基于预测解释模型和离散化萤火虫算法的中风人群分析

摘要

本文研究中风及各因素对中风影响的问题，利用 XGBoost 分类与 TreeSHAP 事后解释器，分析各因素对中风影响情况，选择主要指标并进行分组，建立双目标背包规划模型，利用改进的离散化萤火虫算法求解得到最易中风小群体，最终利用 XGBoost 分类模型预测中风可能性，并进行模型检验。

针对问题一，本文进行了特征处理，并利用 XGBoost 算法与 TreeSHAP 事后解释器分析各指标与中风间的关系。本文利用 **KNN 回归**填补 BMI 数据缺失值，利用 **KNN** 分类填补吸烟数据缺失值，并利用 **SMOTENC 过采样**使数据类别达到平衡以及对数据进行了对数化处理。此外，本文利用 **XGBoost 算法**以及 **TreeSHAP 事后解释器**，得到不同指标对中风的影响程度以及指标的不同取值与中风率的关系。其中，影响程度较大的指标为**年龄**、**身体质量指数**、**平均血糖水平**。最后，本文针对定量数据和定类数据分别进行 **K-W 检验**与**回归分析**。

针对问题二，本文确定主要指标对调查对象进行分组，建立双目标背包规划模型，得到最易中风小群体，并从宏观和微观角度分析群体特征。首先，本文确定年龄、身体质量指数、平均血糖水平、吸烟情况、工作类型以及性别 **6 项**主要指标，并利用 **K-Means++** 以及**肘部法则**对 3 项定量指标进行聚类，将调查对象分为 511 组。之后，本文以最大化中风人数和最小化各组差异率作为目标建立**双目标背包规划模型**，利用**改进的离散化萤火虫算法**求解得到满足条件的背包即小群体的 Parero 最优解集。在最优解中选出 3 个互不相交的最易中风人群。最后，利用**主成分分析降维**与**高斯混合聚类**对结果进行**宏观分析**，利用描述性统计对结果进行**微观分析**。

针对问题三，本文初步筛选特征，建立 XGBoost 分类模型预测中风可能性，进行 10-fold 交叉验证与贝叶斯调参求解模型，最终进行模型检验。首先，本文利用**交叉验证-递归消除法**对各因素特征进行初步筛选。之后，本文建立 **XGBoost 分类模型**对附件 2 中 10 个人的中风可能性进行预测，并在模型求解过程中利用 **10-fold 交叉验证**对数据集进行处理，利用**贝叶斯调参**寻找最优模型参数。最有可能中风的对象 id 为 **3**、**4**、**7**。最后，本文结合问题一中的 **TreeSHAP 事后解释器**以及问题二中的**最易中风小群体**对预测结果进行检验。

本文的优势在于： 1. 本文利用 TreeSHAP 结合 K-W 检验与回归分析，使结果更具合理性与可解释性；2. 本文针对问题对传统萤火虫算法进行了改进，使之可以对离散化问题进行求解并且提高了收敛速度。

关键字：XGBoost　TreeSHAP 事后解释器　双目标背包规划　改进的离散化萤火虫算法　RFECV

目录

一、 问题重述

1.1 问题背景

调查显示，中风已成为我国第一位死亡原因，也是中国成年人残疾的首要原因。不同类型的中风，其治疗方式不同。由于一直缺乏有效的治疗手段，目前认为预防是最好的措施。因此，了解了解居民的身体状况与出现中风的联系，进而分析中风的成因，并采取有针对性地措施，有效地预防该疾病[7]。

中风是极度危险的医疗状况，及早发现，及早治疗就能给患者减少痛苦和伤害。大多数的中风是因为患者长期不健康生活，饮食所造成的后果。影响中风的因素常常体现在高血压、心脏病、吸烟状况、血糖、身体质量、住宅类型、工作类型、婚姻状况以及吸烟情况等方面。

图 1　中风及成因背景图

1.2 问题提出

中风是危害人们健康的重大疾病之一，在我国居民死亡原因中高居第一位，而且我国居民的中风的比例也是世界上最高的之一。针对居民身体状况以及中风成因的相关信息，我们对以下几个问题进行研究：

问题一：分析数据集中的各因素对中风的影响程度进行排序，并分析各因素的不同取值与中风发病率的关系。

问题二：利用所给数据集，找到最易出现中风情况的小群体，并分析个体特征。

问题三：分析不同个体中风的可能性大小，并对附件 2 中的 10 个人按中风可能性大小进行排序。

二、 模型假设

（1）所有数据真实可靠。

（2）数据中的各样本是通过合理采样得到，反映了社会的真实情况。

（3）指标中 0 代表无该情况，1 代表有该情况。

三、 符号说明

符号	含义	说明
y	中风指标	
$x_{1,i}, x_{2,i}, \cdots, x_{8,i}$	高血压、心脏病、婚姻、工作、住宅、血糖、身体质量、吸烟指标	
$\phi(x_{j,i})$	XGBoost 预测函数	$\phi(x_{j,i}) = \sum_{k=1}^{K} f_k(x_{j,i}), f_k \in \Gamma$
$g(z')$	TreeSHAP 模型 shapley 值	$g(z') = \psi_0 + \sum_{j=1}^{M} \psi_j$
N_0, n	调查总人数，分类总组数	$N_0 = 5110, n = 511$
w_i	第 i 组人数	$i = 1, 2, \cdots, n$
p_i	第 i 组中风人数	$i = 1, 2, \cdots, n$

四、 问题一的模型建立与求解

4.1 问题一的描述与分析

问题一要求分析数据集中各因素对中风的影响程度，并分析各因素的不同取值与中风发病率的关系。本文首先利用 KNN 算法对数据集中各指标数据的缺失值进行填补。其中，利用 KNN 回归填补 BMI 数据缺失值，利用 KNN 分类填补吸烟数据缺失值。由于中风与不中风数据量差异过大，本文利用 SMOTENC 过采样使数据类别达到平衡。此外，本文还对数据进行了对数化处理以避免数据出现两极化情况。

为分析各因素对中风的影响情况，本文利用 XGBoost 算法以及 TreeSHAP 事后解释器，得到不同指标对中风的影响程度以及指标的不同取值与中风率的关系。最后，本文针对定量数据和定类数据分别进行 K-W 检验与回归分析。

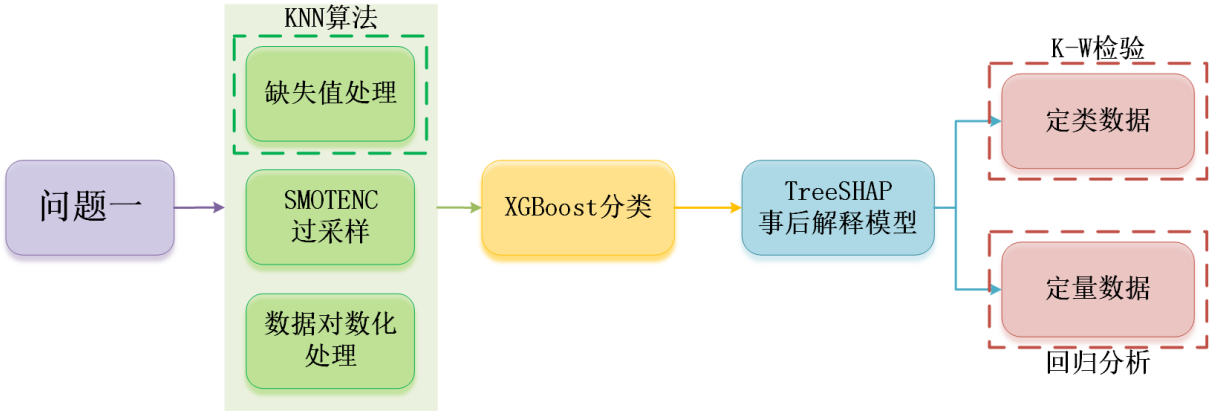


图 2　问题一思路图

4.2 特征处理

4.2.1 基于 KNN 算法的缺失数据处理

本文对数据进行初步分析发现，性别 (gender)、身体质量指数 (BMI) 以及吸烟状况 (smoking status) 的数据中存在缺失值。本文对三种类型的缺失数据分别进行处理。

- **性别数据缺失值**

性别数据中，id 为 56156 的调查对象性别数据为 Other，本文对该项缺失值对应调查对象的数据予以删除。

- **BMI 数据缺失值**

身体质量指数数据中共存在 201 项缺失值，且该数据具有连续的数据标签，因此本文利用 K-Nearest Neighbor(KNN) 回归对该类缺失值进行填补[3]。在回归过程中，本文首先控制变量即除 BMI 指标以外的其他指标**全部相等**，对此时的身体质量指数进行 **KNN 回归**。设某缺失数据对应第 i 个调查对象，该对象的身体质量指数的预测值为 b_i，满足下式：

$$b_i = \frac{b_{i,1} + b_{i,2} + \cdots + b_{i,k}}{k}. \tag{1}$$

其中，$b_{i,1}, b_{i,2}, \cdots, b_{i,k}$ 表示从训练数据集中选择的离该缺失数据点最近的 k 个数据点。

- **吸烟状况数据缺失值**

吸烟状况数据中共存在 1543 项“Unknown”数据，且该类数据具有离散的数据标签，因此本文利用 **KNN 分类**对该类缺失值进行填补。同样，本文首先进行控制变量，即除吸烟状况数据以外其他指标均相同的情况下进行 KNN 分类处理。观察吸烟状况数据发现共有“formerly smoke”、“never smoked”、“smokes”三种数据类型。对于缺失数据，找该数据的 K 个最近邻，统计上述 3 种类别在最近邻中的占比，选取占比最多的类别作为该缺失值的类别[3]。

4.2.2 SMOTENC 数据过采样

分析数据发现，仅有 249 个调查对象中风，而其余 4861 个调查对象没有中风，数据表现不均衡。不平衡样本会导致训练模型侧重样本数目较多的类别，而“轻视”样本数目较少类别，这样模型在测试数据上的泛化能力就会受到影响。因此，为解决不平衡样本问题，我们需对数量少的数据进行过采样操作。

本文采用可处理分类特征过采样方法的 SMOTENC 合成少数类过采样技术对数据进行过采样，这是在随机采样的基础上改进的一种过采样算法[9]，其实现过程为：

首先，从少数类样本中选取一个样本 x_i。其次，按采样倍率 N，从 x_i 的 K 近邻中随机选择 N 个样本 x_{zi}。最后，依次在 x_{zi} 和 x_i 之间随机合成新样本，合成公式如下：

$$x_n = x_i + \beta \times (x_{zi} - x_i). \tag{2}$$

数据过采样前后的数据分布如下图3：

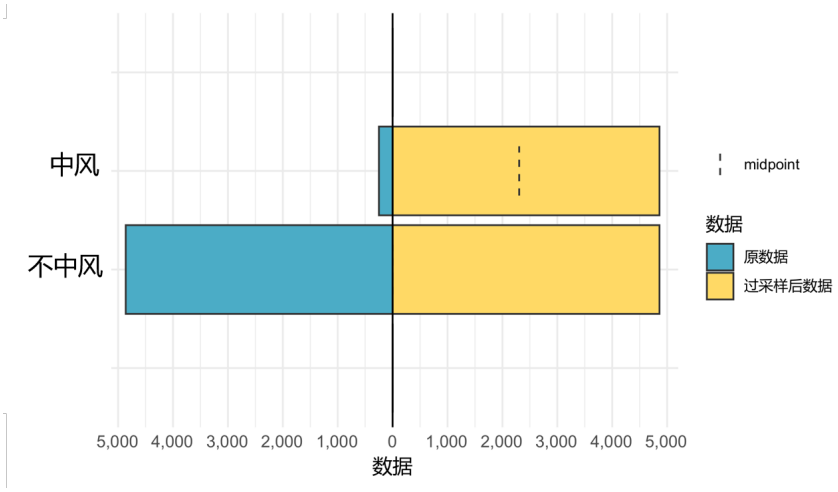


图 3　过采样前后数据分布图

由图可以看出，数据过采样后，中风与未中风的数据量差别减少，数据类别达到平衡，从而避免模型在测试数据上的泛化能力受到影响。

4.2.3 数据对数化处理

由于调查对象的各指标数据中存在差异过大的数据，会对后续分析造成影响，因此，本文利用 lnx 对数据进行取对数处理，避免数据出现两极化的现象。

4.3 基于 XGBoost 算法与 TreeSHAP 事后解释器的影响分析模型建立

- **XGBoost 算法训练和预测**

首先，本文通过 Python 软件利用 XGBoost 分类算法对模型进行训练和预测。设第 i 个调查对象的高血压、心脏病、婚姻、工作、住宅、血糖、身体质量、吸烟数据分别为 $x_{1,i}, x_{2,i}, \cdots, x_{8,i}$，中风情况数据为 y_i。则本文给定数据集为 $D = (x_{1,i}, x_{2,i}, \cdots, x_{8,i}, y_i)$，XGBoost 进行 additive training，学习 K 棵树，采用以下函数对样本进行预测[1]：

$$\tilde{y} = \phi(x_{j,i}) = \sum_{k=1}^{K} f_k(x_{j,i}), f_k \in \Gamma, j \in (1, 2, \cdots, 8). \tag{3}$$

其中 $f(x)$ 是回归树，Γ 是假设空间：

$$\Gamma = \{f(x) = w_{q(x)}\}(q : R^m \to T, w \in R^T). \tag{4}$$

其中，$q(x)$ 表示样本 x 分到了某个叶子节点上，w 是叶子节点的分数，所以 $w_{q(x)}$ 表示回归树对中风情况样本的预测值。

- **TreeSHAP 事后解释器**

XGBoost 算法虽然具有较好的效果，但不具有可解释性，即对于特定的一个样本，样本特征值对最终结果的影响情况未知。因此，本文利用来自博弈论的 TreeSHAP 方法，建立 TreeSHAP(tree SHapley Additive exPlanations) 事后解释器，对 XGBoost 的求解结果进行解释[4]。

TreeSHAP 实际是将输出值归因到每一个特征的 shapely 值上，特征的 shapley 值表示该特征对总体特征的边际贡献的期望值。

计算每一个特征的 shapley 值，以此来衡量特征对最终输出值的影响。用公式表示：

$$g(z') = \psi_0 + \sum_{j=1}^{M} \psi_j z'_j. \tag{5}$$

其中，其中 g 是解释模型，M 是输入特征的数目，z 表示相应特征是否存在，ψ 是每个特征的 Shapley 值，ψ_0 是一个常数。

由于 XGBoost 模型的输入是结构化数据，对于样本 x，所有的特征都是存在的，则有：

$$g(z') = \psi_0 + \sum_{j=1}^{M} \psi_j. \tag{6}$$

基于上述 TreeSHAP 事后解释器对 XGBoost 进行解释，计算各指标不同调查对象的 shapley 值，从而得到不同指标对中风的影响程度以及指标的不同取值与中风率的关系。

4.4 模型求解

4.4.1 TreeSHAP 事后解释器求解

在进行了 XGBoost 模型的训练和预测后，本文通过 Python 软件利用 TreeSHAP 事后解释器对 XGBoost 的求解结果进行解释，具体求解的伪代码如下：

Algorithm 1 TreeSHAP

Input: Training and forecasting results of the XGBoost algorithm, Feature samples

Output: **procedure** TS($x, tree = \{v, a, b, t, r, d\}$)

ψ =array of $len(x)$ zeros

procedure RECURSE(j, m, p_z, p_o, p_i)

$m = EXTEND(m, p_z, p_o, p_i)$

if $v_j \neq internal$ **then**

for $i \leftarrow$ 2to $len(m)$ **do**

$w = sum(UNWIND(m, i), w)$, $\psi_{m_i} = \psi_{m_i} + w(m_{i,o} -, _{i,z})v_j$;

end for

if $k \neq nothing$ **then**

$i_z, i_o = (m_k \cdot z, m_k \cdot o)$, $m = UNWIND(m, k)$;

end if

RECURSE($h, m, i_z r_h / r_j, i_o, d_j$), RECURSE($c, m, i_z r_c / r_j, 0, d_j$);

end if

RECURSE(1,[],1,1,0)

return ψ

end procedure

4.4.2 模型求解总步骤

将模型建立中的每一步骤进行求解实现，其具体步骤如下：

Step 1：利用 XGBoost 算法数据集中高血压、心脏病、婚姻、工作、住宅、血糖、身体质量、吸烟数据与中风数据进行训练与预测。

Step 2：利用 XGBoost 算法进行特征选择，并通过随机搜索调节模型参数获得更好的效果。

Step 3：利用 TreeSHAP 事后解释器对 XGBoost 求解结果进行解释，得到各因素不同调查对象的 shapley 值。

Step 4：对指标的不同调查对象 shapley 值取**绝对值**并求和，得到不同指标对中风的影响程度。依据各指标不同取值的 shapley 值**平均值**得到指标的不同取值与中风率的关系，并进行后续分析。

4.5 求解结果与分析

4.5.1 模型效果比较

在进行模型选择时，本文利用 XGBoost、随机森林以及 CatBoost 分别进行了训练和预测，三种分类算法测试集上的 F1 分数表现如下表：

表 1　三种算法效果比较表

	XGBoost	随机森林	CatBoost
F1 分数	0.961	0.872	0.95

由上表可知，XGBoost 的 F1 分数表现为 0.961，明显优于其他两种算法，因此本文选择 XGBoost 算法建立模型。

4.5.2 求解结果

根据上述 XGBoost 算法以及 TreeSHAP 事后解释器求解得到各因素对应的 shapley 值，由此得到各因素对中风的影响程度量化结果如下图4：

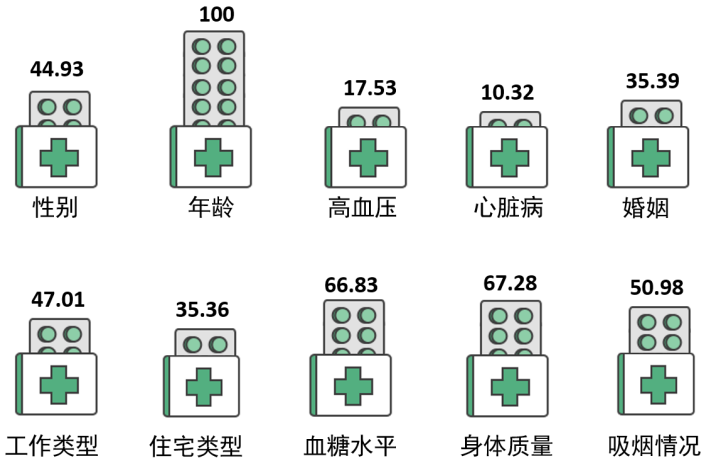


图 4　各因素对中风影响程度量化结果图

由上图可知，年龄对中风的影响程度最大，这是由于随年龄增长人体机能逐渐下降，从而导致中风发病情况的增加。此外，对中风影响程度较大的因素还有身体质量指数、平均血糖水平、吸烟情况等。上述因素对中风的较大影响与中风影响因素的现状基本相符，由此可知本文分析各因素对中风的影响较为准确。

然而，观察结果发现高血压和心脏病指标对中风的影响较小，与实际情况存在差异。这是由于 TreeSHAP 模型消除了各指标间的多重共线性，导致高血压和心脏病指标对中风的影响通过年龄、血糖水平以及 BMI 表现了出来，而未体现在对应量化结果中。

此外，根据各指标不同取值对应 shapley 值的平均值，得到指标取值对中风率的影响情况如下图5。图中定量指标本文均将其按高中低分为三类进行分析。

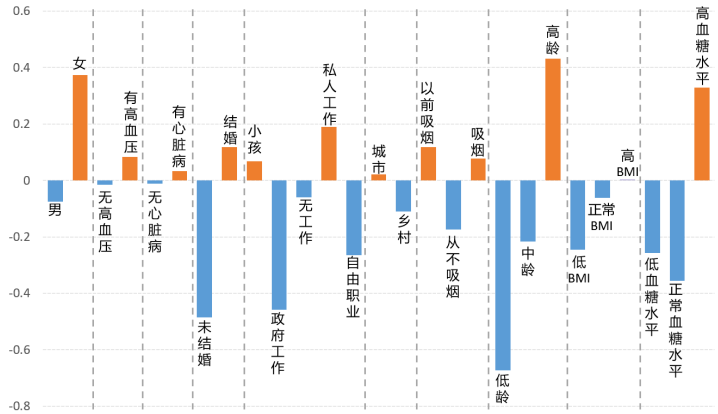


图 5　各指标不同取值影响示意图

由上图数据分布可得到以下结论：

1. 性别指标中女性比男性的中风率更高，本文分析该结果与实际情况存在差异的原因是 TreeSHAP 模型在消除各指标多重共线性的过程中，将男性对中风率的影响体现在了吸烟以及高血压等男性常见特质上，因此导致了男性对中风率的影响较小。

2. 对于高血压以及心脏病指标，患高血压和心脏病均对中风率具有正向的影响，与实际情况相符。

3. 婚姻状态指标显示已婚对中风率呈现正向影响而未婚呈现负向影响，该结果可能是由于已婚人群的年龄普遍大于未婚人群，结合上述量化结果显示的年龄对中风率的显著影响即可证明该结果的合理性。

4. 对于工作类型指标，由图中数据分布可知小孩中也存在部分中风患者，此外私人工作与中风率也有着正向的关系，而政府工作、无工作以及自由职业对中风率的影响为负向。该结果说明，不同的工作压力对中风率的影响不同，对于工作压力较大的人群中风率更高。因此，社会与政府应针对不同岗位的工作者给予有针对性地补贴与福利政策。

5. 对于住宅类型指标，图中数据显示居住地为城市时易导致中风，而居住农村则并未导致中风率的增加。该结果说明，城市中各种类型的污染以及不良的生活习惯会导致中风率的上升，因此在城市居住的人群应该改善生活习惯提高生活质量，而全社会应共同努力保护城市生活环境，减少污染。

6. 对于吸烟情况指标，以前吸烟以及吸烟的情况均易导致中风，而从不抽烟显然与中风不存在正向关系。该结果说明了吸烟对于身体的危害，且易引起中风，社会应加大宣传吸烟对人体的危害以保障广大群众的身体健康。

7. 对年龄、BMI、血糖水平三项定量指标进行分析可以发现，对于年龄指标，低龄与中龄人群均不易中风，而高龄人群中风率较大。身体质量指数较大的人群即肥胖人

群，以及高血血糖水平人群与中风率呈现正向关系，因此日常生活中人们应适当控制饮食并合理锻炼以避免疾病的发生。

4.5.3 拓展分析

- **定类数据 K-W 检验**

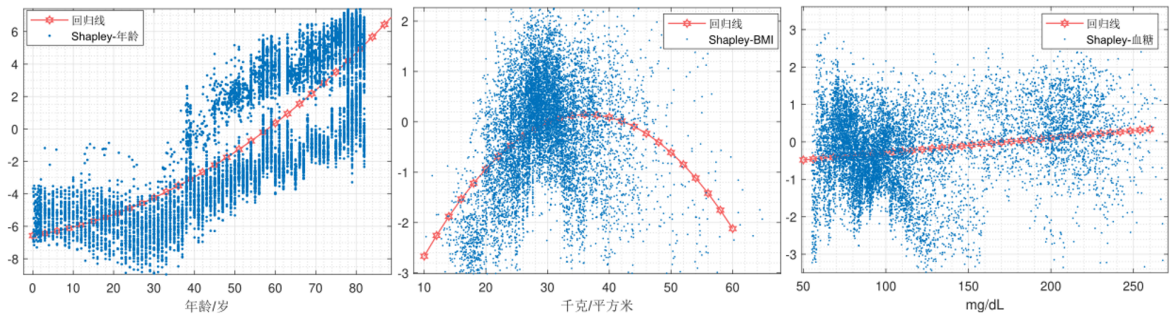
本文对定类指标对应的 shapley 值进行正态检验，发现数据不符合正态分布，因此引入非参数检验对其进行分析。本文对定类指标原数据与 shapley 值进行 Kruskal-Wallis(K-W) 检验，分析各指标在 shapley 值上是否表现出显著差异[5]。检验结果如下：

分析项	性别	高血压	心脏病	婚姻	工作	住宅	吸烟
统计量	2083.235	27.526	17.242	2705.734	1908.57	730.641	1779.647
p 值	0.000***	0.000***	0.000***	0.000***	0.000***	0.000***	0.000***
Cohen's f 值	0.007	0.006	0.003	0.015	0.014	0.004	0.01

观察上表发现，吸烟情况数据的检验结果 p 值为 $0.000^{***} < 0.05$，因此统计结果显著，说明不同的吸烟情况数据在吸烟情况的 shapley 值上存在显著差异。同理可知性别、高血压、心脏病、婚姻、工作以及住宅的定类指标在对应 shapley 值上存在显著差异。差异幅度 Cohen's f 值越大则指标的不同取值对 shapley 值的影响越大。因此，对 f 值进行分析发现，婚姻对应的 f 值为 0.015，工作类型对应 f 值为 0.014，即婚姻与工作类型指标的不同取值对其 shapley 值影响最大。

- **定量数据回归分析**

本文对 3 个定量指标年龄、身体质量指数、平均血糖水平的 shapley 值进行回归分析，结合上述三个指标的数据特征本文分别选择二次、二次和一次函数对其进行回归。具体回归系数见附录。回归结果如下图6：



年龄shapley值回归曲线图　　身体质量shapley值回归曲线图　　血糖shapley值回归曲线图

图 6　定量指标 shapley 值回归曲线图

观察上图回归曲线可以发现，年龄越大其对应 shapley 值越大，即中风率越大，该结果与老年群体中风率较高的事实相符。中国成人的正常 BMI 应该在 18.5-23.9 之间，而 BMI 超过 28 时为肥胖。观察 BMI 回归曲线图发现 BMI 在 30-40 时对应的中风率达到最大，该结果说明肥胖人群更易换中风。对于 BMI 极大时表现出的中风率的下滑可能是由于过度肥胖人群数量较少，样本不足的原因导致的。观察血糖 shapley 值回归曲线可以发现，血糖含量的变化对中风率并无较大的影响。综合上述分析，与实际情况相互对应，印证了本文模型的可靠性。

此外，对于上述曲线的回归效果，年龄对应回归曲线的 R^2 为 0.662，回归效果较好。而由于 shapley 值在身体质量指数和平均血糖水平指标的各取值中分布散乱，回归得到身体质量指数与平均血糖水平对应的 R^2 均小于 0.2，效果有待提高。

五、 问题二的模型建立与求解

5.1 问题二的描述与分析

问题二要求利用提供数据，找到几个最易出现中风的小群体，并分析这些群体的个体特征。首先，本文结合问题一对各指标影响程度的分析确定年龄、身体质量指数、平均血糖水平、吸烟情况、工作类型以及性别 **6 项**主要指标，并利用 **K-Means++** 以及**肘部法则**对年龄、身体质量指数、平均血糖水平 3 项定量指标进行聚类，从而对 5110 个调查对象进行分组。之后，本文以最大化中风人数和最小化各组差异率作为目标建立**双目标背包规划模型**，得到满足条件的背包即小群体的 Parero 最优解集。最后，利用主成分分析降维与高斯混合聚类对结果进行宏观分析，利用描述性统计对结果进行微观分析。

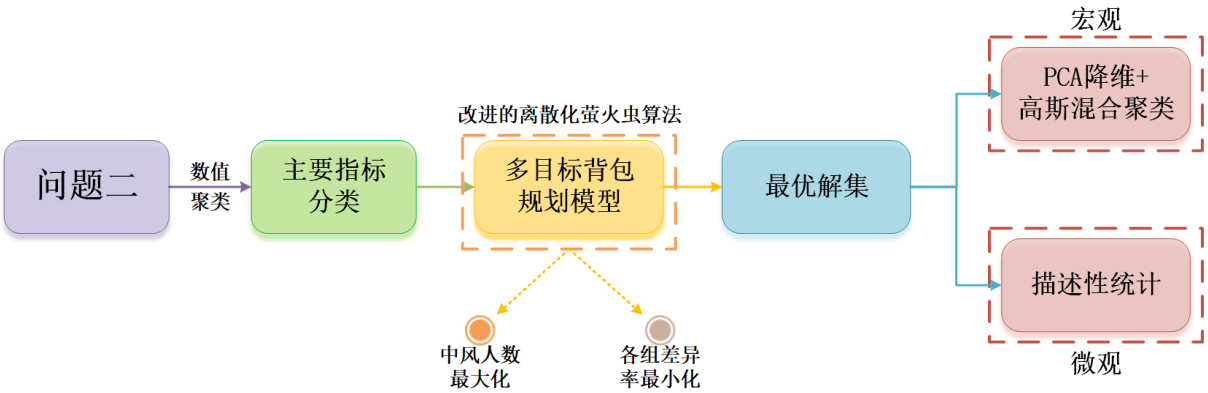


图 7　问题二思路图

5.2 主要指标分组确定考察群体

- **主要指标的选择**

基于问题二的要求，本文首先结合问题一中的结果确定主要指标，再对主要指标进行分组，对每组的中风情况进行后续分析得到最易现中风小群体。

对问题一中对十项指标的量化结果进行分析发现，年龄、身体质量指数、平均血糖水平、吸烟情况、工作类型以及性别 6 项指标对中风的影响程度在十项指标中**占 79%**，可以较大程度的反映整体情况，因此本文以上述 6 项作为主要指标对调查对象分组。

- **定量指标聚类**

本文选择了年龄、身体质量指数、平均血糖水平、吸烟情况、工作类型以及性别 6 项指标对调查对象进行分组，其中年龄、身体质量指数以及平均血糖水平为定量指标。为了更好地进行分组并简化计算，本文利用 K-Means++ 算法对年龄、身体质量指数、平均血糖水平 3 项数值型数据进行聚类，将这三项指标划分为不同类别。

在聚类中心的选择时，K-Means++ 算法采用原则是保证初始聚类中心之间距离尽可能远，从而对 K-Means 法该项权限进行改进[2]。算法流程如下图8：

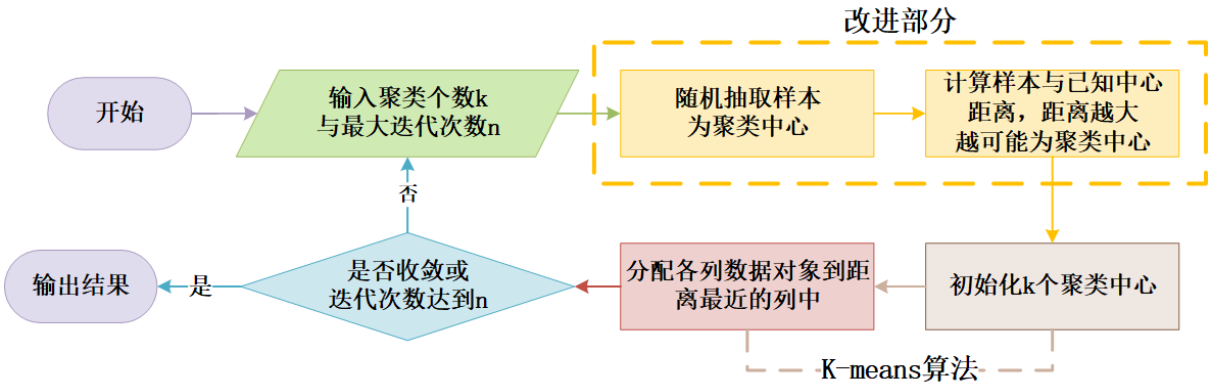


图 8　K-Means++ 聚类算法流程图

针对上述聚类中的 k 值，本文采用肘部法则进行确定。三项指标对应肘部图如下图9所示，其横坐标为聚类类别数 k，纵坐标为畸变程度。由此，本文确定 $k=3$ 、$k=3$、$k=4$ 为分别为年龄、身体质量指数、平均血糖水平的最佳聚类数。

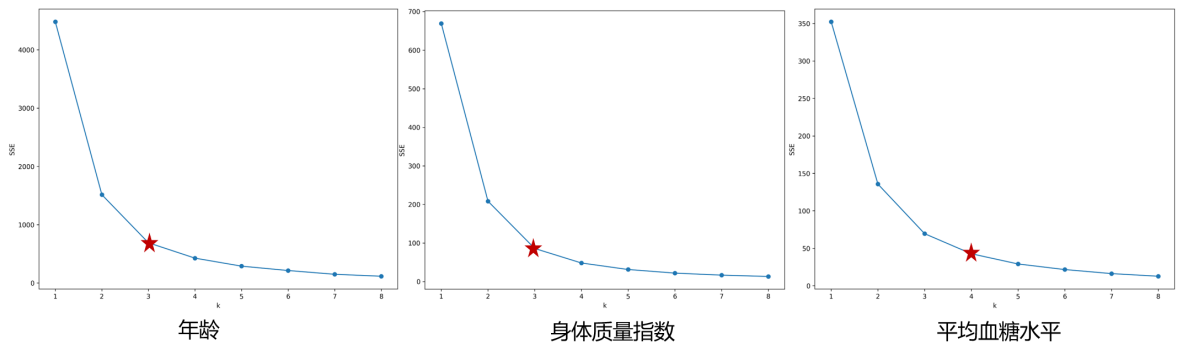


图 9　定类指标聚类肘部图

• **主要指标分组**

基于上述对定量指标的聚类处理，本文将年龄、身体质量指数、平均血糖水平指标分别分为 3 类、3 类和 4 类。而吸烟情况、工作类型、性别指标分别有 3 类、5 类、2 类。由此本文将 5110 名调查对象分为 $3 \times 3 \times 4 \times 3 \times 5 \times 2 = 1080$ 组，而结合数据发现其中仅有 511 组存在数据。因此，本文将调查对象划分为 511 组，并对各组中风情况进行后续分析从而得到几个最易中风的小群体。

5.3 双目标背包规划模型的建立

由5.2可知，本文将 5110 个调查对象划分为 511 组作为分析对象，建立双目标背包规划模型，求解得到符合条件即最易中风的小群体。

设总人数为 N_0，总组数为 n，第 i 组人数为 w_i，第 i 组中风人数为 p_i。首先，引入 0-1 变量 x_i，满足下式：

$$x_i = \begin{cases} 1, \text{表示将第} i \text{组加入小群体} \\ 0, \text{表示不将第} i \text{组加入小群体} \end{cases} \quad (i = 1, 2, \cdots, n). \tag{7}$$

建立双目标背包规划模型如下：

约束条件

• 由题知小群体人数不能超过总人数的 5%，即加入小群体的所有组的人数之和不能超过 $0.05N_0$。因此，需满足约束条件如下式：

$$\sum_{i=1}^{n} w_i x_i \leq 0.05N_0, \quad (i = 1, 2, \cdots, n). \tag{8}$$

• 此外，各组中风人数以及总人数应为自然数，满足下式：

$$p_i, w_i \in N, \quad (i = 1, 2, \cdots, n). \tag{9}$$

目标函数

• 首先，题目要求找到最易中风的小群体，因此在选择组进入背包时，应满足各组中风总人数最大。设小群体内各组中风总人数为 Z_1，则有：

$$\max Z_1 = \sum_{i=1}^{n} p_i x_i, \quad (i = 1, 2, \cdots, n). \tag{10}$$

• 此外，由于一个小群体由多个组组成，求解得到最易中风的小群体，因此小群体需满足其中各组之间的差异率最小，从而保障该小群体组成的合理性。此外，由于各指标对中风的影响程度不同，本文引入问题一中求解得到的**各指标影响程度**量化结果对差异率进行加权。设小群体差异率为 Z_2，则有：

$$\min Z_2 = \sum_{j=1}^{6} \sigma_j W_j, \quad (j = 1, 2, \cdots, 6). \tag{11}$$

其中，W_j 表示第 j 个指标对中风的影响程度，σ_j 表示小群体中各组间第 j 个指标数据的标准差。

各指标对应标准差的计算方法如下：

▷ 对于一项 0-1 变量指标性别指标以及四项定序变量年龄、身体质量指数、平均血糖水平、吸烟状况，对于标准差满足下式：

$$\sigma_j = \sqrt{\frac{\sum_{k=1}^{k_0}(v_{j,k} - \overline{v_j})^2}{k_0}}, \quad (j = 1, 2, 3, 4, 6; k = 1, 2, \cdots, k_0). \tag{12}$$

其中，k_0 为此时小群体中的组数，$v_{j,k}$ 表示小群体中第 k 组的第 j 个指标取值，$\overline{v_j}$ 表示小群体中第 j 个指标对应数据的均值。

▷ 对于工作类型指标，本文采用热编码方式将其转化为 5 组 0-1 变量，计算 5 组 0-1 变量对应的标准差，再对 5 组标准差进行标准化处理。工作类型指标满足下式：

$$\sigma_5 = \frac{\sum_{t=1}^{5} \sigma_{5,t}}{5}, \quad (t = 1, 2, \cdots, 5). \tag{13}$$

其中，$\sigma_{5,t}$ 表示工作类型指标下的第 t 组 0-1 变量的标准差。

综上所述，本文建立双目标背包规划模型展示如下：

$$\min Z = (-Z_1, Z_2).$$

$$\begin{cases} 0 < \sum_{i=1}^{n} w_i x_i \le 0.05 N_0, & (i = 1, 2, \cdots, n), \\ \sigma_j > 0, & (j = 1, 2, \cdots, 6), \\ p_i, w_i \in N, & (i = 1, 2, \cdots, n), \\ \sigma_j = \sqrt{\frac{\sum_{k=1}^{k_0}(v_{j,k} - \overline{v_j})^2}{k_0}}, & (j = 1, 2, 3, 4, 6; k = 1, 2, \cdots, k_0), \\ \sigma_5 = \frac{\sum_{t=1}^{5} \sigma_{5,t}}{5}, & (t = 1, 2, \cdots, 5), \\ Z_1 = \sum_{i=1}^{n} p_i x_i, & (i = 1, 2, \cdots, n), \\ Z_2 = \sum_{j=1}^{6} \sigma_j W_j, & (j = 1, 2, \cdots, 6). \end{cases} \tag{14}$$

5.4 模型求解

5.4.1 目标函数的量纲统一

为了对双目标背包规划模型进行求解，本文对式 (10)(11) 中两个目标函数的量纲进行统一，将其均转化为在 0-1 范围内变化的量。

式 (10) 中最大化中风人数目标转化为：

$$\max Z_1' = \frac{Z_1}{0.05N_0}. \tag{15}$$

式 (11) 中最小化差异率目标转化为：

$$\max Z_2' = \frac{Z_2}{M}. \tag{16}$$

其中，M 为小群体差异率的最大值，即 $M = \max Z_2$。

5.4.2 改进的离散化萤火虫算法求解

基于要解决的双目标背包规划问题，本文针对问题对离散化萤火虫算法进行三个方面的改进如下图10：

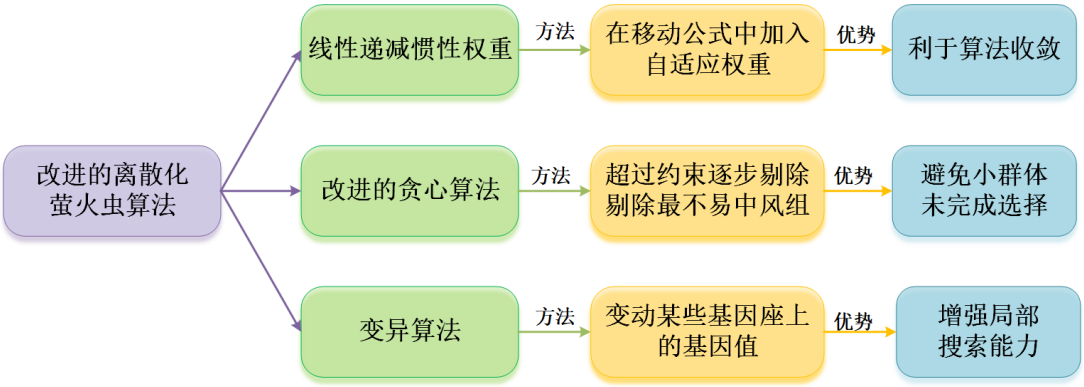


图 10　算法改进示意图

对于线性递减惯性权重，由于较大的权重因子有利于跳出局部最小点，便于全局搜索。而较小的惯性因子则有利于对当前的搜索区域进行精确局部搜索，以利于算法收敛。可以采用线性变化的权重，让惯性权重从最大值 ω_{max} 线性递减到 ω_{min}，从而增加全局搜索和局部搜索能力，随算法迭代次数的变化公式为:

$$\omega_t = \omega_{\max} - \frac{t(\omega_{\max} - \omega_{\min})}{t_{\max}}. \tag{17}$$

加入自适应权重 ω_t 后，有：

$$x_i = \omega_t \times x_i + \beta(x_j - x_i) + \alpha(\text{ rand } - \frac{1}{2}). \tag{18}$$

其中，x_i、x_j 是萤火虫 i、j 的位置，α 为步长因子，$rand$ 为 $[0,1]$ 上均匀分布的随机因子。

对于贪心算法，每次都用一个萤火虫优化算法进行求解，在对一个小群体进行求解时利用贪心算法进行修正，将小群体中中风率最小的小组逐步剔除，直到满足小于 5%，然后再从未选中的小组中，在小于 5% 的约束下，由中风率由高到低添加进入。

变异算法中变异算子的具体操作步骤为：1. 对种群内所有个体以一定变异概率进行判断是否进行变异。2. 对进行变异的个体随机选择变异位置进行变异。3. 对种群中每代最优个体进行变异操作，以增加种群多样性。

综上，得到改进的离散化萤火虫算法伪代码如下：

Algorithm 2 Improved Discretized Firefly Algorithm

1: **Initialize** /* Objective function $f(x)$, $x=(x_1,\cdots,x_d)^T$; Generate initial population of fireflies $x_i(i=1,2,\cdots,n)$; Light intensity I_i at x_i isdetermined by $f(x_i)$; Define light absorption coefficient γ */

2: Optimize the infeasible solution according to the improved greedy algorithm

3: Calculate the objective function value as the maximum width of the firefly and record the optimal position

4: Calculate the relative brightness and attractiveness of fireflies, and determine the direction and distance of fireflies

5: Calculate adaptive weights $x_i=\omega_t\times x_i+\beta(x_j-x_i)+\alpha(\text{ rand }-\frac{1}{2})$

6: Randomly perturb the optimal position and perform mutation operation

7: iteration ← iteration + 1; make a new iteration

8: until the number of iterations reached, output result

5.5 求解结果与分析

5.5.1 规划模型求解结果

本文利用改进的离散化萤火虫算法对上述双目标背包规划模型进行求解，求解过程的迭代曲线如下图11。本文选取最优解集中互不相交的 3 个最优解作为 3 个最易中风小群体，3 个小群体包含的调查对象以及对应特征见附录。上述 3 个小群体共包含了中风人群的 62%，较为全面的体现了调查对象中的中风人群。最优解集表现结果如下图12：

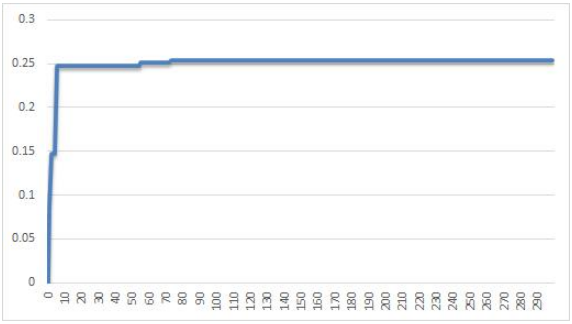


图 11　规划模型求解迭代曲线图

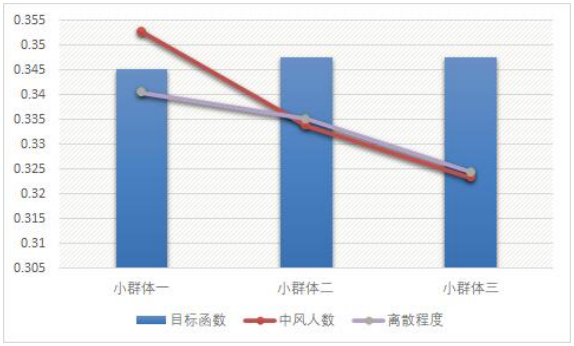


图 12　3 个最易中风小群体示意图

由上图可知三个群体的目标函数分数之间相差很小，可以视为最优解。其中，群体 1 的离散化程度大，但中风人数多，群体 3 离散化程度小，而中风人数少，从侧面也反映出了双目标规划的合理性。

5.5.2 模型检验及拓展分析

- **基于 PCA 降维与高斯混合聚类的宏观分析**

本文求解双目标背包规划问题得到了 3 个最易中风小群体。首先，对每个小群体中的各组进行主成分分析 (PCA)，将其降至二维。再利用高斯混合聚类 (GMM) 对所得两个主成分进行聚类，从而分析模型结果的合理性[5]。对 3 个小群体对应的主成分进行高斯混合聚类得到结果如下图13：

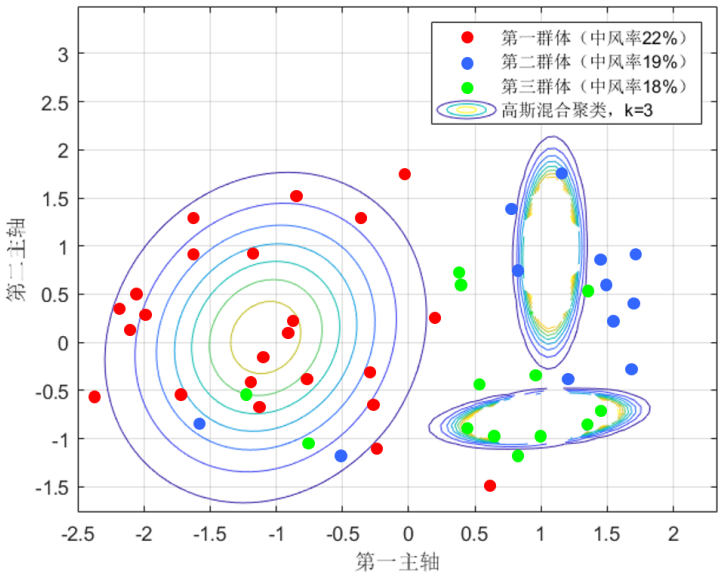


图 13　最易中风小群体主成分聚类图

观察上图可以发现，从**宏观**来看每个最易中风小群体对应点的分布均较为集中，验证了本文规划模型求解结果的可靠性。若按性别和年龄对小群体进行描述，则第一个小

群体表现为高龄男性群体，第二个小群体表现为高龄男女混合群体，第三个小群体表现为高龄女性群体。

- **基于描述性统计的微观分析**

本文对求解所得的三个最易中风小群体进行描述性统计，统计结果如下表：

表 2　描述性统计结果表

	总人数占比	总组数	性别		年龄			血糖					BMI	
			男	女	低龄	中龄	高龄	低	较低	较高	高	低	中	高
1 组	4.677103718	19	19	0	0	0	19	2	0	9	8	5	9	5
2 组	4.931506849	19	11	8	0	0	19	1	2	5	11	6	7	6
3 组	4.883757339	30	0	30	0	0	30	5	4	7	14	3	17	10

表 3　描述性统计结果表

	抽烟频率			工作类型				血糖			
从不	很少	经常	儿童	从不工作	政府工作	私人工作	自体经营	低	较低	较高	高
0	19	0	0	0	1	9	9	2	0	9	8
0	13	6	0	0	3	4	12	1	2	5	11
2	22	6	0	0	17	6	7	5	4	7	14

从**微观**来看，观察以上对于 3 个最易中风小群体各项指标的描述性统计表可以发现，3 个群体中的人群大部分集中在高龄且血糖偏高。第一个小群体均为男性，第二个小群体为男女混合，第 3 个小群体均为女性。此外，3 个小群体各项指标的对应分布均与问题一中求解得到的各指标影响程度基本符合，印证了求解结果的可靠性。

六、 问题三的模型建立与求解

6.1 问题三的描述与分析

问题三要求建立数学模型，分析不同对象的中风可能性大小。首先，本文利用交叉验证-递归消除法对各因素特征进行初步筛选。之后，本文建立 XGBoost 分类模型对

附件 2 中 10 个人的中风可能性进行预测，并在模型求解过程中利用 10-fold 交叉验证对数据集进行处理，利用贝叶斯调参寻找最优模型参数。最后，本文结合问题一中的 TreeSHAP 事后解释器以及问题二中的最易中风小群体对预测结果进行检验。

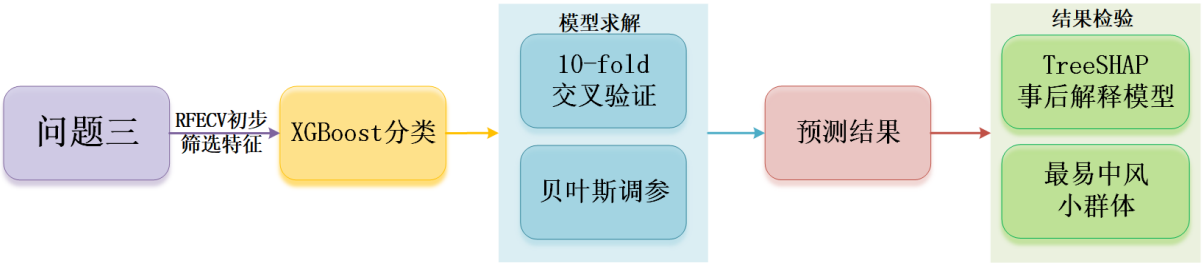


图 14　问题三思路图

6.2 交叉验证-递归消除法初步筛选特征

交叉验证-递归消除法 (RFECV) 是一种与模型相关的变量选择方法，该方法分为递归特征消除和交叉验证两个方面。利用交叉验证-递归消除法完成特征选择后，在递归消除过程中，利用 XGBoost 分类器确定筛选过程中的特征重要性。该方法的步骤展示如下图15：

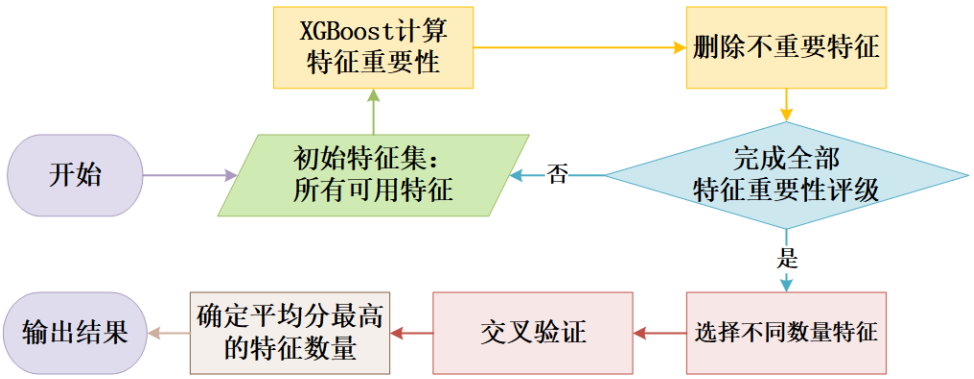


图 15　交叉验证-递归消除法流程图

利用 Python 软件实现上述步骤，10 项因素指标特征重要性评级均为等级 1，由此可知 10 项因素指标均不可排除。因此，本文利用性别、年龄、高血压、心脏病、婚姻、工作类型、住宅类型、平均血糖水平、身体质量指数、吸烟状况 10 项指标进行后续的 XGBoost 分类模型预测。

6.3 基于 XGBoost 的中风可能性分类模型建立

XGBoost 是基于提升树的思想建立的机器学习方法，在 GBDT 的基础上引入了二阶导数和正则化。与同样采用 boosting 思想的 GBDT 算法不同，XGBoost 使用二阶泰勒

展开式使逼近真实值的过程更精确高效[8]。由6.2知不能筛选掉任意一个特征，因此此处的 XGBoost 分类模型对应的预测过程与问题一式 (3)(4) 相同。

模型所采用的目标函数和正则项满足下式：

$$\begin{cases} Y^{(K)} = \sum_{j=1}^{n} \mathrm{L}(y_j, \overline{y}_j^{(K-1)} + f_K(x_j)) + \Omega(f_K) + c, \\ \Omega(f_K) = \gamma T + \frac{1}{2}\lambda \sum_{o=1}^{T} w_0^2 \end{cases} \tag{19}$$

其中，$Y^{(t)}$ 表示第 K 棵树对应的目标函数，$L()$ 表示损失函数，即均方误差，w_0 表示某棵树第 0 个叶子节点的权重。此外，XGBoost 还利用了贪心算法对所有分裂叶子节点进行遍历，并选择目标函数增益最大的节点进行分裂。

基于上述 XGBoost 分类模型，本文对附件 2 中 10 个人的中风可能性进行预测。

6.4 模型求解

在求解上述 XGBoost 分类模型的过程中，本文利用 10-fold 交叉验证对数据集进行处理，并利用贝叶斯调参寻找模型的最优参数。

6.4.1 10-fold 交叉验证

10-fold 交叉验证指的是将数据集划分为 10 等份，每次取其中一组数据作为测试集，其余数据作为训练集进行求解[10]。该方法可以有效地防止过拟合的问题，并找到合适的模型参数。10-fold 交叉验证示意图如下图16：

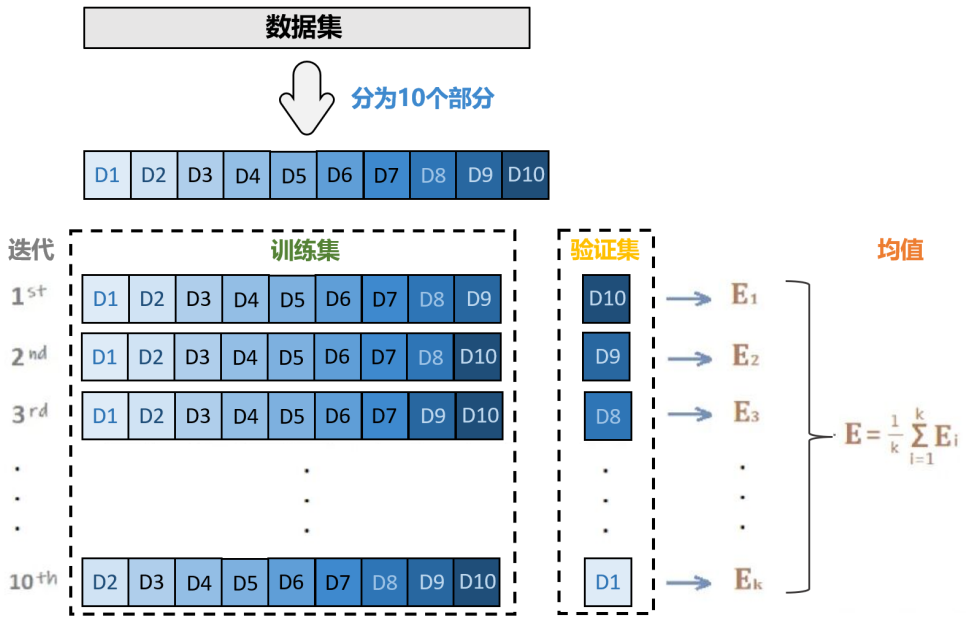


图 16　10-fold 交叉验证示意图

6.4.2 贝叶斯调参

在问题一利用 XGBoost 进行预测的求解过程中，本文利用了随机搜索进行参数调节。为了对 XGBoost 分类模型进行进一步优化，本文引入了贝叶斯调参进行模型参数的调节[6]。

贝叶斯调参流程的伪代码如下：

Algorithm 3 Sequential Model-Based Optimization

Input: f, X, S, M

D $\leftarrow INITSAMPLES(f, X)$

for do $i \leftarrow |D|$ **to** T

  $p(y|x, D) \leftarrow FITMODEL(M, D)$; $x_i \leftarrow arg\max_{x \in X} S(x, p(y|x, D))$

  $y_i \leftarrow f(x_i)$ ▷ Expensive step; $D \leftarrow D \cup (x_i, y_i)$

end for

6.5 求解结果与分析

6.5.1 利用 XGBoost 分类模型的预测结果

将 XGBoost 与随机森林以及 CatBoost 的预测效果进行比较，比较结果如图17。由图可以看出 XGBoost 算法对应 ROC 曲线下方 AUC 面积最接近 1，说明基于 XGBoost 算法的预测模型性能良好，且优于随机森林与 CatBoost 的方法。

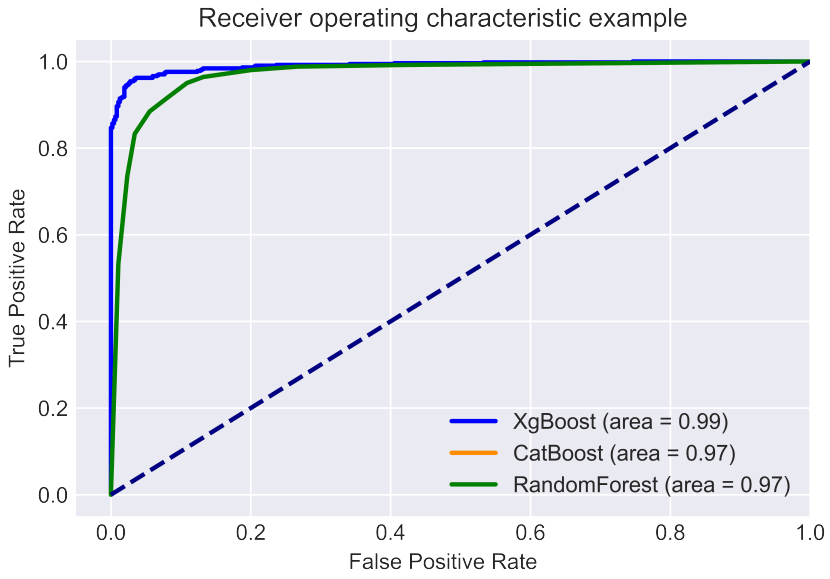


图 17　三种方法对应 ROC 曲线图

本文对上述 XGBoost 分类模型进行了训练和预测，并利用随机森林与 CatBoost 与其进行对比，三种分类算法在交叉验证集上的效果如下表所示：

表 4　三种算法预测效果对比表

	准确率	召回率	精确率	F1
XGBoost	0.964	0.964	0.965	0.964
随机森林	0.869	0.869	0.877	0.868
CatBoost	0.91	0.91	0.913	0.91

由上表数据可知，XGBoost 的准确率、召回率、精确率以及 F1 分数均高于随机森林与 CatBoost。因此，XGBoost 的预测效果优于随机森林与 CatBoost，验证了本文模型选择与建立的合理性、优良性。

本文利用 XGBoost 分类模型对附加 2 中 10 个人的中风可能性预测结果如下表:

表 5　10 人中风可能性预测结果表

排序	id	中风可能性	排序	id	中风可能性
1	3	0.870232	6	5	0.061835
2	4	0.812461	7	9	0.058968
3	7	0.518151	8	8	0.044995
4	10	0.452659	9	2	0.00049
5	1	0.357826	10	6	0.000474

表中 id 为 3、4 的调查对象中风概率显著高于其他对象，分析附件 2 中 3、4 的各项特征发现，id 为 3 的调查对象为高龄男性，该结果与问题二中对最易中风小群体的求解结果相对应，印证了本文建立模型的前后一致性以及预测的合理性。id 为 4 的调查对象平均血糖水平显著高于其他指标且吸烟，基于问题一各指标影响程度结果可知血糖水平与吸烟情况对中风的影响均较大，易知该预测结果具有合理性。

6.5.2 结果检验

- **基于 TreeSHAP 事后解释器的结果检验**

本文结合附件 2 中 10 人的各项指标特征，利用 TreeSHAP 事后解释器求解对应的 shapley 值，从而对上述预测结果进行检验。本文求解 10 个调查对象的 10 项因素指标

对应的 shapley 值。为了方便展示，本文将大于 0 的 shapley 值记为 1，小于 0 的 shapley 值记为-1，对应热力图如下图18:

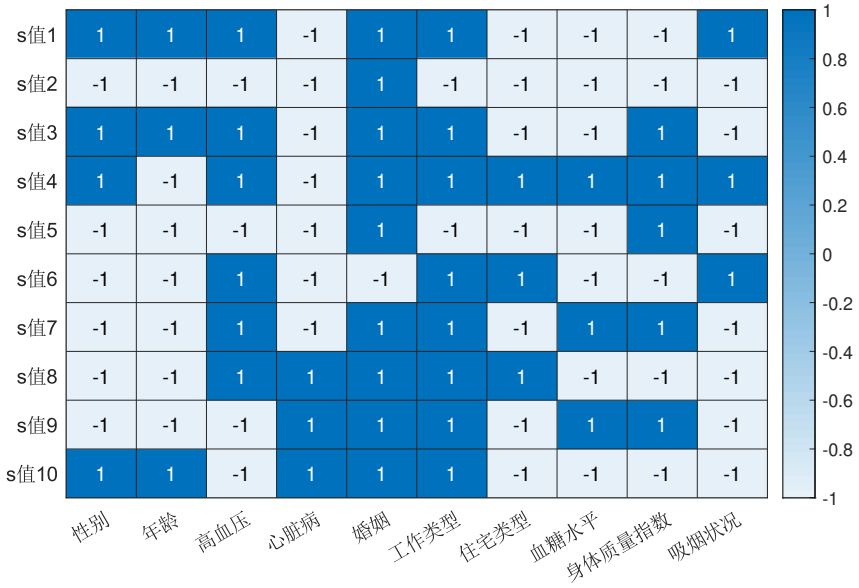


图 18　10 人各因素 shapley 值热力图

上述 shapley 值的热力图显示，id 为 3 和 4 对应的各指标 shapley 值大于 1 的情况最多，该结果与上述预测结果相符。而 id 为 3、4 的调查对象各项指标的 shapley 值与附件 2 中的实际情况基本相符。上述结论印证了本文预测模型的可靠性，也说明了本文问题一与问题三中模型的关联性与一致性。

- **基于最易中风小群体的结果检验**

利用与问题二中相同的分组方法，本文对附件 2 中的 10 个调查对象进行分类，发现 **id 为 3** 的调查对象属于**第 1 个小群体**，即高龄男性群体，**id 为 4** 的调查对象属于**第 3 个小群体**，即高龄女性群体。因此，本问题的预测结果与问题二中所得最易中风小群体的求解结果相符。上述结论印证了本文预测模型的可靠性，也说明了问题三模型与问题二双目标规划模型的关联性与一致性。

七、 模型评价

7.1 模型总结

本文通过 KNN 回归和分类对缺失数据进行填补，运用 SMOTENC 过采样平衡数据集，建立基于 XGBoost 分类器的 SHAP 学习解释器的指标重要程度解释模型，建立基于改进的离散化萤火虫算法的多背包规划模型求解最易中风的小群体，并从宏微观的角度分析求解结果，结合十折交叉验证和贝叶斯调参，建立基于 RFECV 法的 XGBoost 模型对分类结果进行优化，得到合理预测结果。

7.2 模型优缺点分析

优点：

（1）第一问利用 KNN 分类和回归对缺失值进行合理填补，运用 SHAP 学习解释器对分类结果进行客观解释和分析让结果更合理完善；

（2）第二问建立基于改进的离散化萤火虫算法的多背包规划模型求解最易中风的小群体，得到了较好的结果；

（3）利用 PCA-GMM 混合高斯聚类从宏微观的角度解释第二问的求解结果，验证了结果的合理性；

（4）第三问引入十折交叉验证、贝叶斯调参和 RFECV 法对传统 XGboost 模型进行优化，使分类结果更加精确。

缺点：

（1）SHAP 解释器并没有完全消除指标中的多重共线性，指标之间相互干扰的情况并没有很好的解决；

（2）传统贝叶斯调参存在计算量大，占用内存多的特点，降低了求解速度。

7.3 模型改进与展望

首先，本文利用的 SHAP 学习解释器对分类结果进行解释和分析，但并没有完全消除指标之间的多重共线性，可以在此方面运用降维的思维进行更深层次的讨论。其次 XGBoost 的参数调整仍有上升空间，可以用多种调参工具进行优化和改进。最后，在规划模型方面，考虑到最优解集不唯一的情况，可以分别对求解出的最优解进行讨论和分析。

参考文献

[1] LI ZhuoXuan,SHI XinLi,CAO JinDe,WANG XuDong,HUANG Wei.CPSO-XGBoost segmented regression model for asphalt pavement deflection basin area prediction[J].Science China(Technological Sciences),2022,65(07):1470-1481.

[2] Pavankalyan Pavan,B Vani. Improvised Average Gradient for Predicting Multi-Traffic Scene Night Images Using Linear Regression Algorithm Compared with K Means Clustering Algorithm[J]. Electrochemical Society Transactions,2022,107(1).

[3] Sun Yabin,Wang Jiangting,Liu Haonan,Xu Jiwen.Influences of Ho Doping on Structure,Ferroelectric,Energy Storage and Optical Properties of KNN-SYbN Transparent Ceramics[J].Journal of Wuhan University of Technology(Materials Science),2022,37(04):587-590.

[4] Tian, Xiaohua , et al. Screening for shape memory alloys with narrow thermal hysteresis using combined XGBoost and DFT calculation[[J]. Computational Materials Science.(2022) 211. 111519.

[5] Yan-Qian Su,Zhao-Yang Liu,Guo Wei,Chun-Min Zhang.Topical halometasone cream combined with fire needle pretreatment for treatment of primary cutaneous amyloidosis: Two case reports[J].World Journal of Clinical Cases,2022,10(20):7147-7152.

[6] 高虹雷, 门昌骞, 王文剑. 多核贝叶斯优化的模型决策树算法 [J]. 国防科技大学学报,2022,44(03):67-76.

[7] 廖莉, 黄爱平, 汤爱玲, 范燕玲, 吴丹凤. 中风病人跌倒恐惧水平及其相关影响因素的调查分析 [J]. 全科护理,2022,20(22):3148-3150.

[8] 王坤章, 蒋书波, 张豪, 晁征. 基于 XGBoost 的回归-分类-回归寿命预测模型 [J/OL]. 中国测试:1-8[2022-08-19].

[9] 严远亭, 朱原玮, 吴增宝等. 构造性覆盖算法的 SMOTE 过采样方法 *[J]. 计算机科学与探索,2020, 第 14 卷 (6):975-984.

[10] 杨琪威. 线性模型平均中惩罚因子选择的交叉验证法 [D]. 华东师范大学,2022.

附录 A　支撑材料清单

- **图表**

  -特征重要性示意表

  -回归系数表

  -最易中风小群体 1 特征表

  -最易中风小群体 2 特征表

  -最易中风小群体 3 特征表

- **代码**

  -问题一 python 源程序

  -问题一 matlab 源程序

  -问题二 python 源程序

  -问题二 matlab 源程序

  -问题三 python 源程序

附录 B　图表

特征重要性示意表

shap 值	累计	
年龄	3.902747222	390.2747222
身体质量指数	0.864938922	86.49389219
平均血糖水平	0.847080489	84.70804887
吸烟状况	0.408320772	40.83207718
工作类型	0.340008036	34.0008036
性别	0.308989291	30.8989291
结婚	0.19916487	19.91648698
住宅类型	0.198851044	19.88510437
高血压	0.08749292	8.74929195
心脏病	0.062765364	6.27653638

回归系数表

二次项	一次项	常数
0.001258941	0.039999276	-6.57383147
-0.004051564	0.294560928	-5.20822785
0	0.003905437	-0.67697242

最易中风小群体 1 特征表

组别	性别	年龄	血糖	BMI	很少抽烟	从不抽烟	天天抽烟
1	男	高龄	较高	中	1	0	0
1	男	高龄	较高	低	1	0	0
1	男	高龄	较高	高	0	0	0
1	男	高龄	高	中	1	0	0
1	男	高龄	高	中	0	0	0
1	男	高龄	低	中	1	0	0
1	男	高龄	低	低	1	0	0
1	男	高龄	较高	低	1	0	0
1	男	高龄	较高	高	1	0	0
1	男	高龄	高	中	1	0	0
1	男	高龄	高	高	1	0	0
1	男	高龄	高	高	0	0	0
1	男	高龄	较高	中	1	0	0
1	男	高龄	较高	低	1	0	0
1	男	高龄	较高	低	1	0	0
1	男	高龄	较高	高	0	0	0
1	男	高龄	高	中	1	0	0
1	男	高龄	高	中	0	0	0
1	男	高龄	高	中	0	0	0

最易中风小群体 2 特征表

组别	性别	年龄	血糖	BMI	很少抽烟	从不抽烟	天天抽烟
2	女	高龄	高	中	0	1	0
2	女	高龄	高	低	0	1	0
2	女	高龄	高	高	0	1	0
2	男	高龄	较高	低	0	1	0
2	男	高龄	较高	高	0	0	1
2	男	高龄	高	中	0	0	1
2	男	高龄	高	中	0	0	1
2	男	高龄	高	低	0	0	1
2	男	高龄	高	高	0	1	0
2	女	高龄	低	高	0	1	0
2	女	高龄	高	中	0	1	0
2	男	高龄	较高	中	0	1	0
2	男	高龄	较低	高	0	1	0
2	女	高龄	较低	低	0	1	0
2	女	高龄	高	中	0	1	0
2	女	高龄	高	中	0	1	0
2	男	高龄	较高	低	0	1	0
2	男	高龄	较高	高	0	0	1
2	男	高龄	高	低	0	0	1

最易中风小群体 3 特征表

组别	性别	年龄	血糖	BMI	很少抽烟	从不抽烟	天天抽烟
3	女	高龄	较高	中	1	0	0
3	女	高龄	较高	中	0	0	0
3	女	高龄	较高	高	0	0	1
3	女	高龄	高	中	0	0	1
3	女	高龄	高	中	1	0	0
3	女	高龄	高	中	0	0	0
3	女	高龄	高	中	0	0	0
3	女	高龄	高	低	0	0	1
3	女	高龄	高	高	0	0	0
3	女	高龄	低	中	0	0	1
3	女	高龄	低	中	0	0	1
3	女	高龄	低	中	1	0	0
3	女	高龄	低	低	1	0	0
3	女	高龄	较高	中	0	0	0
3	女	高龄	较高	高	1	0	0
3	女	高龄	较低	中	1	0	0
3	女	高龄	较低	中	0	0	0
3	女	高龄	高	高	1	0	0
3	女	高龄	高	高	1	0	0
3	女	高龄	高	高	1	0	0
3	女	高龄	低	中	1	0	0
3	女	高龄	较高	中	0	0	0
3	女	高龄	较高	高	0	0	1
3	女	高龄	较低	中	0	0	0
3	女	高龄	较低	低	1	0	0

附录 C　问题一–python 源程序

```
# 第一问代码数据初分析
import numpy as np
import pandas as pd
import pandas_profiling

data = pd.read_csv("附件1.csv")
data.head()

dataReport = data.profile_report(title="dataAna")
dataReport.to_file("dataAna.html")

# 问题一数据处理
# 引入数据处理和可视化包
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 引入机器学习包
from imblearn.over_sampling import SMOTENC
from sklearn.neighbors import KNeighborsRegressor

data = pd.read_csv("附件1.csv")
# 去除性别other
data.drop(data[data['gender'] == 'Other'].index,inplace=True)
# 去除id列
data.drop(["id"],axis=1,inplace=True)
# 显示有缺失的行
data[data.isna().T.any()]

# 对性别进行处理;Male=1,Female=0
data.iloc[data["gender"] == "Male", 0] = 1
data.iloc[data["gender"] == "Female",0] = 0
# 对ever_married处理;Yes=1,No=0
data.iloc[data["ever_married"] == "Yes",4] = 1
data.iloc[data["ever_married"] == "No",4] = 0
# 对Residence_type进行处理;Rural=1,Urban=0
data.iloc[data["Residence_type"] == "Rural",6] = 1
data.iloc[data["Residence_type"] == "Urban",6] = 0

# 对avg_glucose_level和bmi进行log处理
for col in ['avg_glucose_level', 'bmi','age']:
    data[col] = np.log(data[col])
```

```
def knn_impute(values):
    # 没有缺失值，直接返回
    if ~values.isnull().values.any():
        return values
    # 有缺失值
    df = values.copy()
    num_df = df[["avg_glucose_level","bmi","age"]].copy()
    y_train = num_df.loc[num_df["bmi"].isna()==False,"bmi"]
    X_train = num_df.loc[num_df["bmi"].isna()==False,["avg_glucose_level","age"]]

    X_test = num_df.loc[num_df["bmi"].isna()==True,["avg_glucose_level","age"]]

    knn = KNeighborsRegressor()
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    values.loc[values["bmi"].isna() == True, "bmi"] = y_pred
    return values

dataInter =
     data.groupby(["gender","hypertension","heart_disease","ever_married","work_type","Residence_type"])
# 填补缺失值后的数据
dataInterRes = dataInter.apply(knn_impute)

# 分析分组后,有缺失值组的数量
for key, value in dataInter:
    if ~value.isnull().values.any():
        print(value.shape[0])

dataInterRes.head()
dataInterRes.to_csv("dataInterRes.csv")

# 进行过采样
sm = SMOTENC(random_state=42, categorical_features=[0,2,3,4,5,6,9])
X = dataInterRes.drop(["stroke"],axis=1)
y = dataInterRes["stroke"]
X_over, y_over = sm.fit_resample(X, y)

dataSmote = X_over.copy()
dataSmote['stroke'] = y_over.values
# smote过采样后数据保存
dataSmote.to_csv("dataSmote.csv")

# 两种编码方式来备用
# 哑变量
dataInterRes_Dum = pd.get_dummies(dataInterRes,drop_first=True)
```

```
# label处理
dataInterRes_Lab = dataInterRes.copy()
dataInterRes_Lab['work_type'] =
    dataInterRes_Lab['work_type'].replace({'Private':0,'Self-employed':1,'Govt_job':2,'children':-1,'Never_work
dataInterRes_Lab['smoking_status'] = dataInterRes_Lab['smoking_status'].replace({'never
    smoked':0,'Unknown':1,'formerly smoked':2,'smokes':3})

# label_test处理
dataSmoteTest = dataSmote.copy()
dataSmoteTest['work_type'] =
    dataSmoteTest['work_type'].replace({'Private':0,'Self-employed':1,'Govt_job':2,'children':-1,'Never_worked'
dataSmoteTest['smoking_status'] = dataSmoteTest['smoking_status'].replace({'never
    smoked':0,'Unknown':1,'formerly smoked':2,'smokes':3})
dataSmoteTest.to_csv("dataSmoteTest.csv")
```

附录 D　问题一--matlab 源程序

```
%%第一问
clc;clear
load('shuzhi')
%年龄 ; bmi; 血糖

x1=shuzhi(:,1);
y1=shuzhi(:,2);
x2=shuzhi(:,3);
y2=shuzhi(:,4);
x3=shuzhi(:,5);
y3=shuzhi(:,6);


[f1,g1]=erci(x1,y1);
close
[f2,g2]=erci(x2,y2);

[f3,g3]=yici(x3,y3);

a =   0.001259 ;
b =       0.04;
c =     -6.574 ;
X1=[0:3:90];
Y1=a.*X1.^2+b.*X1+c;

X2=[10:2:60];
Y2=f2.a.*X2.^2+f2.b.*X2+f2.c;
```

```
X3=[50:6:260];
Y3=f3.p1*X3+f3.p2;

plot(X3,Y3,'r-o')
hold on
plot(x3,y3,'.');
legend('回归线','Shape值-bmi')
hold on


figure(1)
createfigure(X1,Y1,x1,y1)
xlabel(['年龄/岁'])


figure(2)
createfigure1(X2,Y2,x2,y2)
xlabel(['千克/平方米'])

figure(3)

createfigure1(X3,Y3,x3,y3)
xlabel(['mg/dL'])

%%回归

function [fitresult, gof] = yici(x1, y1)
%% 一次回归


%% Fit: 'untitled fit 1'.
[xData, yData] = prepareCurveData( x1, y1 );

% Set up fittype and options.
ft = fittype( 'poly1' );

% Fit model to data.
[fitresult, gof] = fit( xData, yData, ft );

% Plot fit with data.
figure( 'Name', 'untitled fit 1' );
h = plot( fitresult, xData, yData );
legend( h, 'y1 vs. x1', 'untitled fit 1', 'Location', 'NorthEast', 'Interpreter', 'none' );
% Label axes
xlabel( 'x1', 'Interpreter', 'none' );
ylabel( 'y1', 'Interpreter', 'none' );
grid on
```

```
function [fitresult, gof] = erci(x1, y1)



%% 开始回归
[xData, yData] = prepareCurveData( x1, y1 );

ft = fittype( 'a*x^2+b*x+c', 'independent', 'x', 'dependent', 'y' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.StartPoint = [0.81551031710201 0.186453042863665 0.490147895980207];


[fitresult, gof] = fit( xData, yData, ft, opts );


figure( 'Name', 'untitled fit 1' );
 plot( fitresult,'-o')
hold on
h=plot( xData, yData );
legend( h, 'y1 vs. x1', 'untitled fit 1', 'Location', 'NorthEast', 'Interpreter', 'none' );

xlabel( 'x1', 'Interpreter', 'none' );
ylabel( 'y1', 'Interpreter', 'none' );
grid on
hold off


%% 画图
function createfigure1(X1, Y1, X2, Y2)
%CREATEFIGURE1(X1, Y1, X2, Y2)
%  X1:  x 数据的向量
%  Y1:  y 数据的向量
%  X2:  x 数据的向量
%  Y2:  y 数据的向量

%  由 MATLAB 于 18-Aug-2022 09:44:28 自动生成

% 创建 figure
figure3 = figure('Name','untitled fit 1');

% 创建 axes
axes1 = axes('Parent',figure3);
hold(axes1,'on');
```

```
% 创建 plot
plot(X1,Y1,'DisplayName','回归线','Marker','hexagram','LineWidth',1,...
    'Color',[0.96078431372549 0.364705882352941 0.364705882352941]);

% 创建 plot
plot(X2,Y2,'DisplayName','Shape值-bmi','MarkerSize',4,'Marker','.',...
    'LineWidth',0.2,...
    'LineStyle','none');

% 取消以下行的注释以保留坐标区的 X 范围
% xlim(axes1,[8.11613388072617 75.7344659618682]);
% 取消以下行的注释以保留坐标区的 Y 范围
% ylim(axes1,[-3.39395281912622 2.6165655880864]);
box(axes1,'on');
% 设置其余坐标区属性
set(axes1,'XGrid','on','XMinorGrid','on','YGrid','on','YMinorGrid','on');
% 创建 legend
legend1 = legend(axes1,'show');
set(legend1,...
    'Position',[0.672738099019223 0.807698426650755 0.216785717282977 0.0869047637212844]);

function createfigure(X1, Y1, X2, Y2)
%CREATEFIGURE(X1, Y1, X2, Y2)
%  X1:  x 数据的向量
%  Y1:  y 数据的向量
%  X2:  x 数据的向量
%  Y2:  y 数据的向量



% 创建 figure
figure1 = figure;


% 创建 axes
axes1 = axes('Parent',figure1);
hold(axes1,'on');

% 创建 plot
plot(X1,Y1,'DisplayName','回归线','Marker','hexagram','LineWidth',1,...
    'Color',[0.96078431372549 0.364705882352941 0.364705882352941]);

% 创建 plot
plot(X2,Y2,'DisplayName','Shape值-年龄','Marker','.','LineStyle','none');

box(axes1,'on');
% 设置其余坐标区属性
```

```
set(axes1,'XGrid','on','XMinorGrid','on','YGrid','on','YMinorGrid','on');
% 创建 legend
legend1 = legend(axes1,'show');
set(legend1,...
    'Position',[0.177134045204997 0.813412696596176 0.153602728569751 0.0869047637212843]);
```

附录 E　问题二–python 源程序

```
# 问题二：模型求解/改进的萤火虫算法
# 数据处理,可视化
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
# 常规模块
import copy
import time
import math
import random
# 引入机器学习包
from sklearn.cluster import KMeans

# age
SSE = []
for k in range(1,9):
    estimator = KMeans(n_clusters=k,random_state=0)
    estimator.fit(data["age"].values.reshape(-1, 1))
    SSE.append(estimator.inertia_)
X = range(1,9)
plt.figure(figsize=(10,8),dpi=300)
plt.xlabel('k')
plt.ylabel('SSE')
plt.plot(X,SSE,'o-')
plt.show()
# avg
SSE = []
for k in range(1,9):
    estimator = KMeans(n_clusters=k,random_state=0)
    estimator.fit(data["avg_glucose_level"].values.reshape(-1, 1))
    SSE.append(estimator.inertia_)
X = range(1,9)
plt.figure(figsize=(10,8),dpi=300)
plt.xlabel('k')
```

```
plt.ylabel('SSE')
plt.plot(X,SSE,'o-')
plt.show()
# bmi
SSE = []
for k in range(1,9):
    estimator = KMeans(n_clusters=k,random_state=0)
    estimator.fit(data["bmi"].values.reshape(-1, 1))
    SSE.append(estimator.inertia_)
X = range(1,9)
plt.figure(figsize=(10,8),dpi=300)
plt.xlabel('k')
plt.ylabel('SSE')
plt.plot(X,SSE,'o-')
plt.show()

# 聚类
dataKmeans = data.copy()
km = KMeans(n_clusters=3,random_state=0).fit(data["age"].values.reshape(-1, 1))
dataKmeans['age_k'] = km.labels_
km = KMeans(n_clusters=4,random_state=0).fit(data["avg_glucose_level"].values.reshape(-1, 1))
dataKmeans["avg_k"] = km.labels_
km = KMeans(n_clusters=3,random_state=0).fit(data["bmi"].values.reshape(-1, 1))
dataKmeans["bmi_k"] = km.labels_

data_onehot = pd.get_dummies(dataKmeans, columns=['smoking_status','work_type'])
# 用于分组的数据
dataFor = data_onehot.iloc[:,[8,0,9,10,11,12,13,14,15,16,17,18,19,20]].copy()
dataFor.head(2)
dataFor.to_excel("test.xlsx")

# 分组
dataGroup = dataFor.groupby(dataFor.columns[1:].values.tolist())

def groupInfo(values):
    # 记录index
    recordIn = values.index.tolist()
    # 该组的数量
    num = values.shape[0]
    # 该组的stroke人数
    stroNum = values["stroke"].sum()
    # 最后的结果
    groupI = pd.Series([recordIn, num, stroNum],index=["recordIn", "num_all", "num_stroke"])
    return groupI

# group后的信息记录
groupEnd = dataGroup.apply(groupInfo).reset_index()
```

```
print(groupEnd.shape)
groupEnd.head(2)

# 不同组之间距离的函数
def distance(item1, item2):
    temp1 = np.array(item1[0:12].copy(),dtype='int32')
    temp2 = np.array(item2[0:12].copy(),dtype='int32')
    temp = temp1 - temp2
    tempAbs = np.absolute(temp)
    # k-means后的那些数据
    tempS1 = tempAbs[0:3].sum()
    # smoke的距离
    tempS2 = tempAbs[3:7].sum()
    # work_type的距离
    tempS3 = tempAbs[7:12].sum()
    res = tempS1 + tempS2 + tempS3
    return res

# 计算一个群体离散程度
# 并除以最大离散程度进行0-1化——5
def discrete(values):
    if values.shape[0] == 0:
        return 1
    values = values.copy()
    # gender
    dis1 = np.std(values.iloc[:,0].values)
    # age_k
    dis2 = np.std(values.iloc[:,1].values)
    # avg_k
    dis3 = np.std(values.iloc[:,2].values)
    # bmi_k
    dis4 = np.std(values.iloc[:,3].values)
    if len(values.shape) == 1:
        dis5 = 0
        dis6 = 0
    else:
        # smoking
        ave = np.full(4,values.shape[0]/4)
        rea = values.iloc[:,4:8].values.sum(axis=0).copy()
        stan = (ave - rea)**2
        stan = stan.sum()
        stanMax = (np.array([values.shape[0], 0, 0, 0]) - ave)**2
        stanMax = stanMax.sum()
        dis5 = (1 - stan/stanMax)/2

        # work_type
```

```
        ave = np.full(5,values.shape[0]/5)
        rea = values.iloc[:,8:13].values.sum(axis=0).copy()
        stan = (ave - rea)**2
        stan = stan.sum()
        stanMax = (np.array([values.shape[0], 0, 0, 0, 0]) - ave)**2
        stanMax = stanMax.sum()
        dis6 = (1 - stan/stanMax)/2
    # print(dis1,dis2,dis3,dis4,dis5,dis6)
    res = (dis1 + dis2 + dis3 + dis4 + dis5 + dis6)/5
    return res

# 用于萤火虫算法
groupS = groupEnd.copy()
# 组数
groupNum = groupS.shape[0]
groupNum

# 计算目标函数
def fxCal(xn):
    xn = xn.copy()
    # stroke人数
    num_stroke = (groupS["num_stroke"].values*xn).sum(axis = 1)
    num_stroke /= 255
    dis = np.zeros(xn.shape[0])
    # print(num_stroke)
    # 离散程度
    xn[xn>=0.5] = 1
    xn[xn<0.5] = 0
    for i in range(xn.shape[0]):
        temp = np.arange(xn.shape[1])
        dis[i] = discrete(groupS.iloc[temp[xn[i]==1],:])
    res = 4*num_stroke - dis
    return res
fxCal(np.around(np.random.random((30, 511))))

# 贪心策略修复解
def f_GA(xn, M, V):
    xn = xn.copy()
    # 目前被选中的总人数
    nowPerNum = (groupS["num_all"].values*xn).sum(axis = 1).copy()
    num = xn.shape[1]
    # 占比
    Perrate = groupS["num_stroke"].values/groupS["num_all"].values
    # 第几小的索引
    index = np.argsort(Perrate)
    fn = Perrate[index]
    for i in range(M):
```

```
        if nowPerNum[i] > V:
            for j in range(num):
                if xn[i,index[j]] == 1:
                    nowPerNum[i] = nowPerNum[i] - groupS["num_all"].values[index[j]]
                    if nowPerNum[i] > V:
                        xn[i,index[j]] = 0
                    else:
                        xn[i,index[j]] = 0
                        # print(nowPerNum)
                        break
        # rest有无不重要
        rest = []
        index2 = []
        for k in range(xn.shape[1]):
            if xn[i,index[k]] == 0:
                rest.append(fn[index[k]])
                index2.append(index[k])
        # 比重由大到小排
        rest = np.flipud(rest)
        index2 = np.flipud(index2)
        for j in range(len(rest)):
            nowPerNum[i] += groupS["num_all"].values[index2[j]]
            if nowPerNum[i] <= V:
                xn[i,index2[j]] = 1
                continue
            else:
                nowPerNum[i] -= groupS["num_all"].values[index2[j]]
    return xn

# 对种群进行排序操作
def f_sort(xn):
    xn = xn.copy()
    fx_for = fxCal(xn)
    # print(fx_for)
    # 按照亮度（适应度函数）进行排序
    index = np.argsort(-fx_for)
    # 亮度的排序结果
    fn = fx_for[index]
    # 种群数量
    num = xn.shape[0]
    # 对种群进行排序
    xn_temp = copy.deepcopy(xn)
    for i in range(num):
        xn[i,:] = xn_temp[index[i]]
    return xn, fn

# 萤火虫位置更新
```

```
def f_move(xn, Light, gamma, wt, beta0, alpha):
    xn = xn.copy()
    Light = Light.copy()
    num = xn.shape[0]
    for i in range(num):
        for j in range(num):
            if i==j:
                continue
            # 种群距离函数
            temp = (xn[i,:] - xn[j,:])**2
            dis = pow(temp.sum(), 0.5)
            # 计算亮度
            l_ri = Light[i]*math.exp(-gamma*(dis**2))
            l_rj = Light[j]*math.exp(-gamma*(dis**2))
            # beta计算
            beta = beta0*math.exp(-(gamma*(dis**2)))
            # 随机扰动项
            tmpf = alpha*(random.random()-0.5)
            # 比较亮度
            if l_ri < l_rj:
                xn[i,:] = xn[i]*(wt - beta) + xn[j,:]*beta + tmpf
            else:
                xn[j,:] = xn[j]*(wt - beta) + xn[i,:]*beta + tmpf
    return xn

# 萤火虫变异
# 将处于最优位置的萤火虫进行随机扰动
# 加强局部搜索能力
def f_variation(xn, pm):
    xn = xn.copy()
    M = xn.shape[0]
    N = xn.shape[1]
    temp = np.random.choice(xn.shape[1], 2, replace=False)
    # 对最优值进行随机扰动
    xn[0,temp[0]], xn[0,temp[1]] = xn[0,temp[1]], xn[0,temp[0]]
    # 变异判断
    for i in range(M):
        if random.random() < pm:
            for j in range(N):
                xn[i,j] = random.random()
    return xn

# 萤火虫迭代过程

def f_maxcon(iter_max, xn, alpha, gamma, beta0, wmax, wmin, pm, M, V):
    xn = xn.copy()
    xbest_set = np.zeros((iter_max,xn.shape[1]))
```

```
    LightBest_set = np.zeros(iter_max)
    for i in range(iter_max):
        # 动态权重
        wt = wmax - ((i+1)*(wmax - wmin))/iter_max # 线性
        # 贪心策略修复解
        xn = f_GA(xn=xn , M=M, V=V)
        # 进行种群排序
        xn, LightN = f_sort(xn=xn)
        # 最好的x值
        xbest = xn[0,:].copy()
        # 目标函数值
        LightBest = LightN[0].copy()
        # 记录当前迭代值
        xbest_set[i,:] = xbest
        LightBest_set[i] = LightBest
        # 打印结果
        print("第%d次迭代" % i)
        # print(xbest)
        print(LightBest)
        # 种群更新
        xn = f_move(xn=xn,Light=LightN,gamma=gamma,wt=wt,beta0=beta0,alpha=alpha)
        xn, LightN = f_sort(xn=xn) #再次排序
        # 种群变异
        xn = f_variation(xn=xn,pm=pm)
        # 种群离散化
        xn[xn>=0.5] = 1
        xn[xn<0.5] = 0
    return xbest_set,LightBest_set


# 改进的萤火虫算法求解

# 初始化参数
alpha = 0.1 # 步长因子
gamma = 0.4 # 介质吸收因子
beta0 = 0.8 # 最大吸引度
wmax = 1.0
wmin = 0.4 # 权重
pm = 0.01 # 变异概率

# 最大容量
V = 255 # 小群体的数量，不能超过5%
num = groupNum # 组的个数
# 重量——对应组内人数
# 价值——对应组内中风人数
```

```
iter_max = 300 # 迭代次数

# 这个用于多次运行，并对结果进行求值分析
# 可以先跑一次看看
runtime = 1 # 运行次数

M = 30 # 萤火虫种群个数

# 初始化生成随机解

xn = np.random.choice([0, 1], size=(M, num), p=[.8, .2])
xn = xn.astype(np.float64)
f_best = [] # 存放每次运行的最优值
sum_f_best = [] # 每次运行的最优值求和

# 计算结果
xbest_set,LightBest_set = f_maxcon(iter_max=iter_max,xn=xn,alpha=alpha,\
    gamma=gamma,beta0=beta0,wmax=wmax,wmin=wmin,pm=pm, M=M, V=V)

figData = pd.DataFrame(data=LightBest_set)
figData.to_excel("画递归图.xlsx")

num_strokeGlobal = (groupS["num_stroke"].values*xbest_set).sum(axis = 1)
num_strokeGlobal /= 255
disGlobal = np.zeros(xbest_set.shape[0])
for i in range(xbest_set.shape[0]):
    temp = np.arange(xn.shape[1])
    disGlobal[i] = discrete(groupS.iloc[temp[xbest_set[i]==1],:])

fEND = LightBest_set.max() #最优解
xEND = xbest_set[LightBest_set.argmax()]
num_strokeEND = num_strokeGlobal[LightBest_set.argmax()]
disEND = disGlobal[LightBest_set.argmax()]
temp = groupS.index
groupS.iloc[temp[xEND==1],:].head(2)

groupS.iloc[temp[xEND==1],:].to_excel("群体一.xlsx")
print("目标函数:%.3f 中风率: %.3f 离散程度: %.3f" % (fEND, num_strokeEND, disEND))

# 去除之前的群体后
groupS2 = groupEnd.iloc[temp[xEND!=1],:].copy()
groupS = groupS2.copy()
groupS.shape

# 计算结果
xn = np.random.choice([0, 1], size=(M, groupS.shape[0]), p=[.8, .2])
xn = xn.astype(np.float64)
```

```
xbest_set,LightBest_set = f_maxcon(iter_max=iter_max,xn=xn,alpha=alpha,\
    gamma=gamma,beta0=beta0,wmax=wmax,wmin=wmin,pm=pm, M=M, V=V)



num_strokeGlobal = (groupS["num_stroke"].values*xbest_set).sum(axis = 1)
num_strokeGlobal /= 255
disGlobal = np.zeros(xbest_set.shape[0])
for i in range(xbest_set.shape[0]):
    temp = np.arange(xn.shape[1])
    disGlobal[i] = discrete(groupS.iloc[temp[xbest_set[i]==1],:])

fEND2 = LightBest_set.max() #最优解
xEND2 = xbest_set[LightBest_set.argmax()]
num_strokeEND2 = num_strokeGlobal[LightBest_set.argmax()]
disEND2 = disGlobal[LightBest_set.argmax()]
temp = np.arange(groupS2.shape[0])
groupS.iloc[temp[xEND2==1],:].head(2)

groupS.iloc[temp[xEND2==1],:].to_excel("群体二.xlsx")
print("目标函数:%.3f 中风率: %.3f 离散程度: %.3f" % (fEND2, num_strokeEND2, disEND2))

temp = np.arange(groupS2.shape[0])

# 去除之前的群体后
# groupS2 = groupEnd.iloc[temp[xEND2!=1],:].copy()
# groupS = groupS2.copy()
groupS = groupS.ilco[temp[xEND2!=1],:].copy()
groupS.shape

# 计算结果
xn = np.random.choice([0, 1], size=(M, groupS.shape[0]), p=[.8, .2])
xn = xn.astype(np.float64)
xbest_set,LightBest_set = f_maxcon(iter_max=iter_max,xn=xn,alpha=alpha,\
    gamma=gamma,beta0=beta0,wmax=wmax,wmin=wmin,pm=pm, M=M, V=V)

num_strokeGlobal = (groupS["num_stroke"].values*xbest_set).sum(axis = 1)
num_strokeGlobal /= 255
disGlobal = np.zeros(xbest_set.shape[0])
for i in range(xbest_set.shape[0]):
    temp = np.arange(xn.shape[1])
    disGlobal[i] = discrete(groupS.iloc[temp[xbest_set[i]==1],:])

fEND3 = LightBest_set.max() #最优解
xEND3 = xbest_set[LightBest_set.argmax()]
num_strokeEND3 = num_strokeGlobal[LightBest_set.argmax()]
disEND3 = disGlobal[LightBest_set.argmax()]
temp = np.arange(groupS2.shape[0])
```

```
groupS.iloc[temp[xEND3==1],:].head(2)

groupS.shape

groupS.iloc[temp[xEND3==1],:].to_excel("群体三.xlsx")
print("目标函数:%.3f 中风率: %.3f 离散程度: %.3f" % (fEND3, num_strokeEND3, disEND3))
```

附录 F　问题二–matlab 源程序

```
%%第二问高斯混合聚类

%% 混合高斯聚类
clear;clc
load data
classes=unique(species)
[a,score]=pca(meas,'NumComponents',2);
%% pca降维成两个主成分

options=statset('Display','final');
gm=fitgmdist(score,3,'Options',options);
Mu=gm.mu
gscatter(score(:,1),score(:,2),species)
h=gca
hold on
ezcontour(@(x,y)pdf(gm,[x,y]),[h.XLim h.YLim],100);
title('Pca-GMM聚类')
xlabel('第一主轴')
ylabel('第二主轴')
grid on
box on
% set(gcf,'Position',[100 100 500 400])









%
% gm=fitgmdist(score,3,'Options',options)
%
% clc;clear
% rng default
```

```
% mu1=[1 2]
% sigmal=[3 0.2;0.2 2]
% mu2=[-1 -2];
% sigma2=[2 0;0 1];
% %%高斯样本
% X=[mvnrnd(mu1,sigmal,200);mvnrnd(mu2,sigma2,100)];
% n=size(X,1);
% scatter(X(1:200,1),X(1:200,2),15 , 'ro','filled');
% hold on ;box on
% scatter(X(201:end,1),X(201:end,2),15,'bo','filled');
% set(gcf,'position',[100 100 400 400]);
% set(gca,'FontSize',10);
%
%
%
% hold on
% ezcontour(@(x,y)pdf(gm,[x,y]),[-6 6],[-6 6]);
% title('散点图和GMM')
% set(gcf,'Position',[100 100 450 360]) ;
%
%
```

附录 G　问题三–python 源程序

```
# 问题三:RFECV进行特征选择与贝叶斯调参
# 引入数据处理包
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import seaborn as sns
# 引入机器学习包
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import classification_report
from xgboost import XGBClassifier
from sklearn.feature_selection import RFECV
from sklearn.metrics import roc_auc_score,roc_curve,auc
from sklearn import metrics
# 贝叶斯调参
from bayes_opt import BayesianOptimization
from sklearn.model_selection import cross_val_score
# shap 解释器
import shap
import warnings
```

```
warnings.filterwarnings('ignore')
dataSmote = pd.read_csv("dataSmote.csv",index_col=0)
dataSmote.head(2)
# 分类数据处理
dataSmote['work_type'] =
    dataSmote['work_type'].replace({'Private':0,'Self-employed':1,'Govt_job':2,'children':-1,'Never_worked':-2}
dataSmote['smoking_status'] = dataSmote['smoking_status'].replace({'never
    smoked':0,'Unknown':1,'formerly smoked':2,'smokes':3})
dataSmote.head(2)

# 贝叶斯调参(将调参结果作为参考)

# 简单整理数据用于贝叶斯调参
x_train_all = dataSmote.drop(['stroke'],axis=1)
y_train_all = dataSmote['stroke']

def xgb_cv(max_depth, learning_rate, n_estimators, min_child_weight, subsample,
    colsample_bytree, reg_alpha, gamma):
   val = cross_val_score(estimator=XGBClassifier(max_depth=int(max_depth),
                                        learning_rate=learning_rate,
                                        n_estimators=int(n_estimators),
                                        min_child_weight=min_child_weight,
                                        subsample=max(min(subsample, 1), 0),
                                        colsample_bytree=max(min(colsample_bytree, 1), 0),
                                        reg_alpha=max(reg_alpha, 0), gamma=gamma,
                                            objective='reg:squarederror',
                                        booster='gbtree',
                                        seed=42), X=x_train_all, y=y_train_all,
                                            scoring="roc_auc",
                      cv=10).mean()
   return val

xgb_bo = BayesianOptimization(xgb_cv, pbounds={'max_depth': (1, 10),
                                       'learning_rate': (0.01, 0.3),
                                       'n_estimators': (1, 200),
                                       'min_child_weight': (0, 20),
                                       'subsample': (0.001, 1),
                                       'colsample_bytree': (0.01, 1),
                                       'reg_alpha': (0.001, 20),
                                       'gamma': (0.001, 10)})
xgb_bo.maximize(n_iter=100, init_points=10)

X_train, X_test, y_train, y_test = train_test_split(dataSmote.drop(['stroke'],axis=1),
                                            dataSmote['stroke'],
                                            random_state=0,test_size=0.1)
xgbtest = XGBClassifier(
   seed=42,
```

```
    learning_rate = 0.3,
    max_depth = 7,
    min_child_weight = 5,
    n_estimators = 400,
    gamma = 0,
)

xgbtest.fit(X_train, y_train)
y_pred = xgbtest.predict(X_test)
y_pred_proba = xgbtest.predict_proba(X_test)
print(classification_report(y_test, y_pred))
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred_proba[:,1], pos_label=1)
print(auc(fpr, tpr))

xgbtest.feature_importances_

# RFECV-xgboost计算特征重要性
# 使用贝叶斯调参得出的参考超参数，并经过调整后
model = XGBClassifier(
    seed=42,
    learning_rate = 0.3,
    max_depth = 7,
    min_child_weight = 5,
    n_estimators = 400,
    gamma = 0,
)
selector = RFECV(estimator=model, step=1,cv=10)
selector.fit(x_train_all, y_train_all)
selector.get_support(True)
selector.grid_scores_

pd.DataFrame(data=selector.grid_scores_).to_excel("RFECV.xlsx")
print("Optimal number of features : %d" % selector.n_features_)
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(selector.grid_scores_) + 1), selector.grid_scores_)
plt.show()

print(selector.ranking_)
selector.grid_scores_

# 得出shap值，用于后续分析
explainer = shap.TreeExplainer(xgbtest)
shap.initjs() #初始化JS
shap_values = explainer.shap_values(dataSmote.drop(['stroke'],axis=1))
shape_df = pd.DataFrame(data=shap_values,columns=dataSmote.columns[:-1])
```

```
shape_df.to_excel("shap_data.xlsx")


# 问题三预测结果以及求shap值进行分析
# 引入数据处理包
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import seaborn as sns
# 引入机器学习包
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import classification_report
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV
from sklearn.metrics import roc_auc_score,roc_curve,auc
from sklearn import metrics
# shap 解释器
import shap
import warnings
warnings.filterwarnings('ignore')
dataSmote = pd.read_csv("dataSmote.csv",index_col=0)
dataSmote.head(2)
# 分类数据处理

dataSmote['work_type'] =
    dataSmote['work_type'].replace({'Private':0,'Self-employed':1,'Govt_job':2,'children':-1,'Never_worked':-2}
dataSmote['smoking_status'] = dataSmote['smoking_status'].replace({'never
    smoked':0,'Unknown':1,'formerly smoked':2,'smokes':3})
dataSmote.head(2)
X_train, X_test, y_train, y_test = train_test_split(dataSmote.drop(['stroke'],axis=1),
                                                    dataSmote['stroke'],
                                                    random_state=0,test_size=0.1)
xgbtest = XGBClassifier(
    seed=42,
    learning_rate = 0.3,
    max_depth = 7,
    min_child_weight = 5,
    n_estimators = 400,
    gamma = 0,
)

xgbtest.fit(X_train, y_train)
y_pred_proba = xgbtest.predict_proba(X_test)
```

```
cat = RandomForestClassifier(n_estimators=10)
cat.fit(X_train, y_train)
y_pred_rf = cat.predict(X_test)

rf = RandomForestClassifier(n_estimators=10)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

y_pred_proba_xgb = xgbtest.predict_proba(X_test)
fpr_xgb, tpr_xgb, thresholds = metrics.roc_curve(y_test,y_pred_proba_xgb[:,1], pos_label=1)
roc_auc_xgb = auc(fpr_xgb, tpr_xgb)

y_pred_proba_rf = rf.predict_proba(X_test)
fpr_rf, tpr_rf, thresholds = metrics.roc_curve(y_test,y_pred_proba_rf[:,1], pos_label=1)
roc_auc_rf = auc(fpr_rf, tpr_rf)

y_pred_proba_cat = cat.predict_proba(X_test)
fpr_cat, tpr_cat, thresholds = metrics.roc_curve(y_test,y_pred_proba_rf[:,1], pos_label=1)
roc_auc_cat = auc(fpr_cat, tpr_cat)

plt.style.use('seaborn-darkgrid')
# 4.绘图
plt.figure(dpi=1000)
lw = 2
plt.plot(
    fpr_xgb, tpr_xgb,
    color="blue",
    lw=lw,
    label="XgBoost (area = %0.2f)" % roc_auc_xgb
)
plt.plot(
    fpr_cat, tpr_cat,
    color="darkorange",
    lw=lw,
    label="CatBoost (area = %0.2f)" % roc_auc_cat,
)
plt.plot(
    fpr_rf, tpr_rf,
    color="green",
    lw=lw,
    label="RandomForest (area = %0.2f)" % roc_auc_rf
)

plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([-0.05, 1.0])
plt.ylim([-0.05, 1.05])
```

```
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()

# 预测集给出预测结果
pre = pd.read_excel("附件2.xlsx")
pre.head(2)

pre.drop(["id","stroke"],axis=1,inplace=True)
# 对性别进行处理;Male=1,Female=0
pre.iloc[pre["gender"] == "Male", 0] = 1
pre.iloc[pre["gender"] == "Female",0] = 0
pre["gender"] = pre["gender"].astype("int32")
# 对ever_married处理;Yes=1,No=0
pre.iloc[pre["ever_married"] == "Yes",4] = 1
pre.iloc[pre["ever_married"] == "No",4] = 0
pre["ever_married"] = pre["ever_married"].astype("int32")
# 对Residence_type进行处理;Rural=1,Urban=0
pre.iloc[pre["Residence_type"] == "Rural",6] = 1
pre.iloc[pre["Residence_type"] == "Urban",6] = 0
pre["Residence_type"] = pre["Residence_type"].astype("int32")

# 对avg_glucose_level和bmi进行log处理
for col in ['avg_glucose_level', 'bmi','age']:
    pre[col] = np.log(pre[col])

pre['work_type'] =
    pre['work_type'].replace({'Private':0,'Self-employed':1,'Govt_job':2,'children':-1,'Never_worked':-2}).asty
pre['smoking_status'] = pre['smoking_status'].replace({'never smoked':0,'Unknown':1,'formerly
    smoked':2,'smokes':3}).astype(np.uint8)

pre

xgbtest.predict_proba(pre)

res = pd.DataFrame(data=xgbtest.predict_proba(pre),columns=["未中风","中风"])
res
res.to_excel("第三问结果.xlsx")

# 得出shap值，用于后续分析
explainer = shap.TreeExplainer(xgbtest)
shap.initjs() #初始化JS
shap_values = explainer.shap_values(pre)
shape_df = pd.DataFrame(data=shap_values,columns=pre.columns)
```

```
shape_df.to_excel("shap_data_pre.xlsx")
```