5 &            40 \\\\\n\\bottomrule\n\\end{tabular}\n'
```

[39]:
```
lr_param_frame
lr_param_frame.to_latex()
```

[39]:
```
                C  multi_class     solver
lr1_voice    10.0          ovr       saga
lr2_voice    10.0          ovr      lbfgs
lr3_voice     1.0  multinomial  newton-cg
lr4_voice    10.0  multinomial  newton-cg
lr1_network   1.0  multinomial  newton-cg
lr2_network   0.1  multinomial        sag
lr3_network  10.0  multinomial        sag
```

```
lr4_network  10.0  multinomial      lbfgs
```

[39]:
```
'\\begin{tabular}{lrll}\n\\toprule\n{} &     C &  multi\\_class &     solver
\\\\\n\\midrule\nlr1\\_voice   &  10.0 &          ovr &       saga
\\\\\nlr2\\_voice   &  10.0 &          ovr &      lbfgs \\\\\nlr3\\_voice   &
1.0 &  multinomial &  newton-cg \\\\\nlr4\\_voice   &  10.0 &  multinomial &
newton-cg \\\\\nlr1\\_network &   1.0 &  multinomial &  newton-cg
\\\\\nlr2\\_network &   0.1 &  multinomial &        sag \\\\\nlr3\\_network &
10.0 &  multinomial &        sag \\\\\nlr4\\_network &  10.0 &  multinomial &
lbfgs \\\\\n\\bottomrule\n\\end{tabular}\n'
```

[40]:
```
rfc_param_frame
rfc_param_frame.to_latex()
```

[40]:
```
              max_depth  n_estimators
rfc1_voice            8            40
rfc2_voice            8            20
rfc3_voice            7            10
rfc4_voice            8            10
rfc1_network          8            20
rfc2_network          8            60
rfc3_network          8            40
rfc4_network          8            50
```

[40]:
```
'\\begin{tabular}{lrr}\n\\toprule\n{} &  max\\_depth &  n\\_estimators
\\\\\n\\midrule\nrfc1\\_voice   &          8 &            40 \\\\\nrfc2\\_voice
&          8 &            20 \\\\\nrfc3\\_voice   &          7 &            10
\\\\\nrfc4\\_voice   &          8 &            10 \\\\\nrfc1\\_network &
8 &            20 \\\\\nrfc2\\_network &          8 &            60
\\\\\nrfc3\\_network &          8 &            40 \\\\\nrfc4\\_network &
8 &            50 \\\\\n\\bottomrule\n\\end{tabular}\n'
```

[41]:
```
svc_param_frame
svc_param_frame.to_latex()
```

[41]:
```
               C  gamma
svc1_voice    10   0.10
svc2_voice    10   0.10
```

```
svc3_voice     1   0.10
svc4_voice    10   0.10
svc1_network  10   0.10
svc2_network  10   0.01
svc3_network  10   0.01
svc4_network  10   0.01
```

[41]:
```
'\\begin{tabular}{lrr}\n\\toprule\n{} &   C &  gamma
\\\\\n\\midrule\nsvc1\\_voice   &  10 &   0.10 \\\\\nsvc2\\_voice   &  10 &
0.10 \\\\\nsvc3\\_voice   &   1 &   0.10 \\\\\nsvc4\\_voice   &  10 &   0.10
\\\\\nsvc1\\_network &  10 &   0.10 \\\\\nsvc2\\_network &  10 &   0.01
\\\\\nsvc3\\_network &  10 &   0.01 \\\\\nsvc4\\_network &  10 &   0.01
\\\\\n\\bottomrule\n\\end{tabular}\n'
```

[42]:
```
gbdt_param_frame
gbdt_param_frame.to_latex()
```

[42]:
```
               max_depth  n_estimators
gbdt1_voice            5            40
gbdt2_voice            5            40
gbdt3_voice            5            50
gbdt4_voice            6            50
gbdt1_network          6            50
gbdt2_network          6            40
gbdt3_network          5            50
gbdt4_network          6            50
```

[42]:
```
'\\begin{tabular}{lrr}\n\\toprule\n{} &  max\\_depth &  n\\_estimators
\\\\\n\\midrule\ngbdt1\\_voice   &          5 &            40
\\\\\ngbdt2\\_voice   &          5 &            40 \\\\\ngbdt3\\_voice   &
5 &            50 \\\\\ngbdt4\\_voice   &          6 &            50
\\\\\ngbdt1\\_network &          6 &            50 \\\\\ngbdt2\\_network &
6 &            40 \\\\\ngbdt3\\_network &          5 &            50
\\\\\ngbdt4\\_network &          6 &            50
\\\\\n\\bottomrule\n\\end{tabular}\n'
```

[43]:
```
abc_param_frame
abc_param_frame.to_latex()
```

[43]:

	algorithm	learning_rate	n_estimators
abc1_voice	SAMME.R	0.3	60
abc2_voice	SAMME.R	0.3	60
abc3_voice	SAMME.R	0.3	60
abc4_voice	SAMME.R	0.3	60
abc1_network	SAMME.R	0.3	60
abc2_network	SAMME.R	0.3	60
abc3_network	SAMME.R	0.3	60
abc4_network	SAMME.R	0.3	60

[43]:
```
'\\begin{tabular}{llrr}\n\\toprule\n{} & algorithm &  learning\\_rate &
n\\_estimators \\\\\n\\midrule\nabc1\\_voice   &   SAMME.R &            0.3 &
60 \\\\\nabc2\\_voice   &   SAMME.R &            0.3 &            60
\\\\\nabc3\\_voice   &   SAMME.R &            0.3 &            60
\\\\\nabc4\\_voice   &   SAMME.R &            0.3 &            60
\\\\\nabc1\\_network &   SAMME.R &            0.3 &            60
\\\\\nabc2\\_network &   SAMME.R &            0.3 &            60
\\\\\nabc3\\_network &   SAMME.R &            0.3 &            60
\\\\\nabc4\\_network &   SAMME.R &            0.3 &            60
\\\\\n\\bottomrule\n\\end{tabular}\n'
```

8 模型融合

[ ]:
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

acc_stacking = []
RMSE_stacking = []


for j in [1,2,3,4]:
    stacking = StackingClassifier(estimators=[('ABC%s'%j,eval('ABC%s_voice'%j)),
                                              ␣
 ↪('tree%s'%j,eval('RFC%s_voice'%j)),
                                               ('svc%s'%j,eval('LR%s_voice'%j)),
                                               ('XGB%s'%j,eval('XGB%s_voice'%j)),
                                               ('SVC%s'%j,eval('SVC%s_voice'%j)),
```

```
                                                    ␣
 ↪('GBDT%s'%j,eval('GBDT%s_voice'%j))],
                                    final_estimator=LogisticRegression())
    stacking.fit(eval('x1_%s_train_balance'%j),eval('y1_%s_train_balance'%j))
    acc_stacking.append(stacking.score(x1_test,eval('y1_%s_test'%j)))
    RMSE_stacking.append(RMSE(stacking,x1_test,eval('y1_%s_test'%j)))
    print(stacking.score(x1_test,eval('y1_%s_test'%j)))
    print(RMSE(stacking,x1_test,eval('y1_%s_test'%j)))
    exec('stacking_voice_%s=stacking'%j)

for j in [1,2,3,4]:
    stacking =␣
 ↪StackingClassifier(estimators=[('ABC%s'%j,eval('ABC%s_network'%j)),
                                                    ␣
 ↪('tree%s'%j,eval('RFC%s_network'%j)),
                                                    ␣
 ↪('svc%s'%j,eval('LR%s_network'%j)),
                                                    ␣
 ↪('XGB%s'%j,eval('XGB%s_network'%j)),
                                                    ␣
 ↪('SVC%s'%j,eval('SVC%s_network'%j)),
                                                    ␣
 ↪('GBDT%s'%j,eval('GBDT%s_network'%j))],
                                    final_estimator=LogisticRegression())
    stacking.fit(eval('x2_%s_train_balance'%j),eval('y2_%s_train_balance'%j))
    acc_stacking.append(stacking.score(x2_test,eval('y2_%s_test'%j)))
    RMSE_stacking.append(RMSE(stacking,x2_test,eval('y2_%s_test'%j)))
    print(stacking.score(x2_test,eval('y2_%s_test'%j)))
    print(RMSE(stacking,x2_test,eval('y2_%s_test'%j)))
    exec('stacking_network_%s=stacking'%j)
```

[45]:
```
names=[]
for i in columns_voice_y:
    names.append(i)
for j in columns_network_y:
    names.append(j)
```

```
acc_stackings = pd.DataFrame({
    'Model_stacking': names,
    'acc_stacking': acc_stacking,
    'RMSE_stacking': RMSE_stacking})
acc_stackings.sort_values(by='acc_stacking', ascending=False)
acc_stackings.to_latex()
```

[45]:

	Model_stacking	acc_stacking	RMSE_stacking
0	语音通话整体满意度	0.592456	2.332326
2	语音通话清晰度	0.571297	2.286716
3	语音通话稳定性	0.539098	2.467638
1	网络覆盖与信号强度	0.521619	2.581692
4	手机上网整体满意度	0.442308	2.984051
5	网络覆盖与信号强度	0.415242	2.691524
7	手机上网稳定性	0.412393	2.667067
6	手机上网速度	0.401709	2.667067

[45]:
```
'\\begin{tabular}{llrr}\n\\toprule\n{} & Model\\_stacking &  acc\\_stacking &
RMSE\\_stacking \\\\\n\\midrule\n0 &      语音通话整体满意度 &      0.592456 &
2.332326 \\\\\n1 &      网络覆盖与信号强度 &      0.521619 &      2.581692␣
↪\\\\\n2 &
语音通话清晰度 &      0.571297 &      2.286716 \\\\\n3 &        语音通话稳定性 &
0.539098 &      2.467638 \\\\\n4 &      手机上网整体满意度 &      0.442308 &
2.984051 \\\\\n5 &      网络覆盖与信号强度 &      0.415242 &      2.691524␣
↪\\\\\n6 &
手机上网速度 &      0.401709 &      2.667067 \\\\\n7 &        手机上网稳定性 & ␣
↪    0.412393
&      2.667067 \\\\\n\\bottomrule\n\\end{tabular}\n'
```

9 模型预测

[46]:
```
#train_voice.iloc[:,4:].head(1)
test_voice=test_voice.drop(columns=['用户 id','用户描述','用户描述.1','终端品牌类
型','语音通话-时长（分钟）',
                                    '当月 ARPU','当月 MOU','前 3 月 MOU','GPRS
总流量（KB）','是否不限量套餐到达用户',
```

```
                                    '是否投诉','性别','4\\5G 用户'],axis=1)


#train_network.iloc[:,4:].head(1)
test_network=test_network.drop(test_network.columns[23],axis=1)
test_network=test_network.drop(columns=['用户 id','注明内容','注明内容.1','注明内
容.2','终端品牌类型','语音通话-时长（分钟）','当月 ARPU','当月 MOU','前 3 月
MOU','GPRS 总流量（KB）','是否投诉','4\\5G 用户','当月 MOU',
                            '终端品牌类型','GPRS-国内漫游-流量（KB）','外省流量占
比','前 3 月 ARPU','是否关怀用户','省际漫游-时长（分钟）','是否 4G 网络客户（本地剔
除物联网）',
                            '注明内容.4','注明内容.3','学习强国','学习强国.1','外省
语音占比','是否遇到网络问题'],axis=1)

#train_network.iloc[:,4:].head(1)
test_network=test_network[[i for i in train_network.iloc[:,4:].columns]]
```

[47]:
```
test_voice['是否遇到过网络问题']=test_voice['是否遇到过网络问题'].map({2:0,1:1})
test_voice['终端品牌']=test_voice['终端品牌'].map({'苹果':1,'华为':2,'小米科技':
↪3,'步步高':4,'欧珀':5,'三星':6,'realme':6,0:6,
                                    '万普拉斯':6,'锤子':6,'万普':6,'联通':6,
                                    '中邮通信':6,'其他':6,'中国移动':6,'中兴':6,
                                    '魅族':6,'黑鲨':6,'联想':6,'智🄵互联':6,
                                    '海信':6,'摩托罗拉':6,'乐视移动':6,'天宇朗
通':6,
                                    '奇酷':6,'上海中兴易联通讯股份有限公司':6,'诺
基亚':6})

test_voice['是否关怀用户']=test_voice['是否关怀用户'].map({'否':0,'是':1})
test_voice['是否 4G 网络客户（本地剔除物联网）']=test_voice['是否 4G 网络客户（本地
剔除物联网）'].map({'否':0,'是':1})
test_voice['是否 5G 网络客户']=test_voice['是否 5G 网络客户'].map({'否':0,'是':
↪1})
test_voice['客户星级标识']=test_voice['客户星级标识'].map({'三星':1, '银卡':2,␣
↪'二星':3, '一星':4, '金卡':5, '白金卡':6, '未评级':7, '准星':8, '钻石卡':9})
```

```
test_voice=test_voice.fillna(0)
index = [i for i in test_voice.columns]
index.remove('客户星级标识')
index.remove('终端品牌')
test_voice[index]=test_voice[index].applymap(lambda x:0 if x<=0 else 1)

x1_temp = test_voice
x1_test = pd.DataFrame([])
for i in x1_temp.columns:
    temp=pd.get_dummies(x1_temp[i])
    for j in range(len(temp.columns)):
        temp1 = pd.DataFrame(temp.iloc[:,[j]].values,columns=['%s_%s'%(i,j)])
        x1_test=pd.concat([x1_test,temp1],axis =1)

x1_test['客户星级标识 _9']=0
```

[48]:
```
test_network['终端品牌']=test_network['终端品牌'].map({'苹果':1,'华为':2,'小米科技':3,'步步高':4,'欧珀':5,'三星':6,'realme':6,0:6,
                                    '万普拉斯':6,'锤子':6,'万普':6,'联通':6,
                                    '中邮通信':6,'其他':6,'中国移动':6,'中兴':6,
                                    '魅族':6,'黑鲨':6,'联想':6,'智🄵互联':6,
                                    '海信':6,'摩托罗拉':6,'乐视移动':6,'天宇朗通':6,
                                    '奇酷':6,'上海中兴易联通讯股份有限公司':6,'诺基亚':6,'华硕':6,
                                    '甄十信息科技（上海）有限公司':6,'维图':6, '捷开通讯科技':6, 'TD':6,
                                    '中国电信':6, '北京珠穆朗玛移动通信有限公司':6, '欧博信':6,'飞利浦':6, '酷比':6, '索尼爱立信':6, '金立':6})


test_network['是否不限量套餐到达用户']=test_network['是否不限量套餐到达用户'].map({'否':0,'是':1})
test_network['是否 5G 网络客户']=test_network['是否 5G 网络客户'].map({'否':0,'是':1})
```

```
test_network['客户星级标识']=test_network['客户星级标识'].map({'三星':1, '银卡':
↪2, '二星':3, '一星':4, '金卡':5, '白金卡':6, '未评级':7, '准星':8, '钻石卡':9})
train_network['性别']=train_network['性别'].map({'男':1, '女':2, '性别不详':3})

test_network=test_network.fillna(0)
index = [i for i in test_network.columns]
index.remove('客户星级标识')
index.remove('终端品牌')
index.remove('性别')
test_network[index]=test_network[index].applymap(lambda x:0 if x<=0 else 1)

x2_temp = test_network
x2_test = pd.DataFrame([])
for i in x2_temp.columns:
    temp=pd.get_dummies(x2_temp[i])
    for j in range(len(temp.columns)):
        temp2 = pd.DataFrame(temp.iloc[:,[j]].values,columns=['%s_%s'%(i,j)])
        x2_test=pd.concat([x2_test,temp2],axis =1)
```

[49]:
```
x1.shape,x1_test.shape
x2.shape,x2_test.shape
```

[49]: ((5433, 75), (2599, 75))

[49]: ((7020, 136), (1610, 136))

[50]:
```
for i in [1,2,3,4]:
    for j in ['voice']:
        exec('predict_%s_%s = stacking_%s_%s.
↪predict(x1_test)'%(j,i,j,i),globals())

for i in [1,2,3,4]:
    for j in ['network']:
        exec('predict_%s_%s = stacking_%s_%s.
↪predict(x2_test)'%(j,i,j,i),globals())
```

[51]:
```
voice = pd.DataFrame([])
network = pd.DataFrame([])

for i in [1,2,3,4]:
    voice = pd.concat([voice,pd.DataFrame(eval('predict_voice_%s'%i))],axis=1)
    network = pd.concat([network,pd.
 ↪DataFrame(eval('predict_network_%s'%i))],axis=1)

voice=voice.applymap(lambda x:x+1)
network=network.applymap(lambda x:x+1)

submission_voice.iloc[:,1:] = voice.values
submission_network.iloc[:,1:] = network.values



writer = pd.ExcelWriter("result.xlsx")
submission_voice.to_excel(writer, sheet_name="语音",index=False)
submission_network.to_excel(writer, sheet_name="上网",index=False)
writer.save()
writer.close()
```