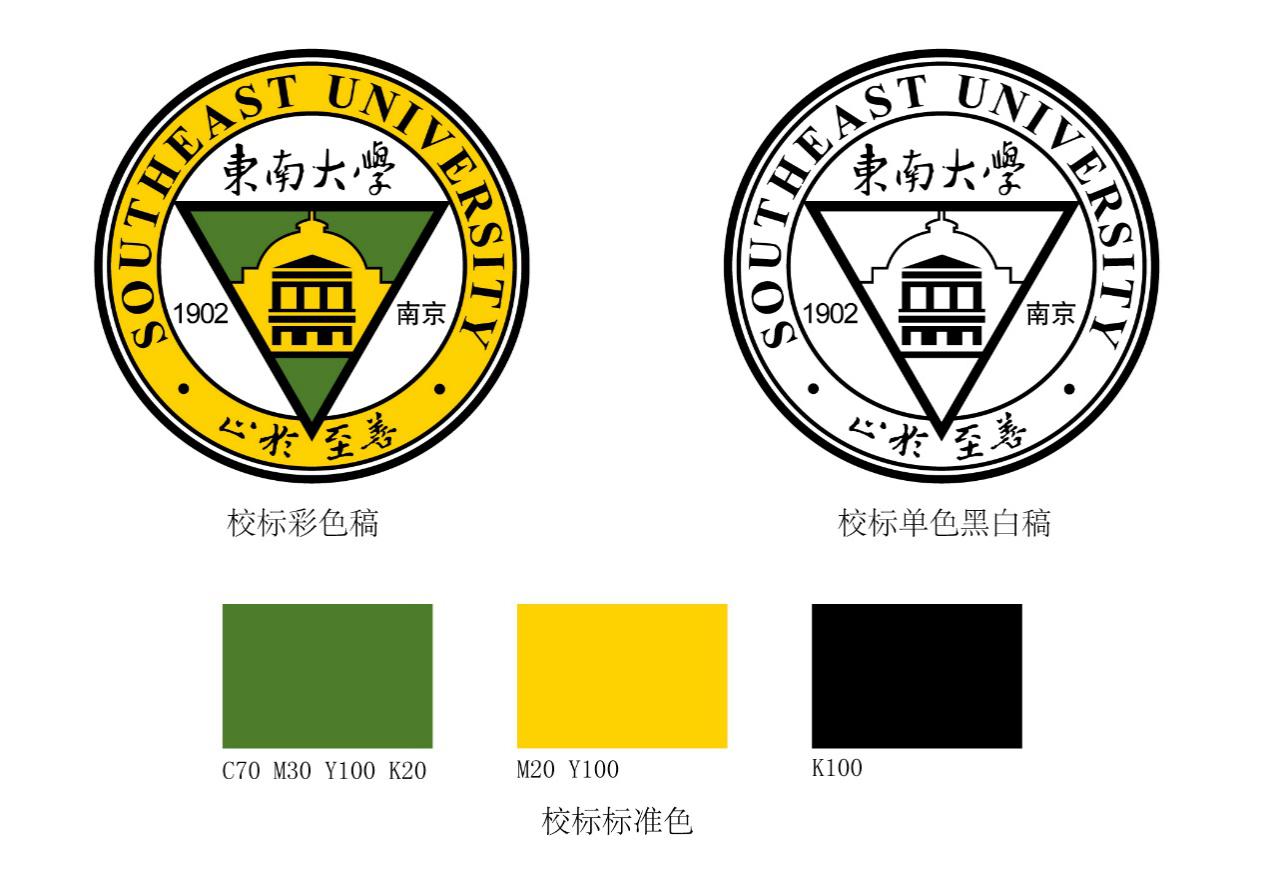


2023-2024学年

数理统计课程

**数据分析报告**



小组成员：张启元

吴章铭

蔺文谢

2023年11月10日

**目 录**

[1. 不同基线数据下患者出院状态分布图及其统计学描述 1](#_Toc150010352)

[1.1. 性别与出院状态分布 1](#_Toc150010353)

[1.2. 年龄与出院状态分布 1](#_Toc150010354)

[1.3. 体重与出院状态分布 2](#_Toc150010355)

[1.4. 身高与出院状态分布 3](#_Toc150010356)

[2. 差异性分析 4](#_Toc150010357)

[2.1. 提出假设 4](#_Toc150010358)

[2.2. 计算卡方值 4](#_Toc150010359)

[2.3. 确定显著性水平 5](#_Toc150010360)

[2.4. 卡方分析结果 5](#_Toc150010361)

[3. 出院死亡概率后验分布以及估计值随时间变化曲线 7](#_Toc150010362)

[3.1. 出院死亡概率*θ*后验分布 7](#_Toc150010363)

[3.2. 出院死亡概率估计随观察量变化 8](#_Toc150010364)

[4. 二元混合高斯分布参数估计与模型拟合 10](#_Toc150010365)

[4.1. 随机数x概率密度 10](#_Toc150010366)

[4.2. 不同来源随机数x的概率密度分布与相对概率 10](#_Toc150010367)

[4.3. 分类错误概率与ROC曲线 11](#_Toc150010368)

[4.4. EM算法估计二元混合高斯分布参数 12](#_Toc150010369)

[4.5. 训练集模型拟合与测试集预测 14](#_Toc150010370)

[5. 相关性分析 18](#_Toc150010371)

[5.1. 数据预处理 18](#_Toc150010372)

[5.2. 皮尔逊相关性分析方法 18](#_Toc150010373)

[5.3. 皮尔逊相关性检验分析 20](#_Toc150010374)

[6. Logit回归模型 22](#_Toc150010375)

[6.1. Logit回归模型介绍 22](#_Toc150010376)

[6.2. Logit 回归模型拟合结果 23](#_Toc150010377)

[7. 多重共线性的解决方案 27](#_Toc150010378)

[7.1. 岭回归 27](#_Toc150010379)

[7.2. 主成分分析 30](#_Toc150010380)

[8. 分类模型 34](#_Toc150010381)

[8.1. 随机森林模型 34](#_Toc150010382)

[8.2. XGBoost 39](#_Toc150010383)

[8.3. 考虑基线数据的模型构建 42](#_Toc150010384)

[附录 45](#_Toc150010385)

**图目录**

[图1 性别与患者出院状态柱状分布图 1](#_Toc150007827)

[图2 年龄与患者出院状态柱状分布图 2](#_Toc150007828)

[图3 体重与患者出院状态柱状分布图 3](#_Toc150007829)

[图4 身高与患者出院状态柱状分布图 3](#_Toc150007830)

[图5 出院状态死亡概率先验分布 7](#_Toc150007831)

[图6 出院状态死亡概率后验分布 8](#_Toc150007832)

[图7 出院状态死亡概率随观察量变化估计 9](#_Toc150007833)

[图8 混合高斯分布随机数x的概率密度分布 10](#_Toc150007834)[图9 (a)观测值x来源于N(0, 2)的概率密度分布 (b) 观测值x来源于N(4, 2)的概率密度分布 (c) 观测值x来源于N(μ1, σ1)的相对概率 11](#_Toc150007835)

[图10 ROC曲线 12](#_Toc150007836)

[图11 真实概率密度与估计概率密度的对比 13](#_Toc150007837)

[图12 ROC曲线（EM算法参数估计） 13](#_Toc150007838)

[图13 ROC曲线(LDA) (a) 训练集 (b) 测试集 15](#_Toc150007839)

[图14 ROC曲线(Fisher判别) (a) 训练集 (b) 测试集 16](#_Toc150007840)

[图15 ROC曲线(QDA) (a) 训练集 (b) 测试集 16](#_Toc150007841)

[图16 ROC曲线(Logistic Regression) (a) 训练集 (b) 测试集 17](#_Toc150007842)

[图17 相关性热力图 21](#_Toc150007843)

[图18 sigmoid函数 22](#_Toc150007845)

[图19 岭回归混淆矩阵 28](#_Toc150007847)

[图20 岭回归ROC曲线 29](#_Toc150007849)

[图21 岭回归精确度-召回率曲线 30](#_Toc150007851)

[图22 主成分数量与解释方差比例关系图 31](#_Toc150007853)

[图23 主成分数量与累计解释方差比例关系图 32](#_Toc150007855)

[图24 主成分分析混淆矩阵 32](#_Toc150007857)

[图25 主成分分析ROC曲线 33](#_Toc150007859)

[图26 主成分分析精确度-召回率曲线 33](#_Toc150007861)

[图27 随机森林示意图 35](#_Toc150007863)

[图28 随机森林混淆矩阵 37](#_Toc150007865)

[图29 随机森林ROC曲线 38](#_Toc150007869)

[图30 随机森林精确度-召回率曲线 38](#_Toc150007871)

[图31 随机森林混淆矩阵 40](#_Toc150007873)

[图32 XGBoost ROC曲线 41](#_Toc150007877)

[图33 XGBoost精确度-召回率曲线 41](#_Toc150007879)

[图34 纳入基线数据的主成分数量与累计解释方差关系图 42](#_Toc150007881)

[图35 纳入基线数据的ROC曲线 43](#_Toc150007883)

[图36 纳入基线数据的精确-召回曲线 43](#_Toc150007885)

**表目录**

[表1 卡方分析的假设 4](#_Toc150010212)

[表2 不同基线数据患者出院状态的卡方检验结果 4](#_Toc150010213)

[表3 卡方分析结果 5](#_Toc150010214)

[表4 后验分布参数与统计指标 8](#_Toc150010215)

[表5 混淆矩阵 11](#_Toc150010216)

[表6 估计初始化参数 12](#_Toc150010217)

[表7 EM估计参数与真实参数对比 12](#_Toc150010218)

[表8 LDA函数参数估计 14](#_Toc150010219)

[表9 相关性检验结果 20](#_Toc150010220)

[表10 出院状态和10个变量单独拟合结果 23](#_Toc150010222)

[表11 模型指标结果 24](#_Toc150010224)

[表12 出院状态以及前10个变量一起拟合结果 25](#_Toc150010226)

# 不同基线数据下患者出院状态分布图及其统计学描述

在提供的数据集mydata中，对性别(gender)、入院年龄(admission\_age)、入院体重(weight\_admit)以及身高(height)这四组基线数据下患者出院状态进行统计分析，绘制了这四组对应基线数据下患者出院状态的分布图，并进行了相应的统计描述。

## 性别与出院状态分布

在提供的数据集中共有6273份患者数据，其中男性（M）有3604份，女性（F）有2669份数据。在男性患者样本中，出院状态为死亡（flag=1）有1152份，死亡率为0.3196。在女性患者样本中，出院状态为死亡（flag=1）有1152份，死亡率为0.3215。上述分析数据分成男女两大类，绘制出堆叠式柱状图以及死亡率折线如图1所示。图中左侧纵轴表示人数，右侧纵轴表示患者出院状态的死亡率，横轴为性别。从图中直观上分析，男性与女性患者出院状态的死亡率差异很小，均在在0.32附近。

图表

描述已自动生成

1. 性别与患者出院状态柱状分布图

Fig.1 Histogram of gender and patient discharge status

## 年龄与出院状态分布

将数据集中6273为患者按照年龄进行分组，除了20岁以下以及90岁以上各自归为一组，年龄梯度按照5岁一个梯度进行划分，因此一共划分出了16个组别。将各个组别的出院状态进行统计并计算对应的死亡率，绘制成分布图如图2所示。对图2进行统计学描述如下：

1）样本数据数量在各年龄层存在一定差异，患者数量集中在40岁以上人群，从侧面反映出入院患者年龄上以中年、老年为主；

2）从出院状态死亡率折线图可以直观发现，患者出院状态死亡率随年龄增长呈现出增长趋势，在20岁以下仅为0.14，90岁以上达到0.41。

图表, 直方图

描述已自动生成

1. 年龄与患者出院状态柱状分布图

Fig.2 Histogram of age and discharge status of patients

## 体重与出院状态分布

将数据集中6273名患者按照体重进行分组，除了30kg以下以及90kg以上各自归为一组，年龄梯度按照10kg一个梯度进行划分，因此一共划分出了8个组别。将各个组别的出院状态进行统计并计算对应的死亡率，绘制成分布图如图3所示。对图3进行统计学描述如下：

1）样本数据中患者的体重集中在40~70kg之间，约占到了97%

2）从出院状态死亡率折线图可以看出不同体重下患者出院状态死亡死亡率最大为0.34(30kg)以下，最低为0.25(70~80kg),差值为0.09。由于差异量并不十分显著，并且30kg以及70~80kg患者的数据量不是很充足，因此无法判断体重过轻对于患者出院状态死亡率是否有直接显著影响，需要后续进一步做差异性分析进行检验。

图表

描述已自动生成

1. 体重与患者出院状态柱状分布图

Fig.3 Histogram of weight and discharge status of patients

## 身高与出院状态分布

将数据集中6273名患者按照身高进行分组，除了150cm以下以及195cm以上各自归为一组，年龄梯度按照5cm一个梯度进行划分，因此一共划分出了11个组别。将各个组别的出院状态进行统计并计算对应的死亡率，绘制成分布图如图3所示。对图3进行统计学描述如下：

1）样本数据中患者的身高集中在150~185cm之间，约占到了90%

2）从出院状态死亡率折线图可以看出不同体重下患者出院状态死亡率最大为0.37(150~155cm)，最低为0.21(195cm)，差值为0.16。由于195cm以上身高患者数据样本数据量很少，导致计算出的最低死亡0.21存在一定偶然性，因此也需要进一步进行差异性分析去判断身高与患者出院状态死亡概率的关联性。

图表, 直方图

描述已自动生成

1. 身高与患者出院状态柱状分布图

Fig.4 Histogram of weight and discharge status of patients

# 差异性分析

在完成了对不同基线数据下患者出院状态的分布图后，需要进一步利用差异性分析方法来对不同基线数据与患者出院状态的关联性进行评估。当前常见的差异性分析方法包括ANOVA(方差分析)、T-test(T检验)以及Chi-Square Analysis(卡方分析)这三种。由于不同基线数据（性别、年龄、体重、身高）与患者出院状态两类数据均为分类数据，综合考虑三种差异性分析方法应用条件，选择Chi-Square Analysis对不同基线数据与患者出院状态进行关联性分析。

## 提出假设

以基线数据中的性别为例，建立原假设H0与备择假设H1，假设内容如表1。原假设认为性别对患者出院状态无显著影响，备择假设认为性别对患者出院状态有显著影响。当原假设被推翻后，则认为备择假设成立。

1. 卡方分析的假设

Tab.1 The hypothesis of the Chi-Square Analysis

|  |  |
| --- | --- |
| Hypothesis | Hypothetical content |
| H0 | Gender has no significant impact on mortality at hospital discharge |
| H1 | Gender has a significant impact on mortality at hospital discharge |

## 计算卡方值

对不同基线数据下患者出院状态的数据进行分类预处理后，利用R中的chiq.test()函数进行计算出各组的χ2(卡方值)以及对应的*P*值(显著性水平)，结果如表2所示。

1. 不同基线数据患者出院状态的卡方检验结果

Tab.2 Chi-square analysis of patient discharge status with different baseline data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Baselinedata | Number | Flaga=0 | Flag=1 | χ2 | *P* |
| Gender |  |  |  |  |  |
| Male | 3604 | 2452 | 1152 | 0.015798 | 0.9 |
| Female | 2669 | 1811 | 858 |
| Age |  |  |  |  |  |
| <=20 | 14 | 12 | 2 | 84.907 | 0.0004998 |
| 20~30 | 167 | 137 | 30 |
| 30~40 | 268 | 209 | 59 |
| 40~50 | 535 | 404 | 131 |
| 50~60 | 1192 | 849 | 343 |
| 60~70 | 1486 | 1006 | 480 |
| 70~80 | 1369 | 892 | 477 |
| 80~90 | 979 | 599 | 380 |
| >90 | 263 | 155 | 108 |
| Weight/kg |  |  |  | 10.004 | 0.2648 |
| <=20 | 63 | 47 | 16 |
| 20~30 | 978 | 639 | 339 |
| 30~40 | 2294 | 1576 | 718 |
| 40~50 | 1781 | 1199 | 582 |
| 50~60 | 715 | 482 | 233 |
| 60~70 | 252 | 182 | 70 |
| 70~80 | 93 | 70 | 23 |
| 80~90 | 52 | 36 | 16 |
| >90 | 45 | 32 | 13 |
| Height/cm |  |  |  |  |  |
| <=150 | 137 | 169 | 68 | 7.4 | 0.1925 |
| 150~160 | 1382 | 921 | 461 |
| 160~170 | 1942 | 1294 | 648 |
| 170~180 | 1929 | 1333 | 596 |
| 180~190 | 674 | 465 | 209 |
| >190 | 109 | 81 | 28 |

aFlag denotes patients discharge status. “Flag=1” denotes discharge status is death.

## 确定显著性水平

在进行假设检验前，需要事先设定显著性水平阈值α用于评估各组计算出的*P*值是否足够小以拒绝原假设。本文设置显著性水平阈值值α为0.05。当*P*值小于α时，表示认为有足够的证据拒绝原假设，反之认为观察到的差异是不显著的，无足够证据拒绝原假设。

## 卡方分析结果

将表2中各组数据计算出的*P*值与*α*值进行比较，结果如表3所示。从表中可以看出年龄组的P值<*α*，这表明年龄组的原假设被推翻，它的备择假设成立，即年龄对患者出院状态有显著影响。性别、体重以及身高组的P值>*α*，则原假设成立，即性别、体重以及身高对患者出院状态没有显著性影响。

1. 卡方分析结果

Tab.3 Chi-Square Analysis Results

|  |  |  |
| --- | --- | --- |
| Baseline data | *P* | Test Results |
| Gender | >*α* | Gender has no significant impact on mortality at hospital discharge |
| Age | <*α* | Age has a significant impact on mortality at hospital discharge |
| Weight | >*α* | Weight has no significant impact on mortality at hospital discharge |
| Height | >*α* | Height has no significant impact on mortality at hospital discharge |

# 出院死亡概率后验分布以及估计值随时间变化曲线

## 出院死亡概率*θ*后验分布

首先构建一个参数为*θ*的伯努利分布概率模型*X*~*bernoulli*(*θ*)，其中*X*为样本数据中患者出院状态。由贝叶斯定理有：

 (1)

式中为所求的后验分布，与分别给定*θ*下的数据样本的似然估计函数与*θ*的先验分布，分母的积分项为与*θ*无关的归一化常数c，常称为证据。因此理论上要求解后验分布的需要分别计算出、以及c即可求出后验分布。

结合伯努利分布似然函数以及Beta分布特点，当给定一个服从伯努利分布*bernoulli*(*θ*)的数据样本*X*=(*x*1,…,*x*n)以及某个共轭先验分布*θ~Beta*(*α,β*)时，可以得到后验分布：

 (2)

以前100名患者的出院状态作为经验，假设患者出院状态死亡概率先验分布服从参数为(alpha=35, beta=65)的Beta分布，概率密度函数与累积分布函数如图5所示。

图表, 折线图

描述已自动生成

1. (b)
2. 出院状态死亡概率先验分布

(a)概率密度函数 (b) 累积分布函数

Fig.5 Prior Beta distribution of mortality probability at discharge state

(a) Probability density function. (b) Cumulative density function

利用患者出院状态的观测数据更新先验分布，得到最新的后验分布的概率密度函数与累积分布函数如图6所示。后验分布仍然服从Beta分布，参数为*θ~Beta*(2012*,*4268)，其统计学指标如表4所示。

图表, 直方图

描述已自动生成图表, 折线图

描述已自动生成

1. (b)
2. 出院状态死亡概率后验分布

a)概率密度函数 b)累积分布函数

Fig.6 Posterior distribution of mortality probability at discharge state

a)Probability density function. b)Cumulative density function

1. 后验分布参数与统计指标

Tab.4 Posterior distribution parameters and statistical indicators

|  |  |
| --- | --- |
| Statistical indicators | *θ* value |
| Mean | 0.3203822 |
| Median | 0.3203631 |
| Lower Confidence Interval(95%) | 0.308897 |
| Upper confidence interval(95%) | 0.3319757 |

## 出院死亡概率估计随观察量变化

3.1中直接给出了最终由先验分布最终更新得到的后验分布，本节将给出后验分布更新过程中死亡概率*θ*的估计值，具体过程如下：

1）首先选取前100个观测量计算出院死亡概率*θ*的先验Beta分布，得到初始的先验Beta分布参数为Beta（*θ,*35,65）；

2）在先验分布基础上根据数据集中患者出院状态不断更新观测，调整后验Beta分布的参数，直至更新完所有样本，更新过程中的*θ*变化如图8所示。图中蓝色曲线为更新过程中*θ*的期望估计，灰色区域为*θ*估计的95%置信区间，红色虚线为更新完所有样本后*θ*的后验分布期望，其值为0.3204。

图表, 直方图

描述已自动生成

1. 出院状态死亡概率随观察量变化估计

Fig. 7 Estimation of the probability of death at discharge with observation

# 二元混合高斯分布参数估计与模型拟合

## 随机数x概率密度

根据数据集中患者出院状态y生成一列符合混合高斯分布的随机数x，其中y=0所对应的x服从参数为N(0,2)的高斯分布，y=1所对应的x服从参数为N(4,2)的高斯分布，最终得到了具有混合高斯分布的随机数x的概率密度如8所示，其中图8(a)为全体x的概率密度图，8(b)为x在不同y类别下的概率密度图。

图表, 折线图

描述已自动生成 图表, 直方图

描述已自动生成

1. (b)
2. 混合高斯分布随机数x的概率密度分布

(a) 不按y分类 (b) 按y分类

Fig. 8 Probability density distribution plot of mixed Gaussian distribution random number x. (a) Not categorized by y. (b) Categorized by y

## 不同来源随机数x的概率密度分布与相对概率

针对4.1节中生成的随机数序列x分别计算其来源于高斯分布N(0, 2)与N(4, 2)的概率密度f0与f1以及其来源于出院状态y=1的相对概率f，结果如图9(a-c)所示。从图9(a, b)中不难发现x在两个高斯分布中的概率密度峰值出现在两个不同值，这也与4.1节生成的混合高斯分布相吻合。图9(c)中相对概率f从随着x值增大逐渐由0增至1。

图表, 折线图

描述已自动生成

1. (a)观测值x来源于N(0, 2)的概率密度分布 (b) 观测值x来源于N(4, 2)的概率密度分布 (c) 观测值x来源于N(μ1, σ1)的相对概率

Fig.9 (a) Probability density distribution of observed value x from N(μ0, σ0) (b) Probability density distribution of observed value x from N(μ1, σ1) (c) Relative probability of observation x derived from N(μ1, σ1).

## 分类错误概率与ROC曲线

根据4.2节计算出的相对概率*f*，设定判断阈值为0.5，即观测值对应的y=1，*f*>0.5则认为分类正确，反之分类错误。由此可以得到混淆矩阵如表5所示。由表5中TP、FP、FN与TN的值进一步计算出当前阈值条件下的第一类错误、第二类错误概率以及F1值分别为0.084、0.079、0.879。

1. 混淆矩阵

Tab.5 Confusion matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Actual | |
| Death | Survival |
| Predicted | Judging death | True Positive=1841 | False Positive=337 |
| Judging survival | False Negative=169 | True Negative=3926 |

当模型判定阈值由0以步长为0.01不断变化时，根据模型敏感性（FPR）与精确性（TPR）计算公式得到ROC曲线如图10所示，图中显示了当前预测模型AUC值(十次平均结果)为0.9771，蓝色虚线为AUC=0.5的ROC线。由于当前模型的AUC=0.9771>0.5，这表明该分类模型在合理设定阈值的条件下具有预测价值。

图表

描述已自动生成

1. ROC曲线

Fig.10 ROC curve

## EM算法估计二元混合高斯分布参数

对4.1节生成的符合二元混合高斯分的随机数x利用EM(Expectation-Maximum)算法进行参数估计。首先设置估计的初始化参数如表6：

1. 估计初始化参数

Tab.6 Estimate initialization parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Distribution | Mean | Variance | Weight |
| 1 | -0.5 | 1.5 | 0.5 |
| 2 | 4.5 | 1.5 | 0.5 |

进一步在上述初始条件下进行EM算法迭代估计，由于每次迭代结果略有差异，因此取三次估计结果的均值作为最终得到随机数x混合高斯分布的参数估计与实际参数对比如表7所示：

1. EM估计参数与真实参数对比

Tab.7 EM estimation parameters compared with real parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Distribution | Mean | Variance | Weight |
| 1(Real) | 0 | 2 | 0.680 |
| 2(Real) | 4 | 2 | 0.320 |
| 1(Estimated) | 0.052 | 1.438 | 0.689 |
| 2(Estimated) | 4.12 | 1.401 | 0.311 |

利用EM算法估计得到的分布参数对y进行概率密度估计，得到真实y的概率密度与估计分布参数的概率密度图对比分别如图11(a, b)所示。

图表, 折线图

描述已自动生成

1. 真实概率密度与估计概率密度的对比

Fig.11 True probability density versus estimated probability density

利用估计出的分布参数，根据4.3节中的步骤求解出当前分类模型的ROC曲线如图12所示，图中曲线的Auc取10次平均后为0.9768。图12与4.3中利用真实y分布得到的ROC曲线相比基本一致，Auc值前者略低一些，但都在0.97左右大于0.5，表明EM算法估计出的高斯混合分布参数与原分布具有较高的一致性，得到的预测分布模型可以在合理设定阈值的条件下具有预测价值。

图表

描述已自动生成

1. ROC曲线（EM算法参数估计）

Fig.12 ROC curve (EM algorithm parameter estimation)

## 训练集模型拟合与测试集预测

1)线性判别(Linear Discriminate Analysis, LDA)

线性判别分析进行建模前需要假设需要分类的*k*个总体分别服从高斯分布且各总体方差相同，然后分别求解自变量在*k*个类中的概率作为判断依据，将概率进行对数化并舍去不含*k*的共同项，得到线性判别函数如式(3)：

 (3)

式中的参数、与为待估计建模的参数，估计公式如式(4)：

 (4)

随机选取70%的样本作为训练集代入上式进行参数估计，结果如表8所示。

1. LDA函数参数估计

Tab.8 LDA function parameter estimation

|  |  |
| --- | --- |
| Parameter | Estimated value |
|  | 0.679 |
|  | 0.321 |
|  | -0.027 |
|  | 4.013 |
|  | 2.031 |

根据估计的参数，分别得到随机数分属于两个类别的判别函数，将训练集与测试集代入判别函数，可以计算出两个数据集中随机数属于状态1的相对概率，按照0.01的步进不断改变判断阈值，得到两数据集的ROC曲线如图13(a, b)所示，其中图13(a)为训练样本集的ROC曲线，对应Auc值为0.9773，图13(b)为测试样本集的ROC曲线，对应Auc值为0.977。

图片包含 散点图

描述已自动生成

1. ROC曲线(LDA) (a) 训练集 (b) 测试集

Fig.13 ROC curve(LDA) (a) training set (b) test set

2）Fisher判别

Fisher判别也是一种线性判别分析方法，基本思想是将样本进行投影，使投影后类内方差最小，类间方差最大。在R中调用Mass库中的lda()函数可以实现模型的预测，随机选取样本中70%的样本作为训练集代入进行训练，得到模型的线性判别系数为0.719。在此基础上将训练样本集以及测试样本集代入训练完成的模型进行预测，最终得到了图14(a, b)所示的ROC曲线，其中图14(a)为训练样本集的ROC曲线，对应Auc值为0.9779，图14(b)为测试样本集的ROC曲线，对应Auc值为0.9697。



1. ROC曲线(Fisher判别) (a) 训练集 (b) 测试集

Fig.14 ROC curve(Fisher Discrimination) (a) training set (b) test set

3）二次判别（Quadratic Discriminant Analysis Algorithm, QDA）

在进行LDA分析时假设的是各分类总体具有相同的协方差矩阵，而利用QDA则更加灵活，允许每个类别协方差矩阵独立估计，但参数估计相对复杂。在R中调用qda()函数进行模型预测，随机选取70%的样本作为训练集代入进行训练，最终得到训练集与测试集样本的ROC曲线分别如图15(a, b)所示，对应的Auc值均为0.9776。

图片包含 图形用户界面

描述已自动生成

1. ROC曲线(QDA) (a) 训练集 (b) 测试集

Fig.15 ROC curve(QDA) (a) training set (b) test set

4）逻辑回归（Logistic Regression）

逻辑回归常用于解决二分类问题，利用逻辑函数(Sigmoid函数)的变换来建立模型，将输入的线性组合通过一个非线性映射，将结果限制在0和1之间，预测样本属于某个类别的概率。在R中利用glm()函数实现逻辑回归预测，同样随机选取70%的样本作为训练集代入进行训练，最终得到训练集与测试集样本的ROC曲线分别如图16(a, b)所示，对应的Auc值分别为0.9787与0.9740。

图片包含 图形用户界面

描述已自动生成

1. ROC曲线(Logistic Regression) (a) 训练集 (b) 测试集

Fig.16 ROC curve(Logistic Regression) (a) training set (b) test set

由于本次进行训练的随机数服从的二元混合高斯分布，并且两高斯分布的方差相同，因此利用四种模型拟合方法得到的ROC曲线Auc值没有显著差异，而实际应用需要根据不同的分布选择对应合适的模型拟合方法进行分类处理方能实现更好的效果。

# 相关性分析

## 数据预处理

1、数据剔除

查看原始数据以及变量解释表可知，Admitted time为患者进入医院时间；discharged time为患者离开医院的时间；Length of stay in hospital为患者在医院停留的时间，本文认为在预测患者是否发生院内死亡时，只需要关注患者在医院停留的时间，因此将患者进入医院时间和患者离开医院的时间剔除；同理，剔除icu\_intime（患者入ICU的时间）和icu\_outtime（患者离开ICU的时间），只保留los\_icu（患者在ICU的停留时间）。另外，本文考虑到Date of death（患者死亡时间），为保护患者隐私，经过了加密处理，无法直接使用，因此将Date of death也进行剔除。

2、填补缺失值

原始数据中，albumin\_min、albumin\_max、globulin\_min等特征存在缺失值，因此在使用数据前需要对缺失值进行填充，考虑到只有数值型特征存在缺失值，本文使用每列特征的平均值进行填充。

3、标准化

标准化的过程涉及将原始数据的每个特征（列）按照一定的规则进行缩放，使其具有零均值和单位方差。包括以下步骤：

* 计算每个特征的均值（mean）和标准差（standard deviation）。
* 对每个特征进行减去均值，然后除以标准差，以实现标准化。

这一过程使得数据的分布更加接近正态分布，有助于提高某些机器学习算法的性能，特别是那些对输入数据的尺度和分布敏感的算法。

## 皮尔逊相关性分析方法

皮尔逊相关性检验（Pearson Correlation Test）是一种用于评估两个连续变量之间线性相关性的统计检验方法。它衡量了两个变量之间的线性关系的强度和方向，通常用于确定两个变量是否在一定程度上呈正相关（值增加时两者同时增加）、负相关（一个值增加时另一个值减少）或不相关（两者之间没有明显的线性关系）。

以下是有关皮尔逊相关性检验的重要信息：

1、假设：皮尔逊相关性检验基于以下假设：

* 零假设（H0）：两个变量之间没有线性相关性。
* 备择假设（H1）：两个变量之间存在线性相关性。

2、相关系数：皮尔逊相关系数（Pearson correlation coefficient），通常表示为"r"，取值范围在-1到1之间。

如果r=1，表示存在完全正线性关系。这意味着两个变量之间存在完美的正相关关系，其中一个变量的增加总是伴随着另一个变量的增加。

如果r=-1，表示存在完全负线性关系。这意味着两个变量之间存在完美的负相关关系，其中一个变量的增加总是伴随着另一个变量的减小。

如果r=0，表示两个变量之间不存在线性关系。这意味着两个变量之间没有线性模式或趋势。

3、计算方法：皮尔逊相关系数是通过以下公式计算的：

其中，*X*和*Y*分别表示两个变量的观测值，而和*Ȳ*分别表示它们的均值。通过计算样本数据中的相关系数，我们可以得到一个介于-1到1之间的值，从而判断两个变量的线性关系强弱。

4、假设检验：皮尔逊相关性检验的目标是检验相关系数是否显著不同于零。为此，通常使用t检验来评估相关系数的显著性。

5、显著性水平：在进行皮尔逊相关性检验时，需要选择一个显著性水平（通常为0.05或0.01），以确定是否拒绝零假设。如果计算得到的t统计值的绝对值大于t分布临界值，则可以拒绝零假设，表示两个变量之间存在显著的线性相关性。

6、解释结果：皮尔逊相关性检验的结果通常包括相关系数（r）、t统计值、自由度、p值和置信区间。p值用于判断是否拒绝零假设，较小的p值表示较强的证据支持拒绝零假设。

7、皮尔逊相关系数的主要特点和用途包括：

度量线性关系：皮尔逊相关系数主要用于度量两个变量之间的线性关系。它只能捕捉线性关系，而不能捕捉非线性关系。

方向和强度：皮尔逊相关系数提供了关于线性关系方向（正相关或负相关）和强度（相关性的程度）的信息。

范围：皮尔逊相关系数的范围在-1到1之间，使其易于解释。接近1表示强正相关，接近-1表示强负相关，接近0表示无相关性。

敏感性：皮尔逊相关系数对异常值敏感。异常值可以显著影响相关性的计算。

用途：皮尔逊相关系数常用于数据探索、特征选择、回归分析、因果关系研究等领域。

此外，皮尔逊相关系数的计算假设数据是正态分布的，对于非正态分布的数据，相关系数的解释可能需要谨慎考虑。如果数据的分布不满足正态分布假设，可以考虑使用非参数方法或其他相关性测量方法。

## 皮尔逊相关性检验分析

对给出的mydata数据中，首日检验数据以及出院状态共109个变量的相关性检验分析，得出与出院状态相关性最高的前10个变量如下表所示：

1. 相关性检验结果

Table 9 Results of correlation test

|  |  |
| --- | --- |
| 与出院状态相关性最高的前10个变量 | |
| aniongap\_min | 0.224521 |
| aniongap\_max | 0.220208 |
| spo2\_min | -0.180098 |
| sbp\_min | -0.173176 |
| ptt\_max | 0.172143 |
| mbp\_min | -0.169785 |
| bicarbonate\_min | -0.164854 |
| spo2\_mean | -0.164674 |
| temperature\_min | -0.164546 |
| temperature\_mean | -0.160319 |

从结果中可以看出，与出院状态相关性最高的变量是aniongap\_min，相关系数是0.224521。

绘制包括出院状态共11个变量的相关性图，如下图所示，并标注相关系数，且相关性越强，颜色越深。

图表, 树状图

描述已自动生成

1. 相关性热力图

Fig.17 Heat map of correlation

# Logit回归模型

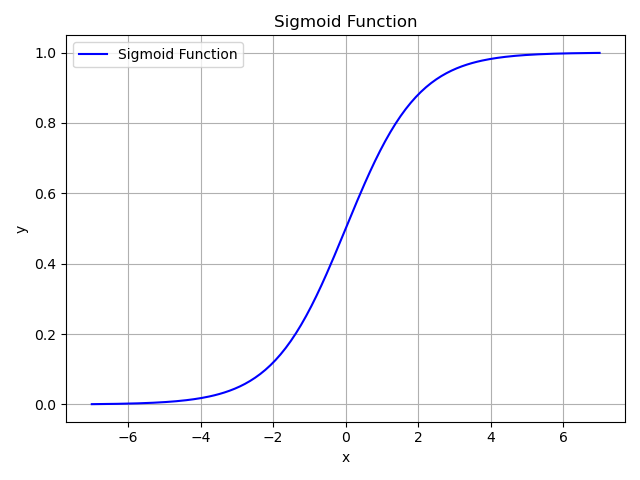
## Logit回归模型介绍

Logit模型是一种用于建模二元分类问题的统计模型。它是逻辑回归模型的基础，用于预测二分类变量的概率。

Logit模型基于以下假设：每个观测样本都有一个恰当的响应变量（通常为0或1），并且该响应变量是基于一组预测变量的概率。模型通过对预测变量的线性组合进行转换，将其映射到[0,1]范围内的概率。

具体地说，Logit模型使用逻辑函数（也称为S形函数）将线性组合的结果转换为概率。逻辑函数是一个sigmoid函数，它具有以下形式：

其中，是观测为类别1的概率，是一组预测变量，是指数函数，是预测变量的线性组合。



1. sigmoid函数

Figure 18 sigmoid function

为了得到模型的参数估计，常用的方法是最大似然估计。通过最大似然估计，我们能够找到一组最佳的参数值，使得观测数据在模型下出现的概率最大化。

Logit模型的优点包括可以建模非线性关系和处理高维数据。它也具有较好的解释性，可以提供关于每个预测变量对分类结果的影响程度的信息。

## Logit 回归模型拟合结果

本文将数据按照训练集80%，测试集20%的比例进行划分，然后对出院状态以及和出院状态相关性最高的前10个变量单独做拟合，并且对单独使用每个变量做拟合的结果进行对比，描述各变量的系数、显著性的区别，如下表所示：

1. 出院状态和10个变量单独拟合结果

Table 10 Discharge status and individual fitting results for 10 variables

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | coef | std err | z | P>|z| | [0.025 | 0.975] |
| const | -0.81095 | 0.031346 | -25.8707 | 1.4E-147 | -0.87238 | -0.74951 |
| aniongap\_min | 0.465527 | 0.032165 | 14.47315 | 1.79E-47 | 0.402485 | 0.528568 |
| const | -0.80944 | 0.031317 | -25.8465 | 2.7E-147 | -0.87082 | -0.74806 |
| aniongap\_max | 0.45798 | 0.031422 | 14.57524 | 4.04E-48 | 0.396394 | 0.519566 |
| const | -0.79153 | 0.030906 | -25.6113 | 1.1E-144 | -0.85211 | -0.73096 |
| spo2\_min | -0.37285 | 0.033174 | -11.2391 | 2.62E-29 | -0.43787 | -0.30783 |
| const | -0.80654 | 0.031171 | -25.8746 | 1.3E-147 | -0.86764 | -0.74545 |
| sbp\_min | -0.39496 | 0.032712 | -12.074 | 1.45E-33 | -0.45908 | -0.33085 |
| const | -0.79623 | 0.030897 | -25.7708 | 1.9E-146 | -0.85679 | -0.73567 |
| ptt\_max | 0.320798 | 0.028842 | 11.12246 | 9.76E-29 | 0.264268 | 0.377328 |
| const | -0.80191 | 0.031048 | -25.8283 | 4.3E-147 | -0.86276 | -0.74105 |
| mbp\_min | -0.3698 | 0.031127 | -11.8805 | 1.5E-32 | -0.43081 | -0.3088 |
| const | -0.80182 | 0.03104 | -25.8316 | 3.9E-147 | -0.86266 | -0.74099 |
| bicarbonate\_min | -0.35163 | 0.031994 | -10.9904 | 4.25E-28 | -0.41433 | -0.28892 |
| const | -0.79041 | 0.030861 | -25.6121 | 1.1E-144 | -0.85089 | -0.72992 |
| spo2\_mean | -0.371 | 0.03476 | -10.6732 | 1.36E-26 | -0.43913 | -0.30287 |
| const | -0.79567 | 0.030908 | -25.7431 | 3.9E-146 | -0.85625 | -0.73509 |
| temperature\_min | -0.34806 | 0.031347 | -11.1036 | 1.21E-28 | -0.4095 | -0.28663 |
| const | -0.80158 | 0.031045 | -25.8198 | 5.3E-147 | -0.86243 | -0.74073 |
| temperature\_mean | -0.3756 | 0.032939 | -11.4029 | 4.05E-30 | -0.44015 | -0.31104 |

现在，对这些统计指标进行分析：

coef：这是回归模型中自变量的系数（也称为回归系数或斜率）。在这两种情况下，它们表示自变量对因变量的影响。例如对于最后一个变量："const"，coef为-0.785678993，对于"creatinine\_max"，coef为0.205343695。

std err：这是系数估计的标准误差，用于衡量估计的不确定性。较小的std err表示对系数估计的更高置信度。

z-statistic (z)：这是coef除以std err的结果，通常用于检验系数是否显著不等于零。较大的z值表明系数估计可能更显著。

P-value (P>|z|)：P-value是z-statistic的双尾假设检验的p值。它用于衡量系数是否统计显著。通常，如果P-value小于显著性水平（通常是0.05），则系数被认为是显著的，即与因变量之间存在关系。

[0.025 0.975]：这是系数的置信区间，通常以95%的置信度计算。它表示系数的估计范围，即该系数的真实值以95%的概率位于这个区间内。

从上表可以看出，与出院状态相关性最高的10个变量的P-value均小于显著性水平，因此认为这10个变量均与出院状态存在关系。

本文使用准确率(Accuracy)、召回率(Recall)、精确率(Precision)、F1分数(F1Score)来评估模型在测试数据集上的表现。以下是这些性能指标的解释：

准确率(Accuracy)：这是模型正确预测的观测值的比例。它表示模型在测试数据集上正确预测的比例。

召回率(Recall)：也称为灵敏度（Sensitivity）或真正例率（TruePositiveRate），它是真实正例中被模型正确预测为正例的比例。召回率用于评估模型在正例类别上的覆盖程度。

精确率(Precision)：精确率是模型在预测为正例的观测中，实际为正例的比例。精确率用于评估模型的预测准确性。

F1分数(F1 Score)：F1分数是精确率和召回率的调和平均值，用于综合评估模型的性能。它考虑了精确率和召回率之间的权衡，对不平衡类别问题尤为有用。

统计上述模型四个指标的结果如下表所示：

1. 模型指标结果

Table 11 Results of model indicators

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 准确率 | 召回率 | 精确率 | F1分数 |
| aniongap\_min | 0.680 | 0.123 | 0.697 | 0.209 |
| aniongap\_max | 0.675 | 0.123 | 0.646 | 0.206 |
| spo2\_min | 0.673 | 0.072 | 0.775 | 0.131 |
| sbp\_min | 0.666 | 0.051 | 0.710 | 0.095 |
| ptt\_max | 0.669 | 0.097 | 0.618 | 0.168 |
| mbp\_min | 0.669 | 0.079 | 0.654 | 0.140 |
| bicarbonate\_min | 0.658 | 0.019 | 0.615 | 0.036 |
| spo2\_mean | 0.674 | 0.067 | 0.829 | 0.124 |
| temperature\_min | 0.665 | 0.072 | 0.608 | 0.128 |
| temperature\_mean | 0.665 | 0.042 | 0.750 | 0.079 |

从上表中可以看出，10个变量与出院状态拟合的准确率在0.65~0.7之间，所有变量的召回率和F1分数均比较低，精确率在0.60~0.85之间，变化幅度比较大；其中，模型对于aniongap\_min与出院状态拟合的F1分数最大，因此拟合效果相对较好。

然后，使用Logit回归模型对出院状态以及前10个变量做拟合，拟合的结果如下表所示：

1. 出院状态以及前10个变量一起拟合结果

Table 12 Discharge status and the results of fitting the first 10 variables together

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **index** | **Coef.** | **Std.Err.** | **z** | **P>|z|** | **[0.025** | **0.975]** |
| const | -0.84934 | 0.032851 | -25.8545 | 2.2E-147 | -0.91373 | -0.78496 |
| aniongap\_min | 0.280961 | 0.04836 | 5.809835 | 6.25E-09 | 0.186178 | 0.375744 |
| aniongap\_max | 0.112772 | 0.054789 | 2.058298 | 0.039562 | 0.005388 | 0.220156 |
| spo2\_min | -0.10741 | 0.043045 | -2.4953 | 0.012585 | -0.19177 | -0.02304 |
| sbp\_min | -0.14488 | 0.042157 | -3.4367 | 0.000589 | -0.22751 | -0.06225 |
| ptt\_max | 0.232649 | 0.031144 | 7.4702 | 8.01E-14 | 0.171609 | 0.293689 |
| mbp\_min | -0.17272 | 0.039823 | -4.33714 | 1.44E-05 | -0.25077 | -0.09467 |
| bicarbonate\_min | -0.02656 | 0.042485 | -0.62514 | 0.531878 | -0.10983 | 0.05671 |
| spo2\_mean | -0.23777 | 0.048256 | -4.92723 | 8.34E-07 | -0.33235 | -0.14319 |
| temperature\_min | -0.07646 | 0.049732 | -1.53739 | 0.124197 | -0.17393 | 0.021015 |
| temperature\_mean | -0.23997 | 0.050491 | -4.75277 | 2.01E-06 | -0.33893 | -0.14101 |

其中，bicarbonate\_min、temperature\_min两个变量的P-value大于0.05，因此认为这两个变量与出院状态不存在关系。10个变量一起回归预测的准确率为0.720、召回率为0.275、精确率为0.753、F1分数为0.403，相比于每个变量单独做回归预测有一定的提升。

# 多重共线性的解决方案

## 岭回归

岭回归（Ridge Regression）是一种线性回归的改进方法，用于解决多重共线性（multicollinearity）问题。多重共线性是指自变量之间存在高度相关性的情况，它可能导致普通最小二乘线性回归（Ordinary Least Squares Linear Regression）模型的不稳定性和过度拟合问题。岭回归通过引入正则化项来解决这些问题，从而提高模型的泛化能力。

以下是岭回归的关键特点和原理：

1、正则化项：岭回归在线性回归模型的损失函数中引入了一个正则化项，这个项是自变量的平方和与一个常数（通常称为λ或alpha）的乘积。这个正则化项被添加到模型的最小二乘损失函数中，以约束系数的大小。

2、系数缩减：正则化项会导致回归系数的缩减，使其趋向于零。这有助于减小多重共线性问题的影响，同时可以改善模型的稳定性。在岭回归中，即使自变量之间高度相关，回归系数也能得到合理的估计。

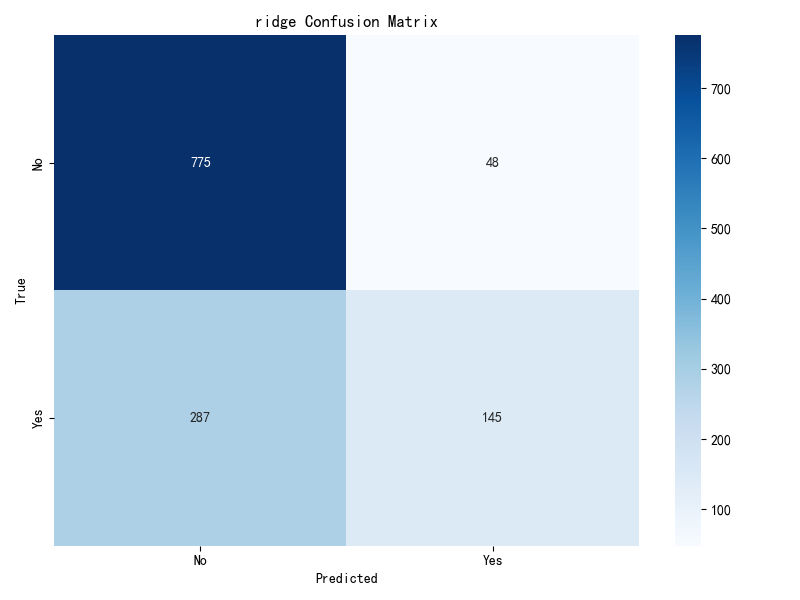
3、参数alpha的选择：参数alpha的选择是岭回归中的关键问题。alpha的值越大，系数的缩减效应越强，模型越简化，但可能失去一些预测能力。alpha的值越小，模型越接近普通最小二乘回归，但可能无法解决多重共线性问题。通常，需要进行交叉验证来选择最合适的alpha值。

4、解决过度拟合：岭回归有助于减小过度拟合问题，特别是当训练数据集中的自变量之间存在高度相关性时。它能够提高模型的泛化能力，降低测试数据上的误差。

5、数学原理：岭回归的数学原理涉及到求解带有正则化项的线性回归问题。通常使用线性代数和数值优化方法来找到最佳系数。

总之，岭回归是一种用于解决多重共线性和过度拟合问题的有用工具，它通过引入正则化项来约束回归系数的大小。这种方法有助于提高线性回归模型的稳定性和泛化能力，特别是在处理高维数据和相关性强的自变量时。

对数据中除了基线数据以外的109个变量进行岭回归，模型回归预测的混淆矩阵如下图：



1. 岭回归混淆矩阵

Figure 19 Ridge regression confusion matrix

岭回归的准确率为0.733、召回率为0.336、精确率为0.751、F1分数为0.464，对比未处理多重共线性的结果有显著提升。

ROC（Receiver Operating Characteristic）曲线是用于评估二元分类模型性能的一种常用工具。它提供了在不同阈值下，真正例率（True Positive Rate，又称召回率）与假正例率（False Positive Rate）之间的权衡。ROC曲线通常是在二元分类问题中使用的，其中一类被标记为“正类”，另一类被标记为“负类”。

以下是ROC曲线的关键概念和特点：

1、真正例率(True Positive Rate，召回率)：召回率是指在所有实际正例中，模型正确识别为正例的比例。它计算如下：

召回率=真正例数/(真正例数+假负例数)

召回率度量了模型对正例的识别能力。

2、假正例率(False Positive Rate)：假正例率是指在所有实际负例中，模型错误识别为正例的比例。它计算如下：

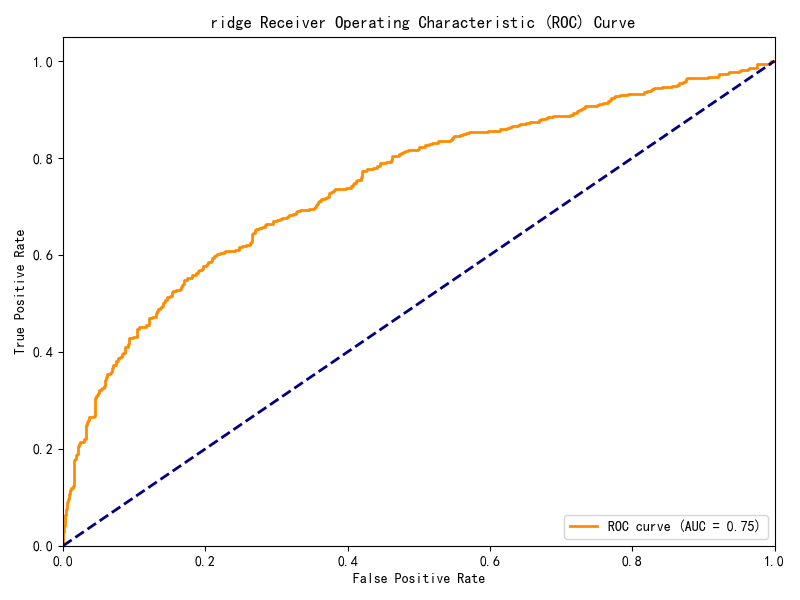
假正例率=假正例数/(真负例数+假正例数)

假正例率度量了模型的误报能力。

3、ROC曲线：ROC曲线是真正例率（召回率）与假正例率之间的关系图。ROC曲线通常是一个以真正例率为纵坐标，假正例率为横坐标的曲线。

4、AUC（Area Under the ROC Curve）：AUC是ROC曲线下面积的度量，用于总体评估分类器性能。AUC的取值范围在0到1之间，AUC越接近1，表示模型的性能越好。通常来说，AUC值大于0.5表示模型有一定的分类能力，而AUC等于1表示模型完美分类。

绘制岭回归的ROC曲线，如下图所示：



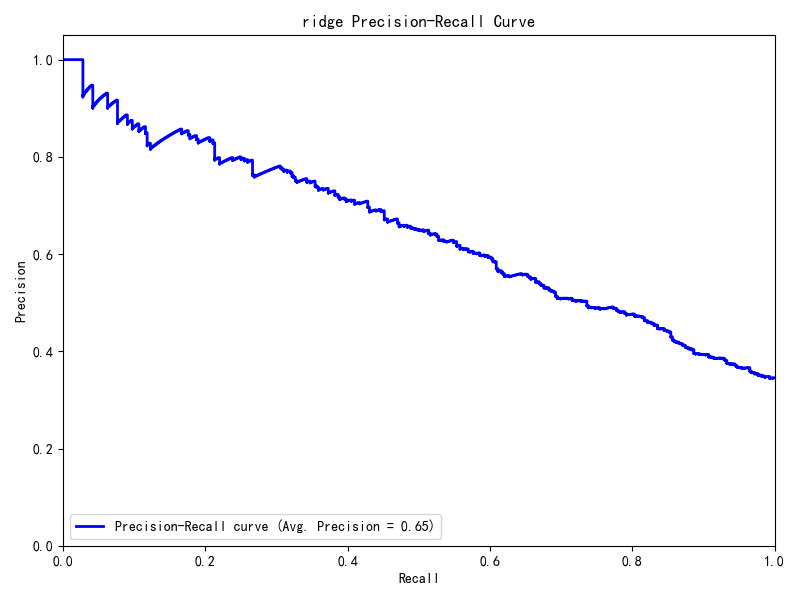
1. 岭回归ROC曲线

Figure 20 Ridge regression ROC curve

从图中可以看出，岭回归的AUC值为0.75，表明该模型具有一定的分辨能力。

精确度-召回率曲线（Precision-Recall Curve）是用于评估二元分类模型性能的一种可视化工具，它显示了模型在不同阈值下的精确度和召回率之间的权衡关系。这种曲线通常用于处理不平衡数据集，其中一类的样本数量远远超过另一类的样本数量。

本文考虑到出院状态为1（死亡）的患者相对较少，因此绘制精确度-召回率曲线考察模型的效果，如下图所示：



1. 岭回归精确度-召回率曲线

Fig. 21 Ridge regression precision-recall curve

从图中可以看出，在不同召回率水平下的平均精确度是0.65。

## 主成分分析

主成分分析（Principal Component Analysis，简称PCA）是一种多元统计分析方法，用于降低数据维度并提取数据中的主要信息。它是一种无监督学习技术，广泛用于数据降维、可视化、噪声滤除和特征提取等领域。以下是主成分分析的详细介绍：

主成分分析的基本原理：PCA的主要目标是找到数据中的主成分，这些主成分是原始特征的线性组合。主成分通常按方差的大小进行排序，因此第一个主成分包含最大方差，第二个主成分包含次大方差，以此类推。通过保留较少数量的主成分，可以降低数据的维度，同时尽量保留原始数据中的信息。

主成分分析的基本步骤包括：

1、标准化数据：首先，对数据进行标准化，确保各个特征具有相似的尺度。这通常涉及将数据特征的均值归零，方差归一。

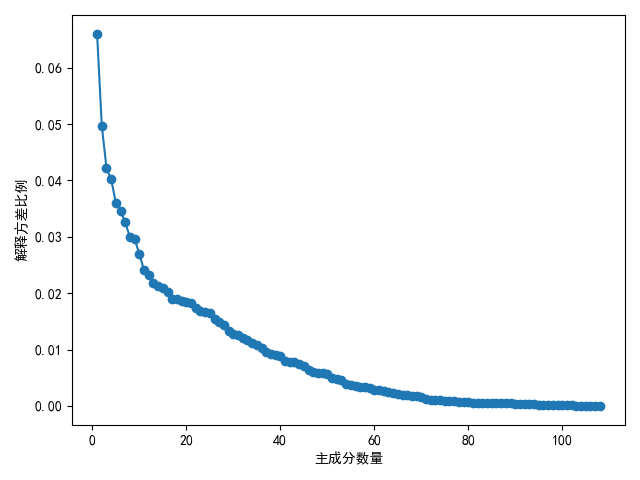
2、计算协方差矩阵：然后，计算数据特征之间的协方差矩阵。协方差矩阵描述了特征之间的线性关系。

3、计算特征值和特征向量：通过解决协方差矩阵的特征值问题，可以找到主成分。特征向量是主成分的方向，而特征值表示相应方向上的方差。

4、选择主成分：根据特征值的大小，选择要保留的主成分数量。通常，只保留前几个主成分，以保留大部分原始数据的方差。

5、投影数据：使用所选的主成分，将数据投影到新的主成分空间。这会生成降维后的数据集。

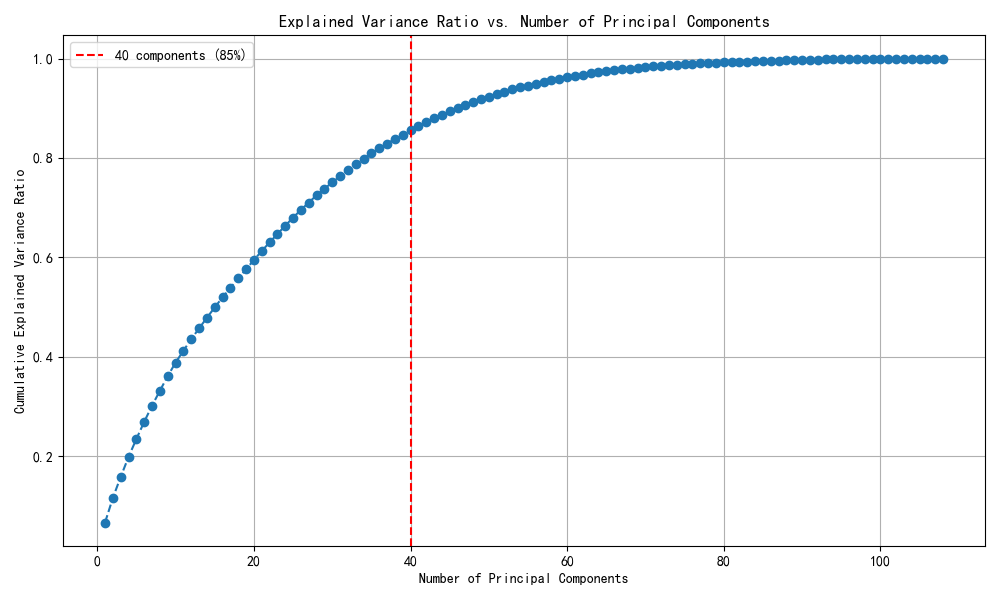
绘制109个变量的主成分数量与解释方差比例的关系图如下所示：



1. 主成分数量与解释方差比例关系图

Fig. 22 Plot of the number of principal components versus the proportion of variance explained

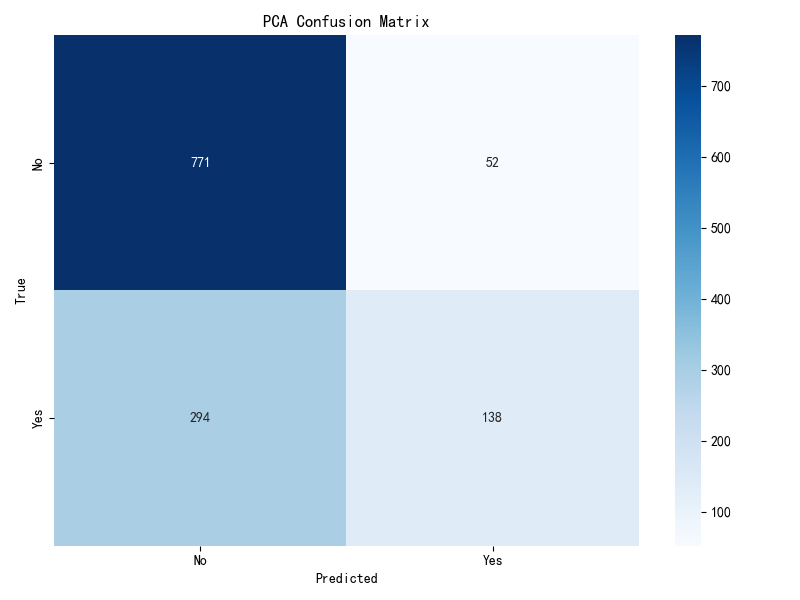
从中可以看出，解释方差比例逐渐减小。第一个主成分包含数据中的大部分方差，第二个主成分包含次大部分，前几个主成分贡献了大部分数据方差。绘制主成分数量与累计解释方差比例的关系图，如下图所示：



1. 主成分数量与累计解释方差比例关系图

Fig. 23 Plot of the number of principal components versus the proportion of cumulative explained variance

根据上图可知，当主成分数量达到40个时，累计的解释方差比例可以达到85%，因此，对数据中除了基线数据以外的109个变量先进行主成分降维（主成分数量为40个），然后使用Logit模型回归预测，混淆矩阵如下图：

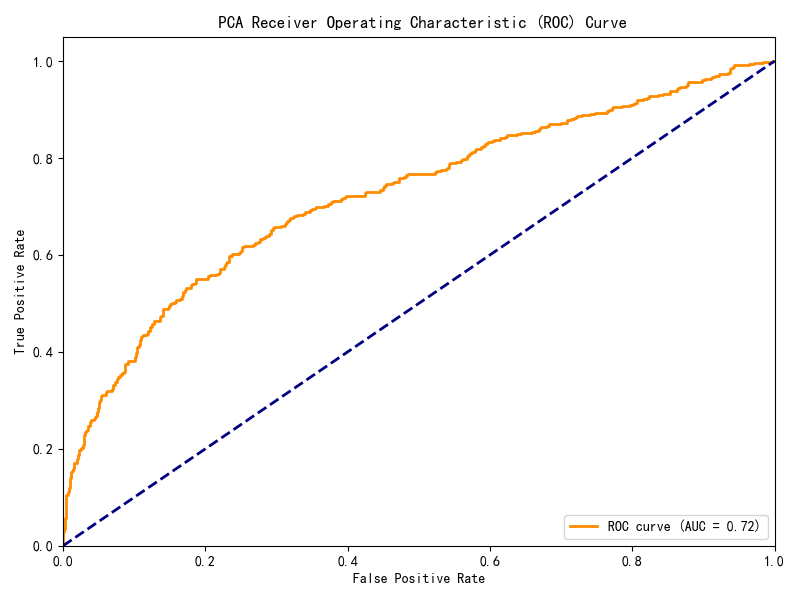


1. 主成分分析混淆矩阵

Figure 24 Principal Component Analysis Confusion Matrix

主成分降维后使用Logit模型回归的准确率为0.724、召回率为0.319、精确率为0.726、F1分数为0.444。

绘制ROC曲线和精确度-召回率曲线如下所示：



1. 主成分分析ROC曲线

Figure 25 Principal component analysis ROC curve

图表, 折线图

描述已自动生成

1. 主成分分析精确度-召回率曲线

Figure 26 Principal Component Analysis Precision-Recall Curve

从图中可以看出，AUC值为0.72，平均精确度为0.63。

# 分类模型

## 随机森林模型

随机森林（Random Forest）是一种强大的集成学习算法，用于解决分类和回归问题。它结合了决策树的强大性能和降低过拟合风险的方法，通过构建多个决策树并汇总它们的预测结果来提高模型的准确性和鲁棒性。以下是关于随机森林的详细介绍：

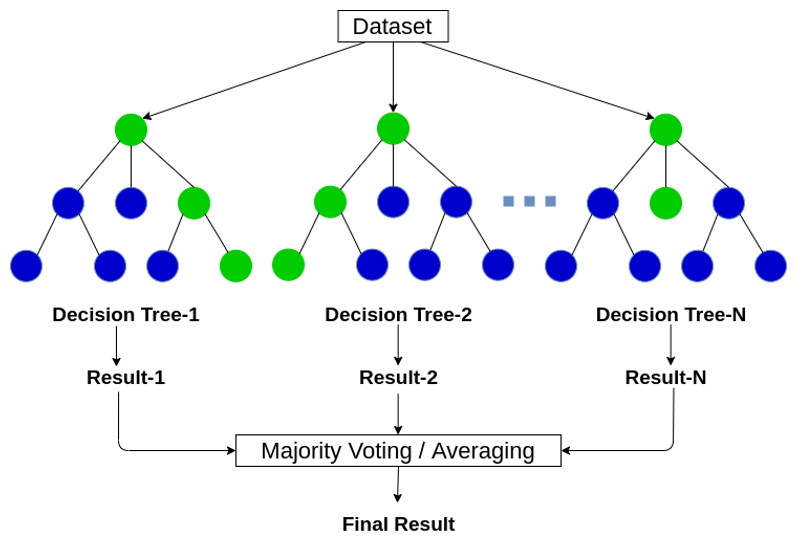
1、决策树的基础：随机森林的基础是决策树，决策树是一种树状结构，用于根据一系列特征来进行决策。每个决策树将数据集分成多个节点，每个节点根据一个特征进行分割。最终，叶节点包含一个特定的类别或回归值。

2、多个决策树的集成：随机森林通过构建多个决策树来进行集成学习。每个决策树都是独立构建的，使用不同的数据子集和特征子集。这种独立性使得每个决策树都有自己的特点，减少了过拟合的风险。

3、Bagging (Bootstrap Aggregating)：在构建每个决策树时，随机森林使用Bagging技术，即Bootstrap Aggregating。Bagging从原始数据集中随机选择多个子集，这些子集可能包含重复的样本。每个决策树都是在一个子集上进行训练，这样可以增加模型的多样性。

4、特征子集选择：在每个节点分裂时，随机森林不考虑所有特征，而是从所有特征中随机选择一个子集。这种特征子集的选择进一步增加了模型的多样性，有助于降低模型的方差。

5、预测过程：对于分类问题，随机森林的预测是通过投票方式进行的，即每个决策树投票选择一个类别，最终的预测结果是获得最多投票的类别。对于回归问题，随机森林的预测是基于多个决策树的平均或加权平均。



1. 随机森林示意图

Figure 27 Schematic diagram of a random forest

**优点**：

1、随机森林对于高维数据和大型数据集具有很好的性能。

2、由于采用Bagging和特征子集选择，随机森林对过拟合有较好的抵抗能力。

3、随机森林可以处理分类和回归问题，并且对于非线性关系的数据拟合效果较好。

**缺点**：

1、随机森林可能不太适用于具有大量类别的分类问题，因为它在多类别分类时可能不够敏感。

2、随机森林在处理稀疏数据（例如文本数据）时效果可能不如其他方法。

3、重要性评估：随机森林可以估计特征的重要性，通过观察哪些特征在多个决策树中对模型性能有较大影响来进行评估。这有助于特征选择和解释模型。

总之，随机森林是一种强大的机器学习算法，适用于各种问题，尤其在复杂数据集和高维数据中表现出色。它的能力来自于构建多个独立决策树，并通过集成它们的预测结果来减少过拟合风险和提高模型性能。

网格搜索（Grid Search）和交叉验证（Cross-Validation）是机器学习中常用的技术，用于帮助选择最佳的模型参数以及评估模型性能。

**网格搜索 (Grid Search)**:

1、网格搜索是一种参数调优的方法，用于确定模型的最佳超参数（超参数是在模型训练之前需要设置的参数）组合。

2、基本思想是将不同的超参数组合成一个参数网格，然后对每一种组合进行模型训练和评估。

3、网格搜索会遍历所有可能的参数组合，因此它是一种穷举搜索方法。

4、通常结合交叉验证一起使用，以确保模型性能评估的可靠性。

**交叉验证 (Cross-Validation)**:

1、交叉验证是一种用于评估模型性能的技术，可以减小过拟合风险。

2、基本思想是将数据集分为多个子集，然后多次训练模型，每次使用不同的子集作为验证集，其余子集作为训练集。

3、最常见的交叉验证是k折交叉验证，其中数据分成k个子集，依次使用其中一个子集作为验证集，其余k-1个子集作为训练集，重复k次。

4、交叉验证可以提供更可靠的模型性能评估，因为它对模型在不同数据子集上的表现进行了多次测试。

通常，网格搜索和交叉验证结合使用，用于选择最佳的模型超参数并评估模型性能。具体步骤如下：

1、定义参数网格：确定需要调优的模型超参数和其可能的取值范围。

2、创建交叉验证对象：选择适当的交叉验证策略，如k折交叉验证。

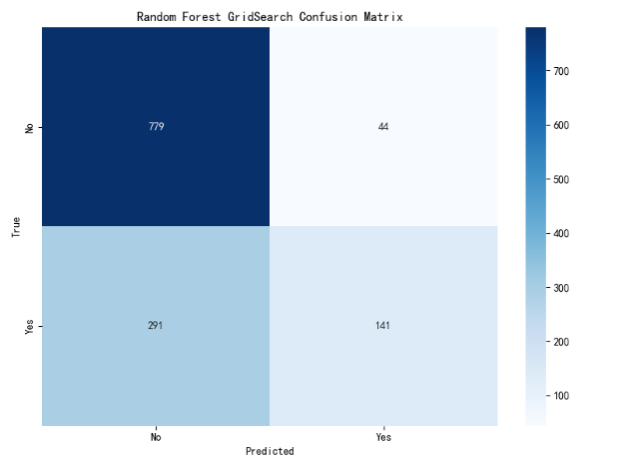
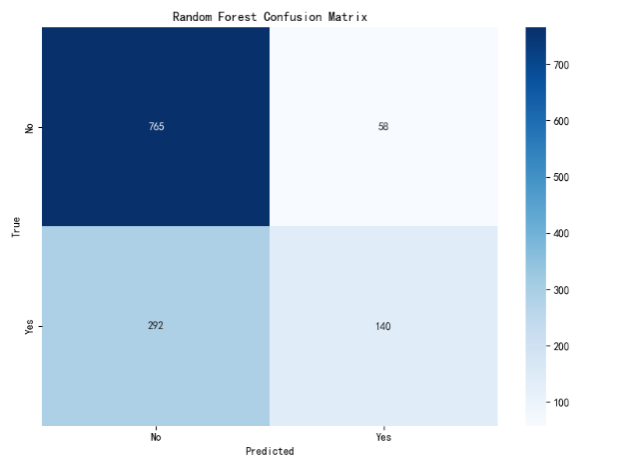
3、执行网格搜索：遍历参数网格中的所有组合，对每个组合执行交叉验证，记录模型性能指标，如准确度、F1分数等。

4、选择最佳参数组合：根据交叉验证结果，选择性能最佳的参数组合。

5、使用最佳模型：使用最佳参数组合训练模型，并在测试集上评估模型性能。

6、这种方法能够提高模型的泛化能力，避免模型在特定数据集上过度拟合，从而更好地适应新的未见数据

使用随机森林模型进行拟合，预测结果的混淆矩阵如下图所示：



1. 随机森林混淆矩阵

左图：默认参数 右图：最佳参数

Figure 28 Random Forest Confusion Matrix

a) default parameters b) optimal parameters

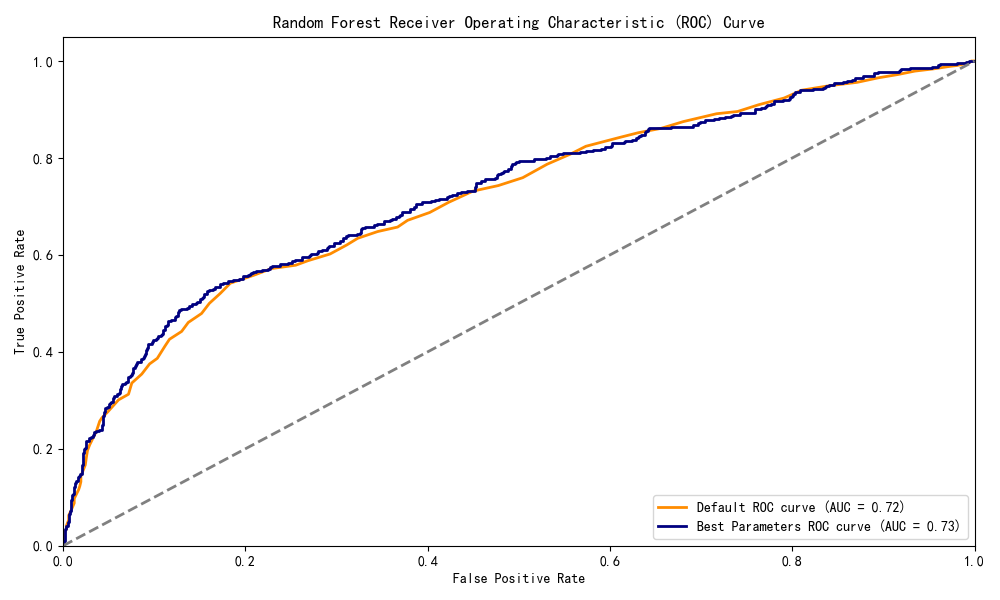
使用随机森林模型（默认参数）回归的准确率为0.721、召回率为0.324、精确率为0.707、F1分数为0.444。

随机森林模型的默认参数：

* n\_estimators（默认值：100）：这个参数表示随机森林中包含的决策树数量。增加决策树数量通常会提高模型性能，但也会增加计算成本。通常，增加树的数量会提高模型的鲁棒性，减少过拟合的风险。
* max\_depth（默认值：None）：这个参数控制每棵决策树的最大深度。如果不指定深度，决策树会生长直到每个叶子节点都包含少于min\_samples\_split个样本或满足min\_samples\_leaf的条件为止。控制树的深度可以防止过拟合。
* min\_samples\_split（默认值：2）：这是一个用于控制决策树分裂的最小样本数。如果一个节点的样本数量小于等于min\_samples\_split，则不再继续分裂。增加此参数的值可以使模型更稳定，减少过拟合风险。
* min\_samples\_leaf（默认值：1）：这个参数控制叶子节点上的最小样本数。如果一个叶子节点上的样本数量小于等于min\_samples\_leaf，则该节点不会再分裂。增加此参数的值可以进一步减少模型的过拟合风险。

定义网格搜索的范围：'n\_estimators': [100, 200, 300, 400]，'max\_depth': [None,5, 10, 20, 30]，'min\_samples\_split': [2, 5, 10, 15]，'min\_samples\_leaf': [1, 2, 4]。搜索结果中最佳参数组合是{'max\_depth': 20, 'min\_samples\_leaf': 4, 'min\_samples\_split': 15, 'n\_estimators': 100}。随机森林模型（最佳参数参数）回归的准确率为0.733、召回率为0.326、精确率为0.762、F1分数为0.457。

对比使用默认参数和最佳参数的ROC曲线，如下图所示：



1. 随机森林ROC曲线

Figure 29 Random Forest ROC curve

从图中可以看出，默认参数的曲线整体低于最佳参数的曲线，默认参数下AUC值为0.72，最佳参数下AUC值为0.73。

进一步对比精确度-召回率曲线，如下图所示：

图表, 折线图

描述已自动生成

1. 随机森林精确度-召回率曲线

Fig. 30 Random forest precision-recall curve

从图中可以看出，默认参数的曲线整体低于最佳参数的曲线，默认参数下平均精确度为0.60，最佳参数下平均精确度为0.62。

## XGBoost

XGBoost（eXtreme Gradient Boosting）是一种高效的梯度提升树（Gradient Boosting Tree）算法，用于解决分类和回归问题。它是一种集成学习方法，以极高的性能和可扩展性而闻名。以下是对XGBoost的详细介绍：

1、梯度提升树（Gradient Boosting Tree）的基础：XGBoost的基础是梯度提升树，它是一种集成学习方法，通过逐步构建决策树来提高模型性能。每颗决策树都是在前一颗树的残差（误差的负梯度）基础上构建的。

2、正则化：XGBoost引入了正则化项，包括L1（Lasso）和L2（Ridge）正则化，以控制模型的复杂度和降低过拟合风险。这使得XGBoost在处理高维数据时更加稳健。

3、自定义损失函数：XGBoost允许用户自定义损失函数，这意味着可以根据问题的特性创建适合的损失函数。这使得XGBoost适用于各种问题，包括排序、推荐系统等。

4、特征选择：XGBoost提供了特征选择的工具，可以估计特征的重要性，帮助用户识别哪些特征对于模型的性能最重要。

5、可解释性：XGBoost提供了可视化工具，可以解释模型的预测过程，例如特征的重要性和分裂点。

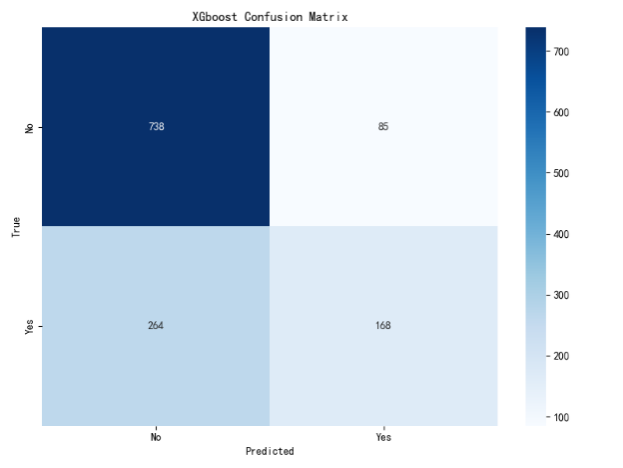
6、逐步增强性：XGBoost通过逐步增加树的数量来提高模型性能，可以设置停止条件以避免过拟合。

7、多语言支持：XGBoost有多种语言的接口，包括Python、R、Java、Scala等，因此适用于不同的数据科学和机器学习环境。

8、XGBoost的速度：XGBoost以其高度优化的算法和并行计算而闻名，通常比其他梯度提升方法快得多。

总之，XGBoost是一种强大的机器学习算法，适用于各种问题，尤其在大规模和高维数据集中表现出色。它的特性包括高效的计算、正则化、自定义损失函数和可解释性，使其成为数据科学家和机器学习工程师的首选工具之一。XGBoost在解决分类和回归问题中取得了卓越的性能，并且在实际应用中被广泛采用。

使用XGBoost进行拟合，预测结果的混淆矩阵如下图所示：

图表, 树状图

描述已自动生成

1. 随机森林混淆矩阵

左图：默认参数 右图：最佳参数

Figure 31 Random Forest Confusion Matrix

a) default parameters b) optimal parameters

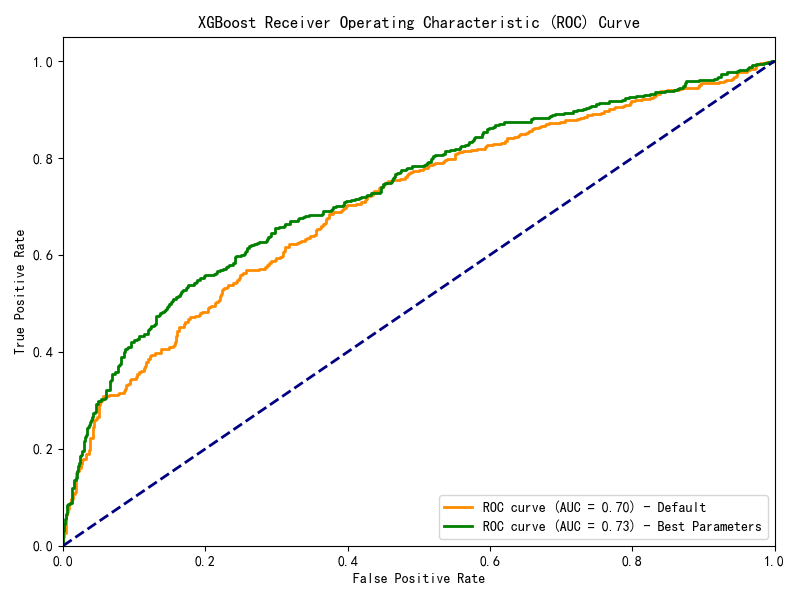
使用XGBoost（默认参数）回归预测的准确率为0.706、召回率为0.366、精确率为0.625、F1分数为0.461。

XGBoost的默认参数：

* n\_estimators（默认值：100）：树的数量，即弱学习器的数量。
* max\_depth（默认值：6）：每棵树的最大深度。它控制每棵树的生长深度，可以用于防止过拟合。
* learning\_rate（默认值：0.3）：学习率，控制每棵树的贡献程度。较小的学习率会使模型更加稳定，但需要更多的树来达到相同的性能。
* subsample（默认值：1.0）：用于训练每棵树的数据子样本的比例。较小的值可以防止过拟合。
* colsample\_bytree（默认值：1.0）：用于训练每棵树的特征子样本的比例。它可以控制每棵树对特征的随机选择，有助于模型的多样性。

定义网格搜索的范围：'n\_estimators': [100, 200, 300]，'max\_depth': [3, 4, 5]，'learning\_rate': [0.01, 0.1,0.3]，'subsample': [0.8, 0.9, 1.0]，'colsample\_bytree': [0.8, 0.9, 1.0]；模型的最佳参数组合是{'colsample\_bytree': 0.8, 'learning\_rate': 0.1, 'max\_depth': 4, 'n\_estimators': 100, 'subsample': 0.8}。XGBoost（最佳参数参数）回归的准确率为0.730、召回率为0.359、精确率为0.714、F1分数为0.478。

对比使用默认参数和最佳参数的ROC曲线，如下图所示：



1. XGBoost ROC曲线

Figure 32 XGBoost ROC curve

从图中可以看出，默认参数的曲线整体低于最佳参数的曲线，默认参数下AUC值为0.70，最佳参数下AUC值为0.73。

进一步对比精确度-召回率曲线，如下图所示：

图表, 折线图

描述已自动生成

1. XGBoost精确度-召回率曲线

Fig. 33 XGBoost precision-recall curve

从图中可以看出，默认参数的曲线整体低于最佳参数的曲线，默认参数下平均精确度为0.59，最佳参数下平均精确度为0.63。

## 考虑基线数据的模型构建

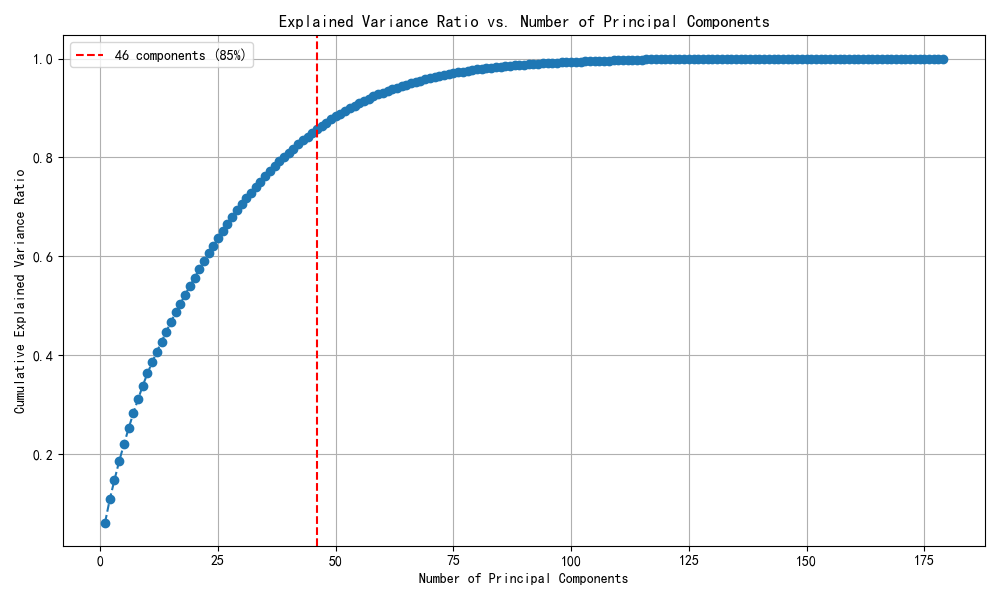
将mydata中的基线数据纳入现有解释变量中，进行最佳参数组合下的模型构建，考虑到基线数据中含有多个类别型变量，因此使用独热编码。

独热编码（One-Hot Encoding）是一种常用的数据编码技巧，用于将分类数据（也称为离散数据或类别数据）转换为数值数据，以便在机器学习模型中使用。它通常用于处理分类特征，如颜色、国家、性别等，以便将它们包括在模型训练中，因为大多数机器学习算法要求输入是数值型的。

独热编码的工作原理如下：

* 对于每个不同的类别特征，创建一个新的二进制特征（也称为虚拟特征）。
* 对于原始特征中的每个样本，将相应的二进制特征设置为1，表示该样本属于该类别，而其他二进制特征都设置为0。

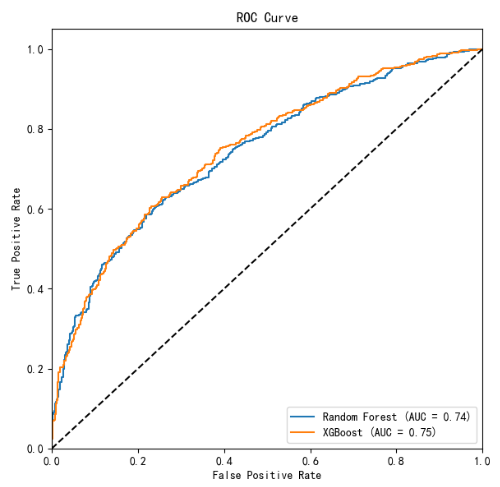
将gender、ethnicity、first\_hosp\_stay、first\_icu\_stay、long\_title、hospstay\_seq这些特征进行独热编码，然后绘制纳入基线数据的主成分数量与累计解释方差比例的关系图，如下图所示：



1. 纳入基线数据的主成分数量与累计解释方差关系图

Figure 34 Number of principal components included in the baseline data plotted against cumulative explained variance

因此，将主成分的数量设置为46。然后使用网格搜索找到随机森林和XGBoost模型的最佳参数，在最佳参数组合下进行回归预测，预测的ROC曲线和精确-召回曲线如下图所示：



1. 纳入基线数据的ROC曲线

Figure 35 ROC curves incorporating baseline data

图表, 折线图

描述已自动生成

1. 纳入基线数据的精确-召回曲线

Figure 36 Precision-recall curves incorporating baseline data

从中可以看出，最佳参数组合下，XGBoost模型的AUC值（0.75）大于随机森林的AUC值（0.74），两个模型的平均精确度均为0.64。

# 附录

**问题1代码**

#导入相关库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

library(reshape2)

#读取目录下mydata.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#########进行性别与出院死亡率数据合成

#取出列名为gender的数据赋给Var\_Gender,类型为字符数组.同理其他

Var\_Gender=mydata\_mean$gender

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

#合并成数据框

Fre\_Gender\_Exflag=as.data.frame(table(Var\_Gender,Var\_ex\_flag))

#重命名数据框列名称

colnames(Fre\_Gender\_Exflag)=c("gender","flag","number")

#因子化flag变量，不把他当成一个数值量

Fre\_Gender\_Exflag$flag=factor(Fre\_Gender\_Exflag$flag,levels = c("1","0"))

#计算不同性别的死亡比例

ratio = c(Fre\_Gender\_Exflag$number[3]/(Fre\_Gender\_Exflag$number[1]+Fre\_Gender\_Exflag$number[3]),Fre\_Gender\_Exflag$number[4]/(Fre\_Gender\_Exflag$number[2]+Fre\_Gender\_Exflag$number[4]),Fre\_Gender\_Exflag$number[3]/(Fre\_Gender\_Exflag$number[1]+Fre\_Gender\_Exflag$number[3]),Fre\_Gender\_Exflag$number[4]/(Fre\_Gender\_Exflag$number[2]+Fre\_Gender\_Exflag$number[4]))

#保留4位小数，四舍五入

ratio = round(ratio,4)

#将data2与ratio合并为数据框

Fre\_Gender\_Exflag = data.frame(Fre\_Gender\_Exflag,ratio)

######性别与出院状态关系图开始

g1=ggplot(data = Fre\_Gender\_Exflag,aes(x=gender,y=number,fill=flag)) +

geom\_bar(position = "stack",stat = "identity",color="white",alpha=0.8)+

scale\_fill\_manual(values=c('#fec79e','#8ec4cb'))+ #给柱状图填充指定颜色

geom\_text(aes(label=number),position=position\_stack(),vjust=-0.3,hjust=0.5)+ #添加数据标注并调整位置

labs(x='gender',y='Number of people counted')+ #设置坐标

theme\_test(base\_size = 15)+

theme(legend.position = 'none', panel.border = element\_rect(size=2,fill = 'transparent'), axis.text = element\_text(color='black'))+ geom\_rect(aes(xmin=0.7,xmax=0.9,ymin=3500,ymax=3600), fill='#8ec4cb',color='#8ec4cb')+geom\_rect(aes(xmin=0.7,xmax=0.9,ymin=3700,ymax=3800), fill='#fec79e',color='#fec79e')+geom\_rect(aes(xmin=0.6,xmax=1.3,ymin=3400,ymax=3900), fill='#D3D3D3',color='#D3D3D3',alpha=0)+

annotate(geom='text',x=1.1,y=3570,label='flag=0',size=4.5)+

annotate(geom='text',x=1.1,y=3770,label='flag=1',size=4.5)+

#设置双Y轴，对第二个坐标轴进行相应的放大，设置间隔

scale\_y\_continuous(limits=c(0,4000),

breaks = seq(0,4000,500),

sec.axis = sec\_axis(~./4000,name = 'ratio',breaks=seq(0,1,0.1)))+

geom\_point(data=Fre\_Gender\_Exflag, aes(factor(gender),ratio\*4000),size=4)+

geom\_line(data=Fre\_Gender\_Exflag, aes(factor(gender),ratio\*4000),group=1,cex=0.8)+

geom\_text(aes(x=gender,y=ratio\*4000,label=ratio),vjust=-0.8,hjust=0.5)+

scale\_color\_manual(values = c('#ff8c3e'))

#输出为tiff格式图片，分辨率600，按lzw进行压缩。

ggsave("gender\_flag1.tiff",dpi=600, compression = 'lzw')

dev.off()

######性别与出院状态关系图结束

#########进行年龄与出院死亡率数据合成

Var\_age=floor(mydata\_mean$admission\_age)

Fre\_age\_Exflag=data.frame(Var\_age,mydata\_mean$hospital\_expire\_flag)

Fre\_age\_Exflag$agegroup=cut(Fre\_age\_Exflag$Var\_age,breaks=c(0,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,100),labels=c(20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95),right=TRUE,include.lowest=FALSE)

#合并成数据框

Fre\_agegroup\_Exflag=as.data.frame(table(Fre\_age\_Exflag$agegroup,Fre\_age\_Exflag$mydata\_mean.hospital\_expire\_flag))

#重命名数据框列名称

colnames(Fre\_agegroup\_Exflag)=c("Age","flag","number")

#因子化flag变量，不把他当成一个数值量

Fre\_agegroup\_Exflag$flag=factor(Fre\_agegroup\_Exflag$flag,levels = c("1","0"))

#去因子化,这样对X轴进行更改可以利用相关函数操作,否则比较麻烦

Fre\_agegroup\_Exflag$Age=as.numeric(as.character(Fre\_agegroup\_Exflag$Age))

#计算各年龄层与死亡率的关系

age\_ratio=rep(1.0,32)

for (i in 1:32){

if(i<17)

{

age\_ratio[i]=Fre\_agegroup\_Exflag$number[i+16]/(Fre\_agegroup\_Exflag$number[i+16]+Fre\_agegroup\_Exflag$number[i])

}

else

{

age\_ratio[i]=Fre\_agegroup\_Exflag$number[i]/(Fre\_agegroup\_Exflag$number[i-16]+Fre\_agegroup\_Exflag$number[i])

}

}

age\_ratio = round(age\_ratio,2)

Fre\_agegroup\_Exflag = data.frame(Fre\_agegroup\_Exflag,age\_ratio)

#########进行年龄与出院死亡率数据合成结束

######年龄与出院状态关系图开始

g2=ggplot(data = Fre\_agegroup\_Exflag,aes(x=Age,y=number,fill=flag)) +

geom\_bar(position = "stack",stat = "identity",alpha=0.9)+

scale\_fill\_manual(values=c('#fec79e','#8ec4cb'))+ #给柱状图填充指定颜色

# geom\_text(aes(label=number),position=position\_stack(),vjust=-0.3,hjust=0.5)+ #添加数据标注并调整位置

labs(x='age',y='Number of people counted')+ #设置坐标

theme\_test(base\_size = 20)+

theme(legend.position = 'none',

panel.border = element\_rect(size=2,fill = 'transparent'),

axis.text = element\_text(color='black'),

axis.title.y.right =element\_text(vjust=2) )+

geom\_rect(aes(xmin=22,xmax=28,ymin=660,ymax=680),

fill='#8ec4cb',color='#8ec4cb')+

geom\_rect(aes(xmin=22,xmax=28,ymin=700,ymax=720),

fill='#fec79e',color='#fec79e')+

geom\_rect(aes(xmin=20,xmax=45,ymin=630,ymax=745),#211,211,211

fill='#D3D3D3',color='#D3D3D3',alpha=0)+

annotate(geom='text',x=36,y=675,label='flag=0',size=4.5)+

annotate(geom='text',x=36,y=715,label='flag=1',size=4.5)+

scale\_x\_continuous(limits = c(15,100),breaks = seq(20,100,10))+

# xlim(10,100)+

#scale\_x\_discrete(limits = c("20", "25 ","30", "35 ","40", "45 ","50", "55 ","60", "65 ","70", " 75","80", "85 ","90", " 95"))

#设置双Y轴，对第二个坐标轴进行相应的放大，设置间隔

scale\_y\_continuous(limits=c(0,800),

breaks = seq(0,800,200),

#expand=c(0,0), #坐标轴不留空白

sec.axis = sec\_axis(~./1000,name = 'ratio',breaks=seq(0,0.8,0.1)))+

geom\_point(data=Fre\_agegroup\_Exflag, aes(Age,age\_ratio\*1000),size=2)+

geom\_line(data=Fre\_agegroup\_Exflag, aes(Age,age\_ratio\*1000),cex=1)+

geom\_text(aes(x=Age,y=age\_ratio\*1000,label=age\_ratio),size=3.5,vjust=-2,hjust=0.6)

#输出为tiff格式图片，分辨率600，按lzw进行压缩。

#之所以没有规定大小，由于图中文字导出不随图形大小改变，按照默认大小输出，然后具体按比例缩放可以在其它工具进行如画图，PS中。

ggsave("age\_flag.tiff",dpi=600, compression = 'lzw')

dev.off()

######性别与出院状态关系图结束

####体重与出院关系图处理

Var\_weight=floor(mydata\_mean$weight\_admit)

Fre\_weight\_Exflag=data.frame(Var\_weight,flag=mydata\_mean$hospital\_expire\_flag)

Fre\_weight\_Exflag$weightgroup=cut(Fre\_weight\_Exflag$Var\_weight,breaks=c(0,60,80,100,120,140,160,180,Inf),labels=c(60,80,100,120,140,160,180,200),right=TRUE,include.lowest=FALSE)

#合并成数据框

Fre\_weightgroup\_Exflag=as.data.frame(table(Fre\_weight\_Exflag$weightgroup,Fre\_weight\_Exflag$flag))

#重命名数据框列名称

colnames(Fre\_weightgroup\_Exflag)=c("Weight","flag","number")

#因子化flag变量，不把他当成一个数值量

Fre\_weightgroup\_Exflag$flag=factor(Fre\_weightgroup\_Exflag$flag,levels = c("1","0"))

#去因子化,这样对X轴进行更改可以利用相关函数操作,否则比较麻烦

Fre\_weightgroup\_Exflag$Weight=as.numeric(as.character(Fre\_weightgroup\_Exflag$Weight))

#计算各年龄层与死亡率的关系

weight\_ratio=rep(1.0,length(Fre\_weightgroup\_Exflag$Weight))

for (i in 1:length(Fre\_weightgroup\_Exflag$Weight)){

if(i<9)

{

weight\_ratio[i]=Fre\_weightgroup\_Exflag$number[i+8]/(Fre\_weightgroup\_Exflag$number[i+8]+Fre\_weightgroup\_Exflag$number[i])

}

else

{

weight\_ratio[i]=Fre\_weightgroup\_Exflag$number[i]/(Fre\_weightgroup\_Exflag$number[i-8]+Fre\_weightgroup\_Exflag$number[i])

}

}

weight\_ratio = round(weight\_ratio,2)

Fre\_weightgroup\_Exflag = data.frame(Fre\_weightgroup\_Exflag,weight\_ratio)

#########进行年龄与出院死亡率数据合成结束

######体重与出院状态关系图开始

g3=ggplot(data = Fre\_weightgroup\_Exflag,aes(x=Weight,y=number,fill=flag)) +

geom\_bar(position = "stack",stat = "identity",alpha=0.9)+

scale\_fill\_manual(values=c('#fec79e','#8ec4cb'))+ #给柱状图填充指定颜色

# geom\_text(aes(label=number),position=position\_stack(),vjust=-0.3,hjust=0.5)+ #添加数据标注并调整位置

labs(x='weight',y='Number of people counted')+ #设置坐标

theme\_test(base\_size = 20)+

#theme\_bw()+

theme(legend.position = 'none',

panel.border = element\_rect(size=2,fill = 'transparent'),

axis.text = element\_text(color='black'),

axis.title.y.right =element\_text(vjust=2) )+

geom\_rect(aes(xmin=155,xmax=166,ymin=2580,ymax=2680),

fill='#8ec4cb',color='#8ec4cb')+

geom\_rect(aes(xmin=155,xmax=166,ymin=2770,ymax=2870),

fill='#fec79e',color='#fec79e')+

geom\_rect(aes(xmin=150,xmax=19,ymin=2480,ymax=2970),#211,211,211

fill='#D3D3D3',color='#D3D3D3',alpha=0)+

annotate(geom='text',x=180,y=2630,label='flag=0',size=4.5)+

annotate(geom='text',x=180,y=2820,label='flag=1',size=4.5)+

scale\_x\_continuous(limits = c(50,210),breaks = seq(60,200,20))+

# # xlim(10,100)+

#scale\_x\_discrete(limits = c("60", "80 ","100", " ","40", "45 ","50", "55 ","60", "65 ","70", " 75","80", "85 ","90", " 95"))

# #设置双Y轴，对第二个坐标轴进行相应的放大，设置间隔

#

scale\_y\_continuous(limits=c(0,3000),

breaks = seq(0,3000,500),

#expand=c(0,0), #坐标轴不留空白

sec.axis = sec\_axis(~./6000,name = 'ratio',breaks=seq(0,0.5,0.1)))+

geom\_point(data=Fre\_weightgroup\_Exflag, aes(Weight,weight\_ratio\*6000),size=2)+

geom\_line(data=Fre\_weightgroup\_Exflag, aes(Weight,weight\_ratio\*6000),cex=1)+

geom\_text(aes(x=Weight,y=weight\_ratio\*6000,label=weight\_ratio),size=3.5,vjust=-2,hjust=0.6)

#输出为tiff格式图片，分辨率600，按lzw进行压缩。

#之所以没有规定大小，由于图中文字导出不随图形大小改变，按照默认大小输出，然后具体按比例缩放可以在其它工具进行如画图，PS中。

ggsave("weight\_flag.tiff",dpi=600, compression = 'lzw')

dev.off()

######体重与出院状态关系图结束

####身高与出院关系图处理

Var\_height=floor(mydata\_mean$height)

Fre\_height\_Exflag=data.frame(Var\_height,flag=mydata\_mean$hospital\_expire\_flag)

Fre\_height\_Exflag$heightgroup=cut(Fre\_height\_Exflag$Var\_height,breaks=c(0,150,155,160,165,170,175,180,185,190,195,Inf),labels=c(150,155,160,165,170,175,180,185,190,195,200),right=TRUE,include.lowest=FALSE)

#合并成数据框

Fre\_heightgroup\_Exflag=as.data.frame(table(Fre\_height\_Exflag$heightgroup,Fre\_height\_Exflag$flag))

#重命名数据框列名称

colnames(Fre\_heightgroup\_Exflag)=c("Height","flag","number")

#因子化flag变量，不把他当成一个数值量

Fre\_heightgroup\_Exflag$flag=factor(Fre\_heightgroup\_Exflag$flag,levels = c("1","0"))

#去因子化,这样对X轴进行更改可以利用相关函数操作,否则比较麻烦

Fre\_heightgroup\_Exflag$Height=as.numeric(as.character(Fre\_heightgroup\_Exflag$Height))

#计算各年龄层与死亡率的关系

height\_ratio=rep(1.0,length(Fre\_heightgroup\_Exflag$Height))

for (i in 1:length(Fre\_heightgroup\_Exflag$Height)){

if(i<12)

{

height\_ratio[i]=Fre\_heightgroup\_Exflag$number[i+11]/(Fre\_heightgroup\_Exflag$number[i+11]+Fre\_heightgroup\_Exflag$number[i])

}

else

{

height\_ratio[i]=Fre\_heightgroup\_Exflag$number[i]/(Fre\_heightgroup\_Exflag$number[i-11]+Fre\_heightgroup\_Exflag$number[i])

}

}

height\_ratio = round(height\_ratio,2)

Fre\_heightgroup\_Exflag = data.frame(Fre\_heightgroup\_Exflag,height\_ratio)

#########进行身高与出院死亡率数据合成结束

######身高与出院状态关系图开始

g4=ggplot(data = Fre\_heightgroup\_Exflag,aes(x=Height,y=number,fill=flag)) +

geom\_bar(position = "stack",stat = "identity",alpha=0.9)+

scale\_fill\_manual(values=c('#fec79e','#8ec4cb'))+ #给柱状图填充指定颜色

# geom\_text(aes(label=number),position=position\_stack(),vjust=-0.3,hjust=0.5)+ #添加数据标注并调整位置

labs(x='height',y='Number of people counted')+ #设置坐标

theme\_test(base\_size = 20)+

#theme\_bw()+

theme(legend.position = 'none',

panel.border = element\_rect(size=2,fill = 'transparent'),

axis.text = element\_text(color='black'),

axis.title.y.right =element\_text(vjust=2) )+

geom\_rect(aes(xmin=185,xmax=190,ymin=1190,ymax=1220),

fill='#8ec4cb',color='#8ec4cb')+

geom\_rect(aes(xmin=185,xmax=190,ymin=1270,ymax=1300),

fill='#fec79e',color='#fec79e')+

geom\_rect(aes(xmin=182,xmax=203,ymin=1140,ymax=1340),#211,211,211

fill='#D3D3D3',color='#D3D3D3',alpha=0)+

annotate(geom='text',x=196,y=1210,label='flag=0',size=4.5)+

annotate(geom='text',x=196,y=1290,label='flag=1',size=4.5)+

scale\_x\_continuous(limits = c(145,205),breaks = seq(150,200,10))+

# # xlim(10,100)+

#scale\_x\_discrete(limits = c("60", "80 ","100", " ","40", "45 ","50", "55 ","60", "65 ","70", " 75","80", "85 ","90", " 95"))

# #设置双Y轴，对第二个坐标轴进行相应的放大，设置间隔

#

scale\_y\_continuous(limits=c(0,1350),

breaks = seq(0,1250,250),

#expand=c(0,0), #坐标轴不留空白

sec.axis = sec\_axis(~./2700,name = 'ratio',breaks=seq(0,0.5,0.1)))+

geom\_point(data=Fre\_heightgroup\_Exflag, aes(Height,height\_ratio\*2600),size=2)+

geom\_line(data=Fre\_heightgroup\_Exflag, aes(Height,height\_ratio\*2600),cex=1)+

geom\_text(aes(x=Height,y=height\_ratio\*2600,label=height\_ratio),size=3.5,vjust=-2,hjust=0.6)

#输出为tiff格式图片，分辨率600，按lzw进行压缩。

#之所以没有规定大小，由于图中文字导出不随图形大小改变，按照默认大小输出，然后具体按比例缩放可以在其它工具进行如画图，PS中。

ggsave("height\_flag.tiff",dpi=600, compression = 'lzw')

dev.off()

######身高与出院状态关系图结束

**问题2代码**

#导入相关库

library(openxlsx)

#读取目录下mydata\_mean.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#####性别与出院状态的卡方分析

#取出列名为gender的数据赋给Var\_Gender,类型为字符数组.同理其他

Var\_Gender=mydata\_mean$gender

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

#统计性别与出院状态的频数表

Tab\_Gender\_Exflag=table(Var\_Gender,Var\_ex\_flag)

#调用卡方分析函数

Result\_Gender\_Exflag=chisq.test(Tab\_Gender\_Exflag)

#####性别与出院状态的卡方分析

#####年龄与出院状态的卡方分析

Var\_age=floor(mydata\_mean$admission\_age)

Agegroup=cut(Var\_age,breaks=c(0,20,30,40,50,60,70,80,90,100),labels=c(20,30,40,50,60,70,80,90,100),right=TRUE,include.lowest=FALSE)

#统计年龄与出院状态的频数表

Tab\_Age\_Exflag=table(Agegroup,Var\_ex\_flag)

#调用卡方分析函数.由于理论计算值中含有小于5的值，利用修正的卡方分析或者fisher检验

ChiResult\_Age\_Exflag=chisq.test(Tab\_Age\_Exflag,simulate.p.value = TRUE)

FisResult\_Age\_Exflag=fisher.test(Tab\_Age\_Exflag,simulate.p.value = TRUE)

#####年龄与出院状态的卡方分析

#####体重与出院状态的卡方分析

Var\_weight=floor(mydata\_mean$weight\_admit)

weightgroup=cut(Var\_weight,breaks=c(0,40,60,80,100,120,140,160,180,Inf),labels=c(40,60,80,100,120,140,160,180,200),right=TRUE,include.lowest=FALSE)

#统计体重与出院状态的频数表

Tab\_Weight\_Exflag=table(weightgroup,Var\_ex\_flag)

#调用卡方分析函数

Result\_Weight\_Exflag=chisq.test(Tab\_Weight\_Exflag)

#####体重与出院状态的卡方分析

#####身高与出院状态的卡方分析

Var\_height=floor(mydata\_mean$height)

heightgroup=cut(Var\_height,breaks=c(0,150,160,170,180,190,Inf),labels=c(150,160,170,180,190,200),right=TRUE,include.lowest=FALSE)

#统计身高与出院状态的频数表

Tab\_height\_Exflag=table(heightgroup,Var\_ex\_flag)

#调用卡方分析函数

Result\_height\_Exflag=chisq.test(Tab\_height\_Exflag)

#####身高与出院状态的卡方分析

**问题3代码**

**###求后验分布**

#导入相关库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

#读取目录下mydata\_mean.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

# 步骤1：设定先验分布

prior\_alpha <- 35 # 先验分布的参数alpha

prior\_beta <- 65 # 先验分布的参数beta

#后验分布参数计算（直接利用伯努利分布与Beta分布的特点，得到后验分布仍然为Beta分布）

posterior\_alpha <- prior\_alpha + sum(Expire\_flag)

posterior\_beta <- prior\_beta + length(Expire\_flag) - sum(Expire\_flag)

# 后验分布的概率密度函数

posterior\_density2 <- dbeta(probs, posterior\_alpha, posterior\_beta)

ggplot(data.frame(probs,posterior\_density2),aes(probs,posterior\_density2))+

geom\_line(colour="blue",size=1)+ labs(x='Mortality θ ',y='Mortality Posterior Density')

ggsave("后验概率概率密度分布\_新.tiff",dpi=600, compression = 'lzw')

**###求画出θ的点估计值随时间变动的曲线图，（3）并添加相应的贝叶斯可信区间**

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

#读取目录下mydata\_mean.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#选取前100个Y的观测数据作为先验

# 步骤1：设定先验分布

prior\_alpha <- sum(mydata\_mean$hospital\_expire\_flag[1:100]) # 先验分布的参数alpha

prior\_beta <- 100-prior\_alpha # 先验分布的参数beta

#步骤二更新后验分布

posterior\_mean=rep(1.0,length(mydata\_mean$hospital\_expire\_flag)-99)

posterior\_mean[1]=prior\_alpha / (prior\_alpha + prior\_beta)

posterior\_alpha =rep(1.0,length(mydata\_mean$hospital\_expire\_flag)-100)

posterior\_beta =rep(1.0,length(mydata\_mean$hospital\_expire\_flag)-100)

posterior\_interval =matrix(1.0,length(mydata\_mean$hospital\_expire\_flag)-99,2)

posterior\_interval[1,]=qbeta(c(0.025, 0.975), prior\_alpha, prior\_beta)

for (i in 101:length(mydata\_mean$hospital\_expire\_flag))

{

posterior\_alpha[i-100]=sum(mydata\_mean$hospital\_expire\_flag[1:i])

posterior\_beta[i-100]=i-sum(mydata\_mean$hospital\_expire\_flag[1:i])

posterior\_mean[i-100+1]= posterior\_alpha[i-100]/(posterior\_alpha[i-100]+ posterior\_beta[i-100])

posterior\_interval[i-100+1,] <- qbeta(c(0.025, 0.975), posterior\_alpha[i-100], posterior\_beta[i-100])

}

observation=100:length(mydata\_mean$hospital\_expire\_flag)

ggplot(data.frame(posterior\_mean,observation),aes(x=observation,y=posterior\_mean))+

geom\_line(colour="blue",size=0.5)+

geom\_ribbon(aes(ymin = posterior\_interval[,1], ymax = posterior\_interval[,2]), alpha = 0.3)+ # 添加置信区间

theme\_bw()+

theme\_test(base\_size = 15)+

labs(x=' Observation ',y='Mortality θ') +

scale\_x\_continuous(limits = c(-100,6500),breaks = seq(0,6000,2000))+

scale\_y\_continuous(limits=c(0.25,0.47),breaks = seq(0.25,0.45,0.05))+

geom\_hline(yintercept = posterior\_mean[6174], linetype = "dashed", color = "red")+

geom\_text(x = 4500, y = 0.26, label = "θ = 0.3204", color = "black",family = "sans",size=5)+

theme(text = element\_text(family ="sans"))

ggsave("概率随观察量变换分布.tiff",dpi=600, compression = 'lzw')

**问题4代码**

导入相关库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

library(mixtools)

library(MASS)

#读取目录下mydata.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#取出患者出院状态这一列

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

# 设置混合高斯分布的参数

mu0 <- 0

sigma0 <- 2

mu1 <- 4

sigma1 <- 2

# 生成对应状态的符合混合高斯分布的随机数 x

x <- ifelse(Var\_ex\_flag == 0, mvrnorm(sum(Var\_ex\_flag == 0), mu = mu0, Sigma = sigma0),

mvrnorm(sum(Var\_ex\_flag == 1), mu = mu1, Sigma = sigma1))

# 将 x 转换为数据框

data\_x <- data.frame(x)

# **任务4—1\_1 绘制x的概率密度图**

ggplot(data\_x, aes(x = x)) +

geom\_density( color = "darkblue", alpha = 0.7,linewidth=1) +

# labs(title = "Density Plot of x") +

xlab("x") +

ylab("density")

ggsave("随机数x概率密度分布.tiff",dpi=600, compression = 'lzw')

# 任务4—1\_2 将y和x转换为数据框

data\_Var\_ex\_flag\_X <- data.frame(Var\_ex\_flag, x)

flag=as.factor(Var\_ex\_flag)

# 绘制 x 在给定 y 的情况下各类别的概率密度图

ggplot(data\_Var\_ex\_flag\_X, aes(x = x, fill = flag)) +

geom\_density(color = "darkblue", alpha = 0.6) +

scale\_fill\_manual(values = c("lightblue", "lightgreen")) +

# labs(title = "Density Plot of x by y", fill = "y") +

xlab("x") +

ylab("density") +

theme(

legend.text = element\_text(),

legend.position = c(0.85,0.85),

legend.background =element\_blank() )

ggsave("随机数x概率密度分布\_分类.tiff",dpi=600, compression = 'lzw')

#分别取出出院状态为0与1的数据

flag1\_data <- data\_Var\_ex\_flag\_X[data\_Var\_ex\_flag\_X$Var\_ex\_flag == 1, ]

flag0\_data <- data\_Var\_ex\_flag\_X[data\_Var\_ex\_flag\_X$Var\_ex\_flag == 0, ]

# 绘制flag=1 x 的概率密度图

flag1\_data\_x=data.frame(flag1\_data$x)

colnames(flag1\_data\_x)=c("x")

ggplot(flag1\_data\_x, aes(x = x)) +

geom\_density(fill = "lightblue", color = "darkblue", alpha = 0.7) +

labs(title = "Density Plot of x") +

xlab("x") +

ylab("Density")

# 绘制flag=0 x 的概率密度图

flag0\_data\_x=data.frame(flag0\_data$x)

colnames(flag0\_data\_x)=c("x")

ggplot(flag0\_data\_x, aes(x = x)) +

geom\_density(fill = "lightgreen", color = "darkblue", alpha = 0.7) +

labs(title = "Density Plot of x") +

xlab("x") +

ylab("Density")

library(cowplot)

**# 任务4\_2**

**# 计算每个观测值来源于N(μ0, σ0)和N(μ1, σ1)的概率密度**

f0 <- dnorm(x, mean = mu0, sd = sigma0)

f1 <- dnorm(x, mean = mu1, sd = sigma1)

# 计算来源于1的相对概率f

f <- f1 / (f0 + f1)

p1=ggplot(data.frame(x,f0,f1,f))+

geom\_line(aes(x,f0),color="darkblue",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(a)", vjust = 1.5, size = 4.5)

p2=ggplot(data.frame(x,f0,f1,f))+

geom\_line(aes(x,f1),color="#0073C2FF",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(b)", vjust = 1.5, size = 4.5)

p3=ggplot(data.frame(x,f0,f1,f))+

geom\_line(aes(x,f),color="darkgreen",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(c)", vjust = 1.5, size = 4.5)

cowplot::plot\_grid(p1,p2,p3,nrow=3)

ggsave("观测值x不同来源概率密度分布3合1.tiff",dpi=600, compression = 'lzw')

library(pROC)

#以阈值为0.5为界

thresholds <- 0.5

# 初始化存储各指标的向量

fpr <- rep(0,1)

tpr <- rep(0,1)

f1\_score <- rep(0,1)

# 计算混淆矩阵和各指标

# 根据选择的阈值对样本进行分类

predicted <- ifelse(f > thresholds, 1, 0)

# 计算混淆矩阵

confusion\_matrix <- table(predicted, Var\_ex\_flag)

# 计算真阳性率、假阳性率和F1值

tp <- confusion\_matrix[2, 2]

tn <- confusion\_matrix[1, 1]

fp <- confusion\_matrix[2, 1]

fn <- confusion\_matrix[1, 2]

fpr <- fp / (fp + tn)

tpr\_sen\_recall <- tp / (tp + fn)

precision = tp / (tp + fp)

specificity = tn / (fp+tn)

accuracy=(tp+tn)/(tp+fp+tn+fn)

f1\_score = 2 \* precision\*tpr\_sen\_recall / (precision + tpr\_sen\_recall)

youden\_index = tpr\_sen\_recall + specificity -1

# 将相对概率f进行排序作为阈值

threshold\_f = seq(0, 1, by = 0.01)

# 初始化存储各指标的向量

fpr\_f <- rep(0, length(threshold\_f))

tpr\_f <- rep(0, length(threshold\_f))

for (i in 1:length(threshold\_f)) {

# 根据选择的阈值对样本进行分类

predicted <- ifelse(f >= threshold\_f[i], 1, 0)

# 计算混淆矩阵

confusion\_matrix\_f <- table(predicted, Var\_ex\_flag)

if(dim(confusion\_matrix\_f)[1]==2)

{

# 计算真阳性率、假阳性率和F1值

tp <- confusion\_matrix\_f[2, 2]

tn <- confusion\_matrix\_f[1, 1]

fp <- confusion\_matrix\_f[2, 1]

fn <- confusion\_matrix\_f[1, 2]

fpr\_f[i] <- fp / (fp + tn)

tpr\_f[i] <- tp / (tp + fn)

}

else

{

if(predicted[i]==1)

{

fpr\_f[i] <- 1

tpr\_f[i] <- 1

}

else

{

fpr\_f[i] <- 0

tpr\_f[i] <- 0

}

}

}

# 使用roc函数创建ROC曲线对象

roc\_obj <- roc(Var\_ex\_flag, f)

# 计算AUC值

auc\_value <- auc(roc\_obj)

print(paste("AUC:", auc\_value))

#绘制ROC曲线

ggplot(data.frame(fpr\_f,tpr\_f))+

geom\_line(aes(fpr\_f,tpr\_f),color="#0073C2FF",linewidth =1)+

xlab("False Positive Rate(FPR)") +

ylab("True Positive Rate(TPR)")+

annotate("text", x = 0.25, y = 0.75,

label = "Auc = 0.9769 ", vjust = 1.5, size = 5)+

# 添加水平参考线

geom\_segment(aes(x = 0, y = 0, xend = 1, yend = 1), linetype = "dashed", color = "darkblue",linewidth=0.8)+

geom\_ribbon(aes(x=fpr\_f,y=tpr\_f,xmin=0,xmax=1,ymin = 0, ymax = tpr\_f), fill = "lightblue", alpha = 0.3)

ggsave("ROC曲线.tiff",dpi=600, compression = 'lzw')

**###**

**#任务4—3利用EM算法进行参数估计**

**###**

#导入相关库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

library(mixtools)

library(MASS)

#读取目录下mydata.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#取出患者出院状态这一列

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

# 设置混合高斯分布的参数

mu0 <- 0

sigma0 <- 2

mu1 <- 4

sigma1 <- 2

alpha=0.679

# 生成对应状态的符合混合高斯分布的随机数 x

x <- ifelse(Var\_ex\_flag == 0, mvrnorm(sum(Var\_ex\_flag == 0), mu = mu0, Sigma = sigma0),

mvrnorm(sum(Var\_ex\_flag == 1), mu = mu1, Sigma = sigma1))

data\_x <- data.frame(x)

# 绘制 x 的概率密度图

ggplot(data\_x, aes(x = x)) +

geom\_density( color = "darkblue", alpha = 0.7,linewidth=1) +

# labs(title = "Density Plot of x") +

xlab("x") +

ylab("density")

# 初始化估计参数

initial\_params <- list(

"normal" = list(mean = c(-0.5, 4.5), var = c(1.5, 1.5), p = c(0.5, 0.5)),

"mix" = list(p = c(0.5, 0.5))

)

**# 使用EM算法估计参数**

fit <- normalmixEM(x, lambda = initial\_params$mix$p, mu = initial\_params$normal$mean, sigma = sqrt(initial\_params$normal$var))

# 输出估计得到的参数值

estimated\_params <- cbind(fit$lambda, fit$mu, sqrt(fit$sigma^2))

rownames(estimated\_params) <- c("Component 1", "Component 2")

colnames(estimated\_params) <- c("Alpha", "Mean", "Standard Deviation")

print(estimated\_params)

mu0\_es=estimated\_params[1,2]

mu1\_es=estimated\_params[2,2]

sigma0\_es=estimated\_params[1,3]

sigma1\_es=estimated\_params[2,3]

alpha\_est=estimated\_params[1,1]

# 输出真实参数值

true\_params <- cbind(c(0.679, 0.320), c(0, 4), c(2, 2))

rownames(true\_params) <- c("Component 1", "Component 2")

colnames(true\_params) <- c("Alpha", "Mean", "Standard Deviation")

print(true\_params)

####对比ROC曲线

library(cowplot)

# 计算每个观测值来源于N(μ0, σ0)和N(μ1, σ1)的概率密度

f0 <- dnorm(x, mean = mu0, sd = sigma0)

f1 <- dnorm(x, mean = mu1, sd = sigma1)

f0\_es=dnorm(x, mean = mu0\_es,sd=sigma0\_es)

f1\_es=dnorm(x, mean = mu1\_es,sd=sigma1\_es)

# 计算估计y的概率密度

y\_est <- alpha\_est \* dnorm(x, mean = mu0\_es, sd = sigma0\_es) + (1 - alpha\_est) \* dnorm(x, mean = mu1\_es, sd = sigma1\_es)

y\_true <- alpha \* dnorm(x, mean = mu0, sd = sigma0) + (1 - alpha) \* dnorm(x, mean = mu1, sd = sigma1)

y\_es\_plot=ggplot(data.frame(x,y\_est), aes(x = x)) +

geom\_line(aes(x,y\_est),color="darkblue",linewidth =1)+

# labs(title = "Density Plot of x") +

xlab("x") +

ylab("y\_est")

y\_tr\_plot=ggplot(data.frame(x,y\_true), aes(x = x)) +

geom\_line(aes(x,y\_true),color="darkblue",linewidth =1)+

# labs(title = "Density Plot of x") +

xlab("x") +

ylab("y\_true")

cowplot::plot\_grid(y\_tr\_plot,y\_es\_plot,nrow=2)

# 计算来源于1的相对概率f

f <- f1 / (f0 + f1)

f\_es=f1\_es / (f0\_es + f1\_es)

p1=ggplot(data.frame(x,f0,f1,f))+

geom\_line(aes(x,f0),color="darkblue",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(a)", vjust = 1.5, size = 4.5)

p2=ggplot(data.frame(x,f0,f1,f))+

geom\_line(aes(x,f1),color="#0073C2FF",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(b)", vjust = 1.5, size = 4.5)

p3=ggplot(data.frame(x,f0,f1,f))+

geom\_line(aes(x,f),color="darkgreen",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(c)", vjust = 1.5, size = 4.5)

cowplot::plot\_grid(p1,p2,p3,nrow=3)

p4=ggplot(data.frame(x,f0\_es,f1\_es,f\_es))+

geom\_line(aes(x,f0\_es),color="darkblue",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(a)", vjust = 1.5, size = 4.5)

p5=ggplot(data.frame(x,f0\_es,f1\_es,f\_es))+

geom\_line(aes(x,f1\_es),color="#0073C2FF",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(b)", vjust = 1.5, size = 4.5)

p6=ggplot(data.frame(x,f0\_es,f1\_es,f\_es))+

geom\_line(aes(x,f\_es),color="darkgreen",linewidth =1)+

annotate("text", x = -3.5, y = Inf-0.05,

label = "(c)", vjust = 1.5, size = 4.5)

cowplot::plot\_grid(p4,p5,p6,nrow=3)

library(pROC)

# 将相对概率f进行排序作为阈值

threshold\_f = seq(0, 1, by = 0.01)

# 初始化存储各指标的向量

fpr\_f\_es <- rep(0, length(threshold\_f))

tpr\_f\_es <- rep(0, length(threshold\_f))

for (j in 1:length(threshold\_f)) {

# 根据选择的阈值对样本进行分类

predicted\_es <- ifelse(f\_es >= threshold\_f[j], 1, 0)

# 计算混淆矩阵

confusion\_matrix\_f\_es <- table(predicted\_es, Var\_ex\_flag)

if(dim(confusion\_matrix\_f\_es)[1]==2)

{

# 计算真阳性率、假阳性率和F1值

tp <- confusion\_matrix\_f\_es[2, 2]

tn <- confusion\_matrix\_f\_es[1, 1]

fp <- confusion\_matrix\_f\_es[2, 1]

fn <- confusion\_matrix\_f\_es[1, 2]

fpr\_f\_es[j] <- fp / (fp + tn)

tpr\_f\_es[j] <- tp / (tp + fn)

}

else

{

if(predicted\_es[j]==1)

{

fpr\_f\_es[j] <- 1

tpr\_f\_es[j] <- 1

}

else

{

fpr\_f\_es[j] <- 0

tpr\_f\_es[j] <- 0

}

}

}

# 使用roc函数创建ROC曲线对象

roc\_obj\_es <- roc(Var\_ex\_flag, f\_es)

# 计算AUC值

auc\_value\_es=auc(roc\_obj\_es)

print(paste("AUC\_ES:", auc\_value\_es))

ggplot(data.frame(fpr\_f\_es,tpr\_f\_es))+

geom\_line(aes(fpr\_f\_es,tpr\_f\_es),color="#7773C2FF",linewidth =1)+

xlab("False Positive Rate(FPR)") +

ylab("True Positive Rate(TPR)")+

annotate("text", x = 0.25, y = 0.75,

label = "Auc = 0.9758 ", vjust = 1.5, size = 5)+

# 添加水平参考线

geom\_segment(aes(x = 0, y = 0, xend = 1, yend = 1), linetype = "dashed", color = "darkblue",linewidth=0.8)+

geom\_ribbon(aes(x=fpr\_f\_es,y=tpr\_f\_es,xmin=0,xmax=1,ymin = 0, ymax = tpr\_f\_es), fill = "lightblue", alpha = 0.3)

###

**#使用 x 与 y 的值，在将数据分为训练样本集和测试样本集的前提下**

**#通过线性判别对训练样本集进行模型拟合，后用于测试样本进行预测，给出训练样本#集、测试样本集的 ROC 曲线**

**###**

#导入库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

library(mixtools)

library(MASS)

#读取目录下mydata.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#取出患者出院状态这一列

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

# 设置混合高斯分布的参数

mu0 <- 0

sigma0 <- 2

mu1 <- 4

sigma1 <- 2

# 生成对应状态的符合混合高斯分布的随机数 x

x <- ifelse(Var\_ex\_flag == 0, mvrnorm(sum(Var\_ex\_flag == 0), mu = mu0, Sigma = sigma0),

mvrnorm(sum(Var\_ex\_flag == 1), mu = mu1, Sigma = sigma1))

#创建数据框

data\_new= tibble(Var\_ex\_flag, x)

# 分割数据集

#以70%样本作为训练集求解

train\_prop <- 0.7 # 训练样本集比例

train\_indices <- sample(1:nrow(data\_new), round(train\_prop \* nrow(data\_new)))

train\_data <- data\_new[train\_indices, ]

test\_data <- data\_new[-train\_indices, ]

#线性判别函数参数求解

pi\_1=sum(train\_data$Var\_ex\_flag==1)/length(train\_data$Var\_ex\_flag)

pi\_0=sum(train\_data$Var\_ex\_flag==0)/length(train\_data$Var\_ex\_flag)

#假设各总体方差相同

#求muk\_1

x1=0

for (i in 1:length(train\_data$Var\_ex\_flag)){

if(train\_data$Var\_ex\_flag[i]==1){

x1=x1+train\_data$x[i]

}

}

muk\_1=x1 / sum(train\_data$Var\_ex\_flag==1)

#求muk\_0

x0=0

for (i in 1:length(train\_data$Var\_ex\_flag)){

if(train\_data$Var\_ex\_flag[i]==0){

x0=x0+train\_data$x[i]

}

}

muk\_0=x0 / sum(train\_data$Var\_ex\_flag==1)

#求sigma\_es

i=0

index\_0=which(train\_data$Var\_ex\_flag==0)

index\_1=which(train\_data$Var\_ex\_flag==1)

sum0=0

sum1=0

for(i in 0:1 ){

if(i==0)

{

for(j in 1:length(index\_0)){

sum0 = sum0+(train\_data$x[index\_0[j]]-muk\_0)\*(train\_data$x[index\_0[j]]-muk\_0)

}

}

else{

for(j in 1:length(index\_1)){

sum1 = sum1+(train\_data$x[index\_1[j]]-muk\_1)\*(train\_data$x[index\_1[j]]-muk\_1)

}

}

}

sigma\_es=(sum0+sum1)/(length(train\_data$Var\_ex\_flag)-2)

#线性判别函数

#log(pi)

train\_flag=train\_data$Var\_ex\_flag

tr\_desicision0 = rep(0,length(train\_data$Var\_ex\_flag))

tr\_desicision1 = rep(0,length(train\_data$Var\_ex\_flag))

#训练集中状态为1的相对概率

f\_train = rep(0,length(train\_data$Var\_ex\_flag))

train\_result=rep(0,length(train\_data$Var\_ex\_flag))

for(i in 1:length(train\_data$Var\_ex\_flag)){

tr\_desicision0[i]=log(pi\_0)+train\_data$x[i]\*muk\_0/(sigma\_es\*sigma\_es)-muk\_0\*muk\_0/(2\*sigma\_es\*sigma\_es)

tr\_desicision1[i]=log(pi\_1)+train\_data$x[i]\*muk\_1/(sigma\_es\*sigma\_es)-muk\_1\*muk\_1/(2\*sigma\_es\*sigma\_es)

f\_train[i] = exp(tr\_desicision1[i])/(exp(tr\_desicision1[i])+exp(tr\_desicision0[i]))

if(tr\_desicision0[i]>tr\_desicision1[i]){

train\_result[i] = 0

}

else{

train\_result[i] = 1

}

}

test\_flag=test\_data$Var\_ex\_flag

te\_desicision0 = rep(0,length(test\_data$Var\_ex\_flag))

te\_desicision1 = rep(0,length(test\_data$Var\_ex\_flag))

test\_result=rep(0,length(test\_data$Var\_ex\_flag))

#测试集中状态为1的相对概率

f\_test=rep(0,length(test\_data$Var\_ex\_flag))

for(i in 1:length(test\_data$Var\_ex\_flag)){

te\_desicision0[i]=log(pi\_0)+test\_data$x[i]\*muk\_0/(sigma\_es\*sigma\_es)-muk\_0\*muk\_0/(2\*sigma\_es\*sigma\_es)

te\_desicision1[i]=log(pi\_1)+test\_data$x[i]\*muk\_1/(sigma\_es\*sigma\_es)-muk\_1\*muk\_1/(2\*sigma\_es\*sigma\_es)

f\_test[i] = exp(te\_desicision1[i])/(exp(te\_desicision1[i])+exp(te\_desicision0[i]))

if(te\_desicision0[i]>te\_desicision1[i]){

test\_result[i] = 0

}

else{

test\_result[i] = 1

}

}

# 概率阈值

threshold\_f = seq(0, 1, by = 0.01)

# 初始化存储各指标的向量

fpr\_f <- rep(0, length(threshold\_f))

tpr\_f <- rep(0, length(threshold\_f))

for (i in 1:length(threshold\_f)) {

# 根据选择的阈值对样本进行分类

predicted <- ifelse(f\_test >= threshold\_f[i], 1, 0)

# 计算混淆矩阵

confusion\_matrix\_f <- table(predicted, test\_flag)

if(dim(confusion\_matrix\_f)[1]==2)

{

# 计算真阳性率、假阳性率和F1值

tp <- confusion\_matrix\_f[2, 2]

tn <- confusion\_matrix\_f[1, 1]

fp <- confusion\_matrix\_f[2, 1]

fn <- confusion\_matrix\_f[1, 2]

fpr\_f[i] <- fp / (fp + tn)

tpr\_f[i] <- tp / (tp + fn)

}

else

{

if(predicted[i]==1)

{

fpr\_f[i] <- 1

tpr\_f[i] <- 1

}

else

{

fpr\_f[i] <- 0

tpr\_f[i] <- 0

}

}

}

library(pROC)

#Auc值计算

#auc\_value <- auc(roc(train\_flag,f\_train))

##测试集

#0.9788 0.9776 0.9729 0.9742 0.9721 0.979 0.9776 0.9804 0.9805 0.9773 /10 =0.977

#测试集ROC曲线

p1=ggplot(data.frame(fpr\_f,tpr\_f))+

geom\_line(aes(fpr\_f,tpr\_f),color="#0043C2FF",linewidth =1)+

xlab("False Positive Rate(FPR)") +

ylab("True Positive Rate(TPR)")+

annotate("text", x = 0.25, y = 0.75,

label = "Test Auc = 0.977 ", vjust = 1.5, size = 5)+

annotate("text", x = 0.75, y = 0.25,

label = "(b) ", vjust = 1.5, size = 5)+

# 添加水平参考线

geom\_segment(aes(x = 0, y = 0, xend = 1, yend = 1), linetype = "dashed", color = "darkblue",linewidth=0.8)+

geom\_ribbon(aes(x=fpr\_f,y=tpr\_f,xmin=0,xmax=1,ymin = 0, ymax = tpr\_f), fill = "lightblue", alpha = 0.3)

for (i in 1:length(threshold\_f)) {

# 根据选择的阈值对样本进行分类

predicted <- ifelse(f\_train >= threshold\_f[i], 1, 0)

# 计算混淆矩阵

confusion\_matrix\_f <- table(predicted, train\_flag)

if(dim(confusion\_matrix\_f)[1]==2)

{

# 计算真阳性率、假阳性率和F1值

tp <- confusion\_matrix\_f[2, 2]

tn <- confusion\_matrix\_f[1, 1]

fp <- confusion\_matrix\_f[2, 1]

fn <- confusion\_matrix\_f[1, 2]

fpr\_f[i] <- fp / (fp + tn)

tpr\_f[i] <- tp / (tp + fn)

}

else

{

if(predicted[i]==1)

{

fpr\_f[i] <- 1

tpr\_f[i] <- 1

}

else

{

fpr\_f[i] <- 0

tpr\_f[i] <- 0

}

}

}

#0.9716 0.9769 0.977 0.9743 0.9773 0.9812 0.979 0.9748 0.9831 0.9782 /10 = 0.9773

#训练集ROC曲线

p2=ggplot(data.frame(fpr\_f,tpr\_f))+

geom\_line(aes(fpr\_f,tpr\_f),color="#00C2FF",linewidth =1)+

xlab("False Positive Rate(FPR)") +

ylab("True Positive Rate(TPR)")+

annotate("text", x = 0.25, y = 0.75,

label = "Train Auc = 0.9773 ", vjust = 1.5, size = 5)+

annotate("text", x = 0.75, y = 0.25,

label = "(a) ", vjust = 1.5, size = 5)+

# 添加水平参考线

geom\_segment(aes(x = 0, y = 0, xend = 1, yend = 1), linetype = "dashed", color = "darkblue",linewidth=0.8)+

geom\_ribbon(aes(x=fpr\_f,y=tpr\_f,xmin=0,xmax=1,ymin = 0, ymax = tpr\_f), fill = "lightblue", alpha = 0.3)

cowplot::plot\_grid(p2,p1,nrow=2)

ggsave("LDA估计分布ROC曲线.tiff",dpi=600, compression = 'lzw')

**###Fisher判别核心代码**

###导入库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

library(mixtools)

library(MASS)

#读取目录下mydata.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#取出患者出院状态这一列

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

# 设置混合高斯分布的参数

mu0 <- 0

sigma0 <- 2

mu1 <- 4

sigma1 <- 2

# 生成对应状态的符合混合高斯分布的随机数 x

x <- ifelse(Var\_ex\_flag == 0, mvrnorm(sum(Var\_ex\_flag == 0), mu = mu0, Sigma = sigma0),

mvrnorm(sum(Var\_ex\_flag == 1), mu = mu1, Sigma = sigma1))

#创建数据框

data\_new= tibble(Var\_ex\_flag, x)

# 分割数据集

train\_prop <- 0.7 # 训练样本集比例

train\_indices <- sample(1:nrow(data\_new), round(train\_prop \* nrow(data\_new)))

train\_data <- data\_new[train\_indices, ]

test\_data <- data\_new[-train\_indices, ]

# 线性判别分析

lda\_model <- lda(Var\_ex\_flag ~ x, data = train\_data)

# 显示模型结果

lda\_model

# 预测

train\_pred <- predict(lda\_model, newdata = train\_data)

train\_prob <- train\_pred$posterior[, 2] # 获取属于类别1的概率作为预测概率

test\_pred <- predict(lda\_model, newdata = test\_data)

test\_prob <- test\_pred$posterior[, 2] # 获取属于类别1的概率作为预测概率

**###二次判别核心代码**

###导入库

###导入库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

library(mixtools)

library(MASS)

#读取目录下mydata.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#取出患者出院状态这一列

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

# 设置混合高斯分布的参数

mu0 <- 0

sigma0 <- 2

mu1 <- 4

sigma1 <- 2

# 生成对应状态的符合混合高斯分布的随机数 x

x <- ifelse(Var\_ex\_flag == 0, mvrnorm(sum(Var\_ex\_flag == 0), mu = mu0, Sigma = sigma0),

mvrnorm(sum(Var\_ex\_flag == 1), mu = mu1, Sigma = sigma1))

#创建数据框

data\_new= tibble(Var\_ex\_flag, x)

# 分割数据集

train\_prop <- 0.7 # 训练样本集比例

train\_indices <- sample(1:nrow(data\_new), round(train\_prop \* nrow(data\_new)))

train\_data <- data\_new[train\_indices, ]

test\_data <- data\_new[-train\_indices, ]

# 线性判别分析

qda\_model <- qda(Var\_ex\_flag ~ x, data = train\_data)

# 显示模型结果

qda\_model

# 预测

train\_pred <- predict(qda\_model, newdata = train\_data)

train\_prob <- train\_pred$posterior[, 2] # 获取属于类别1的概率作为预测概率

test\_pred <- predict(qda\_model, newdata = test\_data)

test\_prob <- test\_pred$posterior[, 2] # 获取属于类别1的概率作为预测概率

**###Logistic判别核心代码**

###导入库

library(openxlsx)

library(ggplot2)

library(patchwork)

library(tidyverse)

library(mixtools)

library(MASS)

#读取目录下mydata.csv表格中的数据,存储在mydata中

mydata\_mean=read.xlsx( "mydata\_mean.xlsx")

#取出患者出院状态这一列

Var\_ex\_flag= mydata\_mean$hospital\_expire\_flag

# 设置混合高斯分布的参数

mu0 <- 0

sigma0 <- 2

mu1 <- 4

sigma1 <- 2

# 生成对应状态的符合混合高斯分布的随机数 x

x <- ifelse(Var\_ex\_flag == 0, mvrnorm(sum(Var\_ex\_flag == 0), mu = mu0, Sigma = sigma0),

mvrnorm(sum(Var\_ex\_flag == 1), mu = mu1, Sigma = sigma1))

#创建数据框

data\_new= tibble(Var\_ex\_flag, x)

# 将数据分为训练集和测试集

# 分割数据集

train\_prop <- 0.7 # 训练样本集比例

train\_indices <- sample(1:nrow(data\_new), round(train\_prop \* nrow(data\_new)))

train <- data\_new[train\_indices, ]

test <- data\_new[-train\_indices, ]

# 使用 Logistic 回归模型进行训练

logistic\_model <- glm(Var\_ex\_flag ~ x, data = train, family = "binomial")

# 在训练集上进行预测

train$pred <- predict(logistic\_model, newdata = train, type = "response")

# 在测试集上进行预测

test$pred <- predict(logistic\_model, newdata = test, type = "response")

**问题5代码**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from matplotlib import rcParams

import statsmodels.api as sm

from sklearn.preprocessing import StandardScaler

plt.rcParams['font.family'] = 'SimHei'

# plt.rcParams['font.family'] = 'DejaVu Sans'

# 读取CSV文件

data = pd.read\_excel('mydata\_new.xlsx', sheet\_name='Sheet1', engine='openpyxl')

# 提取首日检验数据和出院状态共125个变量

columns\_of\_interest = data.columns[13:]

subset\_data = data[columns\_of\_interest]

# 创建标准化处理器

scaler = StandardScaler()

# 对数据进行标准化

scaled\_data = scaler.fit\_transform(subset\_data)

# 将标准化后的数据转换为DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=columns\_of\_interest)

# 计算相关系数矩阵

correlation\_matrix = scaled\_df.corrwith(data['hospital\_expire\_flag'])

# 获取与出院状态相关性最高的前10个变量

top\_10\_correlated = correlation\_matrix.abs().sort\_values(ascending=False).head(10)

# 输出与出院状态相关性最高的前10个变量

print("与出院状态相关性最高的前10个变量：")

print(top\_10\_correlated)

# 提取与出院状态相关性最高的前10个变量以及出院状态的数据

columns\_of\_interest\_10 = top\_10\_correlated.index.tolist()

subset\_data\_11 = scaled\_df[columns\_of\_interest\_10]

hospital=data['hospital\_expire\_flag']

merged\_data = pd.concat([subset\_data\_11, hospital], axis=1)

# 计算11个变量之间的相关系数矩阵

correlation\_matrix\_11 = merged\_data.corr()

# 绘制相关性图

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix\_11, annot=True, cmap="coolwarm", square=True)

plt.title("与出院状态相关性最高的前10个变量的相关性图")

plt.show()

**问题6代码**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib import rcParams

import statsmodels.api as sm

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

rcParams['font.family'] = 'SimHei'

import numpy as np

from sklearn.metrics import recall\_score, precision\_score, f1\_score

# 读取CSV文件

data = pd.read\_excel('mydata\_new.xlsx',sheet\_name='Sheet1', engine='openpyxl')

# 提取首日检验数据和出院状态共125个变量

columns\_of\_interest = data.columns[13:]

subset\_data = data[columns\_of\_interest]

# 创建标准化处理器

scaler = StandardScaler()

# 对数据进行标准化

scaled\_data = scaler.fit\_transform(subset\_data)

# 将标准化后的数据转换为DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=columns\_of\_interest)

y=data['hospital\_expire\_flag']

columns\_of\_interest\_11=['aniongap\_min','aniongap\_max','spo2\_min','sbp\_min','ptt\_max','mbp\_min',

'bicarbonate\_min','spo2\_mean','temperature\_min','temperature\_mean']

X\_10=scaled\_df[['aniongap\_min','aniongap\_max','spo2\_min','sbp\_min','ptt\_max','mbp\_min',

'bicarbonate\_min','spo2\_mean','temperature\_min','temperature\_mean']]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_10, y, test\_size=0.2, random\_state=42)

accuracy\_final=[]

recall\_scores=[]

precision\_scores=[]

f1\_scores=[]

for variable in columns\_of\_interest\_11[0:10]:

X = sm.add\_constant(X\_train[variable])

logit\_model = sm.Logit(y\_train, X)

result = logit\_model.fit\_regularized()

print('{}回归结果参数：'.format(variable), result.summary())

# 添加常数列

X\_test\_1 = sm.add\_constant(X\_test[variable])

# 在测试集上进行预测

y\_pred\_prob = result.predict(X\_test\_1)

# 将概率转化为类别（0或1）

y\_pred\_class = np.where(y\_pred\_prob >= 0.5, 1, 0)

# 计算准确率

accuracy = (y\_test == y\_pred\_class).mean()

print('{}的预测准确率：'.format(variable), accuracy)

accuracy\_final.append(accuracy)

# 计算召回率

recall = recall\_score(y\_test, y\_pred\_class)

print('{}的召回率：'.format(variable), recall)

recall\_scores.append(recall)

# 计算准确率

precision = precision\_score(y\_test, y\_pred\_class)

print('{}的精确率：'.format(variable), precision)

precision\_scores.append(precision)

# 计算F1分数

f1 = f1\_score(y\_test, y\_pred\_class)

print('{}的F1分数：'.format(variable), f1)

f1\_scores.append(f1)

print("每个变量单独回归的预测准确率", accuracy\_final)

print("每个变量单独回归的召回率", recall\_scores)

print("每个变量单独回归的精确率", precision\_scores)

print("每个变量单独回归的F1分数", f1\_scores)

# 使用所有变量拟合Logit回归模型并记录结果

X = sm.add\_constant(X\_train)

logit\_model = sm.Logit(y\_train, X)

result\_all = logit\_model.fit\_regularized()

X\_test = sm.add\_constant(X\_test)

# 在测试集上进行预测

y\_pred\_prob = result\_all.predict(X\_test)

# 将概率转化为类别（0或1）

y\_pred\_class = np.where(y\_pred\_prob >= 0.5, 1, 0)

# 计算准确率

accuracy = (y\_test == y\_pred\_class).mean()

print('10个变量一起回归预测准确率：', accuracy)

recall = recall\_score(y\_test, y\_pred\_class)

print('10个变量一起回归预测的召回率：', recall)

# 计算准确率

precision = precision\_score(y\_test, y\_pred\_class)

print('10个变量一起回归预测的精确率：', precision)

# 计算F1分数

f1 = f1\_score(y\_test, y\_pred\_class)

print('10个变量一起回归预测的F1分数：', f1)

print('10个变量回归结果参数：', result\_all.summary())

import pandas as pd

import statsmodels.api as sm

from openpyxl import Workbook

# ... 这里是你的数据处理和回归模型代码 ...

# 创建一个空的列表来存储每个回归结果的表格

result\_tables = []

for variable in columns\_of\_interest\_11[0:10]:

X = sm.add\_constant(X\_train[variable])

logit\_model = sm.Logit(y\_train, X)

result = logit\_model.fit\_regularized()

# 获取回归结果的表格数据

result\_table = result.summary2().tables[1]

result\_table.reset\_index(inplace=True)

result\_table['Variable'] = variable

# 将结果表格添加到列表中

result\_tables.append(result\_table)

# 创建一个Excel工作簿

workbook = Workbook()

excel\_writer = pd.ExcelWriter('regression\_results.xlsx', engine='openpyxl')

excel\_writer.book = workbook

# 将每个回归结果表格添加到Excel文件的不同工作表

for i, result\_table in enumerate(result\_tables):

result\_table.to\_excel(excel\_writer, sheet\_name=f'Result\_{i + 1}', index=False)

result\_table1 = result\_all.summary2().tables[1]

result\_table1.reset\_index(inplace=True)

result\_table1.to\_excel(excel\_writer, sheet\_name=f'Result\_{13}', index=False)

# 保存Excel文件

excel\_writer.save()

**问题7代码**

**1、PCA代码**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib import rcParams

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

import statsmodels.api as sm

import numpy as np

rcParams['font.family'] = 'SimHei'

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import recall\_score, precision\_score, f1\_score

# 读取CSV文件

data = pd.read\_excel('mydata\_new.xlsx',sheet\_name='Sheet1', engine='openpyxl')

# 提取首日检验数据和出院状态共125个变量

columns\_of\_interest = data.columns[13:]

subset\_data = data[columns\_of\_interest]

# 创建标准化处理器

scaler = StandardScaler()

# 对数据进行标准化

scaled\_data = scaler.fit\_transform(subset\_data)

# 将标准化后的数据转换为DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=columns\_of\_interest)

accuracy\_final=0

k=0

pca = PCA(n\_components=40) # 保留i个主成分

pca\_data = pca.fit\_transform(scaled\_df)

y = data['hospital\_expire\_flag']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(pca\_data, y, test\_size=0.2, random\_state=42)

X = sm.add\_constant(X\_train)

y = y\_train

model = sm.Logit(y, X)

result = model.fit()

# 添加常数列

X\_test = sm.add\_constant(X\_test)

# 在测试集上进行预测

y\_pred\_prob = result.predict(X\_test)

# 将概率转化为类别（0或1）

y\_pred\_class = np.where(y\_pred\_prob >= 0.5, 1, 0)

# 计算准确率

accuracy = (y\_test == y\_pred\_class).mean()

print('PCA 预测准确率：', accuracy)

recall = recall\_score(y\_test, y\_pred\_class)

print('PCA 预测的召回率：', recall)

# 计算准确率

precision = precision\_score(y\_test, y\_pred\_class)

print('PCA 预测的精确率：', precision)

# 计算F1分数

f1 = f1\_score(y\_test, y\_pred\_class)

print('PCA 预测的F1分数：', f1)

# 可视化

from sklearn.metrics import roc\_curve, roc\_auc\_score, precision\_recall\_curve, average\_precision\_score

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# 计算混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_class)

# 创建热图

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('PCA Confusion Matrix')

plt.show()

# 计算ROC曲线

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

# 计算AUC值

roc\_auc = roc\_auc\_score(y\_test,y\_pred\_prob)

# 绘制ROC曲线

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title(' PCA Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.show()

# 计算精确-召回曲线

precision, recall, \_ = precision\_recall\_curve(y\_test, y\_pred\_prob)

# 计算平均精确度

average\_precision = average\_precision\_score(y\_test, y\_pred\_prob)

# 绘制精确-召回曲线

plt.figure(figsize=(8, 6))

plt.step(recall, precision, color='b', where='post', lw=2, label=f'Precision-Recall curve (Avg. Precision = {average\_precision:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('PCA Precision-Recall Curve')

plt.legend(loc='lower left')

plt.show()

**2、Ridge代码**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib import rcParams

import statsmodels.api as sm

from sklearn.decomposition import PCA

from sklearn.linear\_model import Ridge, Lasso

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

rcParams['font.family'] = 'SimHei'

from sklearn.metrics import mean\_squared\_error

import numpy as np

from sklearn.metrics import recall\_score, precision\_score, f1\_score

# 读取CSV文件

data = pd.read\_excel('mydata\_new.xlsx',sheet\_name='Sheet1', engine='openpyxl')

# 提取首日检验数据和出院状态共125个变量

columns\_of\_interest = data.columns[13:]

subset\_data = data[columns\_of\_interest]

# 创建标准化处理器

scaler = StandardScaler()

# 对数据进行标准化

scaled\_data = scaler.fit\_transform(subset\_data)

# 将标准化后的数据转换为DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=columns\_of\_interest)

y=data['hospital\_expire\_flag']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(scaled\_df, y, test\_size=0.2, random\_state=42)

# 使用Ridge回归

ridge = Ridge()

ridge.fit(X\_train, y\_train)

print(len(columns\_of\_interest))

print(len(ridge.coef\_))

ridge\_coefs = pd.DataFrame({'变量': X\_train, '系数': ridge.coef\_})

print('Ridge回归系数：')

print(ridge\_coefs)

y\_pred = ridge.predict(X\_test)

print('预测结果\_Ridge：', y\_pred)

y\_pred\_class = np.where(y\_pred>= 0.5, 1, 0)

accuracy = (y\_test == y\_pred\_class).mean()

print('ridge 预测准确率：', accuracy)

recall = recall\_score(y\_test, y\_pred\_class)

print('ridge 预测的召回率：', recall)

# 计算准确率

precision = precision\_score(y\_test, y\_pred\_class)

print('ridge 预测的精确率：', precision)

# 计算F1分数

f1 = f1\_score(y\_test, y\_pred\_class)

print('ridge 预测的F1分数：', f1)

# 可视化

from sklearn.metrics import roc\_curve, roc\_auc\_score, precision\_recall\_curve, average\_precision\_score

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# 计算混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_class)

# 创建热图

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('ridge Confusion Matrix')

plt.show()

# 计算ROC曲线

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# 计算AUC值

roc\_auc = roc\_auc\_score(y\_test,y\_pred)

# 绘制ROC曲线

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title(' ridge Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.show()

# 计算精确-召回曲线

precision, recall, \_ = precision\_recall\_curve(y\_test, y\_pred)

# 计算平均精确度

average\_precision = average\_precision\_score(y\_test, y\_pred)

# 绘制精确-召回曲线

plt.figure(figsize=(8, 6))

plt.step(recall, precision, color='b', where='post', lw=2, label=f'Precision-Recall curve (Avg. Precision = {average\_precision:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('ridge Precision-Recall Curve')

plt.legend(loc='lower left')

plt.show()

# # 使用PCA进行特征降维

# scaler = StandardScaler()

# scaled\_data = scaler.fit\_transform(scaled\_df)

# pca = PCA()

# pca.fit(scaled\_data)

# explained\_variance\_ratio = pca.explained\_variance\_ratio\_

# plt.plot(range(1, len(explained\_variance\_ratio) + 1), explained\_variance\_ratio, marker='o')

# plt.xlabel('主成分数量')

# plt.ylabel('解释方差比例')

# plt.show()

**问题8、问题9代码**

**1、随机森林代码**

from matplotlib import rcParams

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV

import pandas as pd

import numpy as np

from sklearn.metrics import recall\_score, precision\_score, f1\_score

rcParams['font.family'] = 'SimHei'

# 读取CSV文件

data = pd.read\_excel('mydata\_new.xlsx', sheet\_name='Sheet1', engine='openpyxl')

# 提取首日检验数据和出院状态共125个变量

columns\_of\_interest = data.columns[13:]

subset\_data = data[columns\_of\_interest]

# 创建标准化处理器

scaler = StandardScaler()

# 对数据进行标准化

scaled\_data = scaler.fit\_transform(subset\_data)

# 将标准化后的数据转换为DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=columns\_of\_interest)

pca = PCA(n\_components=40) # 保留5个主成分

pca\_data = pca.fit\_transform(scaled\_df)

y = data['hospital\_expire\_flag']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(pca\_data, y, test\_size=0.2, random\_state=42)

# 定义参数网格

param\_grid = {

'n\_estimators': [100, 200, 300, 400],

'max\_depth': [None,5, 10, 20, 30],

'min\_samples\_split': [2, 5, 10, 15],

'min\_samples\_leaf': [1, 2, 4],

}

# 创建随机森林分类器

rf\_classifier = RandomForestClassifier(random\_state=42)

# 创建GridSearchCV对象

grid\_search = GridSearchCV(estimator=rf\_classifier, param\_grid=param\_grid, cv=5, scoring='accuracy')

# 在训练集上执行网格搜索

grid\_search.fit(X\_train, y\_train)

# 输出最佳参数组合

best\_params = grid\_search.best\_params\_

print("最佳参数组合：", best\_params)

# 获取最佳模型

best\_rf\_classifier = grid\_search.best\_estimator\_

# 在测试集上进行预测

y\_pred = best\_rf\_classifier.predict(X\_test)

y\_pred\_class = np.where(y\_pred >= 0.5, 1, 0)

# 计算准确率

accuracy = (y\_test == y\_pred\_class).mean()

print('Random Forest GridSearch 预测准确率：', accuracy)

recall = recall\_score(y\_test, y\_pred\_class)

print('Random Forest GridSearch 预测的召回率：', recall)

# 计算准确率

precision = precision\_score(y\_test, y\_pred\_class)

print('Random Forest GridSearch 预测的精确率：', precision)

# 计算F1分数

f1 = f1\_score(y\_test, y\_pred\_class)

print('Random Forest GridSearch 预测的F1分数：', f1)

# 最佳参数组合： {'max\_depth': 20, 'min\_samples\_leaf': 4, 'min\_samples\_split': 15, 'n\_estimators': 100}

# Random Forest GridSearch 预测准确率： 0.7330677290836654

# Random Forest GridSearch 预测的召回率： 0.3263888888888889

# Random Forest GridSearch 预测的精确率： 0.7621621621621621

# Random Forest GridSearch 预测的F1分数： 0.4570502431118315

# 可视化

from sklearn.metrics import roc\_curve, roc\_auc\_score, precision\_recall\_curve, average\_precision\_score

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# 计算混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_class)

# 创建热图

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Random Forest GridSearch Confusion Matrix')

plt.show()

# 计算ROC曲线

fpr, tpr, thresholds = roc\_curve(y\_test, best\_rf\_classifier.predict\_proba(X\_test)[:, 1])

# 计算AUC值

roc\_auc = roc\_auc\_score(y\_test, best\_rf\_classifier.predict\_proba(X\_test)[:, 1])

# 绘制ROC曲线

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Random Forest GridSearch Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.show()

# 计算精确-召回曲线

precision, recall, \_ = precision\_recall\_curve(y\_test, best\_rf\_classifier.predict\_proba(X\_test)[:, 1])

# 计算平均精确度

average\_precision = average\_precision\_score(y\_test, best\_rf\_classifier.predict\_proba(X\_test)[:, 1])

# 绘制精确-召回曲线

plt.figure(figsize=(8, 6))

plt.step(recall, precision, color='b', where='post', lw=2, label=f'Precision-Recall curve (Avg. Precision = {average\_precision:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('Random Forest GridSearch Precision-Recall Curve')

plt.legend(loc='lower left')

plt.show()

**2、XGBoost代码**

import xgboost as xgb

from sklearn.model\_selection import GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import pandas as pd

import numpy as np

from sklearn.metrics import recall\_score, precision\_score, f1\_score

# 读取CSV文件

data = pd.read\_excel('mydata\_new.xlsx', sheet\_name='Sheet1', engine='openpyxl')

# 提取首日检验数据和出院状态共125个变量

columns\_of\_interest = data.columns[13:]

subset\_data = data[columns\_of\_interest]

# 创建标准化处理器

scaler = StandardScaler()

# 对数据进行标准化

scaled\_data = scaler.fit\_transform(subset\_data)

# 将标准化后的数据转换为DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=columns\_of\_interest)

# PCA降维

from sklearn.decomposition import PCA

pca = PCA(n\_components=40) # 保留5个主成分

pca\_data = pca.fit\_transform(scaled\_df)

y = data['hospital\_expire\_flag']

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(pca\_data, y, test\_size=0.2, random\_state=42)

# 创建XGBoost分类器

xgb\_model = xgb.XGBClassifier()

# 定义要搜索的超参数组合

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [3, 4, 5],

'learning\_rate': [0.01, 0.1,0.3],

'subsample': [0.8, 0.9, 1.0],

'colsample\_bytree': [0.8, 0.9, 1.0]

}

# 创建GridSearchCV对象，用于超参数搜索

grid\_search = GridSearchCV(estimator=xgb\_model, param\_grid=param\_grid, cv=3, n\_jobs=1, scoring='accuracy')

# 执行超参数搜索

grid\_search.fit(X\_train, y\_train)

# 输出最佳超参数组合

print("Best Parameters:", grid\_search.best\_params\_)

# 使用最佳超参数的模型进行训练和预测

best\_xgb\_model = grid\_search.best\_estimator\_

y\_pred = best\_xgb\_model.predict(X\_test)

y\_pred\_class = np.where(y\_pred >= 0.5, 1, 0)

# 计算准确率

accuracy = (y\_test == y\_pred\_class).mean()

print('XGboost GridSearch 预测准确率：', accuracy)

# Best Parameters: {'colsample\_bytree': 0.8, 'learning\_rate': 0.1, 'max\_depth': 4, 'n\_estimators': 100, 'subsample': 0.8}

# XGboost GridSearch 预测准确率： 0.7338645418326694

# 可视化

from sklearn.metrics import roc\_curve, roc\_auc\_score, precision\_recall\_curve, average\_precision\_score

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# 计算混淆矩阵

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_class)

# 创建热图

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('XGboost GridSearch Confusion Matrix')

plt.show()

# 计算ROC曲线

fpr, tpr, thresholds = roc\_curve(y\_test, best\_xgb\_model.predict\_proba(X\_test)[:, 1])

# 计算AUC值

roc\_auc = roc\_auc\_score(y\_test, best\_xgb\_model.predict\_proba(X\_test)[:, 1])

# 绘制ROC曲线

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('XGboost GridSearch Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.show()

# 计算精确-召回曲线

precision, recall, \_ = precision\_recall\_curve(y\_test, best\_xgb\_model.predict\_proba(X\_test)[:, 1])

# 计算平均精确度

average\_precision = average\_precision\_score(y\_test, best\_xgb\_model.predict\_proba(X\_test)[:, 1])

# 绘制精确-召回曲线

plt.figure(figsize=(8, 6))

plt.step(recall, precision, color='b', where='post', lw=2, label=f'Precision-Recall curve (Avg. Precision = {average\_precision:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('XGboost GridSearch Precision-Recall Curve')

plt.legend(loc='lower left')

plt.show()

**问题10代码**

from matplotlib import rcParams

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

from sklearn.metrics import recall\_score, precision\_score, f1\_score

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from xgboost import XGBClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import roc\_curve, roc\_auc\_score, precision\_recall\_curve, average\_precision\_score

import matplotlib.pyplot as plt

import seaborn as sns

rcParams['font.family'] = 'SimHei'

# 读取CSV文件

data = pd.read\_excel('mydata\_new.xlsx', sheet\_name='Sheet1', engine='openpyxl')

# 提取首日检验数据和出院状态共125个变量

columns\_of\_interest = data.columns[1:]

subset\_data = data[columns\_of\_interest]

# 确定0-1二进制变量的列

binary\_columns = ['gender', 'ethnicity', 'first\_hosp\_stay', 'first\_icu\_stay', 'long\_title', 'hospstay\_seq']

binary\_data=data[binary\_columns]

# 确定非二进制变量的列

non\_binary\_columns = [col for col in columns\_of\_interest if col not in binary\_columns]

# 对非二进制变量进行标准化

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(subset\_data[non\_binary\_columns])

# 将标准化后的数据转换为DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=non\_binary\_columns)

merged\_data = pd.concat([binary\_data, scaled\_df], axis=1)

encoded\_data = pd.get\_dummies(merged\_data, columns=binary\_columns)

pca = PCA(n\_components=46) # 保留5个主成分

pca\_data = pca.fit\_transform(encoded\_data)

y = data['hospital\_expire\_flag']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(pca\_data, y, test\_size=0.2, random\_state=42)

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

from xgboost import XGBClassifier

from sklearn.metrics import roc\_curve, roc\_auc\_score, precision\_recall\_curve, average\_precision\_score

import matplotlib.pyplot as plt

# 随机森林参数网格

rf\_param\_grid = {

'n\_estimators': [100, 200, 300, 400],

'max\_depth': [None,5, 10, 20, 30],

'min\_samples\_split': [2, 5, 10, 15],

'min\_samples\_leaf': [1, 2, 4],

}

# XGBoost参数网格

xgb\_param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [3, 4, 5],

'learning\_rate': [0.01, 0.1, 0.2],

'subsample': [0.8, 0.9, 1.0],

'colsample\_bytree': [0.8, 0.9, 1.0]

}

# 随机森林

rf = RandomForestClassifier(random\_state=42)

rf\_grid\_search = GridSearchCV(estimator=rf, param\_grid=rf\_param\_grid, cv=3, scoring='accuracy')

rf\_grid\_search.fit(X\_train, y\_train)

best\_rf\_model = rf\_grid\_search.best\_estimator\_

# XGBoost

xgb = XGBClassifier(random\_state=42)

xgb\_grid\_search = GridSearchCV(estimator=xgb, param\_grid=xgb\_param\_grid, cv=3, scoring='accuracy')

xgb\_grid\_search.fit(X\_train, y\_train)

best\_xgb\_model = xgb\_grid\_search.best\_estimator\_

# ROC曲线和精确-召回曲线

plt.figure(figsize=(12, 6))

# 随机森林

y\_pred\_rf = best\_rf\_model.predict\_proba(X\_test)[:, 1]

fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test, y\_pred\_rf)

precision\_rf, recall\_rf, \_ = precision\_recall\_curve(y\_test, y\_pred\_rf)

roc\_auc\_rf = roc\_auc\_score(y\_test, y\_pred\_rf)

avg\_precision\_rf = average\_precision\_score(y\_test, y\_pred\_rf)

# XGBoost

y\_pred\_xgb = best\_xgb\_model.predict\_proba(X\_test)[:, 1]

fpr\_xgb, tpr\_xgb, \_ = roc\_curve(y\_test, y\_pred\_xgb)

precision\_xgb, recall\_xgb, \_ = precision\_recall\_curve(y\_test, y\_pred\_xgb)

roc\_auc\_xgb = roc\_auc\_score(y\_test, y\_pred\_xgb)

avg\_precision\_xgb = average\_precision\_score(y\_test, y\_pred\_xgb)

# 绘制ROC曲线

plt.subplot(1, 2, 1)

plt.plot(fpr\_rf, tpr\_rf, label=f'Random Forest (AUC = {roc\_auc\_rf:.2f})')

plt.plot(fpr\_xgb, tpr\_xgb, label=f'XGBoost (AUC = {roc\_auc\_xgb:.2f})')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc="lower right")

# 绘制精确-召回曲线

plt.subplot(1, 2, 2)

plt.plot(recall\_rf, precision\_rf, label=f'Random Forest (AP = {avg\_precision\_rf:.2f})')

plt.plot(recall\_xgb, precision\_xgb, label=f'XGBoost (AP = {avg\_precision\_xgb:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend(loc="lower left")

plt.tight\_layout()

plt.show()