武漢理工大學

数学建模暑期培训论文

第 5 题

基于数据上采样和贝叶斯优化的 xgboost 集成决策树红酒质量预测模型

第 26 组

姓名	方向
许鸢飞（组长）	编程
尹可汗	编程
李想	建模

2020 年 8 月 24 日

摘要

本文根据红酒的理化性质和数据特征，建立了基于数据上采样技术和贝叶斯优化的xgboost集成决策树红酒质量预测模型，实现了对红酒数据集质量的合理预测。

针对问题一，本文建立了以**决策树分类原理**为基础的**xgboost多分类器红酒预测模型**。首先，本模型对数据集进行了预处理工作，对于**数据缺失**的部分数据，我们采用均值进行填充，然后利用数据**归一化**消除了量纲之间的差异，并画出了各项指标的**分布特征**，然后将处理完成的数据集，通过xgboost构建多分类器系统，利用**集成学习和梯度提升**的思想，通过**最佳分割点搜索算法**，最终得到预测效果最好的多分类器系统。**该模型通过交叉验证在训练集上的预测率为62.8%**。

针对问题二，**本文建立了基于皮尔逊相关性和xgboost叶节点权重的红酒重要指标提取和质量评估模型**。首先本文对问题一中的模型结构和属性进行提取，从而得到该模型在xgboost重要性计算中的多组排序方式，并利用归一化后的指标对其进行**皮尔逊相关性检验**，从而得到各个指标之间还有质量之间的相关度，并结合参考文献得到重要程度较高的前3个量为**密度，酒精，剩余糖分，所占的影响比例分别为11.04%,10.59%,9.72%，**同时对待测数据集进行了利用建立的模型进行了预测，并通过**灵敏度分析**可以得到**在合适的范围内，酒精浓度的增大，红酒质量能得到提升，剩余糖分过多会影响品质。**

针对问题三，本文利用**数据上采样和贝叶斯优化器**对xgboost模型从数据处理和模型参数进行了优化。首先我们通过分析当前数据集中的数据分布状况，可以看出等级为3，4，8，9的数量只占到7.6%, 存在明显的**数据分布不均衡**问题，于是本文利用数据上采样的**SMOTE算法**，对当前少数类的样本以**k近邻搜索和欧氏距离差异**评价的方式构造出相同类型的新数据来填充少数类样本从而使得样本分布均衡化，然后利用**贝叶斯优化器**方法，利用先验概率与后验概率的关系对模型的树结构进行调整。**在上述优化方法下该模型的准确率达到了68.4%,比问题一提升了5.6%，**并对各参数进行了**灵敏度分析**，从而得出了各超参数对于模型准确性的影响。

本文的优点：1. 本文利用集xgboost集成学习的方法，建立多决策树分类器系统，有效防止模型预测出现过拟合。2. 采用数据上采样方法合理构造新数据以扩充数据集保证样本类别的均衡分布，提高预测精度。

关键词：　数据上采样　　贝叶斯优化　　xgboost　　集成学习

目录

一、 问题重述

1.1 问题背景

改革开放 40 年来，中国葡萄酒行业实现了跨越式发展，中国即将超越英国成为仅次于美国的全球葡萄酒消费第二大市场。中国的年轻人已经将引用葡萄酒当做是一种社会时尚, 葡萄酒正在征服年轻人的味蕾，并成为一种身份象征。

然而，市场上红酒的品质参差不齐，传统上葡萄酒的品质坚定大多依靠经验丰富的品酒师进行打分来判定，品酒师主要依据葡萄酒的外观，香气，口感和整体感觉等指标判断葡萄酒的品质. 这种判定方式不仅会消耗大量人工成本和时间成本，而且品酒师的个人喜好也会对葡萄酒品质评价造成影响。在红酒种类越来越丰富的今天，这种评价方式显然难以满足市场的需求，葡萄酒品质评价工作迫切需要一种客观的，高效的鉴定方法. 图1是名声享誉海外的法国波尔多葡萄酒的分级图:

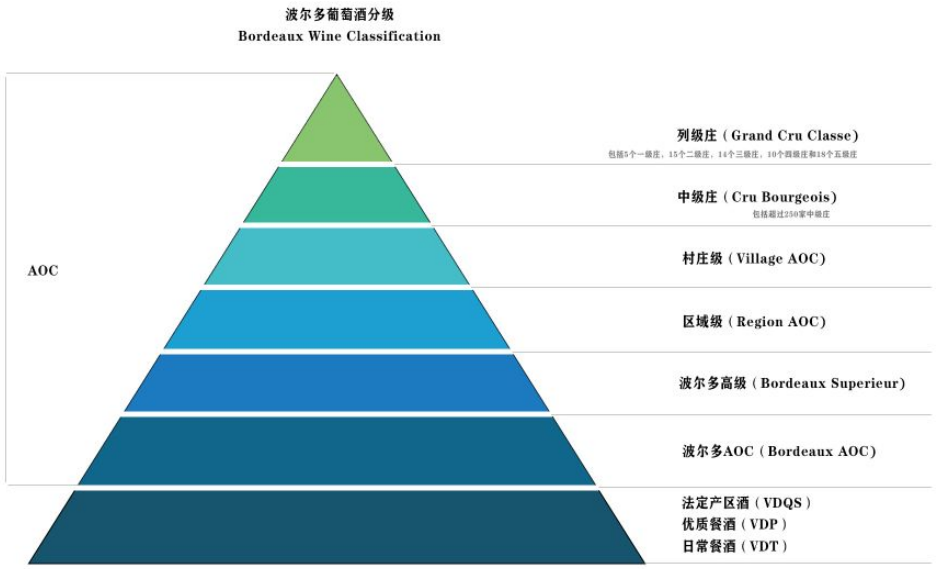

图 **1**　波尔多葡萄酒分级示意图

1.2 待解决的问题

确定葡萄酒质量时一般是通过邀请一批有经验的品酒师对葡萄酒样品进行点评完成的。每个品酒师会对葡萄酒的外观，香气，口感和整体感觉四个方面进行评价，然后给出一个综合评分。这种评价方法的好处是有资质品酒师能凭借其专业水平区分葡萄酒间质量的差异，但缺点是这种评价方式效率低下，花费不菲且受到强烈的个人偏好的影响。葡萄酒质量评定需要一种基于算法的，客观的评价方法. 附件一中给出了 3898 份葡

萄酒样品的理化指标，题目要求我们根据这些特征建立分类模型，对红酒品质进行分类和预测，讨论以下问题：

1. 对附件一中数据进行预处理，利用预处理后的数据建立合理的数学模型.

2. 利用已建立的数学模型对待预测的红酒样品进行品质预测，并给出准确率分析，基于已建立的模型，运用新的模型找出影响红酒品质的前三中理化指标.

3. 对建立的模型进行改进以提高分类的精确率，并展示优化效果.

二、 模型假设

1. 不考虑红酒样品非给定的 11 个理化指标对品质的影响，即红酒品质仅受题目所提供的理化指标数据的影响.
2. xgboost 模型的正确率和模型的超参数服从高斯过程，即具有特点的平滑性. 据此在改进模型时我们根据贝叶斯优化方法对模型的超参数进行调优.

三、 符号说明

符号	含义
X_{ij}	第 j 份红酒样品在第 i 项理化指标上的数据值
X'_{ij}	归一化处理后的数据
$\overline{X}_i$	第 i 项理化指标的平均值
δ_i	第 i 项理化指标数据的标准差
P_{ij}	第 i 个样本在第 j 类理化指标上的预测概率
y_{ij}	第 i 个样本在第 j 类理化指标上的真实得分
T	叶子节点的个数
ω_j	叶子节点在 j 类理化指标上的分数

注：表中未说明的符号以首次出现处为准

四、 问题一模型的建立与求解

4.1 数据预处理

数据集中共有 3898 个红酒样品的数据，包括 11 项红酒理化数据和一项评分数据，各项理化数据中有 31 个数据缺失，需要注意的是，理化数据的指标由于量纲不同，需要进行归一化处理。

4.1.1 缺失值处理

对于数据中存在对的确实现象，本文采用均值替换法对这种确实数据进行处理.

均值替换法就是将该项目剔除异常数据后取剩余数据的平均值来替换异常或缺失数据的方法:

$$\widetilde{X}_{ij} = \frac{1}{\hat{n}} \sum_{D} X_i. \tag{1}$$

其中,$\widetilde{X}_{ij}$ 表示缺失值，$\hat{n}$ 表示完好数据的数量，$\sum_D X_i$ 表示第 i 项理化指标完好数据的加和.

4.1.2 数据归一化处理

不同理化指标的绝对值存在数量级上的差异，可能会影响模型建立的可靠性，我们对数据进行标准化处理，某个数据减去该项指标数据平均值再除以相应的标准差:

对于任意数据 X_{ij},i 表示第 i 项理化指标,j 表示第 j 份红酒样的该项指标上的数据值.$\overline{X}_i$ 表示第 i 项理化指标的平均值.δ_i 表示第 i 项理化指标的标准差, 处理后的数据用 X'_{ij} 表示:

$$X'_{ij} = \frac{X_{ij} - \overline{X}_i}{\delta_i}. \tag{2}$$

4.1.3 数据分布

利用题中所给数据我们进行了可视化处理，画出了 3898 种红酒样品理化指标和分级的直方图与归一化处理后数据的箱线图, 由于图像过大, 我们将其置于目录中. 我们发现：

1.11 项理化指标数据呈现明显的正偏态. 即各项数据中众数 < 中位数 < 平均数. 其中氯化物，挥发性酸和游离二氧化硫的右偏程度较大，而总二氧化硫，酸碱度和酒精度的右偏程度较小。

2. 每项理化数据间的极差相差较大, 总二氧化硫的极差为 357.5，而密度间的极差仅为 0.051, 这说明了数据归一化的重要性.

3. 我们发现红酒的质量评分集中在 5,6,7 等. 品质分级在 4 等以下或 8 等以上的红酒较少.

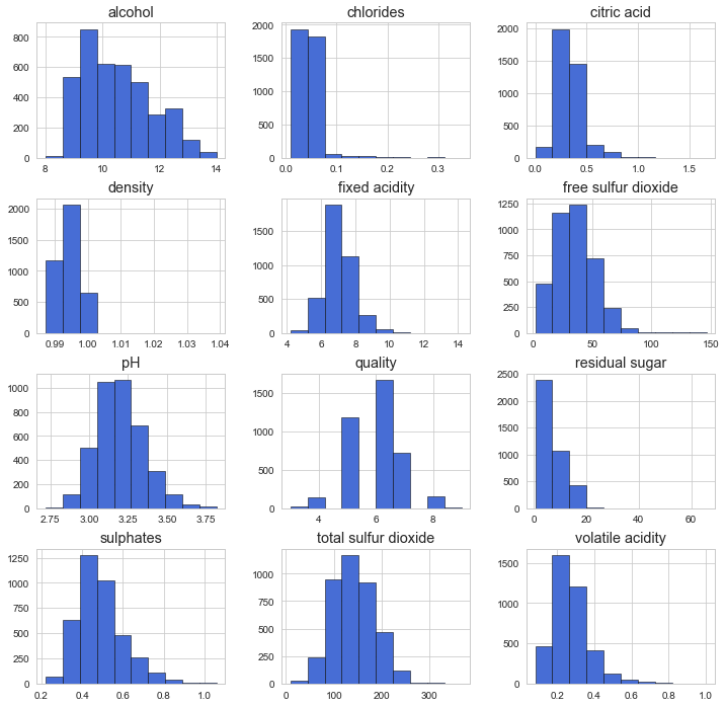

图 **2**　原始数据分布

4.2 问题分析

在问题一中，我们需要对题中给出的训练集进行缺失值，归一化等特征工程处理，然后对预处理以后的数据来建立合理的模型来对红酒数据集进行分类识别，对于分类算法来说比较传统的是决策树分类模型，朴素贝叶斯分类算法，而这些方法的实现往往评判比较单一，且决策树具有天然的过拟合问题，所以在小样本的情况下，容易出现过拟合问题，所以本文采用集成学习的思想，采用 xgboost 构建一个多决策树分类器系统来完成对该问题的分类，xgboost 是一种高度可扩展的端到端的多学习器模型，能够更加高效的解决该问题，并有效防止过拟合。

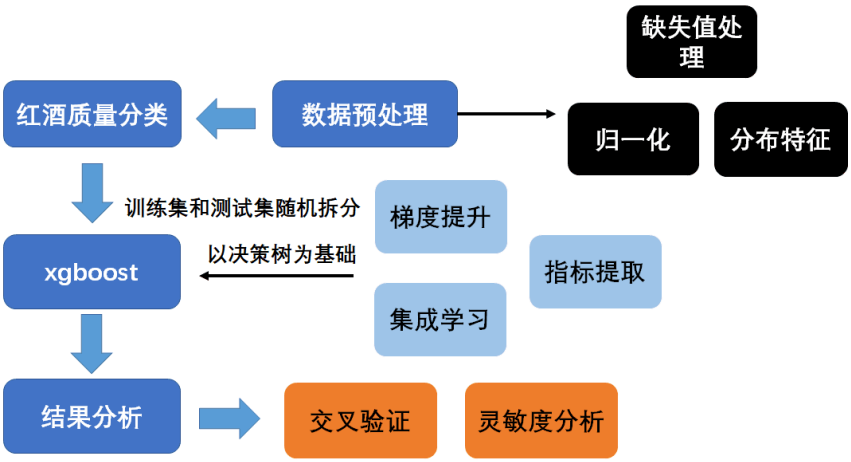

图 **3**　问题一思维导图

4.3 xgboost 集成决策树红酒分类模型

4.3.1 梯度提升法多决策树系统构建

xgboost 构建多个分类器系统的思想是梯度提升的方式，最开始先建立一棵决策树树，然后逐渐迭代，每次迭代过程中都增加一棵树并通过调整使得他能减小当前的预测误差，逐渐形成众多树模型集成的强评估器。其中 f_k 表示第 k 棵决策树的计算函数，x_i 表示样本 i 对应的特征向量, 特征向量则是我们上述归一化后对应的理化指标 $x_i = (X'_{i1}, X'_{i2}, ..., X'_{in})$.

这里我们设 $P_i = P_{i1}, P_{i2}, ..., P_{in}$ 表示第 i 棵决策树对当前样本预测得出的每一类的概率，令 P_{ij} 表示第 i 个样本在第 j 类上的预测概率，而概率最大的自然是被选择的样本预测结果，令 y_{ij} 表示第 i 个样本在第 j 类上的真实得分值，则我们可以构建一个葡萄酒等级的分类序列 $\hat{y}_i = (y_{i1}, y_{i2}, y_{i3}...y_{in})$, 那么我们就能得到 xgboost 中一棵决策树的样本预测结果和各类红酒等级的概率.

$$y_{ij} = \begin{cases} 1, & \text{if } y_{ij}\text{=j.} \\ 0, & \text{else.} \end{cases} \tag{3}$$

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i). \tag{4}$$

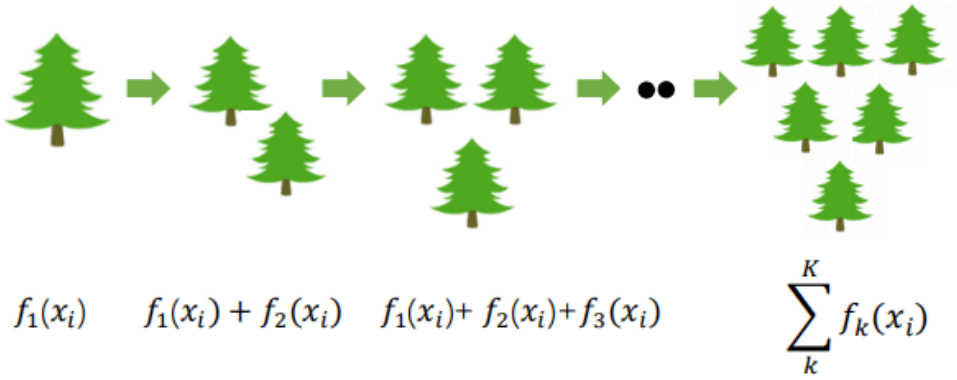

图 4　**xgboost** 梯度提升树构建

4.3.2 交叉熵损失函数

为了缩小样本与预测值之间的差异，我们需要通过构建衡量样本与原始数据差异的损失函数，这里我们采用交叉熵的方式来计算两个样本之间的结果。

$$L = \frac{1}{n}\sum_{i=1}^{N} L_i = \frac{1}{N}\sum_{i=1}^{N} L_i - \sum_{j=1}^{M} y_{ij} log(P_{ij}). \tag{5}$$

其中 M 表示分类的样本数量，P_{ij} 表示第 i 个样本在第 j 类上的预测概率,y_{ij} 表示第 i 个样本在第 j 类上的真实得分, 如果该类别和样本 i 的类别相同就是 1，否则是 0.N 表示样本的个数。

$$\begin{aligned} L(y,\hat{y}) &= \sum_{i=1}^{M} L\left(y_i,\hat{y}_i\right) \\ &= \sum_{i=1}^{M} L\left(\left(y_{i0},y_{i1},\ldots y_{iN-1}\right),\left(\hat{y}_{i0},\hat{y}_{i1},\ldots \hat{y}_{iN-1}\right)\right) \\ &= -\sum_{i=1}^{M}\sum_{n=0}^{N-1} y_{in}\log\left(\frac{e^{\hat{y}_{in}}}{\sum_{c=0}^{N-1} e^{\hat{y}_{ic}}}\right). \end{aligned} \tag{6}$$

4.3.3 模型复杂度惩罚项

同时为了保证该模型树的复杂度，我们在该损失函数中添加了惩罚项来限制树的模型，从而减小模型的叶子节点数目，树的深度，防止过拟合提高模型的泛化能力，也能减小运算的开销。

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2 = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2. \tag{7}$$

T 表示叶子节点的个数，w_j 表示叶子节点 j 上的分数。直观上看，目标要求预测误差尽量小，且叶子节点 T 尽量少 (γ 控制叶子结点的个数)，节点数值 w 尽量不极端 (λ 控制叶子节点的分数不会过大)，防止过拟合。

4.3.4 优化目标函数

为了在保证正确率和模型复杂度两者的情况下，我们通常在目标函数里需要将两者结合起来作为训练迭代的评判依据。

$$Obj = \sum_{i}^{N} L(y,\hat{y}) + \sum_{k}^{K}\Omega(f_k). \tag{8}$$

利用二阶泰勒展开法，可得到

$$Obj = \sum_{i=1}^{N}[f_k(x_i)g_i + \frac{1}{2}(f_k(x_i))^2 h_i)] + \Omega(f_k). \tag{9}$$

其中 $g_i = \frac{\partial L(y,\hat{y}_i^{(t-1)})}{\partial \hat{y}_{in}^{(t-1)}}, h_i = \frac{\partial^2 L(y,\hat{y}_i^{(t-1)})}{\partial \hat{y}_{in}^{(t-1)2}}$ 再通过整理操作得到

$$Obj = \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T. \tag{10}$$

I_j 被定义成每个叶节点 j 上面样本下标的集合 $I_j = \{i|f(x_i) = j\}$, 为了简化公式则把求导部分记为 $G_i = \sum_{i\in I_j} g_i$,$H_i = \sum_{i\in I_j} h_i$.

4.4 模型求解

当我们指定一个树的结构的时候，我们要对树的结构进行优化，我们将优化的程度称为结构分数.

$$Obj = -\frac{1}{2}\sum_{j}\frac{G_j^2}{H_j+\lambda} + 7\gamma. \tag{11}$$

4.4.1 最佳分割点搜索算法

为了确定树的结构，一个理所当然的想法就是不断枚举不同树的结构, 然后利用打分函数来寻一个最优结构的树. 本题提供了 3989 种红酒样品的数据，我们使用贪心法，从树深度为 0 开始，每一个节点都遍历所有的特征, 如柠檬酸含量，酒精度等. 对于某一特征，我们先按照该特征里的值进行排序，然后线性扫描该特征进而确定最好的分点，最后对所有特征进行分割后，我们选择增益 $Gain$ 最高的那个特征.

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma. \tag{12}$$

其中 $\frac{G_L^2}{H_L+\lambda}$ 表示左子树的分数，$\frac{G_R^2}{H_R+\lambda}$ 表示右子树的分数，$\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}$ 表示分割前的分数，γ 表示加入新叶子节点引入的复杂度代价.

对于所有的特征，我们只用做一遍从左到右的扫描就可以枚举出所有分割的梯度和树左右的得分 G_L 和 G_R. 然后计算每个分割方案的得数就可以的.

值得注意的是引入分割不一定会使情况变得更好，所以我们有一个引入新叶子的惩罚项。优化这个目标对应了树的简枝, 当引入的分割带来的增益小于一个阈值 Υ 的时候，则忽略这个分割. 以下是优化树结构的贪心法的伪代码:

Algorithm 1 Exact Greedy Algorithm for Split Finding

Input: I, 当前节点的实例集

Input: d, 特征维度

$gain \leftarrow 0\ G \leftarrow \sum_{i\in I} g_i, H \leftarrow \sum_{i\in I} h_i$

for $k = 1 \quad to \quad m$ **do**

$G_L \leftarrow 0, H_L \leftarrow 0$

for $j \quad in \quad sorted(I, byx_{jk})$ **do**

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$score \leftarrow (score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$的最大值

end for

end for

Output: 最高分数的优化

4.5 结果分析

4.5.1 xgboost 分类结构

通过训练和测试以后，我们利用 python 的 xgboost.to_graphviz 函数将他对分类特征的处理进行详细的可视化，从而得到他的完整分类流程，该模型共生成了 50 多棵树，这里选取其中第一棵树的结构来做一个实例，来反映他的流程。

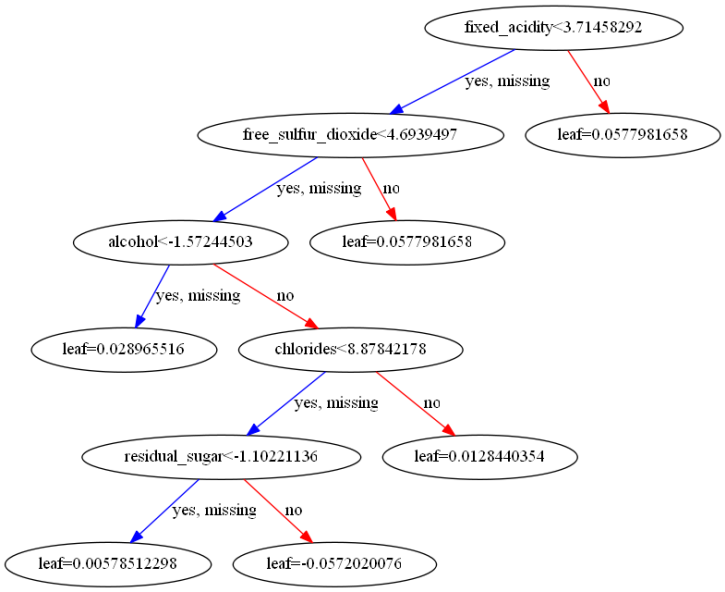

图 **5**　**xgboost** 中第一颗树的分类结构

4.5.2 分类性能

该模型通过交叉验证得到**该模型的正确率为 62.8%**, 当前查准率和查全率都为 0.628 说明，该模型的已经达到了识别分类的平衡点。

表 **1**　模型泛化性能

	微平均	宏平均	加权平均
查准率 precision	0.628	0.496615	0.625077
查全率 recall	0.628	0.413332	0.628
两者的调和平均 F1-score	0.628	0.433917	0.617503

我们画出了红酒分类预测的等级混淆矩阵，如图6, 矩阵对角线颜色较深，整体矩阵颜色较浅，说明预测偏差较少，且多发生相邻等级之间，证明了我们模型预测的可靠性.

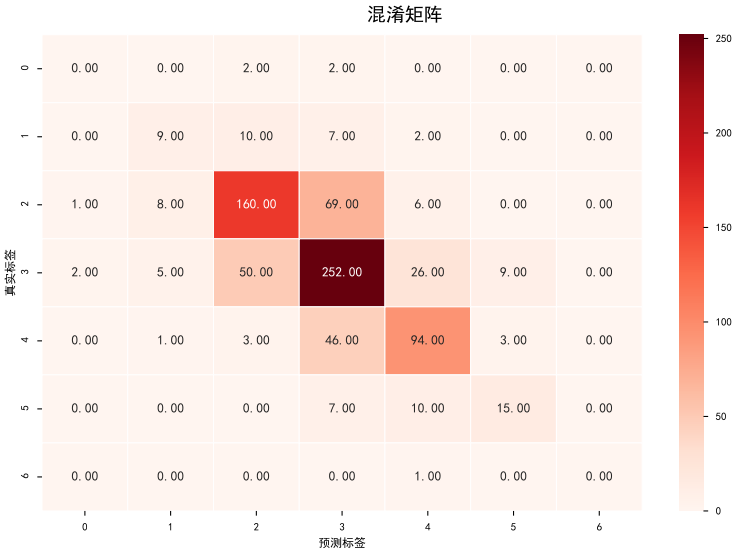

图 **6**　红酒分类等级混淆矩阵

五、 问题二模型的建立与求解

5.1 问题分析

问题二要求我们使用问题一建立的模型对附件中的待预测数据集进行品质预测，同时运用合理的数学模型找出影响红酒品质的理化指标的前三名. 我们使用的基于集成树思想的模型, 利用了不同红酒样品在 11 种理化指标上的差距，对红酒品质完成了分类. 对于模型一中使用的累加优化方法，一个特征出现时覆盖的样本数或者该特征带来的增益效果显然是影响红酒品质的重要指标.

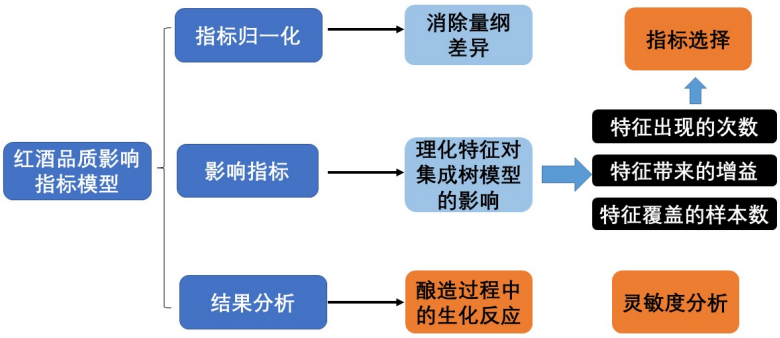

图 **7**　问题二思维导图

5.2 红酒品质影响指标模型的建立

我们认为一个特征在整个集成树模型中出现的次数越多，或者其特征作为树的分裂节点带来的增益总和越多，那么这项特征在对红酒进行分类时的影响就更大。

$xgboost$ 算法中有 5 种常见的计算特征重要性的指标;

- $weight$：表示某种特征在整个树群节点中出现的次数, 出现越多，价值就越高.
- $gain$：某特征在整个树群作为分裂节点的信心增益之和再除以某特征出现的频次，表示该特征出现带来的平均优化效果.
- $cover$：表示在树群中，一个特征作为分裂节点存在时，覆盖的样本数量的平均值.
- $totalgain$ 表示某特征给树群带来的总增益的大小.
- $tatalcover$ 表示覆盖的样本总量.

我们的目的是找出影响红酒品质的前三的理化指标，并解释什么导致一种红酒的品质处于某一等级，所以我们选择的指标要能反映理化指标对红酒样品的平均影响，所以我们选择 $gain$ 和 $cover$ 和 $weight$ 三个指标进行研究.

$$weight_i = N_i, \tag{13}$$

N_i 表示第 i 种理化特征在整个树群节点中出现的次数

$$gain_i = \frac{\sum_D Gain_i}{n_{is}}, \tag{14}$$

其中，n_{is} 表示第 i 种理化指标作为树分裂节点的次数.

$$cover_i = \frac{N_i}{n_{is}}. \tag{15}$$

其中,N_i 表示第 i 个理化指标作为分裂节点时，覆盖的样本总数.

指标归一化

我们已经得到了红酒不同理化指标上述两个指标上的得分情况，但这时的数据还只是得分的绝对表示, 为了研究不同理化指标对红酒品质影响的排列顺序，我们需要将得分的绝对数转化为相对数，即百分比形式.

用 P_{wi} 表示第 i 种理化指标基于 $weight$ 指标的得分情况.

$$P_{wi} = \frac{weight_i}{\sum_D weighti} \times 100\%, i = 1, 2, ...11, \tag{16}$$

用 P_{gi} 表示第 i 种理化指标基于 $gain$ 指标的得分情况:

$$P_{gi} = \frac{gain_i}{\sum_{i=1}^{11} gain_i} \times 100\%, i = 1, 2...11, \tag{17}$$

用 P_{ci} 表示第 i 种理化指标基于 $cover$ 指标的得分情况:

$$P_{ci} = \frac{cover_i}{\sum_{i=1}^{11} cover_i} \times 100\%, i = 1, 2...11. \tag{18}$$

5.3 模型求解与分析

5.3.1 相关性分析

我们画出了各个理化指标之间相关度的**热力图**，如8，可以看到:

1. 红酒的质量评分与酒精、酸碱性、硫酸盐、游离二氧化硫有正的线性相关关系，它们的含量越高，口感评分越高；而与其他特征都呈现负的线性相关关系，这些特征的含量越高，口感评分越低. 关系强度最大为 0.45。

2. 酒精度和密度之间的相关性为-0.77，两者存在较大相关性，且酒精度越高，密度越小。密度和残糖的相关性为 0.84，两者的相关性较大，且残糖越高，密度越大.

3. 从热力图上看，颜色普遍较浅，除极少对理化指标外，各理化指标之间的相关性较弱.

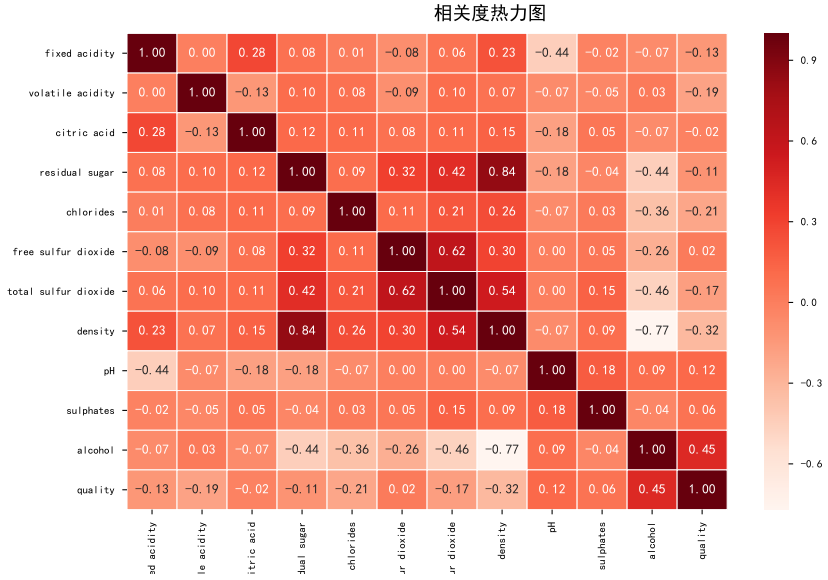

图 **8**　各理化指标之间的相关度

5.3.2 指标选择

我们使用 $feature-importance$ 命令求出了各项理化数据在上述三个指标上的表现, 在进行归一化后对各理化数据在三个指标上的影响分数进行累加，如图9.

我们查阅了相关参考文献[5]，了解了葡萄酒理化指标对葡萄酒品质的影响，在综合各理化指标与质量的相关性和上述三个指标显示的影响得分后，我们最终选择了 *cover* 指标作为计算影响红酒品质的方法. 图10展示了各理化指标在该标准下的影响分数:

我们得到的影响红酒品质属性前三名的理化指标分别为**酒精度, 密度和剩余糖分**, 影响分数分别为:**11.04%,10.59%,9.72%**.

根据参考文献[5][6], 我们对上述三种理化指标对葡萄酒品质产生影响的机理做出解释:

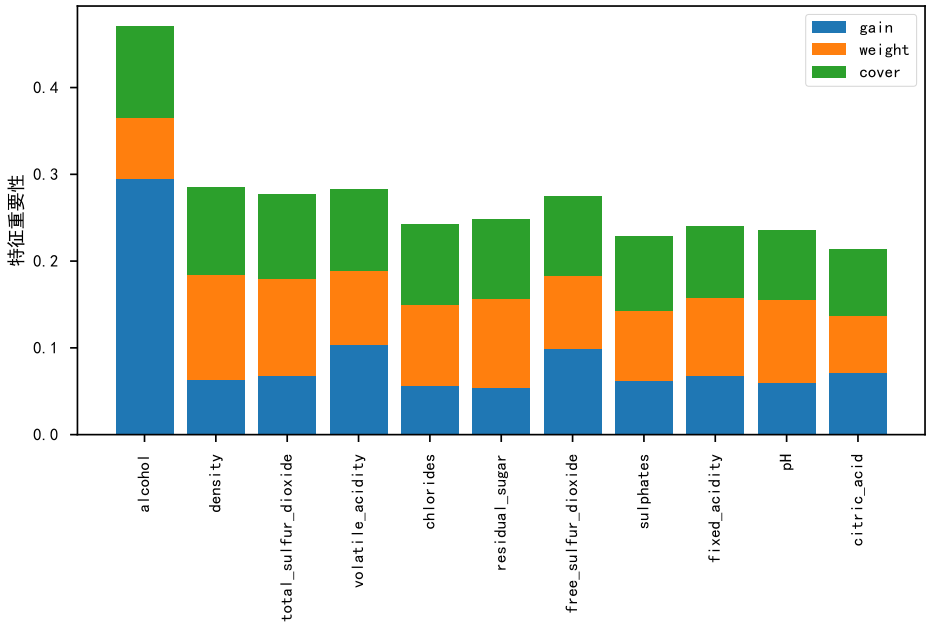

图 **9**　各理化指标影响分数在三个指标上的累加

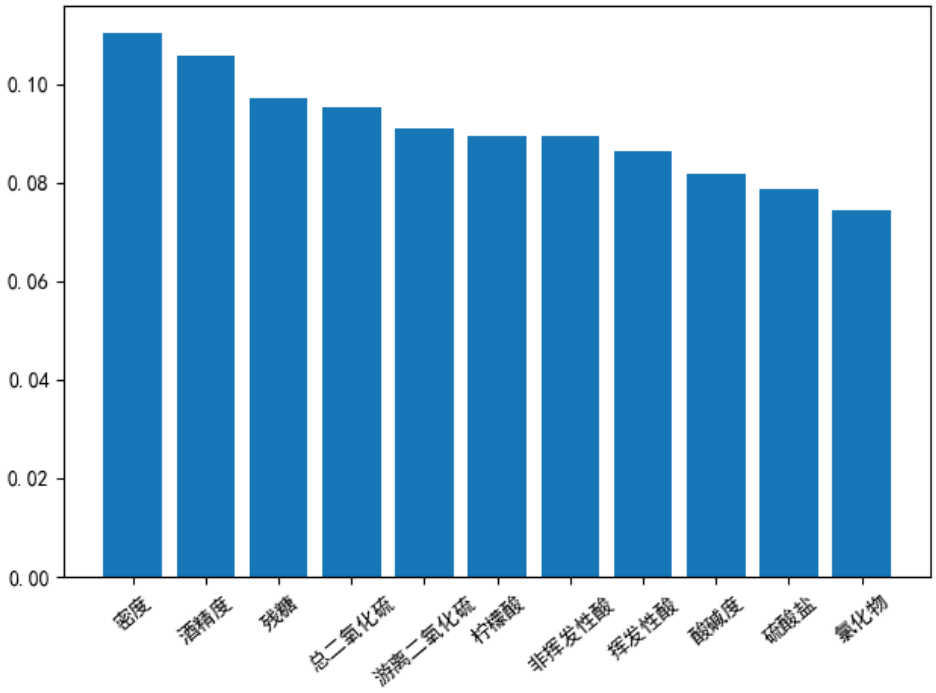

图 **10**　各理化指标影响分数在 **cover** 指标上的体现

1. 酒精度是对葡萄酒品质影响最大的理化指标，在一定范围内，酒精度越高，说明葡萄酒发酵的时间越长，发酵选择的菌种越优良，在相关参考文献[5] 中选出的品质上乘葡萄酒中，酒精度都在 13.74 ± 0.67 度之间，这也解释了样本数据中 80% 的品质为 9 级的红酒，酒精度都在 12.4 度以上.

2. 不同红酒间密度的差异[6] 主要是乙醇，糖类，各种酯类化合物，各种酸，色素造成的，这些经微生物发酵产生的化学物质，极大影响了红酒的颜色，口感，气味，是评价红酒品质的重要组成部分.

3. 在红酒酿造过程中，核心生化反应是通过微生物将糖类转化为乙醇，同时，剩余的糖也会对红酒的口感造成影响，红酒中残存的糖过多，则说明酿造时间过短，糖份过少可能会导致红酒苦涩，残糖量是影响红酒口感的重要指标.

5.3.3 待预测集品质预测

根据问题一中建立的模型，对附件一种待预测集进行品质预测：

表 **2**　预测结果分布

等级	3	4	5	6	7	8	9
数量	5	42	355	260	267	70	1

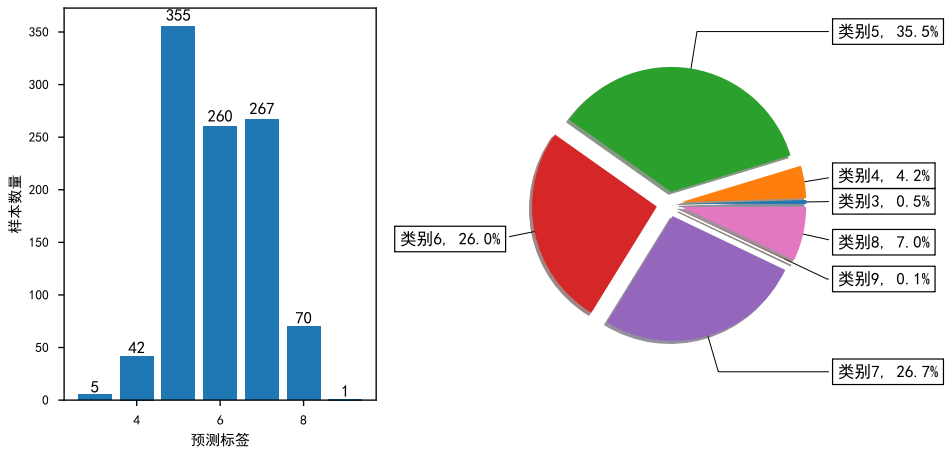

图 **11**　预测数据集质量分布

5.4 灵敏度分析

我们针对所选的重要指标对红酒品质的影响做灵敏度分析，通过随机选取 10 个样本，使其在参考文献所给的酒品正常规格内进行变化，并取这是个样本的均值可以看出. **在合适的范围内，酒精浓度的增大，红酒质量能得到提升，剩余糖分过多会影响品质。**

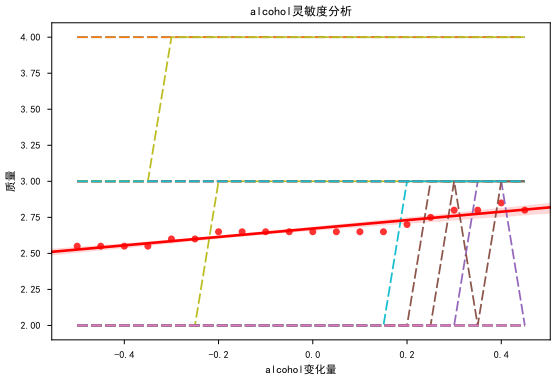

图 **12**　酒精含量变化对酒品质量的影响

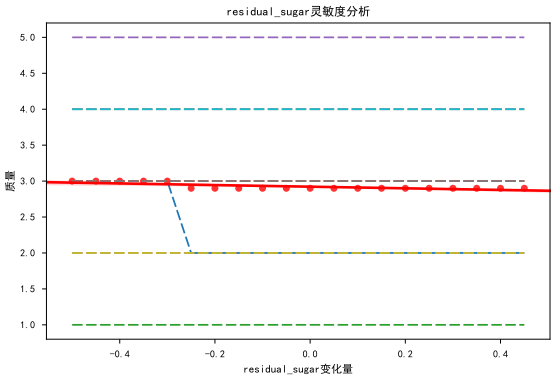

图 **13**　糖分变化对酒品质量的影响

六、 问题三模型的建立与求解

6.1 问题分析

在前两问的基础上我们可以发现我们的模型效果还有比较大的提升空间，分析原因我们可以知道该数据的各个类别在分布上有很明显的差异，有部分等级的数据数量明显偏少，导致样本的分布不均衡，所以使得 xgboost 建树的过程中还是容易受到影响，所以第三问打算使用上采样方法和贝叶斯优化器的方法对该模型的数据集进行模拟扩充和参数调整，从而达到更好的效果。

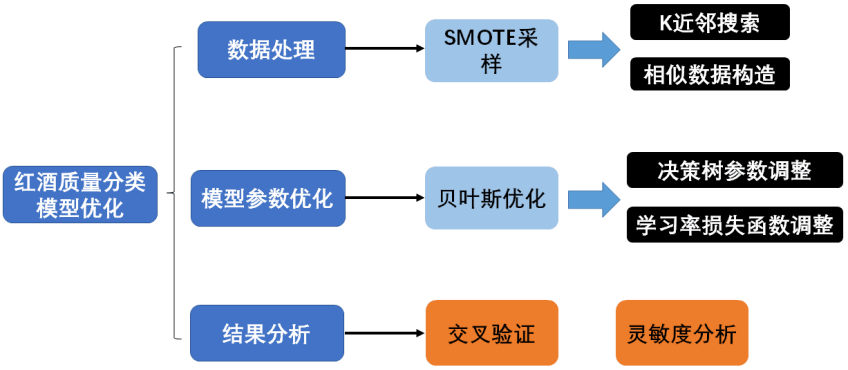

图 **14**　问题三思维导图

6.2 上采样技术与贝叶斯优化器改进的 xgboost 红酒分类模型

6.2.1 合成少数类过采样技术

我们通过观察发现当前数据中，质量所占样本的比例有很大的问题所以导致了预测的准确度有所下降，所以我们采用数据上采样的方法利用已知样本对数据进行构造生成相似的数据，使得当前数据的均衡度保持一致。

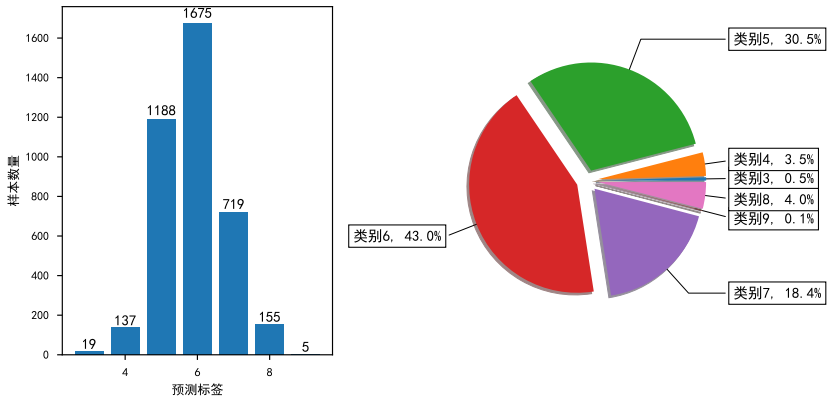

图 **15**　原始数据质量分布不均衡现象

对于少数类中的每一个样本 x，我们利用欧氏距离作为衡量两个样本数据差异的指标，选出它到少数类 S_{min} 中所有样本的距离得到其 k 近邻.

$$d = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + ... + (x_{in} - x_{jn})^2}. \tag{19}$$

根据样本不平衡比例设置一个采样比例以确定采样倍率 n，对于每一个少数类样本 x，从其 k 近邻中随机选择若干个样本，假设选择的近邻为 xn。

对于每一个随机选出的近邻 xn，分别与原样本按照公式20构建新的样本[7]。

$$x_{new} = x + rand(0,1) \times |x - xn|. \tag{20}$$

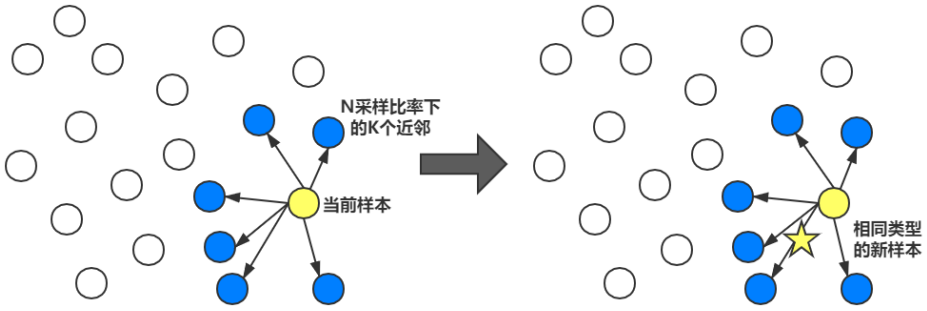

图 **16**　**SMOTE** 采样原理示意图

6.2.2 贝叶斯参数调优

我们将微分方程所有可调节的参数设为 **p**，对于每一个确定的 **p**，我们都能求出 XGBoost 模型 $model.model$ 输入一个数据，输出一个预测标签，即 $y_i = model(x_i)$. 为了对 **p** 进行调优，我们选择针对模型在测试集上的正确率作为衡量模型好坏的标准，所以

定义损失函数为模型预测错误的样本比例：

$$L(\mathbf{p}) = 1 - \frac{1}{n}(\sum_{i=0}^{n-1} CORRECT_i), \tag{21}$$

其中 n 为测试集样本总个数。$CORRECT_i$ 为一个布尔变量，表示第个样本预测正确，即：

$$CORRECT_i = \begin{cases} 1, & \text{if } model(i) = y_i. \\ 0, & \text{if } model(i) \neq y_i. \end{cases} \tag{22}$$

贝叶斯优化用较少的采样数目逼近全局最优值. 贝叶斯优化包含了关于 $L(\mathbf{p})$ 的先验信念，并用从 $L(\mathbf{p})$ 中抽取的样本更新先验，从而得到一个更好地接近 $L(\mathbf{p})$ 的后验. 用于近似目标函数的模型称为“代理模型”. 贝叶斯优化还使用了一个“获取函数”，将采样导向可能优于当前最佳观测的区域.

代理模型 (Surrogate model) 我们使用的代理模型是高斯过程 Gaussian Process (GP). 高斯过程是一个随机过程，其中任意点 $\mathbf{x} \in \mathbb{R}^d$ 被赋予一个随机值 $f(\mathbf{x})$，其中是这些变量的有限数量的联合高斯分布：

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \mathbf{K}), \tag{23}$$

其中 $\mathbf{f} = (f(\mathbf{x}_1), ..., f(\mathbf{x}_N)), \boldsymbol{\mu} = (m(\mathbf{x}_1), ..., m(\mathbf{x}_N)), K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$，$m$ 是均值函数. 高斯过程的函数的分布“平滑度”由 K 定义.

一个 GP 的先验 $p(\mathbf{f}|\mathbf{X})$ 可以在观察到一些数据 y 后转化为 GP 的后验 $p(\mathbf{f}|\mathbf{X}, \mathbf{y})$，根据：

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \int p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{f}) p(\mathbf{f}|\mathbf{X}, \mathbf{y})\, d\mathbf{f} = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*). \tag{24}$$

GPs 定义了我们对目标函数的先验知识，我们可以使用它们来描述关于目标函数的先验信念，例如“平滑性”.GP 后验计算成本低，用于在搜索空间中评估哪些采样点可能产生改进.

获取函数 (Acquisition functions) 在搜索空间中提出采样点是通过获取函数来实现的. 获取函数 $u(\mathbf{x}|\mathcal{D}_{1:t})$ 综合权衡利用 (exploitation) 和勘探 (exploration). 利用是指在替代模型预测的目标高的地方进行采样，勘探是指在预测不确定性高的地方进行采样. 两者会倾向于提高采集函数值，使得获取函数最大化的采样点为下一步的试验点.

$$\mathbf{x}_t = \text{argmax}_{\mathbf{x}} u(\mathbf{x}|\mathcal{D}_{1:t-1}). \tag{25}$$

优化算法 (Optimization algorithm) 贝叶斯优化算法过程如下.

每一次参数试验执行如下步骤:

step1. 通过代理模型 (GP) 求得获取函数 (AF)

step2. 通过获取函 (AF) 数找到下一个采样点 $\mathbf{x}_t = \text{argmax}_{\mathbf{x}} u(\mathbf{x}|\mathcal{D}_{1:t-1})$

step3. 对目标函数进行求值 $y_t = f(\mathbf{x}_t) + \epsilon_t$

step4. 将样本 $(\mathbf{x}_t, y_t)$ 加入先验样本 $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$

step5. 更新代理模型 (GP)

预期提升 (Expected improvement) 定义为：

$$\mathrm{EI}(\mathbf{x}) = \mathbb{E}\max(f(\mathbf{x}) - f(\mathbf{x}^+), 0). \tag{26}$$

其中 $f(\mathbf{x}^+)$ 是目前最优的目标函数值 $\mathbf{x}^+$ 是最优解 $\mathbf{x}^+ = \mathrm{argmax}_{\mathbf{x}_i \in \mathbf{x}_{1:t}} f(\mathbf{x}_i)$.

预期提升可以在 GP 模型下求得解析解：

$$\mathrm{EI}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \text{if } \sigma(\mathbf{x}) > 0. \\ 0, & \text{if } \sigma(\mathbf{x}) = 0. \end{cases} \tag{27}$$

其中：

$$Z = \begin{cases} \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})}, & \text{if } \sigma(\mathbf{x}) > 0. \\ 0, & \text{if } \sigma(\mathbf{x}) = 0. \end{cases} \tag{28}$$

式28中的 ξ 确定优化期间的探究量，较高的 ξ 值将导致更强的探索.ξ 的推荐值是 0.01.

6.3 结果分析

该模型通过数据上采样和贝叶斯优化从而提升了当前模型的准确率，**现在该模型的准确率为 68.4%，相比于第一问的结果提升了 8.9%** 查准率与查全率相等，说明该模型已经达到了平衡点，损失函数也降低到了比较平稳的状态.

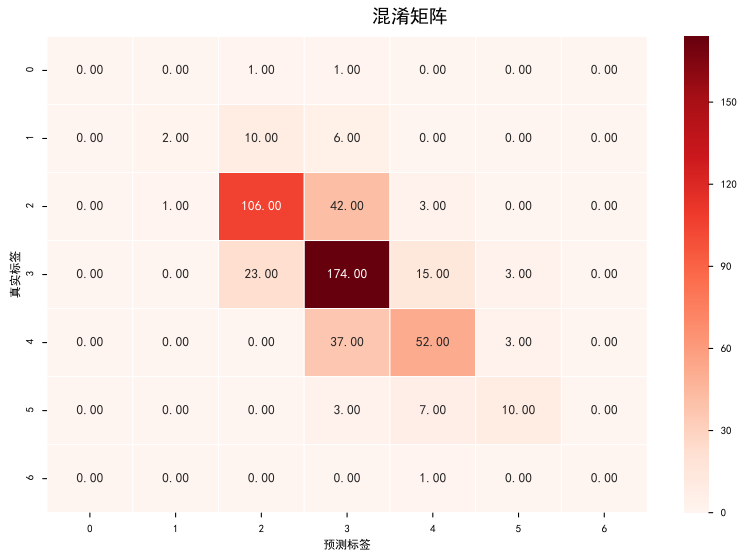

图 **17**　混淆矩阵

表 **3**　模型泛化性能

	微平均	宏平均	加权平均
查准率 precision	0.684	0.559318	0.687673
查全率 recall	0.684	0.438724	0.684
两者的调和平均 f1-score	0.684	0.472472	0.672773

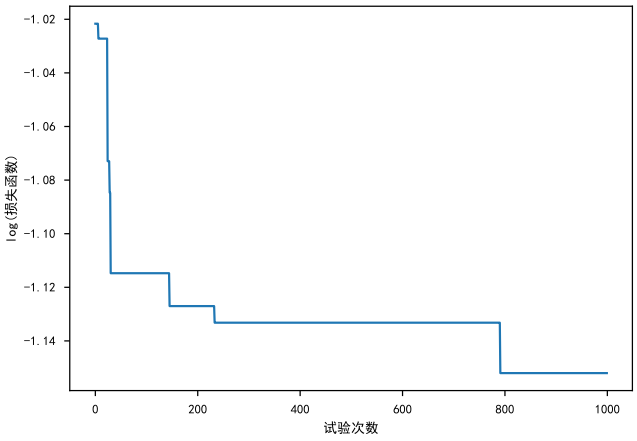

图 **18**　损失函数

6.4 灵敏度分析

我们对模型的每个超参数和他的准确率进行了分析，从而确定了当前的最佳参数，从图像上我们可以看出来各个参数与准确率之间的变化趋势，**很明显当我们的树深度增加的时候，这样对于我们分类红酒的效果会更好**，也能有效防止过拟合。**Gamma 指定了节点分裂所需的最小损失函数下降值，这个参数的值越大，算法越保守**。这个参数的值和损失函数息息相关，分类的效果也更好，**min_child_weight 决定最小叶子节点样本权重和**。XGBoost 的这个参数是最小样本权重的和, 当它的值较大时，可以避免模型学习到局部的特殊样本。但是如果这个值过高，会导致欠拟合, 所以我们可以看到该模型中损失会随着他的增大而增大。

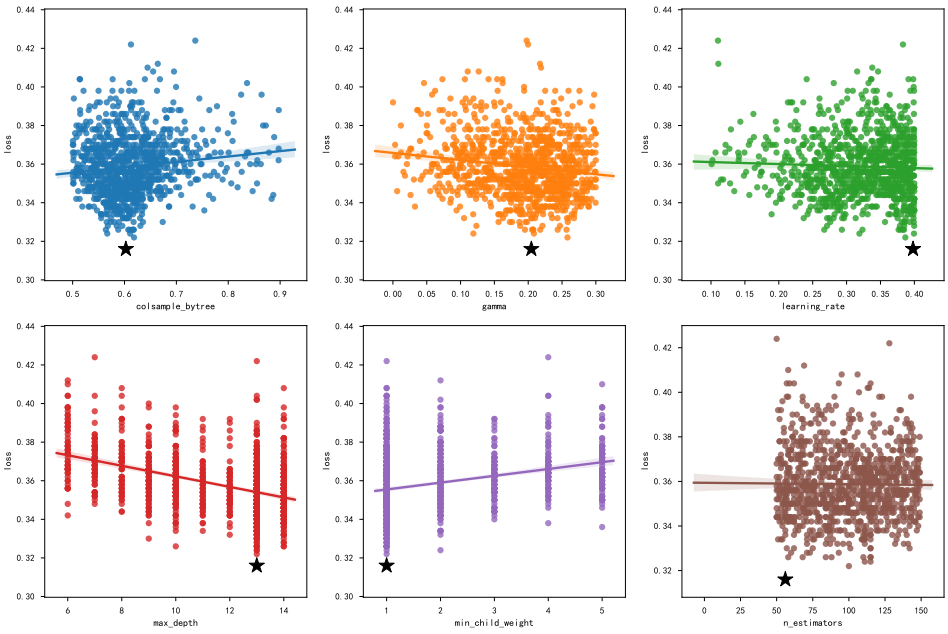

图 **19**　参数对酒品质量的影响

表 **4**　最佳参数设定

XGBoost 超参数		XGBoost 超参数	
colsample_bytree	0.6026232	**max_depth**	13
gamma	0.2046238	**min_child_weight**	1
learning_rate	0.3980428	**n_estimators**	56

七、 模型评价

7.1 模型的优点

1. 考虑到红酒品质预测问题必须首先进行红酒品质的分类，本文通过运用决策树与随机深林的相关理论，建立了集成决策树分类模型，使结果具有更广泛的适应性.

2. 求解问题时使用 xgboost 算法，相较于传统 GBDT 算法，该算法使用二阶泰勒展开，使得求解精度跟高，同时由于进行了正则化处理，有效的控制了模型的复杂度.

3. 综合三类评价指标和相关参考文献，选出了评价红酒品质的最优评价指标，理化指标影响分数的可信度较高.

7.2 模型的缺点

1.xgboost 算法属于集成学习模型，当基分类器的个数较多时，构建模型将比较耗时。而超参数调优又需要对不同参数的模型进行大量试验，所以超参数调优将非常耗时。

7.3 改进与展望

对于较大的数据集，可以采用近似算法降低程序的运行时间，近似算法的核心思想是只考察分裂节点。首先根据特征分布的分位数提出候选划分点，然后将连续型特征映射到由这些候选点划分的桶中，然后聚合统计信息找到所有区间的最佳分裂点。在提出候选切分点时有两种策略，一种是学习每棵树前就提出候选切分点，并在每次分裂时都采用这种分割，第二种策略是每次分裂前将重新提出候选切分点，直观上来看，第一种策略需要更多的计算步骤，而第二种策略因为节点已有划分所以需要更多的候选点.

参考文献

[1] 冯嫱薇. 基于 XGboost 算法的信用债违约风险预测研究 [D]. 电子科技大学,2020.

[2] 刘斌, 陈凯. 基于 SMOTE 和 XGBoost 的贷款风险预测方法 [J]. 计算机与现代化,2020(02):26-30.

[3] 孙逸菲, 袁德成, 王建龙, 白杨. 基于 XGBoost 方法的葡萄酒品质预测 [J]. 沈阳化工大学学报,2018,32(04):372-377.

[4] Shutao Wang,Shiyu Liu,Jingkun Zhang,Xiange Che,Yuanyuan Yuan,Zhifang Wang,Deming Kong. A new method of diesel fuel brands identification: SMOTE oversampling combined with XGBoost ensemble learning[J].Fuel,2020,282.

[5] 李记明, 姜文广. 优质干红葡萄酒中主要质量指标的研究 [J]. 中外葡萄与葡萄酒,2018(06):18-24.

[6] 司合芸. 干红葡萄酒关键工艺研究 [D]. 江南大学,2006.

[7] Han H , Wang W , Mao B . Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning[C]. International Conference on Intelligent Computing. Springer, Berlin, Heidelberg, 2005.

附录 A 代码

A.1 python 源程序

```
import numpy as np
import pandas as pd
import gc
import sys

from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import RandomOverSampler, ADASYN, SMOTE
import matplotlib.pyplot as plt

import lightgbm as lgb
import xgboost as xgb

import plot
import read_data


def LGBM(data, target, N_FOLDS=5):
    X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=0)
    # Training set
    train_set = lgb.Dataset(X_train, label=y_train)
    test_set = lgb.Dataset(X_test, label=y_test)

    model = lgb.LGBMClassifier(random_state=50)
    # Default hyperparamters
    hyperparameters = model.get_params()
    print(hyperparameters)

    model.fit(X_train, y_train)
    preds = np.argmax(model.predict_proba(X_test), axis=1)
    baseline = metrics.accuracy_score(y_test, preds)
    print('The baseline model {} scores {:.5f} accuracy_score on the test set.'.format(
        sys._getframe().f_code.co_name, baseline))


def randomForest(data, target):
    X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=0)

    # ros = SMOTE(k_neighbors=3)
    # X_train, y_train = ros.fit_sample(X_train, y_train)
```

```
    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X_train, y_train)
    y_predict = clf.predict(X_test)
    baseline = metrics.accuracy_score(y_test, y_predict)
    print('The baseline model {} scores {:.5f} accuracy_score on the test set.'.format(
        sys._getframe().f_code.co_name, baseline))
    # print(metrics.classification_report(y_test, y_predict))
    # print(metrics.confusion_matrix(y_test, y_predict))


def sensitivity_analysis(model, X, feature_name, win_size, step_size):
    feature = np.array(X[feature_name])
    steps = np.arange(-win_size, win_size, step_size)
    deltas = np.zeros((X.shape[0], len(steps)))
    result = np.zeros((X.shape[0], len(steps)))
    for i, delta in enumerate(steps):
        deltas[:, i] = delta
        X[feature_name] = feature + delta
        result[:, i] = model.predict(X)
    plot.plot_regression_sensitivity(deltas, result, name=feature_name)


def XGBoostRegressor(data, target, test_features):
    X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=0,
        test_size=500)

    # xgboost模型训练
    model = xgb.XGBRegressor(
        learning_rate=0.3,
        n_estimators=100,
        max_depth=6,
        min_child_weight=2,
        gamma=0.15,
        subsample=0.8,
        colsample_bytree=0.8,
        scale_pos_weight=1,
        n_jobs=-1)
    model.fit(X_train, y_train)
    # digraph = xgb.to_graphviz(model)
    # digraph.format = 'png'
    # digraph.view('../img/xgbtree')
    y_pred = model.predict(X_test)
    print(y_pred.shape)

    # 计算准确率
    baseline = metrics.accuracy_score(y_test, (y_pred+0.5).astype(np.int))
```

```
    print('The baseline model {} scores {:.5f} accuracy_score on the test set.'.format(
        sys._getframe().f_code.co_name, baseline))
    # plot.plot_xgb_fscore(model)
    gain = pd.DataFrame.from_dict(model._Booster.get_score(importance_type="gain"),
         orient='index')
    weight = pd.DataFrame.from_dict(model._Booster.get_score(importance_type="weight"),
         orient='index')
    cover = pd.DataFrame.from_dict(model._Booster.get_score(importance_type="cover"),
         orient='index')
    importance = pd.concat((gain, weight, cover), axis=1)
    importance.columns = ["gain", "weight", "cover"]
    importance = importance/importance.sum()
    importance.to_excel("../cache/importance.xlsx")
    plot.plot_xgb_fscore(importance)
    # print(importance)
    num_analysis = 10
    sensitivity_analysis(model, X_test[:num_analysis], "alcohol", .5, 0.05)
    sensitivity_analysis(model, X_test[:num_analysis], "residual_sugar", .5, 0.05)
    sensitivity_analysis(model, X_test[:num_analysis], "density", .5, 0.05)
    sensitivity_analysis(model, X_test[:num_analysis], "total_sulfur_dioxide", .5, 0.05)



def XGBoost_cv(data, target, test_features):
    # plot.plot_labels_class_counts(target, name="data")
    X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=0,
         stratify=target, test_size=800)

    smote = SMOTE(k_neighbors=3)
    X_train, y_train = smote.fit_sample(X_train, y_train)
    X_train = pd.DataFrame(X_train, columns=X_test.columns)
    # xgboost模型训练
    model = xgb.XGBClassifier(
        learning_rate=0.1,
        n_estimators=100,
        max_depth=10,
        min_child_weight=2,
        gamma=0.1,
        subsample=0.8,
        colsample_bytree=0.8,
        objective='multi:softprob',
        scale_pos_weight=1,
        n_jobs=-1)
    model.fit(X_train, y_train)
    # digraph = xgb.to_graphviz(model)
    # digraph.format = 'png'
    # digraph.view('../img/xgbtree')
```

```
    y_pred = model.predict_proba(X_test)
    print(y_pred.shape)
    y_pred = np.argmax(y_pred, axis=1)
    baseline = metrics.accuracy_score(y_test, y_pred)
    print('The baseline model {} scores {:.5f} accuracy_score on the test set.'.format(
        sys._getframe().f_code.co_name, baseline))

    report = metrics.classification_report(y_test, y_pred, output_dict=True)
    report = pd.DataFrame.from_dict(report)[["micro avg", "macro avg", "weighted avg"]]
    report.columns = ["微平均", "宏平均", "加权平均"]
    report.to_excel("../cache/classification_report.xlsx")
    print(report)
    CM = metrics.confusion_matrix(y_test, y_pred)
    plot.plot_CM(CM, name="problem1")
    test_pred = model.predict(test_features)
    plot.plot_labels_class_counts(test_pred, name="test_pred")
    num_analysis = 30
    sensitivity_analysis(model, X_test[:num_analysis], "alcohol", .5, 0.05)
    sensitivity_analysis(model, X_test[:num_analysis], "residual_sugar", .5, 0.05)
    sensitivity_analysis(model, X_test[:num_analysis], "density", .5, 0.05)
    sensitivity_analysis(model, X_test[:num_analysis], "total_sulfur_dioxide", .5, 0.05)

    gain = pd.DataFrame.from_dict(model._Booster.get_score(importance_type="gain"),
        orient='index')
    weight = pd.DataFrame.from_dict(model._Booster.get_score(importance_type="weight"),
        orient='index')
    cover = pd.DataFrame.from_dict(model._Booster.get_score(importance_type="cover"),
        orient='index')
    importance = pd.concat((gain, weight, cover), axis=1)
    importance.columns = ["gain", "weight", "cover"]
    importance = importance/importance.sum()
    importance.to_excel("../cache/importance.xlsx")
    plot.plot_xgb_fscore(importance)



if __name__ == '__main__':
    df, tgt, test_features = read_data.read_data()
    # LGBM(df, tgt)
    # randomForest(df, tgt)
    XGBoost_cv(df, tgt, test_features)
    # print('Baseline metrics')
    # print(metrics)
```

```
from hyperopt import fmin, tpe, hp, Trials, STATUS_OK
import read_data
```

```
import plot
from common import *

from sklearn import metrics
from sklearn.model_selection import train_test_split
import xgboost as xgb
from imblearn.over_sampling import SMOTE


class ParameterTunung():
    def __init__(self, data, target, max_evals,
                 learning_rate=None, n_estimators=None, max_depth=None,
                 min_child_weight=None, gamma=None, over_sample=None,
                 colsample_bytree=None, objective=None):
        self.max_evals = max_evals
        X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=0,
            stratify=target, test_size=500)

        smote = SMOTE(k_neighbors=3)
        X_train_os, y_train_os = smote.fit_sample(X_train, y_train)
        self.DATA = (X_train, X_test, y_train, y_test, X_train_os, y_train_os)

        self.space = {
            "learning_rate": hp.uniform("learning_rate", 0.25, 0.5) if learning_rate is None
                else learning_rate,
            "n_estimators": hp.choice("n_estimators", range(100, 250)) if n_estimators is None
                else n_estimators,
            "max_depth": hp.choice("max_depth", range(5, 10)) if max_depth is None else
                max_depth,
            "min_child_weight": hp.choice("min_child_weight", range(1, 6)) if min_child_weight
                is None else min_child_weight,
            "gamma": hp.uniform("gamma", 0.1, 0.3) if gamma is None else gamma,
             # "subsample": hp.uniform("subsample", 0.6, 0.9) if subsample is None else
                 subsample,
            "colsample_bytree": hp.uniform("colsample_bytree", 0.5, 0.7) if colsample_bytree is
                None else colsample_bytree,
            "over_sample": hp.choice("over_sample", range(1)) if over_sample is None else
                over_sample,
            "regression": hp.choice("regression", range(2)) if objective is None else objective,
        }
        self.bestParameters = None

    def get_model(self, p):
        # xgboost模型训练
        if p["regression"]:
            model = xgb.XGBRegressor(
                learning_rate=p['learning_rate'],
```

```
            n_estimators=p['n_estimators'],
            max_depth=p['max_depth'],
            min_child_weight=p['min_child_weight'],
            gamma=p['gamma'],
            colsample_bytree=p['colsample_bytree'],
            tree_method='gpu_hist',
            object="reg:squarederror",
            verbosity=0,
            n_jobs=-1)
    else:
        model = xgb.XGBClassifier(
            learning_rate=p['learning_rate'],
            n_estimators=p['n_estimators'],
            max_depth=p['max_depth'],
            min_child_weight=p['min_child_weight'],
            gamma=p['gamma'],
            colsample_bytree=p['colsample_bytree'],
            tree_method='gpu_hist',
            verbosity=0,
            n_jobs=-1)
    if p["over_sample"]:
        model.fit(self.DATA[4], self.DATA[5])
    else:
        model.fit(self.DATA[0], self.DATA[2])
    return model

def opt_parameters(self):
    """
    调用hyperopt库的API接口在搜索空间中确定最佳超参数
    :param max_evals:
    :return:
    """
    def _objective(p):
        """
        模型超参数搜索优化总距离（成本）
        :param p:
        :return:
        """
        model = self.get_model(p)
        y_pred = model.predict(self.DATA[1])
        if p["regression"]:
            y_pred = (y_pred+0.5).astype(np.int)
        loss = 1-metrics.accuracy_score(self.DATA[3], y_pred)
        result = {"total_loss": loss}
        return {'loss': result["total_loss"], "result": result, 'parameters': p, 'status':
            STATUS_OK}
    trials = Trials()
```

```
    best = fmin(fn=_objective, space=self.space, algo=tpe.suggest,
                max_evals=self.max_evals, trials=trials)
    return best, trials

def get_best_parameters(self):
    """
    封装opt_journey解析参数搜索结果
    :param max_evals:
    :return:
    """
    best, trials = self.opt_parameters()
    # Sort the trials with lowest loss first
    trials_list = sorted(trials.results, key=lambda x: x['loss'])
    bestParameters = trials_list[0]['parameters']
    result = trials_list[0]["result"]
    # print(trials_list[0])
    print("best parameter found after {} trials".format(len(trials_list)))
    print(bestParameters)
    print("lowest loss is")
    print(result)
    plot.plot_trials(trials_list)
    loss_step = [step["loss"] for step in trials.results]
    plot.plot_loss_time(loss_step)
    self.bestParameters = bestParameters
    return bestParameters

def eval_best_model(self, p=None):
    if self.bestParameters is None and p is None:
        raise Exception("Have not called get_best_parameters")
    if p is not None and self.bestParameters is None:
        print("Using given Parameters")
        self.bestParameters = p
    best_model = self.get_model(self.bestParameters)
    # 计算准确率
    y_pred = best_model.predict(self.DATA[1])
    y_test = self.DATA[3]
    if self.bestParameters["regression"]:
        y_pred = (y_pred+0.5).astype(np.int)
    baseline = metrics.accuracy_score(y_test, y_pred)
    print('The baseline model XGBoost scores {:.5f} accuracy_score on the test
        set.'.format(baseline))
    CM = metrics.confusion_matrix(y_test, y_pred)
    plot.plot_CM(CM, name="problem3")

    report = metrics.classification_report(y_test, y_pred, output_dict=True)
    report = pd.DataFrame.from_dict(report)[["micro avg", "macro avg", "weighted avg"]]
    report.columns = ["微平均", "宏平均", "加权平均"]
```

```
        report.to_excel("../cache/classification_report_problem3.xlsx")



if __name__ == '__main__':
    df, tgt, test_features = read_data.read_data()
    pt = ParameterTunung(df, tgt, max_evals=50)
    # bestParameters = pt.get_best_parameters()
    bestParameters = bestParametersDict["0.316"]
    pt.eval_best_model(bestParameters)
```

```
import warnings
import pandas as pd
import numpy as np
import matplotlib
matplotlib.rcParams["font.sans-serif"] = ["SimHei"]
matplotlib.rcParams['axes.unicode_minus'] = False
import matplotlib.pyplot as plt
from matplotlib import style
style.use('seaborn-paper')

warnings.filterwarnings("ignore")

WINDOW = 7
DAYS = 150

bestParametersDict = {
    # "0.35": {'colsample_bytree': 0.7477972829697044, 'gamma': 0.18755854462817376,
        'learning_rate': 0.33147590418072514, 'max_depth': 7, 'min_child_weight': 3,
        'n_estimators': 56, 'over_sample': 0, 'regression': 0},
    # "0.346": {'colsample_bytree': 0.829197705656825, 'gamma': 0.12982749219299972,
        'learning_rate': 0.27200281953968064, 'max_depth': 8, 'min_child_weight': 1,
        'n_estimators': 54, 'over_sample': 0, 'regression': 1},
    "0.316": {'colsample_bytree': 0.60262321330833, 'gamma': 0.20462377337987994,
        'learning_rate': 0.3980428446052504, 'max_depth': 13, 'min_child_weight': 1,
        'n_estimators': 56,
            'over_sample': 0, 'regression': 0
            },
    # "0.104":{'colsample_bytree': 0.601809606772758, 'gamma': 0.16651628749041883,
        'learning_rate': 0.3517747514399629, 'max_depth': 10, 'min_child_weight': 1,
        'n_estimators': 61, 'over_sample': 0, 'regression': 0},

}
```

```
def save_para_excel():
    df = pd.DataFrame(bestParametersDict["0.316"], index=["XGBoost超参数"])
    df.T.to_excel("../cache/para.xlsx")
    print(df.T)


if __name__ == '__main__':
    save_para_excel()
```

```

import numpy as np
import pandas as pd

from imblearn.over_sampling import ADASYN, SMOTE

def read_data(over_sample=False):
    df = pd.read_excel("../data/data.xlsx", sheet_name=0).replace("N", np.nan)
    test_features = pd.read_excel("../data/data.xlsx", sheet_name=1).replace("N", np.nan)
    tgt = df["quality"]-3
    df.drop(["quality"], axis=1, inplace=True)
    # 缺失值的计算
    # df.apply(lambda col: col.fillna(col.median()), axis=0)
    df.fillna(df.mean(), inplace=True)
    df = (df-df.mean())/(df.std())
    assert not np.isnan(df).any().any()
    cols = []
    for i in range(len(df.columns)):
        cols.append(df.columns[i].replace(" ", "_"))
    df.columns = cols
    test_features.columns = cols
    test_features = (test_features-test_features.mean())/(test_features.std())
    if over_sample:
        smote = SMOTE(k_neighbors=4)
        df_os, tgt_os = smote.fit_sample(df, tgt)
        df_os = pd.DataFrame(df_os, columns=df.columns)
        # tgt_os = pd.DataFrame(tgt_os, columns=tgt.columns)
        return df_os, tgt_os, test_features
    else:
        print(df.shape)
        return df, tgt, test_features


def read_winequality(over_sample=False):
    df = pd.read_excel("../data/wine quality-white.xls")
    print(df.head())
    print(df.describe())
```

```
if __name__ == '__main__':
    # df, tgt, test_features = read_data()
    # print(df)
    # print(tgt)
    read_winequality()
```

```
from matplotlib.backends.backend_pdf import PdfPages
import seaborn as sns
from common import *

import xgboost as xgb


def plot_mat(mat, save=True, name="mat"):
    plt.figure()
    plt.imshow(mat, cmap=plt.cm.hot,
               # vmin=0, vmax=1
               )
    plt.colorbar()
    if save:
        pdf = PdfPages("img//"+name+".pdf")
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


def get_trials_df(trials, hypers):
    df = pd.DataFrame(columns=hypers+["loss"])
    for trial in trials:
        row_dict = trial['parameters'].copy()
        row_dict["loss"] = trial["loss"]
        df = df.append(row_dict, ignore_index=True)
    return df


def plot_trials(trials, save=True, name=""):
    r, c = 2, 3
    hypers = list(trials[0]['parameters'].keys())[:r*c]
    trials_df = get_trials_df(trials, hypers)
    best_trial = trials_df.iloc[0, :]
```

```
    # assert len(hypers) == r*c
    fig, axs = plt.subplots(r, c, figsize=(4*c, 4*r))
    for i in range(r):
        for j in range(c):
            hyper = hypers[c*i+j]
            sns.regplot(hyper, "loss", data=trials_df[[hyper, "loss"]], ax=axs[i, j])
            axs[i, j].scatter(best_trial[hyper], best_trial["loss"], marker='*', s=200, c='k')
            axs[i, j].set(xlabel='{}'.format(hyper), ylabel="loss")
    plt.tight_layout()
    if save:
        pdf = PdfPages("..//img//"+name+"_trials.pdf")
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


def plot_loss_time(loss_step, save=True, name=""):
    loss_time = []
    bestLoss = loss_step[0]
    for loss in loss_step:
        if loss < bestLoss:
            loss_time.append(loss)
            bestLoss = loss
        else:
            loss_time.append(bestLoss)
    plt.figure()
    plt.plot(list(range(len(loss_time))), np.log(np.array(loss_time)))
    plt.xlabel("试验次数")
    plt.ylabel("log(损失函数)")

    if save:
        pdf = PdfPages("..//img//{}_loss_time.pdf".format(name))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()

def plot_xgb_fscore(importance, save=True, name="xgb"):
    importance.sort_values(by="cover", axis=0, ascending=False, inplace=True)
    importance_types = importance.columns
    x = range(len(importance.index))
    plt.bar(x,
            importance[importance_types[0]],
            label=importance_types[0])
```

```
    plt.bar(x,
            importance[importance_types[1]],
            label=importance_types[1],
            bottom=importance[importance_types[0]])
    plt.bar(x,
            importance[importance_types[2]],
            label=importance_types[2],
            bottom=importance[importance_types[0]]+importance[importance_types[1]])
    plt.legend()
    plt.ylabel('特征重要性')
    plt.xticks(x, importance.index)
    plt.xticks(rotation=90)
    plt.tight_layout()
    if save:
        pdf = PdfPages("..//img//feature_importance_{}.pdf".format(name))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()

    plt.close()
    plt.figure()
    cover = importance[importance_types[2]]
    plt.bar(x, cover, label=importance_types[2],)
    plt.legend()
    plt.ylabel('特征重要性')
    plt.xticks(x, importance.index)
    plt.xticks(rotation=90)
    plt.tight_layout()
    if save:
        pdf = PdfPages("..//img//feature_importance_{}.pdf".format("cover"))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


def plot_corr(save=True, name=""):
    df = pd.read_excel("../data/data.xlsx", sheet_name=0).replace("N", np.nan)
    f, ax = plt.subplots(figsize=(10, 6))
    corr = df.corr()
    hm = sns.heatmap(round(corr, 2), annot=True, ax=ax, cmap="Reds", fmt='.2f',
                     linewidths=.05)
    f.subplots_adjust(top=0.93)
    t = f.suptitle('相关度热力图', fontsize=14)
    if save:
```

```
        pdf = PdfPages("..//img//{}_corr.pdf".format(name))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


def plot_CM(CM, save=True, name=""):
    f, ax = plt.subplots(figsize=(10, 6))
    sns.heatmap(CM, annot=True, ax=ax, cmap="Reds", fmt='.2f', linewidths=.05)
    f.subplots_adjust(top=0.93)
    f.suptitle('混淆矩阵', fontsize=14)
    plt.xlabel("预测标签")
    plt.ylabel("真实标签")
    if save:
        pdf = PdfPages("..//img//{}_CM.pdf".format(name))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


def plot_features(save=True, name=""):
    df = pd.read_excel("../data/data.xlsx", sheet_name=0).replace("N", np.nan)
    df.fillna(df.mean(), inplace=True)
    # df = (df-df.mean())/(df.std())
    assert not np.isnan(df).any().any()

    r, c = 4, 3
    sns.set()
    # sns.set()
    # sns.relplot(data=df, x='alcohol', y="quality", kind='line', height=5, aspect=2,
        color='red')
    # plt.show()
    cols = list(df.columns)
    # assert len(hypers) == r*c
    fig, axs = plt.subplots(r, c, figsize=(4*c, 4*r))
    for i in range(r):
        for j in range(c):
            if c*i+j >= len(cols):
                continue
            col = cols[c*i+j]
            sns.regplot(data=df, x=col, y="quality", ax=axs[i, j])
            # axs[i, j].scatter(df[col], df["quality"])
```

```
    plt.tight_layout()
    if save:
        plt.savefig("..//img//"+name+"_features.png")
        pdf = PdfPages("..//img//"+name+"_features.pdf")
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


def draw_multivarient_plot(dataset=None, rows=4, cols=3, plot_type="violin", save=True,
     name=""):
    if dataset is None:
        dataset = pd.read_excel("../data/data.xlsx", sheet_name=0).replace("N", np.nan)
    # Veri setindeki sütünların isimleri alınıyor
    column_names = dataset.columns.values
    # Kaç tane sütün olduğu bulunuyor
    number_of_column = len(column_names)

    # Satır*sütün boyutlarında alt grafik içeren
    # matris oluşturuluyor. Matrisin genişliği:22 yüksekliği:16
    fig, axarr = plt.subplots(rows, cols, figsize=(22, 16))

    counter = 0 # Çizimi yapılacak özelliğin column_names listesindeki indeks değerini tutuyor
    for i in range(rows):
        for j in range(cols):
            if 'violin' in plot_type:
                sns.violinplot(x='quality', y=column_names[counter], data=dataset, ax=axarr[i][j])
            elif 'box' in plot_type:
                sns.boxplot(x='quality', y=column_names[counter], data=dataset, ax=axarr[i][j])
            elif 'point' in plot_type:
                sns.pointplot(x='quality', y=column_names[counter], data=dataset, ax=axarr[i][j])
            elif 'bar' in plot_type:
                sns.barplot(x='quality', y=column_names[counter], data=dataset, ax=axarr[i][j])

            counter += 1
            if counter == (number_of_column - 1,):
                break
    plt.tight_layout()
    if save:
        pdf = PdfPages("..//img//" + name + "{}_multivarient.pdf".format(plot_type))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()
```

```
def plot_regression_sensitivity(deltas, result, save=True, name=""):
    plt.figure()
    for i in range(len(deltas)):
        plt.plot(deltas[i], result[i], dashes=[6, 2])
    result_mean = np.mean(result, axis=0)
    # plt.scatter(deltas[0], result_mean, marker="s")
    sns.regplot(deltas[0], result_mean, color="red")
    plt.title(name+"灵敏度分析")
    plt.xlabel(name+"变化量")
    plt.ylabel("质量")
    plt.tight_layout()
    if save:
        pdf = PdfPages("..//img//sensitivity_{}.pdf".format(name))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


def autolabel(rects, ax, xpos='center'):
    """
    Attach a text label above each bar in *rects*, displaying its height.

    *xpos* indicates which side to place the text w.r.t. the center of
    the bar. It can be one of the following {'center', 'right', 'left'}.
    """

    xpos = xpos.lower() # normalize the case of the parameter
    ha = {'center': 'center', 'right': 'left', 'left': 'right'}
    offset = {'center': 0.5, 'right': 0.57, 'left': 0.43} # x_txt = x + w*off

    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()*offset[xpos], 1.01*height,
                '{}'.format(height), ha=ha[xpos], va='bottom')


def plot_labels_class_counts(labels, save=True, name=""):
    labels += 3
    x = np.array([3, 4, 5, 6, 7, 9, 8])
    num = len(labels)
    class_counts = []
    for tgt in x:
        class_counts.append((labels == tgt).sum())
```

```
    fig, axs = plt.subplots(1, 2, figsize=(8, 4))
    rects1 = axs[0].bar(x, class_counts)
    autolabel(rects1, axs[0])
    axs[0].set_xlabel("预测标签")
    axs[0].set_ylabel("样本数量")
    wedges, _ = axs[1].pie(class_counts, explode=np.zeros_like(x)+0.1, shadow=True)
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
    kw = dict(arrowprops=dict(arrowstyle="-"),
              bbox=bbox_props, zorder=0, va="center")
    recipe = ["类别{}, {:.1f}%".format(x[i], 100*class_counts[i]/num) for i in range(len(x))]
    for i, p in enumerate(wedges):
        ang = (p.theta2 - p.theta1) / 2. + p.theta1
        y = np.sin(np.deg2rad(ang))
        x = np.cos(np.deg2rad(ang))
        horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
        connectionstyle = "angle,angleA=0,angleB={}".format(ang)
        kw["arrowprops"].update({"connectionstyle": connectionstyle})
        axs[1].annotate(recipe[i], xy=(x, y), xytext=(1.35 * np.sign(x), 1.4 * y),
                    horizontalalignment=horizontalalignment, **kw)

    plt.tight_layout()
    if save:
        pdf = PdfPages("..//img//labels_class_counts_{}.pdf".format(name))
        pdf.savefig()
        pdf.close()
    else:
        plt.show()
    plt.close()


if __name__ == '__main__':
    plot_xgb_fscore()
    # draw_multivarient_plot(plot_type="box")
```