基于 CEEMDAN 和 GRU-随机森林的生命体征监测

摘要

本文研究睡眠阶段生命体征监测问题，首先建立序列聚类分析模型，为了对电子信号进行“转换”和估计分别建立基于 CEEMDAN 的电子信号模态分解模型和 GRU-随机森林预测模型，最后建立电子信号特征分析模型研究疾病导致的电子信号特征变化。

针对问题一，本文建立了**序列聚类分析模型**，分析每个类别的电子信号序列的特征及相应生命体征的特征。首先通过 **ACF 检验**到电子信号序列的变化具有周期性，利用 **CEEDAN 模型**并考虑到部分时刻数据缺失求解得到**周期为 6s**(300 个数据点)。然后计算电子信号序列的熵、偏度、峰度等六个统计特征，再结合 **MCMC-GARCH 模型**进一步分析序列在频域上的特征，进而利用 **K-Means** 进行聚类，并通过**肘部法确定 K 值为 3**. 最后根据聚类结果进行特征分析，并给出每一类别可能所处的睡眠阶段。

针对问题二，本文建立**电子信号模态分解模型**和**生命体征预测模型**，可通过电子信号得知生命体征数据。首先利用 **CEEMDAN** 将电子信号分解成被监测者的心率信号和呼吸信号，利用**快速傅里叶变换**拟合函数关系，建立将电子信号转换成能理解的生命体征数据的数学模型。接着在此基础上，采用 **GRU-随机森林**实现通过电子信号估计生命体征并进行类别划分。求得呼吸频率、心率、体动三项生命体征的预测准确率分别为 **78.76%**、**84.69%**、**80.32%**，与特征感知机、支持向量机、逻辑回归相比效果更优。

针对问题三，本文建立**序列特征分析模型**，针对**阻塞性睡眠呼吸暂停和打鼾**这一类疾病，分析其导致的电子信号特征变化。首先分析两个睡眠呼吸类疾病对三项生命体征的影响，然后基于呼吸信号和生命体征得出 8、28 号被监测者不存在打鼾现象，呼吸较稳定；3、44 号存在打鼾现象；30 号会出现长时间呼吸暂停现象，有较大的患病风险。本文结合之前的分析对 **30 号**进行重点研究，通过**差分序列**和**滑动窗口分析**其电子信号序列特征。最终得到当电子信号出现剧烈变化、数值存在异常下降时会出现打鼾的现象，当出现低标准偏差的连续序列时，电子信号变化较小，可能会出现呼吸暂停的现象。

针对问题四，除企业关注的生命特征，小组认为心率变异性、体温也应受到关注。

本文的亮点有：1. 考虑信号数据特殊性，从时域、统计、频域角度提取特征，提高数据分析准确性；2. 实现时域和频域相互转化，对于疾病诊断和健康监测有很大价值。

关键词：生命体征监测　MCMC-GARCH　CEEMDAN　GRU-随机森林预测

目录

1 问题重述

1.1 问题背景

近年来，随着社会经济的发展，人们对健康的关注度越来越高，国家也对以来医疗行业发展高度重视。而对生命体征的监测是健康维护与医疗救治的重要前提。生命特征主要是由呼吸频率、心率、血压、脉搏等生理参数反映，其中呼吸频率和心率是反映当前人体生理状态的重要指标。随着科技的进步，各类光纤传感、压电传感、红外热成像、生物雷达等非接触式传感技术和可穿戴传感器技术[1]迅速发展，为生命体征的监测提供技术支持，对病情的早期诊断和动态预测有重要意义。

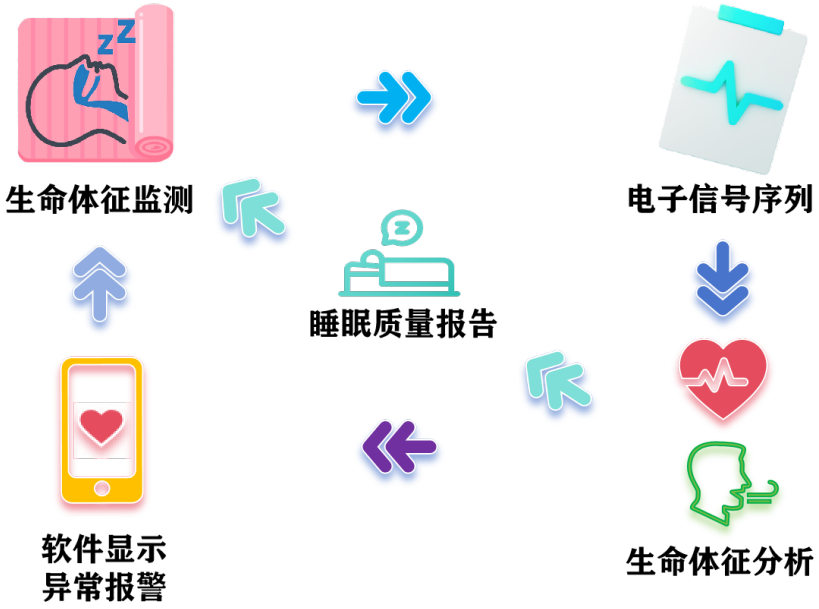

图 1　问题背景图

1.2 问题提出

现有某医疗器械企业利用其开发的生命体征监测垫收集了被监测者睡眠阶段的呼吸频率、心率、体动三种生命体征数据和相应的该时刻光纤反射转换成的电子信号。

问题一：监测到的电子信号长短不一，要求选取足够长的电子信号序列，聚类分析现有的电子信号序列，并分析每一个聚类的电子信号序列和相应的生命体征的特征。

问题二：监测仪器能够实时监测被检测者的电子信号，再根据电子信号转换为生命体征数据，要求建立模型，做到根据电子信号估计生命体征数据。

问题三：选取某一类疾病，经查找资料了解该类疾病的生命特征，从而对其电子信号序列特征进行分析。

问题四：除了呼吸频率、心率、体动三个生命体征，要求给出也需要被关注的其他生命特征。

2 模型假设

1. 假设附件中的一个 json 文件代表一个被监测者在某一时刻的生命体征数据和光纤在一分钟内的一秒钟返回的光功，如“3_2023_05_23_23_59_48.json”表示 3 号被监测者在 2023 年 23 日 23 点 59 分 48 秒对应的生命特征和该时刻光纤反射转换成的电子信号的数据文件；
2. 假设通过生命特征监测垫获得的监测数据均真实可靠;
3. 假设如果电子信号序列存在一定的周期性，附件数据可以当作有序的连续排列数据进行处理.

3 符号说明

符号	说明
$x(t)$	t 时刻的电子信号
$h(t)$	内模态函数
b	呼吸频率
h	心率
m	体动
T	时间序列长度
f	频率
A	幅度

注：表中未列出及重复的符号均以首次出现处为准。

4 问题一模型的建立及求解

4.1 问题一分析

问题一主要分为两个小问。对于第一小问，问题要求选取足够长度的电子信号序列。首先我们对将文件数据整合，得到时间序列，通过 ACF 函数图像发现时间序列的变化具有周期性，利用 CEEMDAN 并考虑到部分时刻数据的缺失确定电子信号序列的周期为 6s。对于第二小问，问题要求对电子信号序列进行聚类分析，首先考虑序列统计和频域两方面，分别选取序列的最大值、最小值、均值、标准差、偏度、峰度、熵七个统计特征，并用 MCMC-GARCH 求得三个反映波动性的特征。之后综合十个特征，利用肘部法确定 K 值，将时间序列聚为 3 类，最后根据聚类结果对电子信号序列的特征及相应生命体征的特征进行分析。

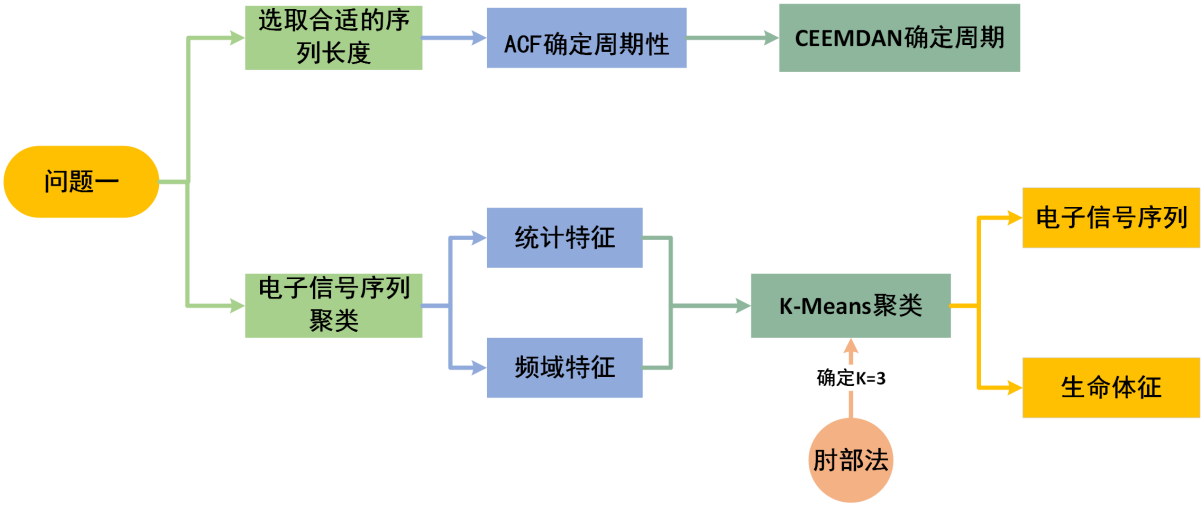

图 2　问题一思路图

4.2 选取序列长度

根据题目，我们需要针对不同的被检测者，选取合适长度的电子信号序列。我们希望该特定长度下的序列几乎能够包含被监测者全部电子信号信息。

为了方便分析电子信号序列信息，本文首先对文件数据进行整合处理，得到各被监测者的电子信号的时间序列。生命体征监测垫每秒钟生成 50 个电子信号，因此得到的时间序列步长为 0.2s.

4.2.1 ACF 检验序列周期性

为检测该时间序列中的周期性和趋势，本文对各被监测者的时间序列求取自相关函数（ACF），以判断电子信号在不同时间点的相关性程度。虽然有部分时刻缺失，但时

间跨度较大，可将电子信号的时间序列大致看作连续序列。自相关系数的公式如下：

$$ACF\left(k\right)=\frac{Cov\left[x\left(t\right),x\left(t-k\right)\right]}{Var\left[x\left(t\right)\right]} \tag{1}$$

其中 $ACF(k)$ 表示滞后期为 k 的自相关系数；$Cov\left[x\left(t\right),x\left(t-k\right)\right]$ 表示时间序列在时刻 t 和时刻 $t-k$ 处的协方差；$Var\left[x\left(t\right)\right]$ 表示时间序列在时刻 t 的方差。

以 3 号和 44 号的前 1200 个数据点为例，获取自相关函数曲线图如下所示。

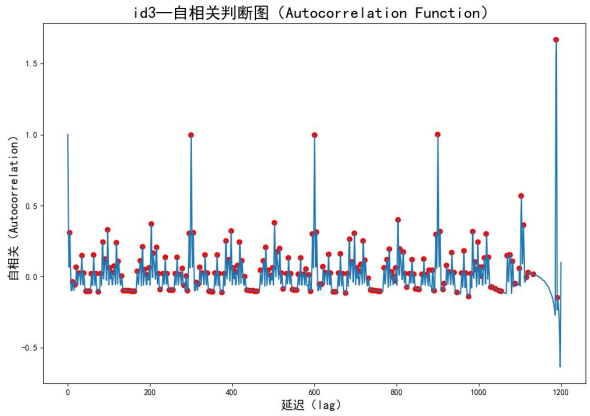

图 3　3 号被监测者的自相关函数图

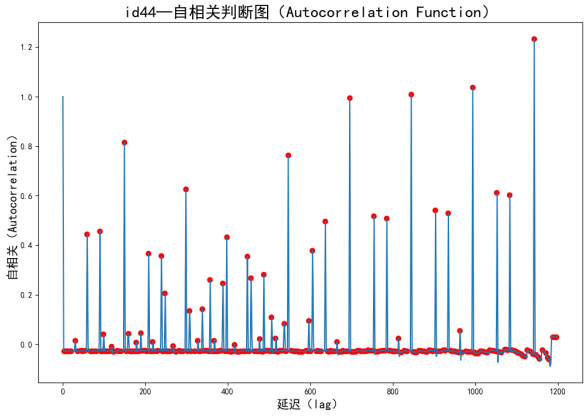

图 4　44 号被监测者的自相关函数图

从图中可以直观看出，自相关图上的峰值位置和衰减情况呈现周期性变化，从而可以初步判断时间序列具有周期性。

4.2.2 CEEMDAN 确定周期

CEEMDAN（完全自适应噪声集合经验模态分解）模型在处理非平稳和非线性信号时分析处理精确，效果较好。其通过在 EMD 过程中引入噪声信息并进行多次迭代提高了分解的鲁棒性和稳定性，避免了传统 EEMD 和 CEEMD 分解算法中白噪声从高频到低频的转移传递问题。

针对本题的电子信号的时间序列具有非线性且不平稳，本文决定采用 CEEMDAN 模型来对时间序列进行分析，估计其变化周期[3]。CEEMDAN 可以将非线性和非平稳数据分解成一组称为固有模态函数 IMF 的时变函数，每个 IMF 代表了数据中的一个局部尺度或频率分量。

CEEMDAN 的主要步骤如下：

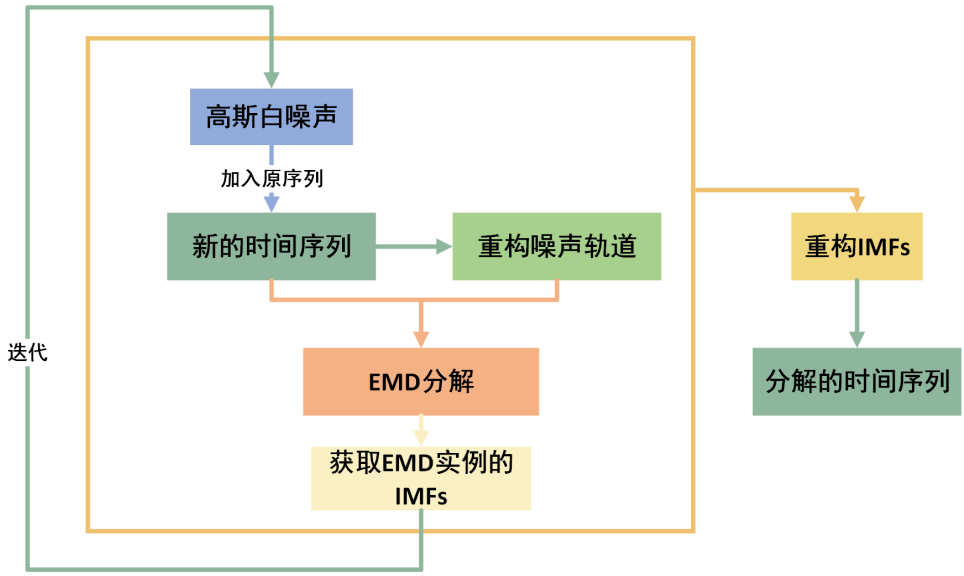

图 5　CEEMDAN 主要步骤

以 3 号和 44 号被监测者为例，利用 CEEMDAN 对两者的电子信号的时间序列进行分解得到如下结果

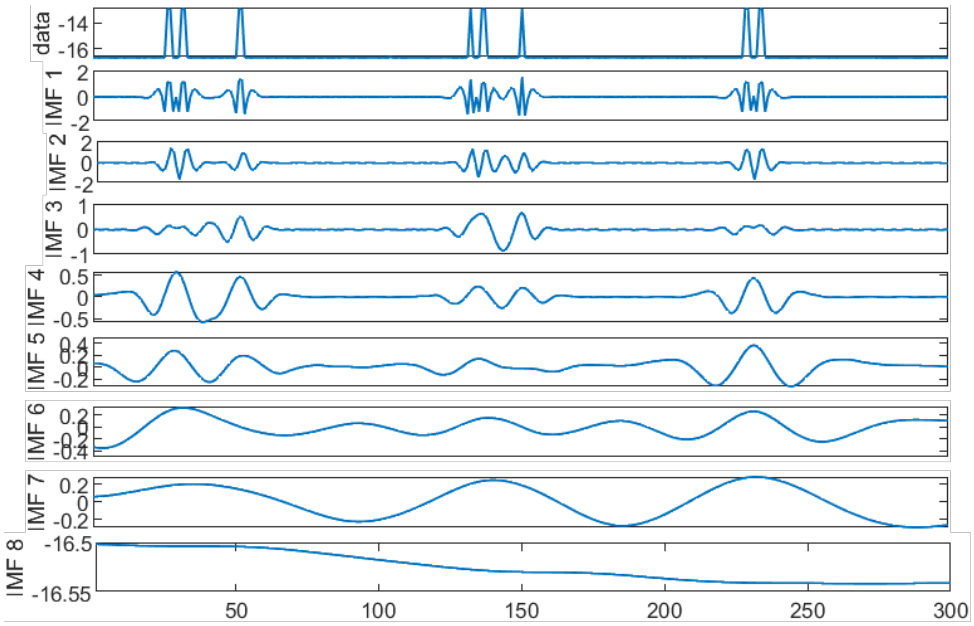

图 6　3 号被监测者的 CEEMDAN 分解结果

根据振幅谱密度图，观察其明显峰值的频率成分，发现峰值位置和衰减情况的变化周期大致为 3s。但可能存在生命体征监测垫运行不稳定等问题，导致部分秒数缺失。为保证选定的一个周期内包含的电子信号信息的完整性，本文选取两个周期作为一个完整周期，因此最终确定最佳周期为 6s，即 300 个样本数据点。

4.3 基于 MCMC-GARCH 的序列聚类分析模型

本文首先将五个被监测者的电子信号序列都按 6s 周期分为若干时间序列片段，由于时间常度不一定是周期的整数倍，将最后近似 6s 的时间序列作为一个周期处理，然后对所有时间序列片段进行聚类分析。

4.3.1 统计特征计算

在统计方面，为了描述不同电子信号序列的趋势和复杂度，本文选取了时间序列的最大值、最小值、均值、标准差、偏度、峰度、熵七个统计特征。其中偏度用来度量时间序列的非对称性，正偏斜表示数据分布的右尾较重，负偏斜表示数据分布的左尾较重；峰度表示度量时间序列的陡峭程度，峰度越大，数据分布相对于相对于正态分布越有尖锐趋势；熵反映评估时间序列的复杂性，熵越高，数据分布越不规则。

4.3.2 频域特征计算

在频域方面,为反映不同电子信号序列的波动性特征差异,本文采用 MCMC-GARCH（马尔可夫链的蒙特卡洛方法——广义自回归条件异方差模型）求得的参数来衡量[4]。

GARCH 模型：

GARCH 是由 ARCH（自回归条件异方差模型）推广得到的。该模型用可变条件方差来衡量序列的波动特征，而条件异方差主要来源于依赖于已知序列信息和滞后条件方差，从而相比 ARCH 具有更灵活的滞后结构，可以更好地体现序列的记忆特征和波动性模式。

本题的 GARCH(p,q) 模型可表示为

$$\begin{cases} \epsilon_t = \eta_t\sqrt{\sigma_t} \\ \sigma_t^2 = \omega + \sum_{i=1}^{p}\alpha_i\epsilon_{t-i}^2 + \sum_{j=1}^{q}\beta_j\sigma_{t-j}^2 \end{cases} \tag{2}$$

式中 σ_t^2 是时刻 t 的条件方差；$\{\eta_t\}$ 是均值为 0 方差为 1 的读题同分布随机变量，ϵ_{t-i}^2 是过去 i 个时刻的残差项。

三个重要参数的具体含义表示为

ω 是 GARCH 模型的常数项，用于描述平均波动率或常态波动；

α_i 是 ARCH 项的系数，用于描述波动性的自回归部分，表示前一期残差平方对当前条件方差的影响程度，它反映了时间序列对新的信息或冲击的反应敏感性；

β_j 是 GARCH 项的系数，用于描述波动性的条件异方差部分，表示表示前一期条件平方对当前条件方差的影响程度，它反映了时间序列波动性的持续性或惯性。

MCMC 估参：

考虑到 GARCH 模型中条件方差存在路径依赖问题，参数估计的常用的极大似然方法并不适用，本文在贝叶斯框架下采用 MCMC 来进行参数估计，MCMC 可以构建一个

马尔可夫链，从复杂的概率分布中进行采样，通过迭代生成样本，并根据接受概率决定是否接受新的状态，从而近似地获得参数估计。

为方便叙述，下面仅对 GARCH（1,1）进行讨论，其它情况类似。

Step1 Gibbs 抽样：上述 GARCH 的三个重要参数为 $\Theta=\{\alpha,\beta,\omega\}$。首先生成马尔科夫链

$$\left\{\Theta^{(0)},\Theta^{(1)},\Theta^{(2)},\ldots\right\} \tag{3}$$

式中序列中下一时刻 $t+1$ 处的 $\Theta^{(t+1)}$ 由先验部分的条件分布 $P\left(X|\Theta^{(t)}\right)$ 产生，仅与时刻 t 处的当前状态有关。其中 ω 的先验为半正态分布，限制了 $\omega>0$;α 和 β 的先验为 Beta 分布，限制了 $\alpha,\beta\in[0,1]$.

Step2 加入随机扰动：为模型可以灵活地适应不同时间段的波动率变化，引入高斯随机游走，表示条件方差的时间变化过程。该高斯随机游走过程会生成波动率的序列，这些序列将与观测数据一起用于后续模型参数的估计。

Step3 估计参数：根据大数定律，样本均值依概率收敛到后验分布下的均值。我们从后验概率分布中采样参数的值。规定了 1500 次迭代（1000 次采样和 500 次调整），生成了一个参数迹线，通过取参数迹线中的 ω、α 和 β 的平均值，用于近似表示它们的后验概率分布，并将这些均值作为结果返回。

4.3.3 K—Means 聚类

本文采用 K—Means 聚类算法来对时间序列进行聚类。它是一种无监督学习，同时也是基于划分的聚类算法，一般用欧式距离作为衡量数据对象间相似度的指标，相似度与数据对象间的距离成反比，相似度越大，距离越小该算法的主要作用是将相似的样本自动归到一个类别中，划分为若干个通常是不相交的子集，每个子集称为一个“簇(cluster)”，聚类既能作为一个单独过程，用于找寻数据内在的分布结构，也可作为分类等其他学习任务的前去过程，一般聚类结果较好[9]。

该算法主要步骤如下图所示：

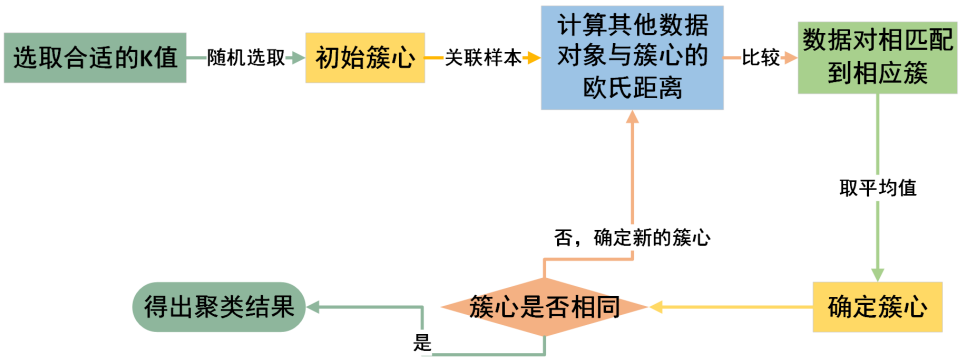

图 7　K-Means 主要步骤

4.3.4 肘部法确定 K 值

在 K-Means 主要步骤中，定义聚类类别数即选取合适的 K 值是 K-Means 聚类算法的关键步骤。本文采用肘部法进行确定 K 值，K 值的最优解是以成本函数最小化为目标，成本函数为各个类畸变程度之和，每个类的畸变程度等于该类重心与其内部成员位置距离的平方和，在这个平方和随 K 值的变化过程中，会出现一个拐点也即“肘”点，下降率突然变小时即认为是最优的 K 值。

本文根据上述十个指标对时间序列进行聚类，绘制肘部图如下图所示，其横坐标为聚类别数 K，纵坐标为畸变程度。

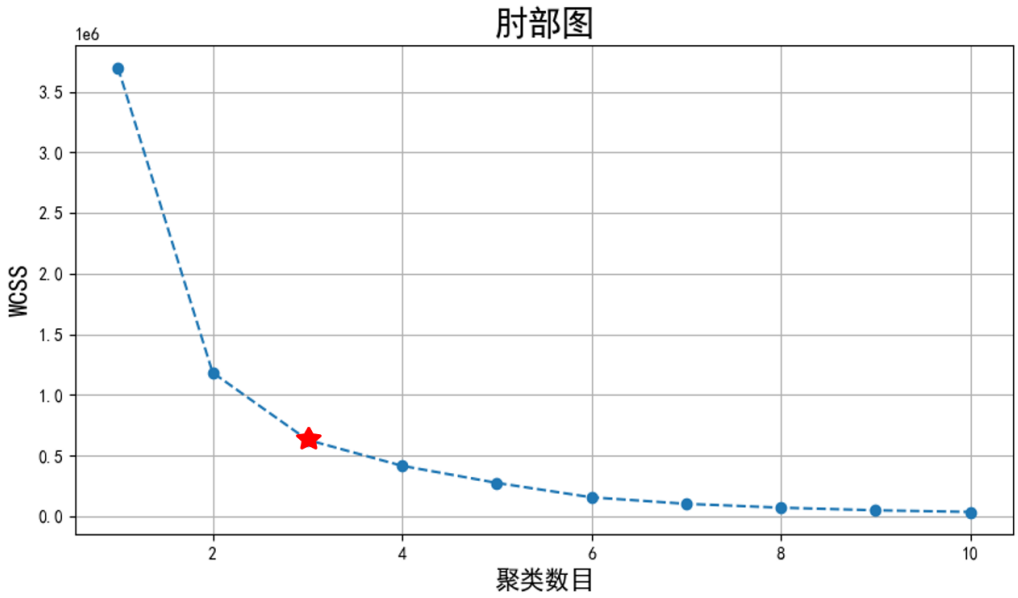

图 8　肘部图

从图中可以看出当类数从 1 增加到 3 时，总畸变程度下降较快；但当类别数超过 3 时，总畸变程度变化趋势变缓。对此，本文确定 K= 3 为最佳聚类数。

4.4 结果分析

结合统计和频域的十个特征指标将时间序列聚为三类，轮廓系数为 0.539，聚类效果较好。得到聚类结果图如下。

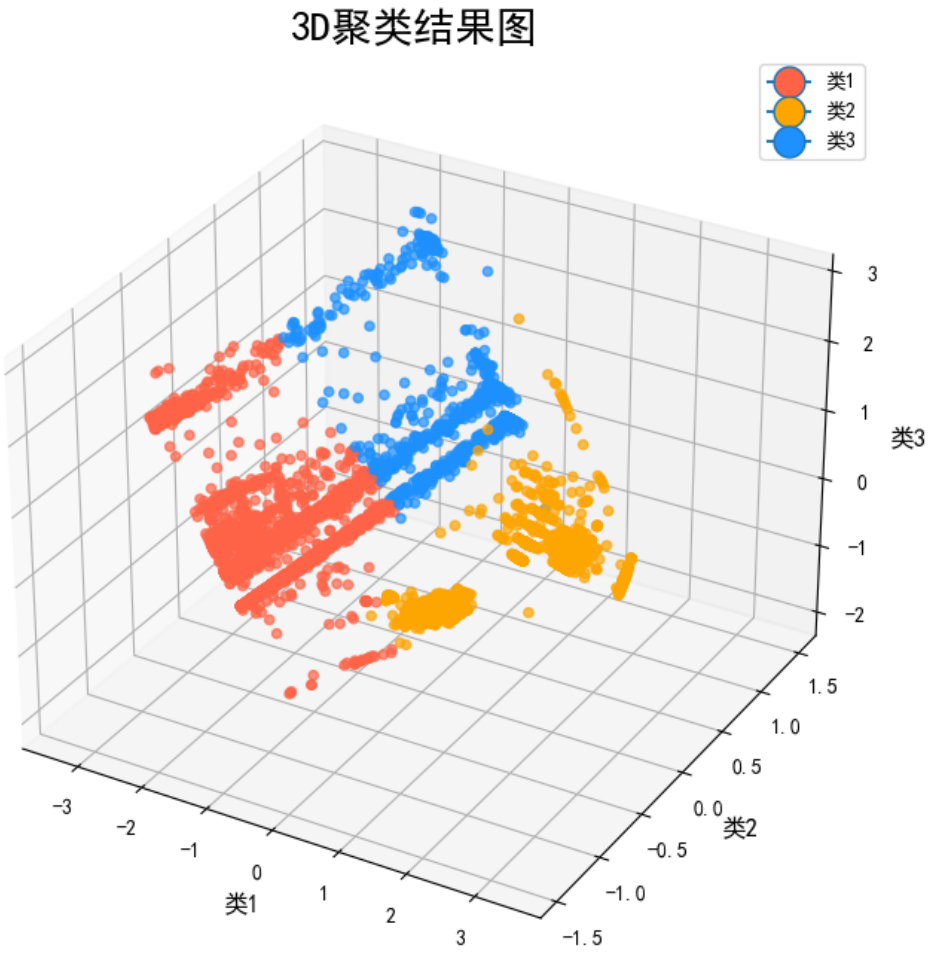

图 9　3D 聚类结果图

4.4.1 电子信号序列的特征分析

三个类别的频域特征和统计特征及分布情况统计如下。

表 1　不同类别的分布情况及其频域特征的平均值

聚类类别	总数	ω	α	β
1	1943	0.1545	0.4636	0.0163
2	4854	0.0546	0.1766	0.5596
3	497	0.0606	0.4574	0.0458

类别 1: 该类别人群的 ω 和标准差都较大，说明类别 1 整体上电子信号序列的波动性较大，不平稳，数据值之间的差异或变化较大，不容易被捕捉到稳定的趋势或信息。

类别 2: 绝大多数时间序列处于类别 2 中，分析类别的频域与统计特征发现，α 较

表 2　不同类别的统计特征的平均值

聚类类别	标准差	偏度	峰度	最大值	最小值	均值	熵
1	0.5264	5.8595	32.8212	-12.9435	-16.1985	-16.0655	0.9247
2	0.3072	0.9973	1.7972	-16.8303	-18.0310	-17.8915	1.9934
3	0.3359	8.3941	70.3776	-12.9715	-15.9124	-15.8147	0.9108

小，β 较大说明该类别的时间序列受新信息的冲击较小，相对稳定，且具有较长的连续性，存在持久波动特征；熵值相较于其他两类较大，说明序列复杂性高，信息量较大。

类别 3: 属于类别 3 的序列占比较小。峰度极大，表明该类别序列是非对称分布，具有长尾分布特征，即在尾部存在着较多的离群点或极端值。偏度也较大，表示时间序列的分布具有明显的偏斜性，在序列中有较多的尖峰，波动幅度更大。

4.4.2 相应生命体征的特征分析

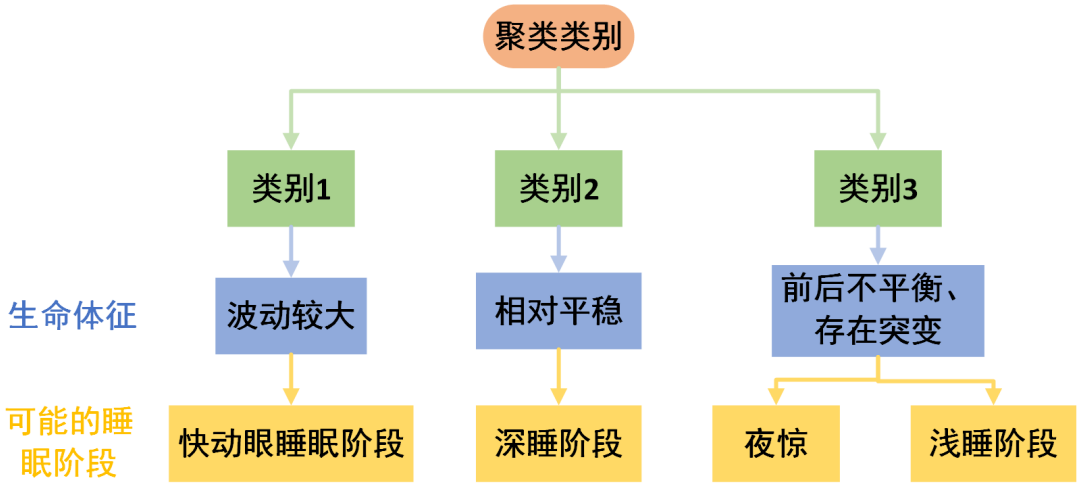

图 10　各类别的生命体征分析图

类别 1: 该类别在睡眠时生命体征变化较大，心率、呼吸等生命体征呈现不规律态势。在该类别的电子信号对应的时间段内，被监测者可能处于 REM（快动眼）睡眠状态，呼吸和心跳变得不规则，肌肉完全瘫痪，并且很难唤醒。

类别 2: 该类别相较于其他两个类别波动性小且该类序列的最大值和最小值都较小，说明其对应的生命体征较稳定。在该类别的电子信号对应的时间段内，被监测者可能处于深睡状态，迷走神经兴奋，心率会明显减慢，呼吸平缓，肌肉充分松弛。

类别 3: 该类别在睡眠时生命体征和前后不平衡情况，被监测者极有可能存在睡眠

障碍，如睡眠呼吸暂停综合征会导致呼吸频率和氧气饱和度的波动，而嗜睡和睡眠过度则可能引起呼吸和心率的减缓，导致生命体征不平稳，从而使序列偏度较高。除此之外，序列数据存在突变情况，一是可能存在夜惊情况，人在深睡中发生觉醒，伴有因强烈恐惧产生的尖叫、交感神经功能亢进等症状，导致呼吸心跳加速、体动增加；二是可能在该类别的电子信号对应的时间段内，被监测者处于入睡阶段或浅睡阶段，容易被惊醒，在被惊醒时，心跳呼吸加速，生命体征发生突变。

5 问题二模型的建立与求解

5.1 问题二分析

问题二要求根据电子信号估计生命体征数据。首先我们利用 CEEMDAN 对电子信号序列进行模态分解，通过 FFT 和呼吸和心率的振幅频率约束，分解得到两个内模态函数，分别反映被监测者的心率信号和呼吸频率信号。但考虑到在监测过程是非接触的，存在振幅损失，仅能反映生命特征的频率成分，无法反映具体数值。因此，我们利用 CEEMDAN 进一步提取电子信号序列特征，利用 GRU-随机森林预测模型，根据电子信号序列特征预测生命体征。

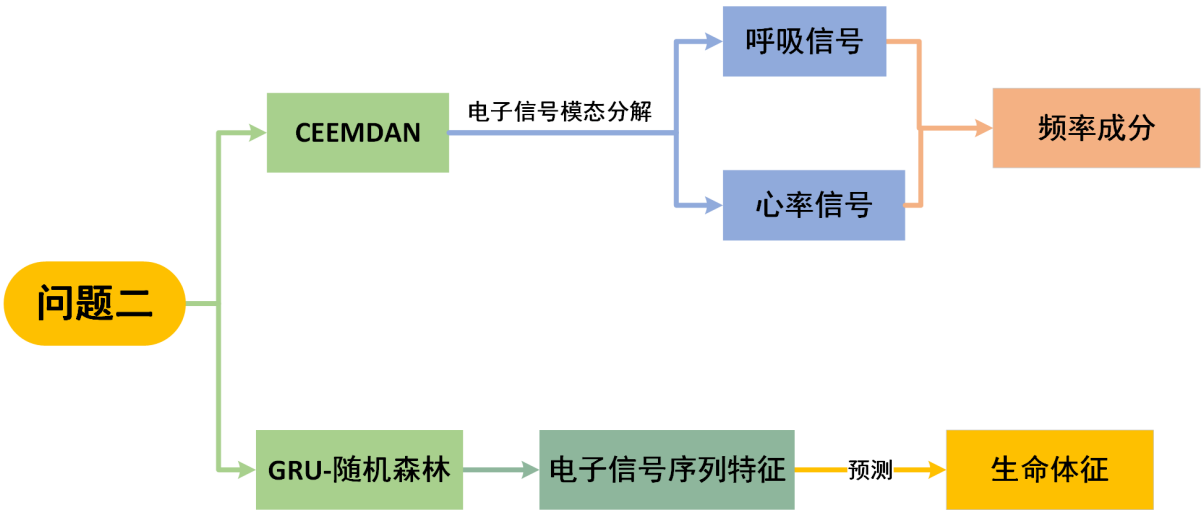

图 11　问题二思路图

5.2 基于 CEEMDAN 的电子信号模态分解模型

5.2.1 模型的建立

本文对电子信号序列进行分离，希望可以从序列 $x(t)$ 中分离出内模态函数，用其来反映被监测者的呼吸频率、心率等，从而实现电子信号转换为生命体征数据。本文决定

采用 CEEMDAN 进行模态分解[2] [10]，该方法在问题一中已被简单应用，下面对其在问题二中应用实现过程进行详述。

本文目标是将电子信号序列 $x(t)$ 分离为内膜态函数 IMF$_s$，用以表示被监测者的呼吸频率、心率。

CEEMDAN 的过程主要是提取 IMF$_s$，将其记作 $h(t)$：

Step1: 首先利用给定的电子信号序列初始化内膜态函数，如下。

$$h_i(t) = x(t) \quad (i = 0) \tag{4}$$

Step2: 从 $h_i(t)$ 中提取上包络和下包络并计算均值包络。上、下包络线分别是指通过对每个 IMF 的局部极大值或极小值进行插值得到连续的曲线，能够可视化并描述信号的局部振动范围，以及每个 IMF 的振幅变化情况。

$$m_i\left(t\right) = \frac{U_i\left(t\right) + L_i\left(t\right)}{2} \tag{5}$$

其中 $m_i\left(t\right)$ 表示第 i 次的内膜态函数的均值包络；$U_i\left(t\right)$ 表示上包络；$L_i\left(t\right)$ 表示下包络。

Step3: 内膜态函数要求在整个数据集中极大值的数量与极小值的数量之差不超过 1，并且在任意时刻下，局部极大值与局部极小值确定的数据均值为 0。为达到此要求，我们对 $h_i(t)$ 进行更新：

$$h_{i+1}(t) = h_i(t) - m_i\left(t\right) \tag{6}$$

重复 Step2—3 至满足 IMF 条件。

通过 FFT（快速傅里叶变换）计算 $h(t)$ 的主导频率。

$$h\left(x\right) = \sum_{n=0}^{T-1} h\left(t\right) e^{\frac{-2\pi ni}{T}} \tag{7}$$

其中 T 表示时间序列长度。经过 FFT 可将时域数据映射至频域，可通过频谱幅度谱，找到最大幅度值对应的频率，从而计算出主导频率。

由于题目所给信息有限，本文假设志愿者都为成年人。经查找资料，正常成人呼吸速率和心率的振幅与频率的约束如下

表 3　生命体征相关指标

生命体征	频率（Hz）	幅度（mm）
呼吸速率	0.1∼0.5	1∼12
心率	0.8∼3.0	0.1∼0.5

为了满足上述特定的呼吸频率和心率的振幅与频率的约束，我们对得到的每个 IMF 进行评估。

$$f\left[h\left(t\right)\right] \in \left[f_{\min}, f_{\max}\right] \tag{8}$$

其中 $f\left[h(t)\right]$ 表示 $h(t)$ 的频率；$f_{\min}$ 表示最小频率；$f_{\max}$ 表示最大频率。

$$A\left[h\left(t\right)\right] \in \left[A_{\min}, A_{\max}\right] \tag{9}$$

其中 $A\left[h\left(t\right)\right]$ 表示 $h(t)$ 的幅度；$A_{\min}$ 表示最小幅度；$A_{\max}$ 表示最大幅度。

5.2.2 求解及结果分析

根据上述 CEEMDAN 可以的得到个时间序列的心率和呼吸信号图。以 3 号被监测者为例，前 10s 的电子信号经模态分解后得到的心率与呼吸频率 IMF 如下图所示。明显可以看出，3 号被检测者的心率与呼吸频率较稳定，呈现周期性变化，基本上没有较大的浮动变化，生命体征较平稳。

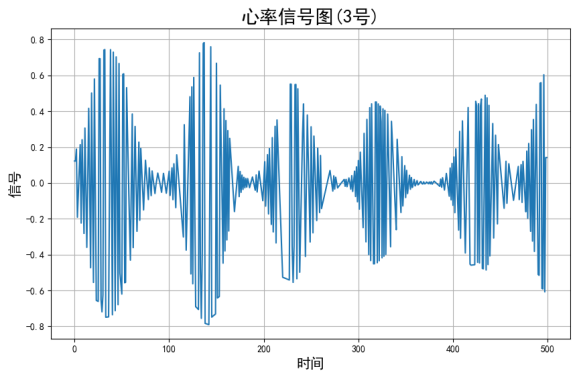

图 12　3 号被监测者的心率信号图

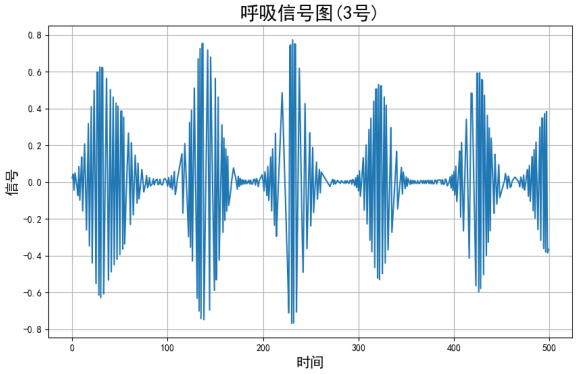

图 13　3 号被监测者的呼吸频率信号图

由于被监测者的体动具有很强的不确定性，不具有特定频率和时态特性，我们很难判断出被监测者的体动体征的频率成分。

除此之外，由于生命体征监测垫是非接触式生命体征检测方法，它的原理是人体轻微的呼吸和心跳会引起胸腔微小浮动，当光在光纤中传播时，可以捕捉到这样的细小变化，从而使光的性质发生改变，如强度、波长和相位等，再经过光探测器、解调器等处理可以获得被测量参数信号。在电子信号的生成过程中，会存在振幅损失。因此，本文通过 IMF_s 仅能初步得到呼吸频率和心率的相关信号成分，无法通过振幅来准确得知其监测的数值。

本文决定进一步分析电子信号序列，以能够得到准确的转换。我们利用 CEEMDAN 进一步获取电子信号序列的统计特征、频域特征、时域特征和非线性特征，试图通过序列特征来详细预测生命体征的具体监测数值，从而更进一步转换电子信号。

5.3 基于 GRU—随机森林的生命体征预测模型

5.3.1 模型的建立

GRU（门控循环单元）是 LSTM（常短期记忆人工神经网络）的一种变型，性能相类似，但结构比 LSTM 更加简洁，仅有两个门控激活函数——更新门以及重置门。而且由于参数的减少，计算效率得到很大提高。并且 GRU 通过引入门控机制有效避免了梯度消失和梯度爆炸问题。

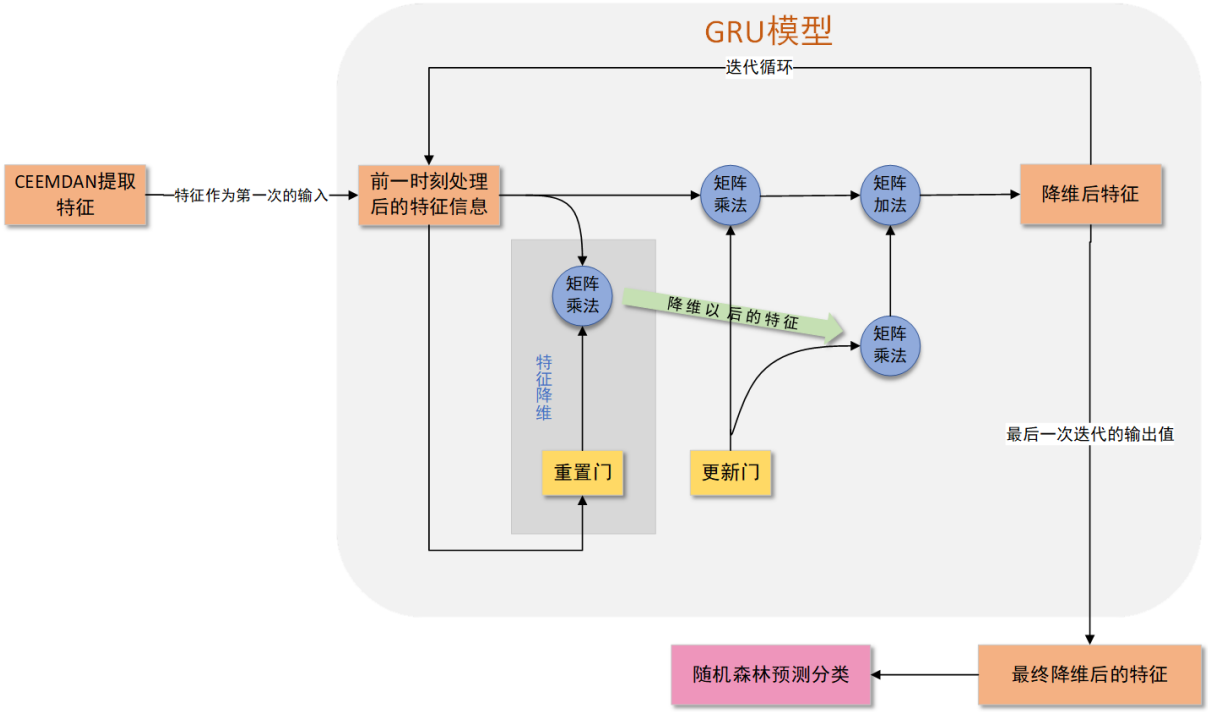

图 14　GRU 内部门控结构图

本文为根据电子信号特征预测生命体征数据，将 GRU 作为自编码器，通过 GRU 网络将输入序列映射到一个低维的隐藏表示空间，然后再将隐藏表示解码回原始输入序列，以实现数据的压缩和重构，同时保留了输入数据的的重要特征。由于得到的低维特征仍具有重要程度之分[5]，为了选取最佳特征进行预测，本文在 GRU 的基础上加入随机森林[11]。主要计算过程为

Step1 输入: 将由上述 CEEMDAN 得出的电子信号序列的特征 a_t 与相应的生命体征 $l_t = b_t, h_t, m_t$ 输入，作为编码器部分，其中 b_t、h_t、m_t 分别表示 t 时刻监测到的呼吸频率、心率、体动；

Step2 重置门计算: 重置门有效更新上一层的隐藏状态，并联系当前输入，在此基础上生成当前神经元的候选状态.

$$z_t = \sigma(W_z l_{t-1} + U_z a_t + c_z) \tag{10}$$

其中 z_t 表示重置门，σ 表示 Sigmoid 激活函数,W_z、U_z 和 c_z 分别是可学习的权重和偏置。

Step3 更新门计算: 通过计算更新门 $r(t)$ 来决定多少新的状态将被融合进来

$$r_t = (W_z l(t) + U_z a_t + c_r) \tag{11}$$

其中 $r(t)$ 表示重置门，W_r、U_r 和 c_r 分别是可学习的权重和偏置。

Step4 更新隐藏状态: 通过计算更新门 $r(t)$ 来决定多少新的状态将被融合进来

$$l_t = (1 - z_t)l_{t-1} + z_t tanh[W_l a(t) + U_l r_t l_{t-1} + c_l] \tag{12}$$

其中 W_l、U_l 和 c_l 分别是可学习的权重和偏置。

Step5 GRU 输出: 通过重复上述步骤，建立长期记忆并反向传播，我们得到了训练好的 GRU 自编码模型，当输入电子信号序列的特征，最后得到降维后的特征。

Step6 随即森林预测： 基于上述 GRU 降维后的特征构造决策树, 通过递归地选择最佳特征和最佳分割点，将数据划分为不同类别。通过投票或多数表决的方式，将所有决策树的预测结果集成结合起来，决定最终的预测类别。

5.3.2 求解及结果分析

基于 GRU 自编码模型，将 43780 个样本集的 80% 作为训练集，20% 作为测试集。得到五位被监测者各生命体征的均方根误差如下表所示，均方根误差不大，说明模型预测效果较好。

表 4　被监测者各生命体征的均方根误差

被监测者	呼吸频率	心率	体动
3	4.4942	6.4520	10.9235
8	7.9641	3.5236	17.4326
28	4.9641	8.0935	15.0989
30	4.2493	4.9877	13.8999
44	6.2783	6.9838	13.8849

经查找资料，本文将根据将各项生命体征分为低、中、高三个类别。划分标准如下

- 心率：小于 55 次/分钟为低，55~70 次/分钟为中，大于 70 次/分钟为高；
- 呼吸频率：小于 12 次/分钟为低，12~20 次/分钟为中，大于 20 次/分钟为高；

● 体动：小于 20 为低，20~60 为中，大于 60 为高。

将所有被监测者出现过的生命特征情况进行频数统计，并利用划分标准划分，绘制频数统计圆环图，如下所示。

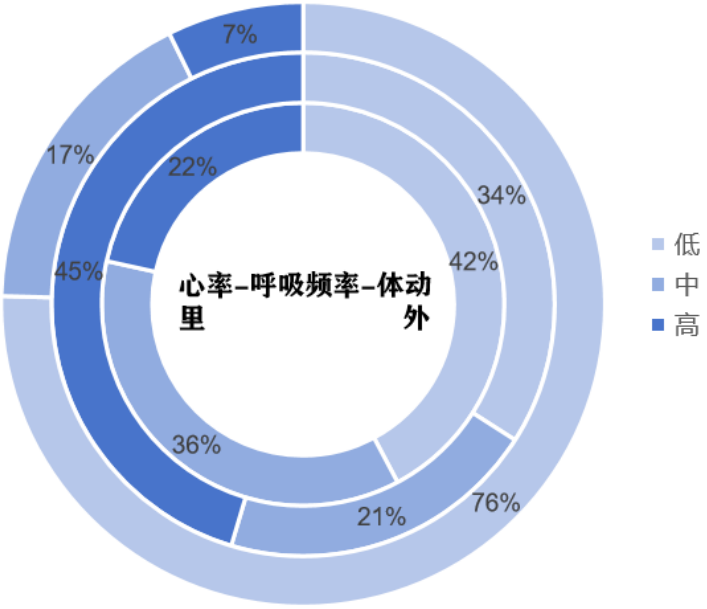

图 15　频数统计圆环图

对 GRU 预测结果进行分类，得到预测的心率、呼吸频率、体动三个生命体征类别准确率分别为 78.76%，为 80.32%，为 84.69%. 三种生命体征的准确率大于 75%，模型实现效果较好。本文还用特征感知机、支持向量机、逻辑回归分类三个模型与 GRU-随机森林进行对比，结果如下。

模型	GRU+ 随机森林	特征感知机	支持向量机	逻辑回归分类
呼吸频率准确率	78.76%	37.57%	43.93%	67.14%
心率准确率	84.69%	27.04%	55.87%	78.79%
体动准确率	80.32%	80.92%	39.28%	40.65%

通过上述两个模型可以有效将电子信号序列信息转换为生命体征。我们可以利用 CEEMDAN 的电子信号模态分解模型，从电子信号提取呼吸与心跳的频率成分，从而可以观测到生命特征的变化趋势情况。基于 GRU 的生命体征预测模型，我们可以实时将电子信号转换为生命体征状况，并进行分类。当监测到生命体征所属类别为高或低时，可以进行报警，告知医护人员和家属，及时进行干预处理，实时保护高危人群健康安全。

6 问题三模型的建立与求解

6.1 问题三分析

问题三要求分析某一类疾病的生命体征和电子信号序列特征。本文选取阻塞性睡眠呼吸暂停和打鼾，建立基于差分序列和滑动窗口的电子信号特征分析模型。首先分析了两种疾病对生命体征变化的影响。之后通过对五位被监测者呼吸信号的分析，得出 30 号被监测者发生阻塞性睡眠呼吸暂停疾病的风险最大。然后对 30 号被监测者的生命体征进行分析，印证了确实存在患病风险。最后通过差分序列检验和滑动窗口，分析该类疾病下电子信号序列的特征。

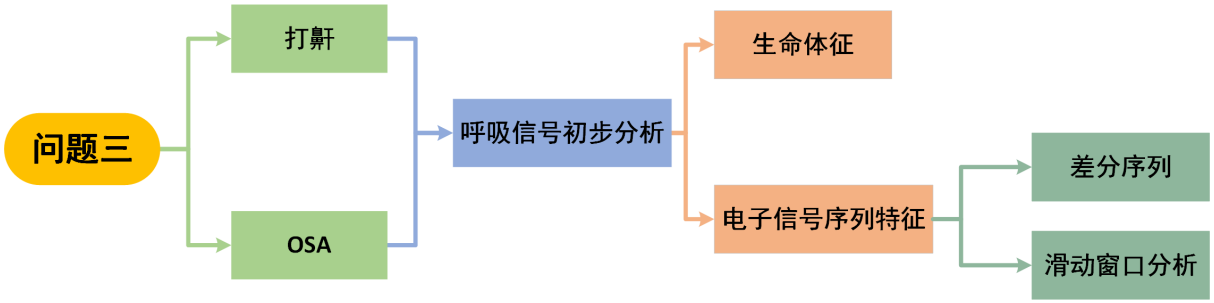

图 16　问题三思路图

6.2 疾病分析

本文主要考虑了打鼾和 OSA（阻塞性睡眠呼吸暂停）两种疾病。在打鼾严重时常伴有 OSA 发生，因此我们先研究打鼾症状再进一步分析 OSA[6]。两种疾病对睡眠生命体征变化的影响如下。

打鼾：

- 呼吸：由于气道部分阻塞，空气流动受阻，产生震颤声，这种震颤可能在电子信号中显示为呼吸的震动或噪音；
- 心率：打鼾可能会导致心跳不规则；
- 体动：打鼾的人在夜间可能会有更多的体动，尤其是这在呼吸恢复或努力重新开始呼吸时。

OSA：

- 呼吸：患病者的呼吸会因为气道受阻完全停止或显著减少，一般会持续 10s 以上；
- 心率：由于缺氧，患者可能会出现心率减慢；当呼吸恢复时，可能会出现心率的短暂增加；
- 体动：与打鼾相似，当呼吸暂停结束时，患者很可能会突然惊动，可能是由于患者尝试调整姿势重新开始呼吸。

6.3 基于呼吸信号的初步分析

通过 CEEMDAN 可以得到被监测者某段时间的呼吸信号图，如下所示。

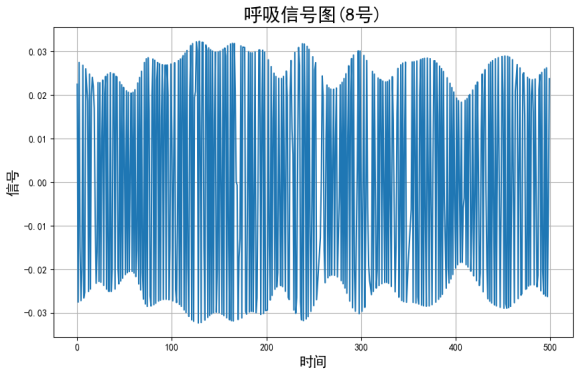

图 17　8 号被监测者的呼吸信号图

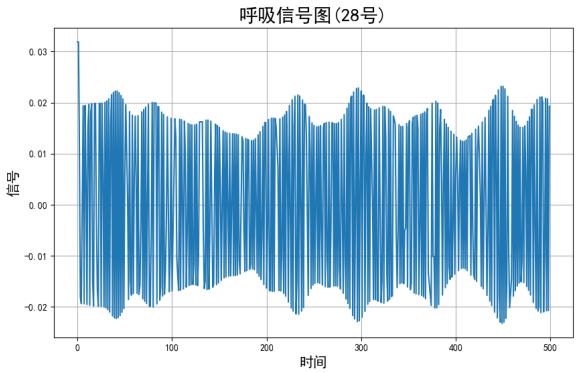

图 18　28 号被监测者的呼吸信号图

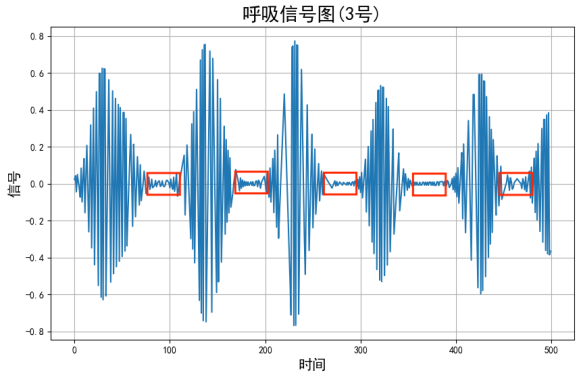

图 19　3 号被监测者的呼吸信号图

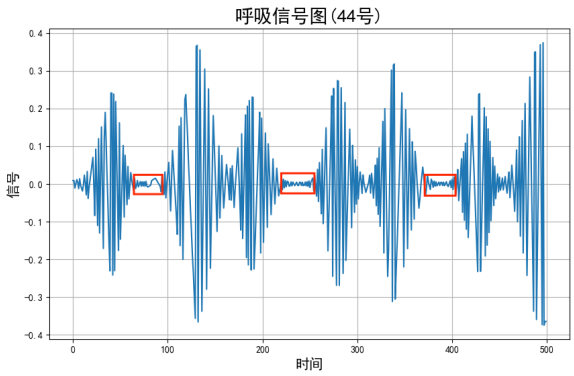

图 20　44 号被监测者的呼吸信号图

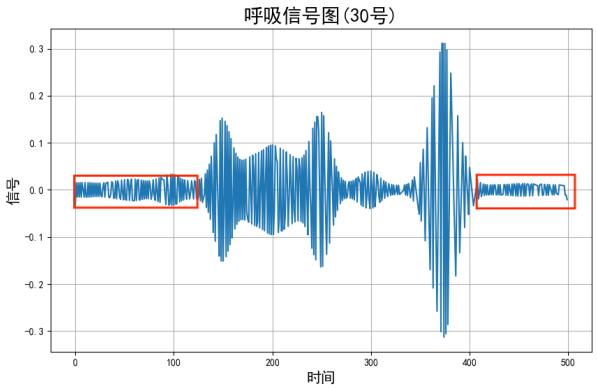

图 21　30 号被监测者的呼吸信号图

根据图像可以初步分析，图中用红色方框标记的区域是呼吸信号连续处于 0 附近的信号区间，可以用来初步判断被监测者的呼吸暂停情况。8 号、28 号的呼吸情况较为稳定，可能不存在打鼾现象；3 号、44 号在呼吸过程中会存在很短暂的、具有周期性的呼

吸暂停，可能是由于打鼾对呼吸情况造成的影响；30 号会存在较长时间的呼吸暂停，可能会有较大的 OSA 患病风险。本文在后面的分析中主要针对 30 号被监测者的情况进行详细分析，并分析其电子信号的特征。

6.4 30 号被监测者生命体征分析

根据 30 号被监测者的生命体征数据，可以绘制呼吸频率、心率、体动随时间的变化图像，如下所示。可以看出 30 号被监测者的呼吸频率存在低于 12 以下的区域，比如 1~1000s，并且对应的心率也低于正常值，且在 1000 秒左右出现体动的突变，在之后呼吸情况、心跳情况都发生了与之前不同的变化，可能是因为被监测者在惊醒后尝试调整姿势重新开始呼吸。根据这样的结果分析也能进一步证明 30 号被监测者确实存在睡眠问题，存在患有 OSA 的风险。

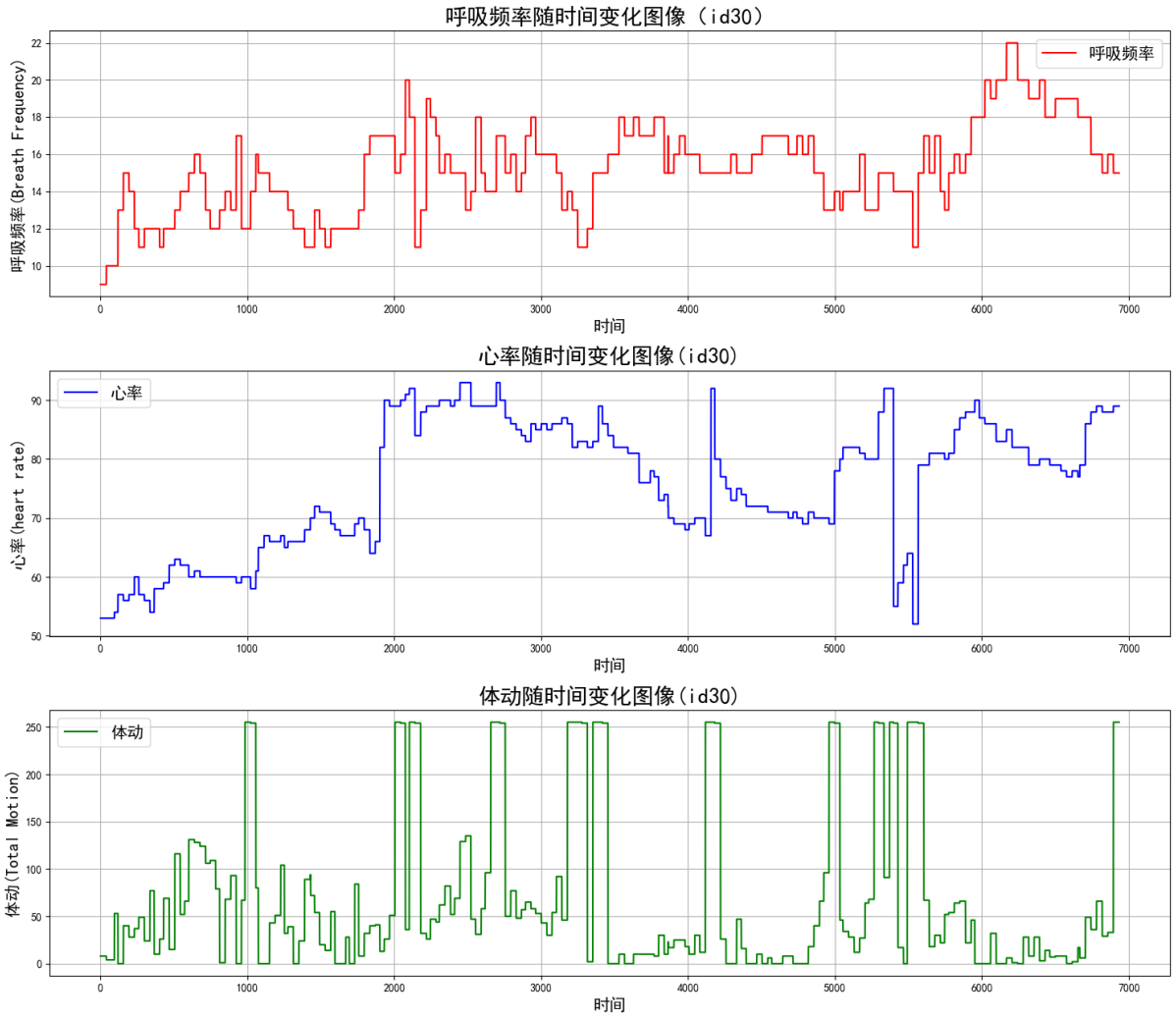

图 22　生命特征随时间变化图

6.5 基于差分序列和滑动窗口的电子信号序列特征分析模型

6.5.1 基于差分序列的打鼾检验

打鼾出现前后，呼吸会存异常变化。本文利用差分法寻找电子信号序列的异常值[7]，以检验 30 号被监测者的打鼾情况。

首先，对电子信号序列数据进行归一化处理；

$$x'(t) = \frac{x(t) - \min\{x(t)\}}{\max\{x(t)\} - \min\{x(t)\}} \tag{13}$$

其中 $x'(t)$ 表示归一化后的电子信号序列。

之后，利用如下公式生成差分序列。

$$d(t) = x'(t+1) - x'(t) \tag{14}$$

其中 $d(t)$ 表示差分后的数据。

根据差分序列的特征并结合问题，本文将差分序列中 $[\mu - 3\sigma, \mu + 3\sigma]$ 将被视为潜在的打鼾区域。最终选取了 30 号被监测者连续 20 秒的数据进行潜在打鼾区间标记，如下所示。

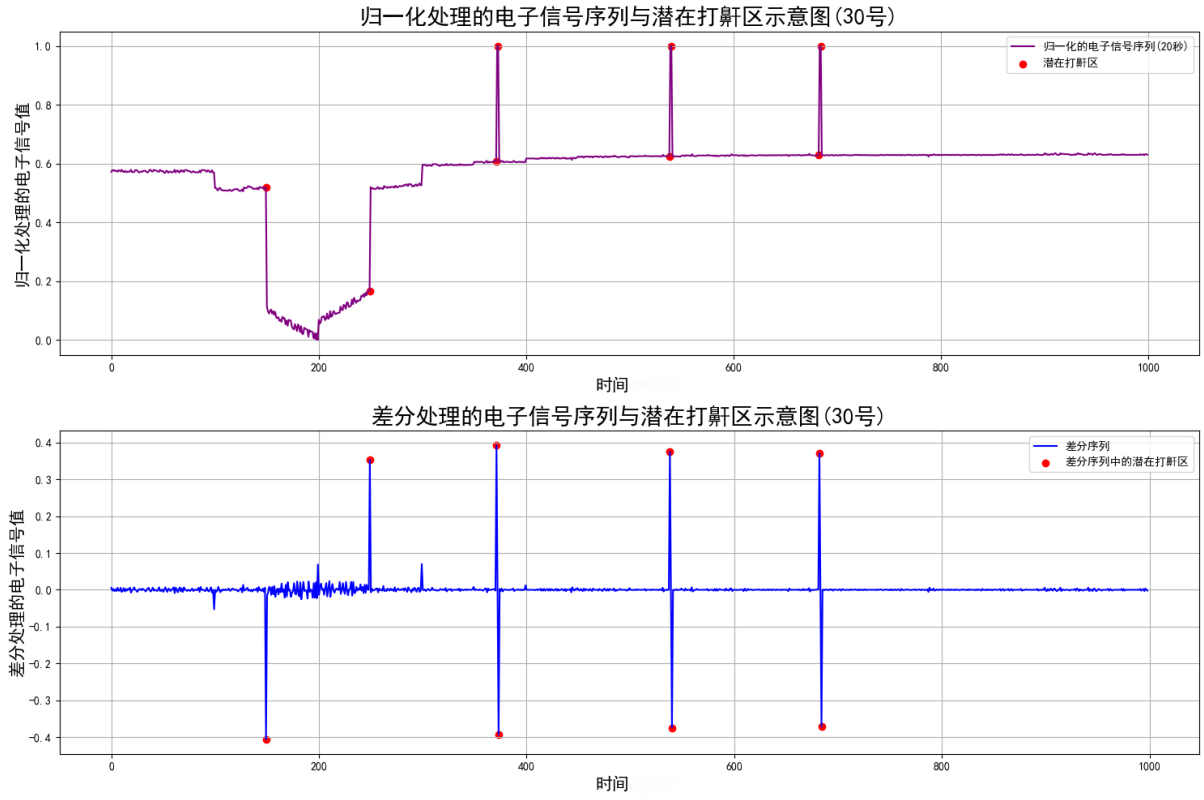

图 23　潜在打鼾区间图

根据图像可以分析，在 2s~6s 之间（第 100~300 个数据点）可以发现存在一段剧烈变化的电子信号，且在 4s 左右（第 200 个数据点附近）的位置降到了很低的信号值。

图中红点的标记位置为潜在的打鼾现象出现点，在 9s~10s 未发现打鼾现象，可能是因为被监测者由于呼吸暂停导致惊醒，这与之前的生命特征分析结果一致，进一步证明了被监测者存在较大的 OSA 患病风险。

6.5.2 基于滑动窗口分析的 OSA 检验

滑动窗口分析法常用于观察和识别序列中的短期模式、周期性或趋势变化，既可以在较小的时间范围内观察序列的局部特征，又能保留序列的整体信息[8]。本文采用滑动窗口分析法对 30 号被监测者电子信号序列进行分析。以电子信号时间序列的初始时间节点作为窗口放置的起始位置，确定 350000 个数据点为滑动窗口的大小，通过循环或迭代的方式实现窗口的不断滑动，在每个滑动窗口的位置，对窗口内的数据进行统计分析，最终获得的结果如下所示。

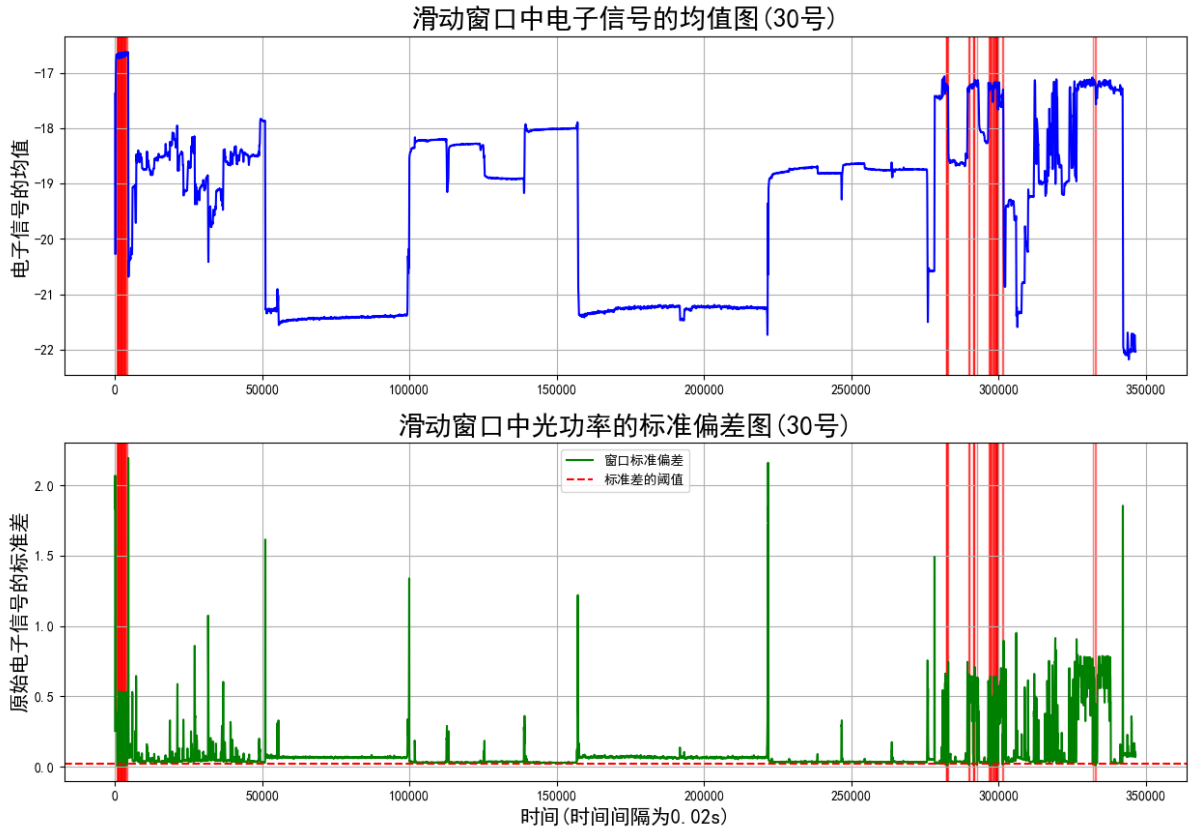

图 24　滑动窗口图 (30 号)

根据滑动窗口分析的结果可以得出以下结论。在标有红色背景的窗口中可以发现具有低标准偏差的连续时间段，电子信号的变化较小，这可能是呼吸暂停的标志，特别是 2s~6s 之间（第 100~300 个数据点）也被标上红色背景，进一步验证了我们之前做出的判断，在这一段时间段里 30 号被监测者确实存在呼吸暂停的现象，存在较大的 OSA 患病风险。

6.6 结果分析

通过对存在 OSA 患病风险的被监测者进行电子信号特征分析，可以得出当电子信号出现剧烈变化、数值存在异常下降时，其的差分序列也会出现剧烈波动，在这种情况下的被监测者往往会出现打鼾的现象，同时呼吸也会出现极为短暂的、具有周期性的呼吸暂停；当电子信号出现低标准偏差的连续序列时，电子信号变化较小，可能会出现呼吸暂停的现象。综合来看，阻塞性睡眠呼吸暂停和打鼾会对生命体征造成不利的影响，被监测者应认真浏览生成睡眠质量报告，通过监测获得的心率和呼吸频率信息判断自己的睡眠情况，采取适当的措施进行疾病预防或治疗。

7 问题四模型的建立与求解

7.1 问题四分析

生命监测垫主要依赖于监测微小的体表震动来测量生命体征。除了常规的生命体征（如心率、呼吸频率、体动）之外，还有其他的生命体征可能会引起足够的体表震动，并能使用生命体征监测垫进行监测。

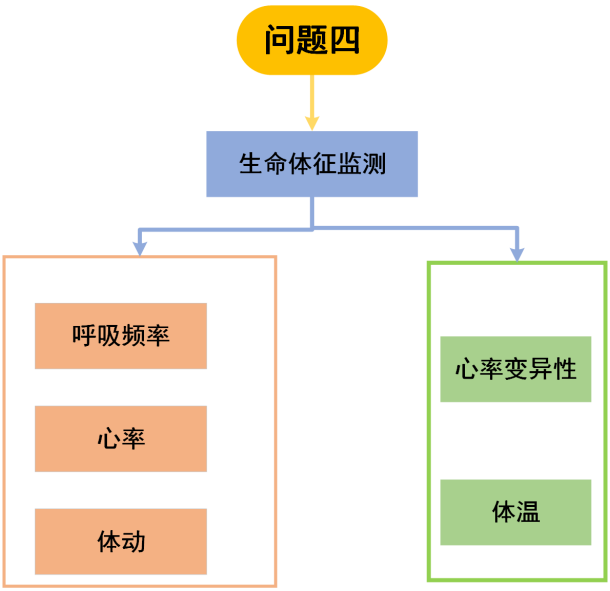

图 25　问题四思路图

7.2 其他生命体征

生命体征的有效监测是医疗救治和健康维护的重要前提，通过监测得到的信号序列能够在一定成度上预测疾病，提前有效避免疾病的发生。本题中生命监测垫主要依赖于监测微小的体表震动来测量生命体征。除了题目中的三项生命体征之外，还有其他的生命体征可能会引起足够的体表震动，并能使用生命体征监测垫进行监测。

心率变异性（HRV）是指连续窦性心动周期之间时间上的微小差异。经查找相关资料得知，人体的 HRV 与自主神经系统密切相关，而自主神经系统对心脏的调节作用，反映了副交感神经张力与交感神经张力的大小及他们之间的平衡关系。HRV 分析对于心室颤动、心源性猝死有较好的预测效果，并且根据相关学者研究 HRV 信号在研究睡眠分期中的具有重要作用。如果在上述睡眠类疾病分析问题中，生命体征加入对 HRV 的监测，得到的电子信号序可能会有更加明显的特征。

体体温的异常升高或降低可能是疾病或感染的迹象，可以反映体内的多种生理和病理变化，因此对体温的监测异常重要。通常情况下，体温的范围在 36.5°C 到 37.5°C 之间，但这可能会受到个体、一天中的时间以及测量位置的影响而有所不同。当体温超出这个正常范围，通常意味着身体正在对某些感染、炎症或其他病理状态做出反应。例如，发热是身体对外部病原体的一种防御机制，而低体温可能是由于在寒冷环境中过度暴露、代谢异常或某些药物的影响。监测垫的设计主要是为了捕捉微小的身体震动，从而获取生命体征信息。因此，虽然它可能无法直接测量体温，但体温的异常会影响到其他生命体征，如心率、呼吸频率和体动。高热可能导致心率加快，而低体温可能导致心率和呼吸速度减慢。因此，通过对这些生命体征的综合分析，我们可以间接地推断出体温的异常，使得医护人员或家属能够及时采取措施。

7.3 未来展望

由于生命体征监测垫必须以依靠震动才能进行监测，从而受到限制，导致脉搏、血糖、血氧饱和度同样十分重要的生命体征无法进行检测。并且容易受到外界震动因素的影响。后续公司可进行技术改进，研发手戴式、雷达监测等监测传感器对生命体征监测垫进行改进，使其形式功能多样化。

8 模型总结与评价

8.1 模型总结

本文通过 ACF 检验和 CEEMDAN 模型对电子序列进行周期性分析与求解，然后建立 MCMC-GARCH 的 K-Means 模型从统计特征和频域特征两个角度求解电子序列和生命体征的特征并进行聚类分析。之后基于第一问的 CEEMDAN 模型建立电子信号估计生命体征数据的数学模型，通过 GRU-随机森林算法进行预测。最后结合上述特征分析结果，采用差分序列法和滑动窗口法，通过分析电子序列特征来预测和初步诊断呼吸暂停、打鼾等疾病。

8.2 模型优点

1) 问题一采用马尔科夫链蒙特卡洛算法对传统的 GARCH 模型做出改进，更好地捕捉本问题时间序列中的非线性关系和波动性聚集特点。

2) 问题一和问题二采用基于 CEEMDAN 的电子信号模态分解模型，可以更好地描述信号的频域特征，且本模型具有自适应能力，能准确处理本问题数据集中非线性、非平稳和非高斯信号，有利于电子信号转换为生命体征数据。

3) 问题二构建基于 GRU 改进的随机森林预测分类模型，采用 GRU 改进随机森林算法对于特征选择和特征构造的部分，实现有效的特征降维，在一定程度上提升了模型性能。

4) 问题三结合差分序列和滑动窗口法对电子信号序列进行特征分析，差分序列强调序列变化趋势，滑动窗口在保留序列整体信息的同时可以兼顾电子信号序列的局部特性，使分析更全面准确。

8.3 模型缺点

1) 问题一运用到的 MCMC-GARCH 模型在计算复杂度上较高，对于初始模型参数有较高的要求，参数估计可能需要较长的计算时间。

2) 问题二中采用的 CEEMDAN 算法也存在一些挑战和限制，如选择合适的参数和对噪声的处理等，需要仔细处理以确保分析的准确性和可靠性。

8.4 模型改进

首先，本文利用 CEEMDAN 模型对电子信号序列进行周期性分析和求解，同时利用该模型在信号分解上的优势，成功分离出心率信号和呼吸频率信号，高质量完成电子信号序列向生命体征数据的转换，后续可以结合更多生命体征的特性，尝试进行分离，挖掘出电子信号中更多的信息，为被检测者的睡眠状态做出更多维全面的评估。

此外，本文基于 GRU 改进的随机森林算法，可以实现电子信号对生命体征的估计，本文不仅实现了使用通过电子信号对生命体征的预测，也结合专业领域中对生命体征的评估，进一步进行分类，后续可以考虑使用分类结果可以作为异常情况的提示，便于医护人员及时干预处理，实时保护被监测者的健康安全。

参考文献

[1] 朱国建, 陈爱英, 王冉冉等. 用于人体健康监测的可穿戴传感器件研究进展 [J]. 分析化学,2022,50(11):1673-1684.

[2] 韩宇, 张兴敢. 基于改进变分模态分解的生命体征检测 [J]. 南京大学学报 (自然科学),2022,58(04):680-688.

[3] 姜山, 周秋鹏, 董弘川等. 考虑数据周期性及趋势性特征的长期电力负荷组合预测方法 [J]. 电测与仪表,2022,59(06):98-104.

[4] 李强, 周婉玲, 董耀武. 马尔可夫状态转换 GARCH 族模型的选择与估计 [J]. 统计与决策,2021,37(18):14-18.

[5] 田丰, 程志华, 侯天育. 基于 CEEMDAN-SE 和 LSSA-GRU 组合的短时交通流量预测 [J]. 公路交通科技,2023,40(06):194-202.

[6] 刘众, 王新安, 李秋平等. 基于 ECG 信号和体动信号的睡眠分期方法研究 [J]. 北京大学学报 (自然科学版),2021,57(05):833-840.

[7] 章小宝, 陈巍. 基于时域有限差分的通信干扰信号广域监测 [J]. 计算机仿真,2021,38(10):227-231.

[8] 马驰远, 雷国庆. 一种 CDC 信号滑动窗口时序分析方法 [J]. 计算机工程与科学,2022,44(02):214-219.

[9] Benaimeche M A , Yvonnet J , Bary B ,et al.A k-means clustering machine learning-based multiscale method for anelastic heterogeneous structures with internal variables[J].International Journal for Numerical Methods in Engineering, 2022, 123(09): 2012-2041.

[10] Huang R .Laser gyro signal filtering by combining CEEMDAN and principal component analysis[J].Journal of Vibroengineering, 2021, 23(08): 1820-1832.

[11] Riedel J , Lettow M , Grabarics M ,et al.Predicting Structural Motifs of Glycosaminoglycans using Cryogenic Infrared Spectroscopy and Random Forest[J].Journal of the American Chemical Society, 2023, 145(14):7859-7868.

附录 A　问题一源代码

1.1　ACF 检验程序

```
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_excel("E:/Desktop/3_data.xlsx")

# t = np.linspace(0, 1198, 1, endpoint=False)
# x = np.sin(0.1*np.pi*t)
fig, ax = plt.subplots()
ax.plot(df.index, df["opticalpower"], color='m', marker='o', markersize=3)
ax.grid(color='c', alpha=0.5, linestyle=':')
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

# 生成示例时间序列数据
time = df.index
x = df["opticalpower"]

# 计算自相关函数
def autocorrelation(x):
    n = len(x)
    variance = np.var(x)
    mean = np.mean(x)
    autocorr = np.correlate(x - mean, x - mean, mode='full')[-n:]
    autocorr /= (variance * (np.arange(n, 0, -1)))
    return autocorr

# 寻找周期
def find_period(autocorr):
    peaks, _ = find_peaks(autocorr)
    return peaks

# 判别最佳周期
def find_best_period(periods):
    best_period = np.max(periods)
    return best_period

# 绘制自相关函数和周期
autocorr = autocorrelation(x)
peaks = find_period(autocorr)
```

```
periods = np.diff(peaks)

# 打印周期值
print("周期值:", periods)

# 判别最佳周期。。
best_period = find_best_period(periods)
print("最佳周期:", best_period)

plt.subplot(2, 1, 1)
plt.plot(time, x)
plt.xlabel('Time')
plt.ylabel('Data')
plt.title('Time Series Data')

plt.subplot(2, 1, 2)
plt.plot(np.arange(1, len(autocorr) + 1), autocorr)
plt.scatter(peaks, autocorr[peaks], color='red')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation Function')

plt.tight_layout()
plt.show()
```

1.2 CEEMDAN 求周期性程序

```
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
import numpy as np
import pandas as pd
from scipy.interpolate import UnivariateSpline
import numpy as np
import matplotlib.pyplot as plt

# CEEMDAN函数实现
def ceemdan(signal, max_imf):
    imfs = []

    for _ in range(max_imf):
        h = signal.copy()
        while True:
            d = h - np.mean(h)
            d = _emd(d)
            r = h - d
```

```
            if _is_imf(r):
                imfs.append(d)
                break
            h = r
    imfs.append(signal - np.sum(imfs, axis=0))
    return imfs

# 辅助函数：判断信号是否为IMF
def _is_imf(signal):
    # 判断极值点个数
    peaks = np.sum((signal[:-2] > signal[1:-1]) & (signal[2:] > signal[1:-1]))
    valleys = np.sum((signal[:-2] < signal[1:-1]) & (signal[2:] < signal[1:-1]))
    return abs(peaks - valleys) <= 1

# 辅助函数：执行EMD
def _emd(signal):
    imf = signal.copy()
    imf_prev = imf.copy() + 1

    while np.sum((imf_prev - imf)**2) > 0.001:
        imf_prev = imf.copy()
        # 分解模式和提取残差
        imf = imf - _mean_envelope(imf)

    return imf

# 辅助函数：计算信号的均值包络
def _mean_envelope(signal):
    upper = []
    lower = []

    # 寻找极值点
    maxima = (signal[:-2] < signal[1:-1]) & (signal[2:] < signal[1:-1])
    minima = (signal[:-2] > signal[1:-1]) & (signal[2:] > signal[1:-1])

    for i, (is_max, is_min) in enumerate(zip(maxima, minima)):
        if is_max:
            upper.append(i+1)
        if is_min:
            lower.append(i+1)

    # 计算上下包络线
    upper_env = np.interp(range(len(signal)), upper, signal[upper])
    lower_env = np.interp(range(len(signal)), lower, signal[lower])

    return (upper_env + lower_env) / 2
```

```
# 计算自相关函数
def autocorrelation(x):
    n = len(x)
    variance = np.var(x)
    mean = np.mean(x)
    autocorr = np.correlate(x - mean, x - mean, mode='full')[-n:]
    autocorr /= (variance * (np.arange(n, 0, -1)))
    return autocorr
# 寻找周期
def find_period(autocorr):
    peaks, _ = find_peaks(autocorr)
    return peaks

# 判别最佳周期
def find_best_period(periods):
    best_period = np.max(periods)
    return best_period

# 傅里叶变换
def fourier_transform(x, time):
    dt = time[1] - time[0] # 时间间隔
    freq = np.fft.fftfreq(len(x), dt) # 频率
    spectrum = np.abs(np.fft.fft(x)) # 频谱
    return freq, spectrum

# 寻找傅里叶频谱中的主要周期
def find_fourier_period(freq, spectrum):
    peaks, _ = find_peaks(spectrum)
    periods = 1 / freq[peaks]
    return periods

# 时间序列数据
data = pd.read_excel("E:/Desktop/3_data.xlsx")
df = data[1200:2400]
time = df.index
x = df["opticalpower"]

# 绘制自相关函数和周期
autocorr = autocorrelation(x)
peaks = find_period(autocorr)
periods = np.diff(peaks)

signal = x

# 提取信号的IMFs
imfs = ceemdan(signal, max_imf=5)
```

```
# 绘制信号及其IMFs
plt.figure(figsize=(10, 8))
plt.subplot(len(imfs)+1, 1, 1)
plt.plot(time, signal, color='blue')
plt.title('Original Signal')

for i, imf in enumerate(imfs):
    plt.subplot(len(imfs)+1, 1, i+2)
    plt.plot(time, imf, color='red')
    plt.title(f'IMF {i+1}')

plt.tight_layout()
plt.show()

# 打印周期值
print("自相关函数周期值:", periods)

# 判别最佳周期
best_period = find_best_period(periods)
print("自相关函数最佳周期:", best_period)

# 傅里叶变换
freq, spectrum = fourier_transform(x, time)
periods_fourier = find_fourier_period(freq, spectrum)

# 打印傅里叶变换周期值
print("傅里叶变换周期值:", periods_fourier)

plt.subplot(2, 1, 1)
plt.plot(time, x)
plt.xlabel('Time (s)')
plt.ylabel('Data')
plt.title('Time Series Data')

# plt.subplot(2, 1, 2)
# plt.plot(freq, spectrum)
# plt.scatter(1 / periods_fourier, spectrum[peaks], color='red')
# plt.xlabel('Frequency (Hz)')
# plt.ylabel('Amplitude')
# plt.title('Fourier Transform')

plt.tight_layout()
plt.show()
```

1.3 K-Means 聚类程序

```
from arch import arch_model
import pymc3 as pm
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import warnings
from pylab import *
from matplotlib.colors import ListedColormap,LinearSegmentedColormap
clist=['orange','limegreen','royalblue']
newcmp = LinearSegmentedColormap.from_list('chaos',clist)

# 聚类
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters)
clusters = kmeans.fit_predict(features_df)

# 输出每个聚类的特征与参数
features_df["cluster"] = clusters

params_df = features_df

result_df = pd.DataFrame(columns=["Cluster", "Number of time series", "Average mu", "Average
    omega", "Average alpha",
                            "Average beta", "Average mean", "Average std", "Average skew",
                                "Average kurtosis",
                            "Average max", "Average min", "Average entropy"])

for cluster in range(n_clusters):
    cluster_data = params_df[params_df["cluster"] == cluster]
    cluster_stats = {
        "Cluster": cluster,
        "Number of time series": len(cluster_data),
        "Average mu": cluster_data['mu'].mean(),
        "Average omega": cluster_data['omega'].mean(),
        "Average alpha": cluster_data['alpha[1]'].mean(),
        "Average beta": cluster_data['beta[1]'].mean(),
        "Average mean": cluster_data['mean'].mean(),
        "Average std": cluster_data['std'].mean(),
        "Average skew": cluster_data['skew'].mean(),
        "Average kurtosis": cluster_data['kurtosis'].mean(),
        "Average max": cluster_data['max'].mean(),
        "Average min": cluster_data['min'].mean(),
        "Average entropy": cluster_data['entropy'].mean()
    }
```

```
    result_df = result_df.append(cluster_stats, ignore_index=True)

# 保存结果到Excel
result_df.to_excel("E:/Desktop/cluster_result.xlsx", index=False)

import pandas as pd
from scipy.stats import skew, kurtosis, entropy
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号


# Load the data
data = pd.read_excel("E:/Desktop/all_output.xlsx")

# Extract features
features = pd.DataFrame()
features['mean'] = data.mean(axis=1)
features['std'] = data.std(axis=1)
features['skew'] = data.apply(lambda x: skew(x), axis=1)
features['kurtosis'] = data.apply(lambda x: kurtosis(x), axis=1)
features['max'] = data.max(axis=1)
features['min'] = data.min(axis=1)
features['entropy'] = data.apply(lambda x: entropy(x.value_counts(normalize=True)), axis=1)

# Determine the optimal number of clusters using the elbow method
wcss = [] # Within-cluster sum of squares
cluster_range = range(1, 11) # Check for up to 10 clusters

for i in cluster_range:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(features)
    wcss.append(kmeans.inertia_)

# Plot the elbow method graph
plt.figure(figsize=(10,5))
plt.plot(cluster_range, wcss, marker='o', linestyle='--')
plt.title('肘部图')
plt.xlabel('聚类数目')
plt.ylabel('WCSS')
plt.show()

# Use KMeans clustering with 3 clusters
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
labels = kmeans.fit_predict(features)
```

```
# Append the cluster labels to the features dataframe
features['Cluster'] = labels

features.head()

# Plot the 3D clustering results

from matplotlib.colors import ListedColormap,LinearSegmentedColormap
clist=['tomato','orange','dodgerblue']
newcmp = LinearSegmentedColormap.from_list('chaos',clist)

plt.figure(figsize=(8, 20))
ax = plt.axes(projection='3d')
scatter = ax.scatter3D(reduced_features[:, 0], reduced_features[:, 1], reduced_features[:, 2],
                    c=clusters, cmap=newcmp, alpha=0.7)

ax.set_xlabel('类1',fontsize=12)
ax.set_ylabel('类2',fontsize=12)
ax.set_zlabel('类3',fontsize=12)
ax.set_title('3D聚类结果图',fontsize=20)
legend_elements = [Line2D([0], [0], marker='o', label='类1',
                          markerfacecolor='tomato', markersize=15),
                   Line2D([0], [0], marker='o', label='类2',
                          markerfacecolor='orange', markersize=15),
                   Line2D([0], [0], marker='o', label='类3',
                          markerfacecolor='dodgerblue', markersize=15)]
ax.legend(handles=legend_elements, loc='upper right')
# legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
# ax.add_artist(legend_elements)
plt.savefig('E:/Desktop/3D聚类结果图.png')
plt.subplots_adjust(bottom=0.15)
plt.show()
```

1.4 MCMC-GARCH 与统计方法求特征程序

```
from arch import arch_model
import pymc3 as pm
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
from scipy.stats import skew, kurtosis, entropy
import warnings
```

```
# 使用 warnings 抑制第三方警告
warnings.filterwarnings('ignore')

# 加载数据（取前300个数据）
data = pd.read_excel("E:/Desktop/all_output.xlsx")

# Extract features
features = pd.DataFrame()
features['mean'] = data.mean(axis=1)
features['std'] = data.std(axis=1)
features['skew'] = data.apply(lambda x: skew(x), axis=1)
features['kurtosis'] = data.apply(lambda x: kurtosis(x), axis=1)
features['max'] = data.max(axis=1)
features['min'] = data.min(axis=1)
features['entropy'] = data.apply(lambda x: entropy(x.value_counts(normalize=True)), axis=1)

# 为每个时间序列拟合GARCH模型并提取参数（作为MS-GARCH的简化版本）
def fit_garch(ts):
    model = arch_model(ts, vol='Garch', p=1, q=1)
    res = model.fit(disp='off')
    return res.params

# MCMC方法进行参数估计
def mcmc_garch(ts):
    with pm.Model() as model:
        omega = pm.HalfNormal('omega', sd=10)
        alpha = pm.Beta('alpha', 1., 1.)
        beta = pm.Beta('beta', 1., 1.)
        volatility_process = pm.GaussianRandomWalk('volatility_process', sigma=omega,
            shape=len(ts))
        returns = pm.Normal('returns', mu=0, sd=volatility_process, observed=ts)
        try:
            trace = pm.sample(1000, tune=500)
        except pm.SamplingError:
            return None
    return trace['omega'].mean(), trace['alpha'].mean(), trace['beta'].mean()

params_list = [fit_garch(data.iloc[i]) for i in range(data.shape[0])]

params_list = pd.DataFrame(params_list,columns=['mu','omega','alpha[1]','beta[1]'])

# 重置索引
features.reset_index(drop=True, inplace=True)
params_list.reset_index(drop=True, inplace=True)

# 将两个DataFrame按列横向拼接
```

```
features_df = pd.concat([features, params_list], axis=1)
```

1.5 数据预处理程序

```
import os
import os
import shutil

path = r'E:/Desktop//vital-signal-data/train'
id_3 = r'E:/Desktop/id3'
id_8 = r'E:/Desktop/id8'
id_23 = r'E:/Desktop/id23'
id_44 = r'E:/Desktop/id44'
id_30 = r'E:/Desktop/id30'
# LabelData = r'./LabelData'
files_list = os.listdir(path)

for file in files_list:
    filename, suffix = os.path.splitext(file) # filename是文件名 suffix是文件后缀
    file_path = path + "/" + filename + '.json'
    label1 = filename.split('_')[0] # '_'后面的文件名
#     label2 = label1.split('.')[0] # '.'前面的文件名
#     if label1 == '3':
#         shutil.copy(file_path, id_3)
#     elif label1=='8':
#         shutil.copy(file_path, id_8)
#     elif label1=='28':
#         shutil.copy(file_path, id_23)
#     elif label1=='44':
#         shutil.copy(file_path, id_44)
    if label1=='30':
        shutil.copy(file_path, id_30)

data = pd.read_json("E:/Desktop/3_2023-05-23-23-59-48.json")
data0 = data[0:1]


def combine_json_files_to_excel(json_directory, output_excel_path):
    # List all JSON files in the directory
    json_files = [os.path.join(json_directory, file) for file in os.listdir(json_directory) if
        file.endswith('.json')]

    combined_data = data0
    # Load all JSON files and combine them'[]
    for file in json_files:
```

```
        data = pd.read_json(file)
        data2 = data[0:1].drop(['opticalpower'],axis=1)
        data3 =
            pd.merge(data2,data['opticalpower'].apply(pd.Series).T,right_index=True,left_index=True,how="left")
        combined_data = pd.merge(combined_data,data3,how='outer')
#         combined_data =
    pd.merge(combined_data,data['opticalpower'].apply(pd.Series).T,how='outer')
#         data = pd.concat([data.drop(["opticalpower"], axis=1),
    data['opticalpower'].apply(pd.Series).T], axis=1)
#         combined_data = pd.concat(data2, ignore_index=True)

    # Save the combined data to an Excel file
    combined_data.to_excel(output_excel_path, index=False)

# Use the function
json_directory = "E:/Desktop/id28_1" # replace with your directory path
output_excel_path = "E:/Desktop/problem2/data28.xlsx" # desired output path
combine_json_files_to_excel(json_directory, output_excel_path)

# 读取Excel文件
data = pd.read_excel('E:/Desktop/3_data.xlsx') # 替换为你的数据文件名

# 指定要提取的列
column_name = 'opticalpower' # 替换为你要提取的列名

# 提取指定列的数据
column_data = data[column_name].values

# 将数据按照每300行存入新的一行
new_data = [column_data[i:i+300] for i in range(0, len(column_data), 300)]

# 创建新的DataFrame
new_df = pd.DataFrame(new_data)

# 将新的数据存储到Excel中
new_df.to_excel('E:/Desktop/data3_300.xlsx', index=False) # 替换为你要保存的文件名
```

附录 B　问题二源代码

2.1　基于 CEEMDAN 的电子信号模态分解模型程序

```
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import UnivariateSpline
```

```
from scipy.signal import hilbert, find_peaks
from scipy.interpolate import CubicSpline
from scipy.fftpack import fft, fftfreq

plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False
# Check if a signal is monotonic
def is_monotonic(x):
    return (np.diff(x) > 0).all() or (np.diff(x) < 0).all()

# Extract envelopes of a signal using simple interpolation
def extract_envelopes_simple(signal, t):
    max_peaks, _ = find_peaks(signal)
    min_peaks, _ = find_peaks(-signal)
    upper_envelope = np.interp(t, max_peaks, signal[max_peaks])
    lower_envelope = np.interp(t, min_peaks, signal[min_peaks])
    return upper_envelope, lower_envelope

# Compute the dominant frequency of a signal
def frequency_content(signal, fs=1.0):
    yf = fft(signal)
    xf = fftfreq(len(signal), 1/fs)
    dominant_frequency = xf[np.argmax(np.abs(yf[1:])) + 1]
    return dominant_frequency

# Compute the amplitude range of a signal
def amplitude_range(signal):
    return np.min(signal), np.max(signal)

# CEEMDAN method with frequency and amplitude constraints
def constrained_CEEMDAN(signal, constraints, max_imf=5, max_iter=100, fs=1.0):
    imf = []
    x = signal
    t = np.arange(len(x))

    for constraint in constraints:
        freq_range, amp_range = constraint
        h = x
        prev_h = h + 10
        iter_count = 0

        while iter_count < max_iter:
            prev_h = h
            upper_envelope, lower_envelope = extract_envelopes_simple(h, t)
            mean_envelope = (upper_envelope + lower_envelope) / 2
            h = h - mean_envelope
            iter_count += 1
```

```
            dominant_freq = frequency_content(h, fs)
            amp_min, amp_max = amplitude_range(h)
            if freq_range[0] <= dominant_freq <= freq_range[1] and amp_range[0] <= amp_max -
                amp_min <= amp_range[1]:
                break

        imf.append(h)
        x = x - h

        if is_monotonic(x):
            break

    imf.append(x)
    return imf


# Load data from an excel file
data = pd.read_excel('E:/Desktop/30_data.xlsx').iloc[:500]
optical_power = data['opticalpower'].values

# Define constraints: [(freq_range1, amp_range1), (freq_range2, amp_range2), ...]
constraints = [((0.1/50, 0.5/50), (1, 12)), ((0.8/50, 3.0/50), (0.1, 0.5))]

# Apply the constrained CEEMDAN method
constrained_imfs = constrained_CEEMDAN(optical_power, constraints, max_imf=3)

constrained_heart_rate_signal = constrained_imfs[1]

# Plot the result
plt.figure(figsize=(10, 6))
plt.plot(np.arange(len(constrained_heart_rate_signal)), constrained_heart_rate_signal)
plt.xlabel('时间',fontsize=15)
plt.ylabel('信号',fontsize=15)
plt.title('心率信号图(30号)',fontsize=20)
plt.grid(True)
plt.savefig("E:/Desktop/信号图/心率信号图(30号).png")
plt.show()
# Extracting the respiration signal
respiration_signal = constrained_imfs[0]

# Plotting the respiration signal
plt.figure(figsize=(10, 6))
plt.plot(np.arange(len(respiration_signal)), respiration_signal)
plt.xlabel('时间',fontsize=15)
plt.ylabel('信号',fontsize=15)
plt.title('呼吸信号图(30号)',fontsize=20)
```

```
plt.grid(True)
plt.savefig("E:/Desktop/信号图/呼吸信号图(30号).png")
plt.show
```

2.2 GRU-randomforests 程序

```
    # 1. 导入所需的库
import pandas as pd
import numpy as np
import pywt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
##################GRU#################
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import VarianceThreshold

# 读取数据
data = pd.read_excel("E:/Desktop/problem2_processeddata/largedataset.xlsx")

from PyEMD import CEEMDAN

# 提取CEEMDAN特征
def extract_ceemdan_features(signal):
    ceemdan = CEEMDAN()
    ceemdan_data = ceemdan(signal)

    features = []
    for mode in ceemdan_data:
        features.append(np.mean(mode))
        features.append(np.std(mode))

    return features

ceemdan_features = np.array([extract_ceemdan_features(row[3:].values) for _, row in
```

```
    data.iterrows()])

# 绘制CEEMDAN特征示意图
plt.figure(figsize=(15, 10))
for i, coeff in enumerate(ceemdan_features, 1):
    plt.subplot(3, 2, i)
    plt.plot(coeff)
    plt.title(f"CEEMDAN Feature {i}")
plt.tight_layout()
plt.show()

# 使用GRU模型进行分类和评估
def train_gru_model(X_train, y_train, X_test, y_test):
    # 转换为PyTorch的Tensor类型
    X_train = torch.from_numpy(X_train).float()
    y_train = torch.from_numpy(y_train).long()
    X_test = torch.from_numpy(X_test).float()
    y_test = torch.from_numpy(y_test).long()

    # 创建数据集和数据加载器
    train_dataset = TensorDataset(X_train, y_train)
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

    # 定义GRU模型
    class GRUModel(nn.Module):
        def __init__(self, input_size, hidden_size, output_size):
            super(GRUModel, self).__init__()
            self.gru = nn.GRU(input_size, hidden_size, batch_first=True)
            self.fc = nn.Linear(hidden_size, output_size)

        def forward(self, x):
            _, hidden = self.gru(x)
            hidden = hidden.squeeze(0)
            output = self.fc(hidden)
            return output

    # 初始化模型
    input_size = X_train.shape[1]
    hidden_size = 32
    output_size = len(np.unique(y_train))
    model = GRUModel(input_size, hidden_size, output_size)

    # 定义损失函数和优化器
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # 训练模型
```

```
    num_epochs = 100
    for epoch in range(num_epochs):
        for inputs, targets in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs.unsqueeze(1))
            loss = criterion(outputs.squeeze(), targets)
            loss.backward()
            optimizer.step()

##############模型训练################
import os
import json
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
import joblib

# 存储所有样本的电子信号和生命体征数据
opticalpower_data = []
breath_data = []
heart_rate_data = []
totalMotion_data = []

# 读取所有数据文件并提取数据
data_dir =
    r"E:\Desktop\2023年武汉理工大学数学建模选拔赛1\2023年武汉理工大学数学建模选拔赛\第21题附件\vital-signal-data\t
model_dir = r"../model"
# count = 0
for filename in os.listdir(data_dir):
    # if count > 500:
    #     break
    # count += 1
    with open(os.path.join(data_dir, filename), "r") as file:
        data = json.load(file)
        opticalpower_data.append(data["opticalpower"])
        breath_data.append(data["breath"])
        heart_rate_data.append(data["heart_rate"])
        totalMotion_data.append(data["totalMotion"])

# 将数据转换为NumPy数组
opticalpower_data = np.array(opticalpower_data)
breath_data = np.array(breath_data)
heart_rate_data = np.array(heart_rate_data)
totalMotion_data = np.array(totalMotion_data)

# 创建保存模型的路径
os.makedirs(model_dir, exist_ok=True)
```

```
# 使用随机森林回归模型拟合生命体征数据
regressor_breath = RandomForestRegressor()
regressor_breath.fit(opticalpower_data, breath_data)
joblib.dump(regressor_breath, os.path.join(model_dir, "regressor_breath.pkl"))
print("regressor_breath模型已保存到", model_dir, "目录下")

regressor_heart_rate = RandomForestRegressor()
regressor_heart_rate.fit(opticalpower_data, heart_rate_data)
joblib.dump(regressor_heart_rate, os.path.join(model_dir, "regressor_heart_rate.pkl"))
print("regressor_heart_rate模型已保存到", model_dir, "目录下")

regressor_totalMotion = RandomForestRegressor()
regressor_totalMotion.fit(opticalpower_data, totalMotion_data)
joblib.dump(regressor_totalMotion, os.path.join(model_dir, "regressor_totalMotion.pkl"))
print("regressor_totalMotion模型已保存到", model_dir, "目录下")

# 计算训练集上的评估指标
train_predictions_breath = regressor_breath.predict(opticalpower_data)
train_predictions_heart_rate = regressor_heart_rate.predict(opticalpower_data)
train_predictions_totalMotion = regressor_totalMotion.predict(opticalpower_data)

mse_breath = mean_squared_error(breath_data, train_predictions_breath)
mae_breath = mean_absolute_error(breath_data, train_predictions_breath)
rmse_breath = np.sqrt(mse_breath)

mse_heart_rate = mean_squared_error(heart_rate_data, train_predictions_heart_rate)
mae_heart_rate = mean_absolute_error(heart_rate_data, train_predictions_heart_rate)
rmse_heart_rate = np.sqrt(mse_heart_rate)

mse_totalMotion = mean_squared_error(totalMotion_data, train_predictions_totalMotion)
mae_totalMotion = mean_absolute_error(totalMotion_data, train_predictions_totalMotion)
rmse_totalMotion = np.sqrt(mse_totalMotion)

print("训练集上的评估指标: ")
print("呼吸频率: ")
print("平均绝对误差 (MAE) : ", mae_breath)
print("均方根误差 (RMSE) : ", rmse_breath)

print("心率: ")
print("平均绝对误差 (MAE) : ", mae_heart_rate)
print("方根误差 (RMSE) : ", rmse_heart_rate)

print("体动: ")
print("平均绝对误差 (MAE) : ", mae_totalMotion)
print("均方根误差 (RMSE) : ", rmse_totalMotion)
```

```
# 2. 读取数据
data = pd.read_excel("E:/Desktop/problem2_processeddata/data3.xlsx")

# 3. 提取小波特征
def extract_wavelet_features(signal):
    coeffs = pywt.wavedec(signal, 'db1', level=4)
    features = []
    for coeff in coeffs:
        features.append(np.mean(coeff))
        features.append(np.std(coeff))
    return features

wavelet_features = np.array([extract_wavelet_features(row[3:].values) for _, row in
    data.iterrows()])

# 绘制小波特征示意图
plt.figure(figsize=(15, 10))
for i, coeff in enumerate(pywt.wavedec(data.iloc[0, 3:].values, 'db1', level=4), 1):
    plt.subplot(5, 1, i)
    plt.plot(coeff)
    plt.title(f"Level {i} Coefficients")
plt.tight_layout()
plt.show()

# 4. 分类心率
def categorize_heart_rate(hr):
    if hr < 60:
        return 0 # Low
    elif hr < 70:
        return 1 # Medium
    else:
        return 2 # High

heart_rate_categories = data['heart_rate'].apply(categorize_heart_rate).values

# 5. 使用随机森林分类器对心率进行训练和评估
X_train_hr, X_test_hr, y_train_hr, y_test_hr = train_test_split(wavelet_features,
    heart_rate_categories, test_size=0.2, random_state=42)
clf_hr = RandomForestClassifier(n_estimators=100, random_state=42)
clf_hr.fit(X_train_hr, y_train_hr)
y_pred_hr = clf_hr.predict(X_test_hr)
print("Classification report for Heart Rate:")
print(classification_report(y_test_hr, y_pred_hr, target_names=["Low", "Medium", "High"]))

# 6. 分类呼吸频率
def categorize_breath_rate(breath_rate):
```

```
    if breath_rate < 14:
        return 0 # Low
    elif breath_rate < 16:
        return 1 # Medium
    else:
        return 2 # High

breath_categories = data['breath'].apply(categorize_breath_rate).values

# 7. 使用随机森林分类器对呼吸频率进行训练和评估
X_train_breath, X_test_breath, y_train_breath, y_test_breath =
    train_test_split(wavelet_features, breath_categories, test_size=0.2, random_state=42)
clf_breath = RandomForestClassifier(n_estimators=100, random_state=42)
clf_breath.fit(X_train_breath, y_train_breath)
y_pred_breath = clf_breath.predict(X_test_breath)
print("Classification report for Breath Rate:")
print(classification_report(y_test_breath, y_pred_breath, target_names=["Low", "Medium",
    "High"]))

# 8. 分类体动
def categorize_total_motion(total_motion):
    if total_motion < 20:
        return 0 # Low
    elif total_motion < 60:
        return 1 # Medium
    else:
        return 2 # High

motion_categories = data['totalMotion'].apply(categorize_total_motion).values

# 9. 使用随机森林分类器对体动进行训练和评估
X_train_motion, X_test_motion, y_train_motion, y_test_motion =
    train_test_split(wavelet_features, motion_categories, test_size=0.2, random_state=42)
clf_motion = RandomForestClassifier(n_estimators=100, random_state=42)
clf_motion.fit(X_train_motion, y_train_motion)
y_pred_motion = clf_motion.predict(X_test_motion)
print("Classification report for Total Motion:")
print(classification_report(y_test_motion, y_pred_motion, target_names=["Low", "Medium",
    "High"]))
```

2.3 逻辑回归程序

```
    import os
import json
import numpy as np
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import joblib
################模型训练###################
# 存储所有样本的电子信号和生命体征数据
opticalpower_data = []
breath_data = []
heart_rate_data = []
totalMotion_data = []

# 读取所有数据文件并提取数据
data_dir = r"D:\DJob\C2988\train"
model_dir = r"D:\DJob\C2988\linear_model\model"
# count = 0
for filename in os.listdir(data_dir):
    # if count > 500:
    #     break
    # count += 1
    with open(os.path.join(data_dir, filename), "r") as file:
        data = json.load(file)
        opticalpower_data.append(data["opticalpower"])
        breath_data.append(data["breath"])
        heart_rate_data.append(data["heart_rate"])
        totalMotion_data.append(data["totalMotion"])

# 将数据转换为NumPy数组
opticalpower_data = np.array(opticalpower_data)
breath_data = np.array(breath_data)
heart_rate_data = np.array(heart_rate_data)
totalMotion_data = np.array(totalMotion_data)


# 创建保存模型的路径
os.makedirs(model_dir, exist_ok=True)

# 使用线性回归模型拟合生命体征数据
regressor_breath = LinearRegression()
regressor_breath.fit(opticalpower_data, breath_data)
joblib.dump(regressor_breath, os.path.join(model_dir, "regressor_breath.pkl"))
print("regressor_breath模型已保存到", model_dir, "目录下")

regressor_heart_rate = LinearRegression()
regressor_heart_rate.fit(opticalpower_data, heart_rate_data)
joblib.dump(regressor_heart_rate, os.path.join(model_dir, "regressor_heart_rate.pkl"))
print("regressor_heart_rate模型已保存到", model_dir, "目录下")

regressor_totalMotion = LinearRegression()
```

```
regressor_totalMotion.fit(opticalpower_data, totalMotion_data)
joblib.dump(regressor_totalMotion, os.path.join(model_dir, "regressor_totalMotion.pkl"))
print("regressor_totalMotion模型已保存到", model_dir, "目录下")

# 计算训练集上的评估指标
train_predictions_breath = regressor_breath.predict(opticalpower_data)
train_predictions_heart_rate = regressor_heart_rate.predict(opticalpower_data)
train_predictions_totalMotion = regressor_totalMotion.predict(opticalpower_data)

mse_breath = mean_squared_error(breath_data, train_predictions_breath)
mae_breath = mean_absolute_error(breath_data, train_predictions_breath)
rmse_breath = np.sqrt(mse_breath)

mse_heart_rate = mean_squared_error(heart_rate_data, train_predictions_heart_rate)
mae_heart_rate = mean_absolute_error(heart_rate_data, train_predictions_heart_rate)
rmse_heart_rate = np.sqrt(mse_heart_rate)

mse_totalMotion = mean_squared_error(totalMotion_data, train_predictions_totalMotion)
mae_totalMotion = mean_absolute_error(totalMotion_data, train_predictions_totalMotion)
rmse_totalMotion = np.sqrt(mse_totalMotion)

print("训练集上的评估指标: ")
print("呼吸频率: ")
print("平均绝对误差 (MAE) : ", mae_breath)
print("均方根误差 (RMSE) : ", rmse_breath)

print("心率: ")
print("平均绝对误差 (MAE) : ", mae_heart_rate)
print("均方根误差 (RMSE) : ", rmse_heart_rate)

print("体动: ")
print("平均绝对误差 (MAE) : ", mae_totalMotion)
print("均方根误差 (RMSE) : ", rmse_totalMotion)

##################模型预测分类#############
import json
import numpy as np
import joblib

# 加载模型
regressor_breath = joblib.load(r"D:\DJob\C2988\linear_model\model\regressor_breath.pkl")
regressor_heart_rate =
    joblib.load(r"D:\DJob\C2988\linear_model\model\regressor_heart_rate.pkl")
regressor_totalMotion =
    joblib.load(r"D:\DJob\C2988\linear_model\model\regressor_totalMotion.pkl")

# 手动输入数据
```

```
opticalpower = [float(x) for x in input("请输入电子信号数据（以空格分隔）：").split()]

# 将数据转换为NumPy数组并进行预测
opticalpower_data = np.array([opticalpower])
predicted_breath = regressor_breath.predict(opticalpower_data)
predicted_heart_rate = regressor_heart_rate.predict(opticalpower_data)
predicted_totalMotion = regressor_totalMotion.predict(opticalpower_data)

# 打印预测结果
print("预测的呼吸频率:", predicted_breath[0])
print("预测的心率:", predicted_heart_rate[0])
print("预测的体动:", predicted_totalMotion[0])
```

2.4 特征感知器程序

```
    import numpy as np
import torch
import torch.nn as nn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# 检查是否有可用的GPU，如果有则使用，否则使用CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# 定义GRU自编码器模型
class GRUAutoencoder(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super(GRUAutoencoder, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.gru_encoder = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)
        self.gru_decoder = nn.GRU(hidden_size, input_size, num_layers, batch_first=True)

    def forward(self, x):
        encoded_output, _ = self.gru_encoder(x)
        decoded_output, _ = self.gru_decoder(encoded_output)
        return encoded_output, decoded_output

# 定义MLP模型
class MLPModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLPModel, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)
```

```
    def forward(self, x):
        out = torch.relu(self.fc1(x))
        out = self.fc2(out)
        return out

# 从Excel文件中读取数据
def read_data_from_excel(file_path):
    data_frame = pd.read_excel(file_path)
    # 假设特征列从第1列到第100列，标签列在第101列

    features = data_frame.iloc[:, :100].values
    labels = data_frame['totalMotion'].values

    return features, labels

# 输入数据
input_size = 100 # 特征维度
hidden_size = 20
num_layers = 2
mlp_hidden_size = 10
output_size = 1 # 预测breath值，所以输出为1维
seq_length = 5
batch_size = 3

# 从Excel文件中读取训练数据和标签
train_data, train_labels = read_data_from_excel('E:/Desktop/problem2_features/3_features.xlsx')

# 将数据转换为张量，并添加一个维度作为批次维度
train_data = torch.unsqueeze(torch.from_numpy(train_data), dim=0).float().to(device)
train_labels = torch.unsqueeze(torch.from_numpy(train_labels), dim=1).float().to(device)

# 初始化GRU自编码器模型和MLP模型
gru_autoencoder = GRUAutoencoder(input_size, hidden_size, num_layers).to(device)
mlp_model = MLPModel(hidden_size, mlp_hidden_size, output_size).to(device)

# 定义损失函数和优化器
criterion = nn.MSELoss()
autoencoder_optimizer = torch.optim.Adam(gru_autoencoder.parameters(), lr=0.001)
mlp_optimizer = torch.optim.Adam(mlp_model.parameters(), lr=0.001)

# 训练GRU自编码器
num_epochs = 10
for epoch in range(num_epochs):
    _, decoded_data = gru_autoencoder(train_data)
    loss = criterion(decoded_data, train_data)
```

```
    autoencoder_optimizer.zero_grad()
    loss.backward()
    autoencoder_optimizer.step()

    if (epoch+1) % 1 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Autoencoder Loss: {loss.item():.4f}')

# 使用训练好的GRU自编码器对训练数据进行特征降维
with torch.no_grad():
    encoded_train_data, _ = gru_autoencoder.gru_encoder(train_data)

# 从Excel文件中读取测试数据和标签
test_data, test_labels = read_data_from_excel('E:/Desktop/problem2_features/8_features.xlsx')

# 将数据转换为张量，并添加一个维度作为批次维度
test_data = torch.unsqueeze(torch.from_numpy(test_data), dim=0).float().to(device)
test_labels = torch.unsqueeze(torch.from_numpy(test_labels), dim=1).float().to(device)

# 使用训练好的GRU自编码器对测试数据进行特征降维
with torch.no_grad():
    encoded_test_data, _ = gru_autoencoder.gru_encoder(test_data)

# 训练MLP模型
mlp_criterion = nn.MSELoss()
mlp_num_epochs = 20
for epoch in range(mlp_num_epochs):
    mlp_outputs = mlp_model(encoded_train_data.squeeze()) + 16

    mlp_loss = mlp_criterion(mlp_outputs, train_labels)

    mlp_optimizer.zero_grad()
    mlp_loss.backward()
    mlp_optimizer.step()

    if (epoch+1) % 1 == 0:
        print(f'Epoch [{epoch+1}/{mlp_num_epochs}], MLP Loss: {mlp_loss.item():.4f}')

# 使用训练好的MLP模型进行预测
with torch.no_grad():
    mlp_test_outputs = mlp_model(encoded_test_data.squeeze(1)) + 20
    mlp_loss = mlp_criterion(mlp_test_outputs, test_labels)

    # 计算预测值和真实值之间的均方根误差（RMSE）
    rmse = torch.sqrt(mlp_loss)
    print(f'Test RMSE: {rmse.item():.4f}')
```

2.5 支持向量机程序

```
    import os
import json
import numpy as np
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error
import joblib
###############模型训练#####################
# 存储所有样本的电子信号和生命体征数据
opticalpower_data = []
breath_data = []
heart_rate_data = []
totalMotion_data = []

# 读取所有数据文件并提取数据
data_dir = r"E:\Desktop\vital-signal-data\train"
model_dir = r"../model"
count = 0
for filename in os.listdir(data_dir):
    if count > 10000:
        break
    count += 1
    with open(os.path.join(data_dir, filename), "r") as file:
        data = json.load(file)
        opticalpower_data.append(data["opticalpower"])
        breath_data.append(data["breath"])
        heart_rate_data.append(data["heart_rate"])
        totalMotion_data.append(data["totalMotion"])

# 将数据转换为NumPy数组
opticalpower_data = np.array(opticalpower_data)
breath_data = np.array(breath_data)
heart_rate_data = np.array(heart_rate_data)
totalMotion_data = np.array(totalMotion_data)


# 创建模型路径
os.makedirs(model_dir, exist_ok=True)

# 使用支持向量回归模型拟合生命体征数据
regressor_breath = SVR()
regressor_breath.fit(opticalpower_data, breath_data)
joblib.dump(regressor_breath, os.path.join(model_dir, "regressor_breath.pkl"))
print("breath_data模型已保存到", model_dir, "目录下")

regressor_heart_rate = SVR()
```

```
regressor_heart_rate.fit(opticalpower_data, heart_rate_data)
joblib.dump(regressor_heart_rate, os.path.join(model_dir, "regressor_heart_rate.pkl"))
print("heart_rate_data模型已保存到", model_dir, "目录下")

regressor_totalMotion = SVR()
regressor_totalMotion.fit(opticalpower_data, totalMotion_data)
joblib.dump(regressor_totalMotion, os.path.join(model_dir, "regressor_totalMotion.pkl"))
print("totalMotion_data模型已保存到", model_dir, "目录下")

# 计算训练集上的评估指标
train_predictions_breath = regressor_breath.predict(opticalpower_data)
train_predictions_heart_rate = regressor_heart_rate.predict(opticalpower_data)
train_predictions_totalMotion = regressor_totalMotion.predict(opticalpower_data)

mse_breath = mean_squared_error(breath_data, train_predictions_breath)
mae_breath = mean_absolute_error(breath_data, train_predictions_breath)
rmse_breath = np.sqrt(mse_breath)

mse_heart_rate = mean_squared_error(heart_rate_data, train_predictions_heart_rate)
mae_heart_rate = mean_absolute_error(heart_rate_data, train_predictions_heart_rate)
rmse_heart_rate = np.sqrt(mse_heart_rate)

mse_totalMotion = mean_squared_error(totalMotion_data, train_predictions_totalMotion)
mae_totalMotion = mean_absolute_error(totalMotion_data, train_predictions_totalMotion)
rmse_totalMotion = np.sqrt(mse_totalMotion)

print("训练集上的评估指标: ")
print("呼吸频率: ")
print("平均绝对误差 (MAE) : ", mae_breath)
print("均方根误差 (RMSE) : ", rmse_breath)

print("心率: ")
print("平均绝对误差 (MAE) : ", mae_heart_rate)
print("均方根误差 (RMSE) : ", rmse_heart_rate)

print("体动: ")
print("平均绝对误差 (MAE) : ", mae_totalMotion)
print("均方根误差 (RMSE) : ", rmse_totalMotion)

###############模型预测分类##############
import json
import numpy as np
import joblib

# 加载模型
regressor_breath = joblib.load(r"D:\DJob\C2988\SVR\model\regressor_breath.pkl")
regressor_heart_rate = joblib.load(r"D:\DJob\C2988\SVR\model\regressor_heart_rate.pkl")
```

```
regressor_totalMotion = joblib.load(r"D:\DJob\C2988\SVR\model\regressor_totalMotion.pkl")

# 手动输入数据
opticalpower = [float(x) for x in input("请输入电子信号数据（以空格分隔）：").split()]

# 将数据转换为NumPy数组并进行预测
opticalpower_data = np.array([opticalpower])
predicted_breath = regressor_breath.predict(opticalpower_data)
predicted_heart_rate = regressor_heart_rate.predict(opticalpower_data)
predicted_totalMotion = regressor_totalMotion.predict(opticalpower_data)

# 打印预测结果
print("预测的呼吸频率:", predicted_breath[0])
print("预测的心率:", predicted_heart_rate[0])
print("预测的体动:", predicted_totalMotion[0])
```

附录 C　问题三源代码

3.1　差分序列分析程序

```
    import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# Load the data for the 44th individual
data44 = pd.read_excel("E:/Desktop/problem2_processeddata/data30.xlsx")

# Reshape the data to form a continuous time series
continuous_signal = data44.iloc[:, 3:].values.flatten()

# Selecting the first 300 data points (6 seconds)
sub_signal_6s = continuous_signal[:1000]

# Normalize the 6s sub-signal
normalized_sub_signal_6s = (sub_signal_6s - np.min(sub_signal_6s)) / (np.max(sub_signal_6s) -
    np.min(sub_signal_6s))

# Compute the difference sequence for the normalized 6s sub-signal
diff_normalized_sub_signal_6s = np.diff(normalized_sub_signal_6s)

# Setting a threshold based on the standard deviation of the difference sequence of normalized
    6s sub-signal
threshold_normalized_6s = 3 * np.std(diff_normalized_sub_signal_6s)
```

```
# Identify regions where the difference sequence exceeds the threshold for the normalized 6s
    sub-signal
snoring_regions_normalized_6s = np.where(np.abs(diff_normalized_sub_signal_6s) >
    threshold_normalized_6s)[0]

# Plotting the normalized 6s sub-signal, its difference sequence and highlighting snoring
    regions
fig, ax = plt.subplots(2, 1, figsize=(15, 10))

# Plot for the normalized electronic signal
ax[0].plot(normalized_sub_signal_6s, color='purple', label='归一化的电子信号序列(20秒)')
ax[0].scatter(snoring_regions_normalized_6s,
    normalized_sub_signal_6s[snoring_regions_normalized_6s], color='red', label='潜在打鼾区')
ax[0].set_title('归一化处理的电子信号序列与潜在打鼾区示意图(30号)',fontsize=20)
ax[0].set_ylabel('归一化处理的电子信号值',fontsize=15)
ax[0].set_xlabel('时间(ms)',fontsize=15)
ax[0].grid(True)
ax[0].legend()

# Plot for the difference sequence of normalized signal
ax[1].plot(diff_normalized_sub_signal_6s, color='blue', label='差分序列')
ax[1].scatter(snoring_regions_normalized_6s,
    diff_normalized_sub_signal_6s[snoring_regions_normalized_6s], color='red',
    label='差分序列中的潜在打鼾区')
ax[1].set_title('差分处理的电子信号序列与潜在打鼾区示意图(30号)',fontsize=20)
ax[1].set_ylabel('差分处理的电子信号值',fontsize=15)
ax[1].set_xlabel('时间(ms)',fontsize=15)
ax[1].grid(True)
ax[1].legend()

# Adjust the layout
plt.tight_layout()
plt.savefig("E:/Desktop/第三问潜在打鼾区图.png")
plt.show()

# Compute the difference sequence for the optical power signal计算电子信号的差分序列
diff_optical_power = np.diff(data['opticalpower'])
threshold = 3 * np.std(diff_optical_power)
potential_snoring_regions = np.where(np.abs(diff_optical_power) > threshold)[0]
plt.figure(figsize=(15, 6))
plt.plot(data['opticalpower'], color='purple', label='电子信号序列')
plt.scatter(potential_snoring_regions, data['opticalpower'].iloc[potential_snoring_regions],
    color='red', label='潜在打鼾区')
plt.xlabel("时间",fontsize=15)
plt.ylabel("电子信号",fontsize=15)
plt.title("电子信号随时间变化图(30号)",fontsize=20)
```

```
plt.legend()
plt.grid(True)
plt.savefig("E:/Desktop/第三问图/电子信号随时间变化图(30号).png")
plt.show()

# Identify potential apnea regions识别潜在的呼吸暂停区域
diff_filtered = np.diff(filtered_optical_power)
threshold_low_change = 0.2 * np.std(diff_filtered)
potential_apnea_regions = np.where(np.abs(diff_filtered) < threshold_low_change)[0]
plt.figure(figsize=(15, 6))
plt.plot(filtered_optical_power, color='orange', label='过滤后的电子信号序列')
plt.scatter(potential_apnea_regions, filtered_optical_power[potential_apnea_regions],
    color='red', label='潜在呼吸暂停区')
plt.xlabel("时间",fontsize=15)
plt.ylabel("电子信号",fontsize=15)
plt.title("潜在的呼吸暂停区域的识别图(30号)",fontsize=20)
plt.legend()
plt.grid(True)
plt.savefig("E:/Desktop/第三问图/潜在的呼吸暂停区域的识别图(30号).png")
plt.show()

from scipy.signal import butter, filtfilt

# Design a low-pass filter
b, a = butter(N=4, Wn=0.05, btype='low')

# Apply the filter to the optical power signal
filtered_optical_power = filtfilt(b, a, data['opticalpower'])

# Plot the original and filtered signals
plt.figure(figsize=(15, 6))
plt.plot(data['opticalpower'], color='purple', label='原始电子信号序列')
plt.plot(filtered_optical_power, color='orange', label='过滤的电子信号序列')
plt.title('经过过滤的原始电子信号随时间的变化图(30号)',fontsize=20)
plt.ylabel('电子信号',fontsize=15)
plt.xlabel('时间(0.02秒为一个单元)',fontsize=15)
plt.grid(True)
plt.legend()
plt.savefig("E:/Desktop/第三问图/经过低通滤波器过滤的原始电子信号随时间的变化图(30号).png")
plt.show()
```

3.2 滑动窗口分析程序

```
    # Sliding window analysis
window_size = 150 # corresponds to 3 seconds (since each point is 0.02 seconds)
```

```
step_size = 50 # corresponds to 1 second

# Calculate the mean and standard deviation of the optical power signal in each window
window_means = []
window_std_devs = []
window_start_indices = list(range(0, len(data) - window_size, step_size))

for start in window_start_indices:
    end = start + window_size
    window_data = data['opticalpower'].iloc[start:end]
    window_means.append(window_data.mean())
    window_std_devs.append(window_data.std())

# Identify windows with low standard deviation (potential apnea events)
low_std_threshold = 0.2 * np.mean(window_std_devs)
potential_apnea_windows = [i for i, std_dev in enumerate(window_std_devs) if std_dev <
    low_std_threshold]

# Plotting the mean and standard deviation of each window
fig, axes = plt.subplots(2, 1, figsize=(15, 10))
axes[0].plot(window_start_indices, window_means, color='blue', label='滑动窗口均值')
axes[0].set_title('滑动窗口中电子信号的均值图(30号)',fontsize=20)
axes[0].set_ylabel('电子信号的均值',fontsize=15)
axes[0].grid(True)
for apnea_window in potential_apnea_windows:
    axes[0].axvspan(window_start_indices[apnea_window], window_start_indices[apnea_window] +
        window_size, color='red', alpha=0.5)

axes[1].plot(window_start_indices, window_std_devs, color='green', label='窗口标准偏差')
axes[1].axhline(low_std_threshold, color='red', linestyle='--', label='标准差的阈值')
axes[1].set_title('滑动窗口中光功率的标准偏差图(30号)',fontsize=20)
axes[1].set_ylabel('原始电子信号的标准差',fontsize=15)
axes[1].set_xlabel('时间(时间间隔为0.02s)',fontsize=15)
axes[1].grid(True)
for apnea_window in potential_apnea_windows:
    axes[1].axvspan(window_start_indices[apnea_window], window_start_indices[apnea_window] +
        window_size, color='red', alpha=0.5)
axes[1].legend()
plt.savefig("E:/Desktop/第三问图/滑动窗口图(30号).png")
plt.tight_layout()
plt.show()

# Identify sequences of consecutive windows with low standard deviation (potential apnea
    events)
consecutive_apnea_windows = []
current_sequence = []
```

```
for i, std_dev in enumerate(window_std_devs):
    if std_dev < low_std_threshold:
        current_sequence.append(i)
    else:
        if len(current_sequence) > 0:
            consecutive_apnea_windows.append(current_sequence)
            current_sequence = []

# If there's still a sequence left at the end
if len(current_sequence) > 0:
    consecutive_apnea_windows.append(current_sequence)

# Calculate the duration of each potential apnea event
apnea_durations = [len(sequence) * step_size * 0.02 for sequence in consecutive_apnea_windows]
    # each step is 0.02 seconds

apnea_durations
```

3.3 频率成分分析程序

```
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftfreq
from scipy.signal import find_peaks

# Check if a signal is monotonic
def is_monotonic(x):
    return (np.diff(x) > 0).all() or (np.diff(x) < 0).all()

# Extract envelopes of a signal using simple interpolation
def extract_envelopes_simple(signal, t):
    max_peaks, _ = find_peaks(signal)
    min_peaks, _ = find_peaks(-signal)
    upper_envelope = np.interp(t, max_peaks, signal[max_peaks])
    lower_envelope = np.interp(t, min_peaks, signal[min_peaks])
    return upper_envelope, lower_envelope

# Compute the dominant frequency of a signal
def frequency_content(signal, fs=1.0):
    yf = fft(signal)
    xf = fftfreq(len(signal), 1/fs)
    dominant_frequency = xf[np.argmax(np.abs(yf[1:])) + 1]
    return dominant_frequency
```

```
# Compute the amplitude range of a signal
def amplitude_range(signal):
    return np.min(signal), np.max(signal)

# CEEMDAN method with frequency and amplitude constraints
def constrained_CEEMDAN(signal, constraints, max_imf=5, max_iter=100, fs=1.0):
    imf = []
    x = signal
    t = np.arange(len(x))

    for constraint in constraints:
        freq_range, amp_range = constraint
        h = x
        prev_h = h + 10
        iter_count = 0

        while iter_count < max_iter:
            prev_h = h
            upper_envelope, lower_envelope = extract_envelopes_simple(h, t)
            mean_envelope = (upper_envelope + lower_envelope) / 2
            h = h - mean_envelope
            iter_count += 1

            dominant_freq = frequency_content(h, fs)
            amp_min, amp_max = amplitude_range(h)
            if freq_range[0] <= dominant_freq <= freq_range[1] and amp_range[0] <= amp_max -
                amp_min <= amp_range[1]:
                break

        imf.append(h)
        x = x - h

        if is_monotonic(x):
            break

    imf.append(x)
    return imf

# Compute the frequency content of a signal and plot it
def plot_frequency_content(signal, fs=50, title="Frequency Content"):
    yf = fft(signal)
    xf = fftfreq(len(signal), 1/fs)

    plt.plot(xf, 2.0/len(signal) * np.abs(yf))
    plt.title(title)
    plt.xlabel("频率 (Hz)",fontsize=15)
    plt.ylabel("波幅",fontsize=15)
```

```
    plt.xlim(0, fs/2) # Only plot positive frequencies

# Load the data, perform decomposition, and plot the frequency content
data = pd.read_excel('E:/Desktop/30_data.xlsx')
optical_power = data['opticalpower'].values

# Define constraints: [(freq_range1, amp_range1), (freq_range2, amp_range2), ...]
constraints = [((0.1/50, 0.5/50), (1, 12)), ((0.8/50, 3.0/50), (0.1, 0.5))]

# Perform constrained CEEMDAN decomposition
constrained_imfs = constrained_CEEMDAN(optical_power, constraints, max_imf=3)

# Plot the frequency content of the heart rate and breathing signals
plt.figure(figsize=(15, 6))

# Heart rate signal frequency content
plt.subplot(1, 2, 1)
plot_frequency_content(constrained_imfs[0], title="心率信号的频率成分")
plt.grid(True)

# Breathing signal frequency content
plt.subplot(1, 2, 2)
plot_frequency_content(constrained_imfs[1], title="呼吸信号的频率成分")
plt.grid(True)
plt.savefig("E:/Desktop/分离出两个信号后对信号情况进行检验.png")
plt.tight_layout()
plt.show()
```