中国研究生创新实践系列大赛
“华为杯”第十七届中国研究生数学建模竞赛

学　　校	南京航空航天大学
参赛队号	**20102870010**
队员姓名	**1.**骆毅
	2.许启强
	3.刘翔

中国研究生创新实践系列大赛
“华为杯”第十七届中国研究生
数学建模竞赛

题 目　面向康复工程的脑电信号分析与判别模型

摘　　要：

人类在进化过程中，大脑逐渐演变成构造最复杂，信息处理最完善的器官，其通过数以亿计的神经元协同合作来传递和处理信息，有效控制着人各种生理与心理活动。脑-机接口旨在人脑和计算机设备之间建立起控制通道，当大脑受到外界某种刺激而产生具有相应特征的生物信号时，对其解码处理，在不依赖正常的外围神经或肌肉组织等输出系统的情况下，以识别大脑意图。本文研究了脑-机接口技术在字符矩阵识别和睡眠分期预测中的应用，通过对事件 P300 脑电信号数据的预处理和特征提取，选择合适的分类模型和算法，减少了实验所需的观测数据，同时保证了目标分类准确率和一定水平的信息传输速率。

针对问题一，对于附件 1 的数据进行预处理，通过经验模态分解（EMD）算法和 FastICA 算法进行去噪声和去伪迹处理，得到较为干净的脑电信号数据集。采用了支持向量机（SVM）分类的识别方法。以特征向量为依据将已知字符的训练集数据分为正负标签进行训练，经测试集检验后，我们以表格形式给出了待测目标字符的结果和 *F1-score* 指标值。

针对问题二，采用了基于 2-范数下的共空间模式（CSP）通道优化模型。对于不同被试的脑电信号时空矩阵，基于 CSP 方法得到对应的投影矩阵，使用 2-范数的通道筛选准则，筛选出投影空间中权重之和大于等于 85%的若干个通道形成通道组合，分别为记为 L1-L5，通道个数依次为 13,11,14,12,12,；重复问题一的实验，发现相比于全部通道，每个被试着的分类指标 *F1-score* 都有一定程度的提升，故说明该方法筛选信号通道的有效性。同时，将筛选出的通道组合与被试者进行 5 折交叉实验，发现 L2 通道（包括通道 1,3,4,5,6,7,9,11,16,17,19）的平均 *F1-score* 为 0.7458 最高，故选取 L2 通道作为问题三的最优组合通道。

针对问题三，对于少量标签样本分类设计了一种基于 KShape 半监督学习的信号序列聚类方法。考虑到 P300 信号的尺度变换性和相位变换性，给出了一种 P300 信号相似度衡量方法。阐述了 P300 信号的重心提取过程，使用 Kshape 算法对正负时序样本进行聚类，得到最高 77.8%聚类准确率，验证了测试集中其余的待识别目标字符。

针对问题四，对所给数据集做了细致的预处理和特征提取，根据对数据的分析结果发现能表征不同睡眠分期的特征组合各有差异，因此采用了由 5 个二分类器组成的混合多分类模型，每个二分类器采用支持向量机模型，负责识别一个睡眠分期，在经过与 BP 神经网络多分类模型、随机森林(Random Forest, RF)的实验对比，验证了该混合模型对于问题四中数据集分类预测相比于其他模型具有更好的精确率，另外做了数据集划分比例实验，用以验证本模型在较少的数据用于训练模型的情况下依旧具有较高的准确率。

关键词：脑电信号，特征提取，支持向量机，共空间模式，Kshape 聚类，多分类

目录

1．问题重述

1.1 研究背景

大脑是人体中高级神经活动的中枢，拥有着数以亿计的神经元，并通过相互连接来传递和处理人体信息。脑电信号按其产生的方式可分为诱发脑电信号和自发脑电信号。诱发脑电信号是通过某种外界刺激使大脑产生电位变化从而形成的脑电活动；自发脑电信号是指在没有外界特殊刺激下，大脑自发产生的脑电活动。

（1）诱发脑电信号（P300 脑-机接口）

P300 事件相关电位是诱发脑电信号的一种，在小概率刺激发生后 300 毫秒范围左右出现的一个正向波峰（相对基线来说呈现向上趋势的波）。由于个体间的差异性，P300 的发生时间也有所不同，图 1.1 表示的是在刺激发生后 450 毫秒左右的 P300 波形。P300 电位作为一种内源性成分，它不受刺激物理特性影响，与知觉或认知心理活动有关，与注意、记忆、智能等加工过程密切相关。基于 P300 的脑-机接口优点是使用者无需通过复杂训练就可以获得较高的识别准确率，具有稳定的锁时性和高时间精度特性。

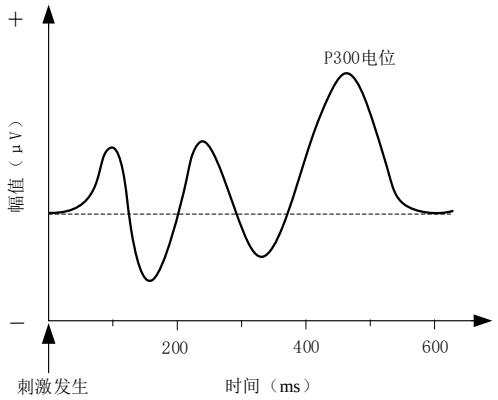
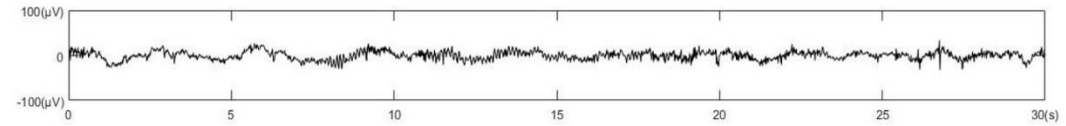


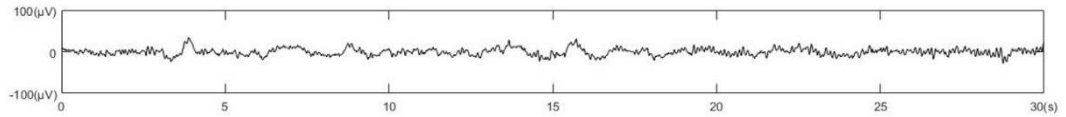
图 1.1 P300 波形示意图

（2）自发脑电信号（睡眠脑电）

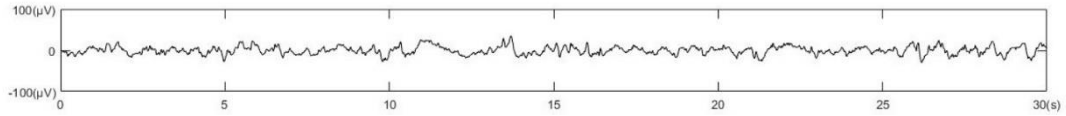
睡眠过程中采集的脑电信号，属于自发型的脑电信号。在国际睡眠分期的判读标准 R&K 中，对睡眠过程中的不同状态给出了划分：除去清醒期以外，睡眠周期是由两种睡眠状态交替循环，分别是非快速眼动期和快速眼动期；在非快速眼动期中，根据睡眠状态由浅入深的逐步变化，又进一步分为睡眠 I 期，睡眠 II 期，睡眠 III 期和睡眠 IV 期；睡眠 III 期和睡眠 IV 期又可合并为深睡眠期。图 2 给出了不同睡眠分期对应的脑电信号时序列，自上而下依次为清醒期、睡眠 I 期、睡眠 II 期、深睡眠和快速眼动期。从图 1.2 中可以观察到，脑电信号在不同睡眠分期所呈现的特点有所不同。基于脑电信号进行自动分期，能够减轻专家医师的人工负担，也是评估睡眠质量、诊断和治疗睡眠相关疾病的重要辅助工具。



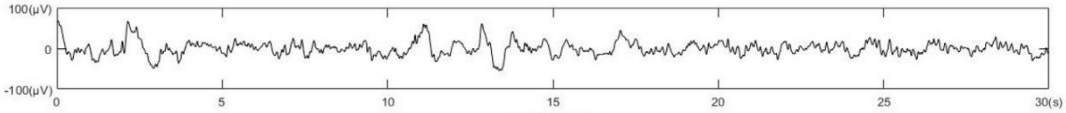
(a) 清醒期



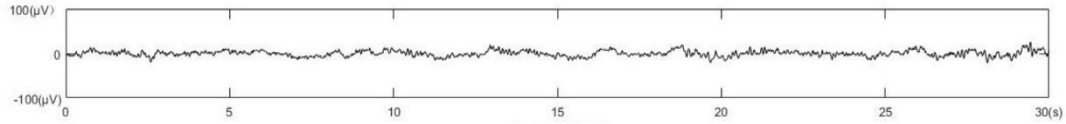
(b) 睡眠 I 期



(c) 睡眠 II 期



(d) 深睡眠期



(e) 快速眼动期

图 1.2　各睡眠分期的睡眠脑电信号时序列

1.2 实验描述

实验提供了 5 个健康成年被试（S1-S5）的 P300 脑-机接口实验数据，平均年龄为 20 岁。在实验的过程中，要求每一位被试（被测试者）集中注意力。P300 脑-机接口实验的设计如下：每位被试能够观察到一个由 36 个字符组成的字符矩阵，如图 1.3 所示，字符矩阵以行或列为单位（共 6 行 6 列）。每轮实验的设计流程：首先，提示被试注视“目标字符”，例如在图 3 的字符矩阵上方，出现的灰色字符“A”；其次，进入字符矩阵的闪烁模式，每次以随机的顺序闪烁字符矩阵的一行或一列，闪烁时长为 80 毫秒，间隔为 80 毫秒；最后，当所有行和列均闪烁一次后，则结束一轮实验。在被试注视“目标字符”的过程中，当目标字符所在行或列闪烁时，脑电信号中会出现 P300 电位；而当其他行和列闪烁时，则不会出现 P300 电位。上述实验流程为 1 轮，共重复 5 轮。

图 1.3 字符矩阵界面

1.3 已知数据信息

（1）附件 1：5 位被试的 P300 脑机接口实验数据。分别包括重复实验 5 次的 12 个已知目标字符的训练集与事件标签，10 个未知目标字符的测试集与事件标签；

（2）附件 2：取自不同健康成年人整夜睡眠过程的 3000 个睡眠脑电特征样本及其特征参数和标签。

1.4 问题提出

问题一：针对所给的测试数据，设计分类识别模型，找出附件 1 中被试的 10 个待识别目标，给出具体过程并说明所设计方法的合理性。

问题二：对于原始脑电信号数据中包含较多的冗余通道数据，设计每位被试最有利分类的通道组合筛选方法，要求通道组合数量大于等于 10 小于 20；并分析选择出一组每位被试都较为适用的最优通道名称组合。

问题三：基于问题二得到的最优通道组合，设计一种学习的方法，减少通道数据中的无效信号数据，缩短有标签样本的训练时间，用问题二中测试数据（char13-char17）检验方法的有效性，并识别其余待识别目标字符。

问题四：根据附件 2 中特征样本，设计一个睡眠分期预测模型，选择尽可能少的训练样本得到较高的预测准确率，给出训练数据和测试数据的选取方式和分配比例，对预测效果进行分析。

2．模型假设

（1）不考虑 P300 脑-机接口的测量误差

（2）假设被试注视“目标字符”的过程不存在延迟

3．符号说明

i	表示被试者
j	表示观测目标字符
dj	表示待观测目标字符
k	表示同一实验内的观测轮次
l	表示第l个观测通道
m	表示 EEG 信号的样本点索引
s	表示经过空间过滤器后的 EEG 信号
x	表示原始 EEG 信号
e	表示论文中所涉及的任意长度的脑电信号

E	表示脑电信号集合
P_{ijk}^{r}	表示第 i 个被试者在观测目标字符 j 时第 k 轮出现 P300 信号的行起始点索引
$P_{ijk}^{r'}$	表示第 i 个被试者在观测目标字符 j 时第 k 轮出现 P300 信号的行终止点索引
P_{ijk}^{c}	表示第 i 个被试者在观测目标字符 j 时第 k 轮出现 P300 信号的列起始点索引
$P_{ijk}^{c'}$	表示第 i 个被试者在观测目标字符 j 时第 k 轮出现 P300 信号的列终止点索引
$\tilde{P}$	表示 P300 信号
C^{*}	表示 P300 信号的重心序列
TP	真实类别为正确，预测为正例的样本个数
TN	真实类别为正确，预测为反例的样本个数
FP	真实类别为错误，预测为正例的样本个数

4. 问题一模型建立与求解

4.1 问题分析

根据脑-机接口的字符矩阵实验设计，被试者在实验开始前被提示关注指定目标字符后，进行随机闪烁一行或一列，只要在闪烁过程中闪烁包含目标字符的行列，脑电信号中就会出现 P300 电位的信号，同时由 P300 波形示意图可知，P300 信号是一小段波长的信号，并非一个时间点信号。

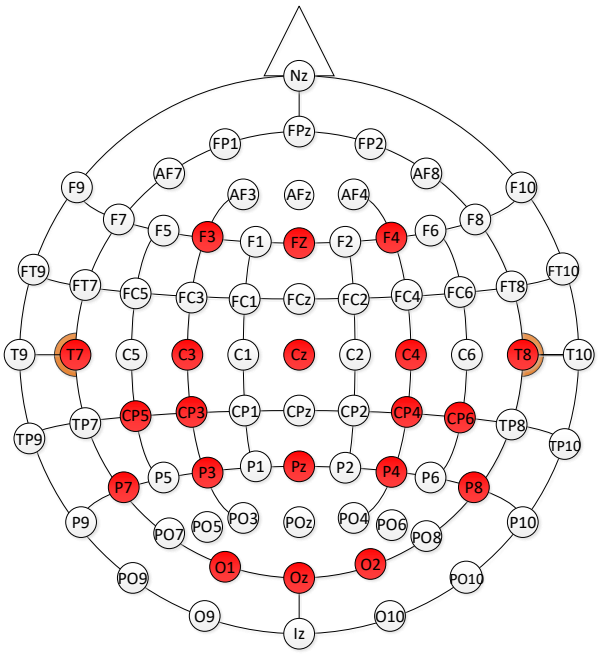


图 4.1 脑电信号采集通道图

标识符	通道名称	标识符	通道名称
1	Fz	**11**	CP5
2	F3	**12**	CP6
3	F4	**13**	Pz
4	Cz	**14**	P3
5	C3	**15**	P4
6	C4	**16**	P7
7	T7	**17**	P8
8	T8	**18**	Oz
9	CP3	**19**	O1
10	CP4	**20**	O2

表 4.1 采集通道的标识符

对附件 1 数据内容的分析：

（1）“train_data”文件中子表（char01-char12）分别代表已知目标字符名称，如图 4.2，其中数据表格包含 20 列，每列代表 1 个记录通道，依次进行编号，表 4.1 为记录通道的标识符，图 4 对应了记录通道的位置，脑电数据表格的每一行表示一个样本点数据，共 20 维，每一维表示一个通道的数据。

（2）“train_event”文件也有 12 个子表（char01-char12），每一个子表表示关注一个确定字符时实验记录的数据，在子表表格中，第 1 列代表随机闪烁行列的标签，如图 4.3 所示，首行数字代表目标字符的标识符，每轮实验以“100”标识符结束；第 2 列代表标签所对应的样本点序号，即对应”train_data”文件中相应字符子表中索引号为该样本点序号的一行电信号数据。

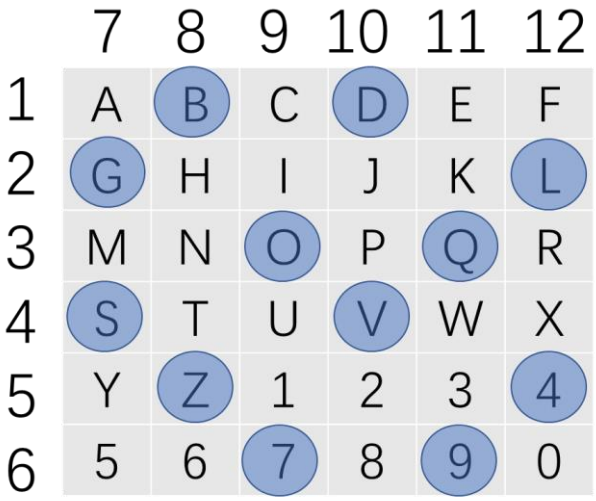


图 4.2 已知的目标字符

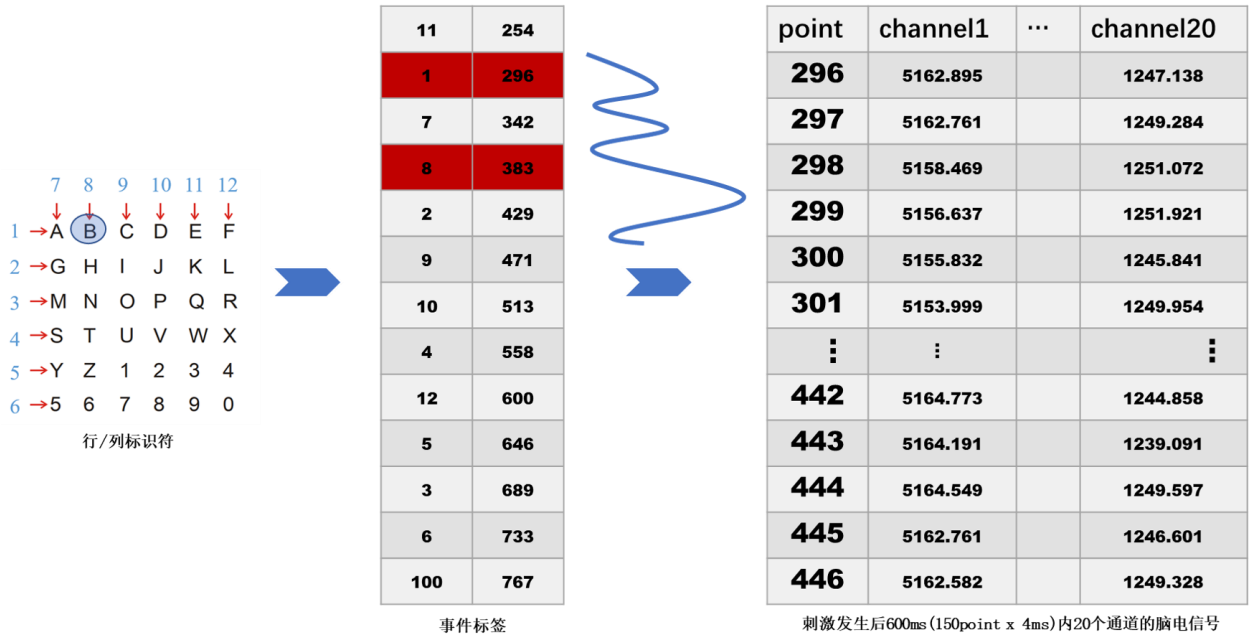


图 4.3 行/列的标识符

在第一问中，题目要求针对所给的测试数据，设计一个分类识别模型，利用这个分类

模型找出附件 1 中被试的 10 个待识别目标分别是什么字符，并给出具体过程并说明所设计方法的合理性。

从问题描述上看，第一问中是要解决一个分类问题，即，需要根据测试集中记录的被试者在看到行或列闪烁后脑电信号数据的变化情况，来判断本次实验被试者关注的目标字符的类别。根据实验描述，进一步分析发现，实际上需要进行分类的是普通电信号和 P300 电信号，因为当被试者关注的目标字符出现在闪烁行或列中时，才会出现 P300 电信号，闪烁行或列没有包含被关注字符时，则为普通信号，因此，当我们检测出此次实验中出现的 P300 电信号所对应的行或列标识符索引，我们就可以根据此索引查找图 6 中行或列标识符对应的字符，即为待识别目标字符。例如，在一个待识别的实际字符为‘B’的一轮实验中，当我们检测到 P300 电信号，查看”test_event”子表中此 P300 电信号起始索引对应的行或列标识符，假设为 1 和 8，此时再查看图 4.3 中索引 1 和 8 对应的字符，查表得出待识别字符为’B’，则问题一得以解决。

经过对问题定义的分析，我们发现最关键的步骤是如何设计一个高效的二分类识别模型，使得该模型能够对输入的一段时间间隔内的脑电信号数据进行识别，将其分类为普通信号和 P300 电信号，从而判断出待识别目标字符。

4.2 模型建立

4.2.1 数据预处理

脑电信号是强随机性、微幅值的非平稳信号，极易受到外界因素干扰，因此采集到的脑电信号中往往惨杂着各种各样的噪声干扰，如工频干扰、心电伪 迹及肢体运动产生的伪迹等，其信噪比低。预处理过程是脑电信号处理过程中的第一步，目的是降低夹杂在原始脑电信号中的噪声和伪迹干扰，使特征提取算法能够提取到反映大脑真实意图的特征。

图 4.4 为 S1 被试观测字符‘B’时 Cz 通道的脑电信号。红色信号为 P300 信号，选择方法在特征提取模块中给出。

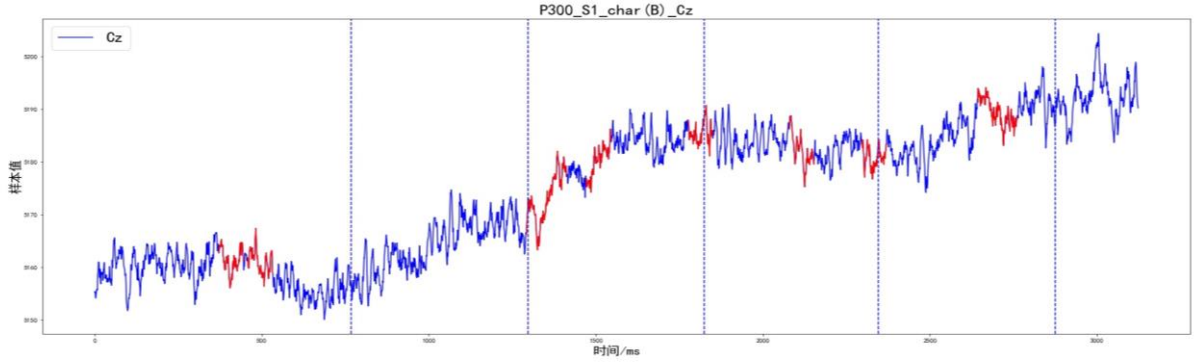


图 4.4 原始脑电信号

使用经验模态分解（Empirical Mode Decomposition，EMD）算法和 FastICA 算法[1]对采集到的 EEG 进行去噪声和伪迹的操作，图 4.5 和图 4.6 为去除噪声和伪迹前后的对比图，原始 EEG 中尖峰处为眼电伪迹，经过预处理后较好地实现了降低噪声和去除伪迹的目标。

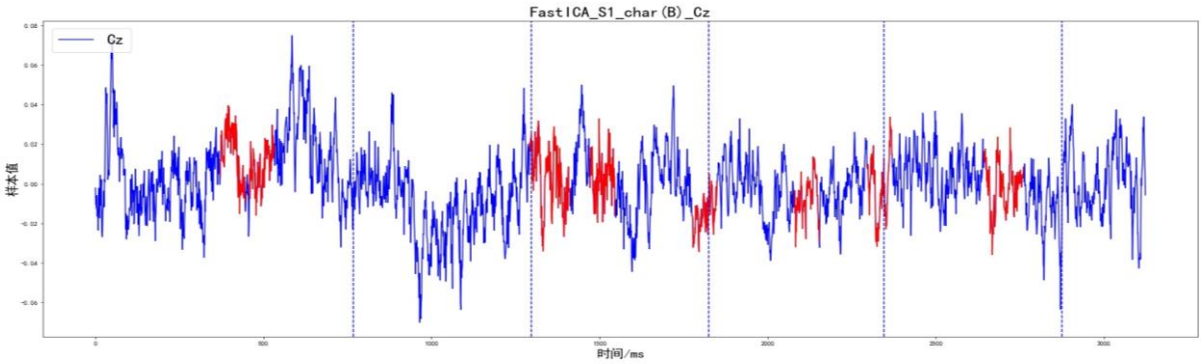


图 4.5 FastICA 去伪迹后的脑电信号

提出的经验模式分解是自适应的，能够较好筛选脑电信号的综合噪声，同时保留了脑电信号的基本信息。

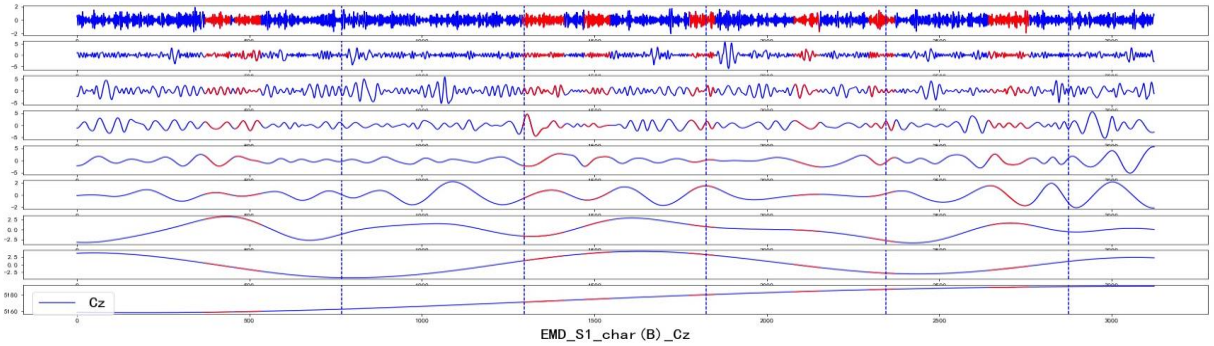


图 4.6 EMD 分解后的脑电信号

4.2.2 特征提取

首先，需要设计一种方法从所给的训练集中找出能表征 P300 电信号和普通信号的特征向量，然后将该特征向量输入到分类器中，得到分类结果，再根据图 4.3 中所示方法，得到待识别目标字符判断结果。由于问题一中只有 P300 电信号和普通信号两种，因此，问题一是一个二分类问题，我们将 P300 电信号视为正例，标签设置为 1， 普通信号视为负例，标签设置为 0，当输入一个特征向量到模型中，模型将输出一个标签 0 或 1，为 0 则被识别为负例，即普通信号，为 1 则被识别为正例，即 P300 电信号。

经过对所给训练集的分析得出，为了找到能表征 P300 电信号和普通信号的特征向量，构造出该二分类模型所需要的标准训练集和测试集，我们可以先找出”train event”文件子表里所有相邻行中第二列数据的值，相邻的两行代表了本次行/列闪烁与下一次行/列闪烁，那么相邻的两行第二列数据的值 index1, index2 则分别对应了”train_data”中起始行电信号数据和结束行电信号数据所在的索引，所以我们只需到找出这两个索引间隔内的电信号数据，则是出现一个闪烁到下一次闪烁的电信号时间序列，由于有 20 个通道，每一个电信号在一个时间点内是一个电信号数据点，因此一个电信号时间序列为一个维度为(index2-index1 ，20)的数值矩阵,而我们所需要的是一个特征向量，因此我们讨论了两种方法：第一种是将该电信号时间序列所有通道的数值进行叠加，如图 4.7，得到一个维度为

index2-index1 维的向量，以此作为二分类模型的输入特征向量，再根据”train_event”中子表第一列给出的该起始索引 index1 对应的行/列标识符，判断是否为该子表对应目标字符所在的行/列标识符，如果是，则将采集到的该特征向量赋予标签为 1，反之为 0。第二种方法是将所有通道的数据拉伸为一个维度为(index2-index1)*20 的列向量作为特征向量， 如图 4.8。由于输入到分类模型的特征向量必须是固定维度的，所以我们经过统计分析索引间隔的大小，发现平均间隔数量为 42，因此我们将该电信号时间序列对应的数值矩阵中行维度进行统一，行维度小于 42 的，后尾补 0，超过 42 的进行截断，以此来达到统一特征向量维度的目的。经过实验结果验证，两种方法均是有效的，都取得了较为理想的准确率。

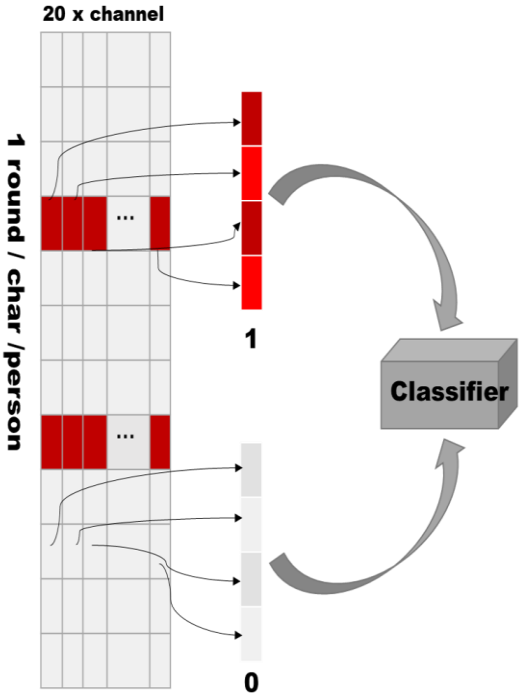


图 4.7

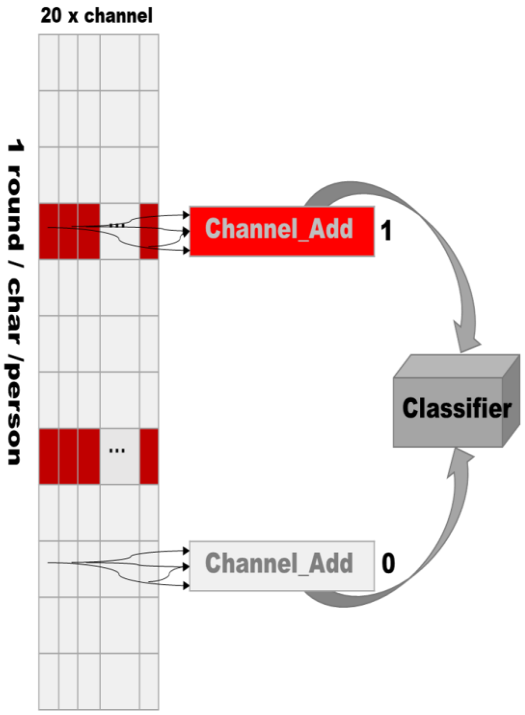


图 4.8

4.2.3 模型训练与识别

将预处理后的 5 轮实验信号数据作为训练集。训练集里正样本（观测字符所在的行/列闪烁）和负样本（不在观测字符所在的行/列的矩阵行/列闪烁）所占比为 1：5。在训练集中，所有正样本的分类结果为 1，负样本的分类结果为 0 。在训练好分类器后，测试集按同样的预处理方式，同样的特征提取方法处理，输入进分类器得出分类结果。分类模型流程图如下图 4.9 所示。

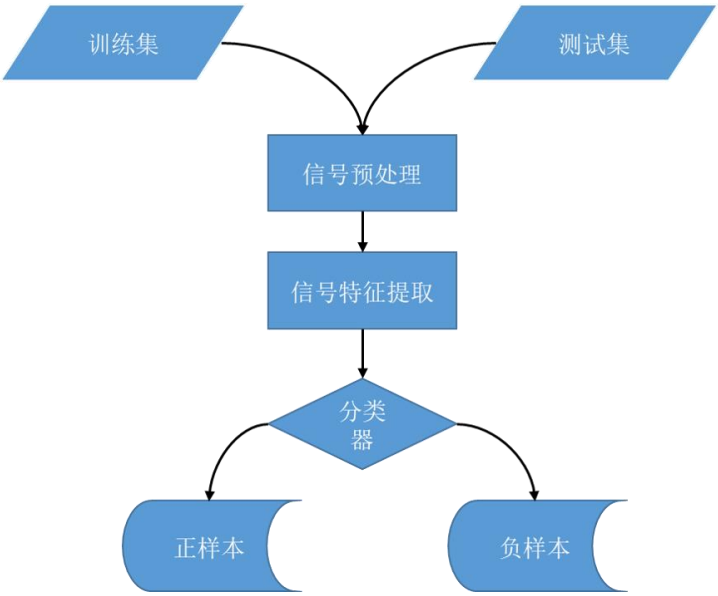


图 4.9 分类模型流程图

本题采用支持向量机（SVM）的分类模型进行实验。

支持向量机是一类按监督学习方式对数据进行二元分类的广义线性分类器。其主要工作是寻找一个超平面，将数据分成两类，同时保证两类数据之间任意元素的距离（margin）最大化，离超平面最近的数据称之为支持向量。具体可分为以下几个步骤：

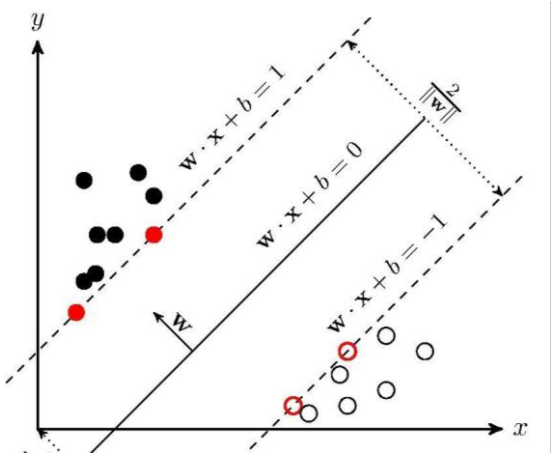


图 4.10 SVM 线性分类示意图

Step1:选取决策边界

在二维平面中，将超平面作如下定义：

$$w\cdot x+b=0, \tag{4.1}$$

在该平面上方的点，定义 $y=1$，在平面下方的点，定义 $y=-1$. w 是垂直于超平面的向量。

Step2：列目标函数

为了保证数据点到该超平面的距离 M 最大化，我们有

$$M=\|x-x'\|=\|\lambda w\|=\frac{|g(x)|}{\|w\|}, \tag{4.2}$$

其中 $g(x)=w\cdot x+b$，当 $w\cdot x+b$=±1时， M 则变为 $\frac{2}{\|w\|}$，称其为 margin. 经过转化发现

$$\max M=\frac{2}{\|w\|}\Leftrightarrow \min\frac{1}{2}w^{T}w. \tag{4.3}$$

Step3：优化目标函数

对于超平面 $w\cdot x+b=0$，首要目标是讲样本数据分类分对，在此基础上再去最大化 margin；故对第一个目标，我们有如下要求

$$\begin{cases} w\cdot x_i+b\geq 1 & if\ \ y_i=+1 \\ w\cdot x_i+b\leq -1 & if\ \ y_i=-1 \end{cases}, \tag{4.4}$$

其等价于

$$y_i(w\cdot x_i+b)-1\geq 0, \tag{4.5}$$

故支持向量机转变成一个优化问题：

$$\begin{aligned} &\min\frac{1}{2}w^{T}w \\ &s.t.\ y_i\left(w\cdot x_i+b\right)\geq 1 \end{aligned}. \tag{4.6}$$

Step4:拉格朗日乘子法求解优化问题

给出上述优化问题的拉格朗日函数

$$L(\alpha,w,b)=\frac{1}{2}\|w\|^{2}-\sum_{i=1}^{l}\alpha_i y_i\left(w\cdot x_i+b\right)+\sum_{i=1}^{l}\alpha_i. \tag{4.7}$$

通过求偏导和化简，可以得到

$$\begin{aligned} w&=\sum_{i}^{l}\alpha_i y_i x_i \\ b&=\frac{1}{N_s}\sum_{s\in S}\left(y_s-\sum_{m\in S}\alpha_m y_m x_m\cdot x_s\right) \end{aligned}, \tag{4.8}$$

其中 (x_s,y_s) 表示任一支持向量。

Step5: 软间隔问题（解决离群点）

在原始数据中，会产生完全分不开的点，故不存在一个超平面使得两类数据完全分开，如图 4.11，因此需要放宽约束条件：

$$y_i\left(w\cdot x_i+b\right)-1+\xi_i\geq 0, \tag{4.9}$$

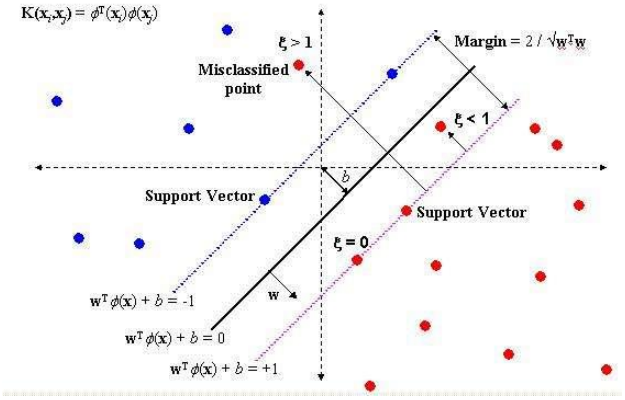


图 4.11　软间隔示意图

但在目标函数中需要加入一个惩罚项：

$$\varphi(w)=\frac{1}{2}\|w\|^2+C\sum_i\xi_i\,, \tag{4.10}$$

该问题同样可以根据拉格朗日乘子法进行求解。

Step6: 核函数变换

上述过程中，设计到的均是线性分类，对于非线性的 SVM，我们需要把原空间的数据通过函数$\psi:x\to\psi(x)$映射到高维空间中，一般称为 Feature Space. 故问题转化为如何找到这样的映射，目前在工程上，已经存在了一些固定的映射函数，不需要针对特定问题去单独构造。常见的有多项式核函数，Gaussian 核函数等。

由于 SVM 的分类思想简单，同时可通过核函数将原始数据映射到高维空间从而解决非线性分类，针对本问题所给的小数据样本可以有较好的分类效果，因此我们选择 SVM 作为本题主要分类器。

4.3 模型结果

在评价分类效果方面，我们选取了精确率（Precision），召回率（Recall）和$F1-score$指标，具体定义如下：

$$precision=\frac{TP}{TP+FP}\,, \tag{4.11}$$

$$recall=\frac{TP}{TP+TN}\quad, \tag{4.12}$$

其中TP表示真实类别为正确，预测为正例的样本个数；TN表示真实类别为正确，预测为反例的样本个数；FP表示真实类别为错误，预测为正例的样本个数；故精确率计算的是“正确被检索的样本数”占所有“实际被检索到的样本数”的比例，召回率计算的是“正确被检索的样本数”占所有“应该检索到的样本数”的比例。根据以上两个指标，我们可以计算评判分类效果的另一个指标：

$$F1\text{-}score=\frac{2\,precision\cdot recall}{precision+recall}\,, \tag{4.13}$$

当$F1-score$越大时，表明分类效果越好。

对原始数据进行上述预处理，特征提取，分类器训练后，将“train_test”数据做相同的预处理后输入模型，得到各个被试对测试目标字符的识别结果如下表 4.2，并给出精确率，召回率和 *F1-score* 指标平均数据。

表 4.2 被试对测试目标字符的识别结果及相应评价指标结果

	S1	S2	S3	S4	S5
Char13	M	M	M	M	M
Char14	F	F	F	F	F
Char15	5	5	5	5	5
Char16	2	2	2	2	2
Char17	I	I	I	I	I
Char18	U	U	X	X	U
Char19	A	A	A	A	E
Char20	1	3	3	1	1
Char21	C	C	C	E	C
Char22	6			6	0
Precision	0.65	0.75	0.68	0.69	0.67
Recall	0.73	0.68	0.77	0.65	0.70
F1-score	0.6877	0.7133	0.7222	0.6694	0.6847

5. 问题二模型建立与求解

5.1 问题分析

在各被试的每轮实验中，“train_data”记录了 20 个通道 3000 个样本的脑电数据，由于 EEG 信号具有非平稳噪声影响较大，个体差异性大，信噪比低等特点，即使在相同实验条件下，每个对象测试得到的 EEG 信号也会出现不一致的情况，且多通道的 EEG 信号势必会造成信息冗余，增加系统的复杂性，降低分类识别的准确率，并造成储存空间的浪费。因此本问题共分为两步骤进行求解：

Step1: 设计一种通道选择算法来筛选每位被试的最优通道组合，并同时保证通道中包含能足够识别目标字符的信息特征。

Step2: 根据筛选出来的 5 组最优通道，对附件 1 中的“test_event”中的（char13-char17）进行 5 折交叉实验，得出 5 组最优通道对 5 个被试的 $F1-score$.

5.2 通道筛选方法

通道筛选可理解为通道降维。在数据分析中，降维方法有缺失值比率，低方差滤波，高相关滤波，主成分分析，随机森林，反向特征消除，前项特征构造等。下面就具体介绍主成分分析，随机森林的方法和本文所采用的共空间模式法[2][3]。

5.2.1 主成分分析（PCA）

该过程通过正交变换，将若干个原始变量换到一组新的互不相关的若干个综合变量中，此数据集为主成分，通过方差来表示信息，方差越大表明提取的信息就越大。数学模型如下：

原始数据矩阵 X 的 p 个变量 $X_1, X_2, ..., X_p$ 作线性组合如下：

$$\begin{cases} F_1 = a_{11}X_1 + a_{12}X_2 + \cdots + a_{1p}X_p \\ F_2 = a_{21}X_1 + a_{22}X_2 + \cdots + a_{2p}X_p \\ \vdots \\ F_p = a_{p1}X_1 + a_{p2}X_2 + \cdots + a_{pp}X_p \end{cases}, \tag{5.1}$$

用矩阵表示为

$$F = AX, \tag{5.2}$$

式中

$$F = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_p \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pp} \end{bmatrix}, \quad X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix}, \tag{5.3}$$

矩阵 A 中的每一行都是单位向量；F_i 与 F_j 之间互不相关，即各主成分之间独立；F_1 在变量 $X_1, ...X_p$ 线性组合下方差最大，即包含信息量最大，$F_2, ..., F_p$ 包含的信息量逐级递减；变量 F 的方差之和等于变量 X 的方差之和，即做正交变换前后的变量均包含全部信息。

5.2.2 随机森林（RF）

主要思想是对目标属性产生许多巨大的树，然后根据对每个属性的统计结果找到信息

量最大的特征子集。其步骤可以分为数据的随机选取和待选特征的随机选取。

首先，从原始的数据集中采取有放回的抽样，构造子数据集，子数据集的数据量是和原始数据集相同的。不同子数据集的元素可以重复，同一个子数据集中的元素也可以重复。第二，利用子数据集来构建子决策树，将这个数据放到每个子决策树中，每个子决策树输出一个结果。最后，如果有了新的数据需要通过随机森林得到分类结果，就可以通过对子决策树的判断结果的投票，得到随机森林的输出结果了。

其次，与数据集的随机选取类似，随机森林中的子树的每一个分裂过程并未用到所有的待选特征，而是从所有的待选特征中随机选取一定的特征，之后再在随机选取的特征中选取最优的特征。这样能够使得随机森林中的决策树都能够彼此不同，提升系统的多样性，从而提升分类性能。

就目前在 EEG 信号通道筛选的研究当中，共空间模式法（CSP）属于空域滤波特征提取算法，针对的是两分类任务，可用于从多通道 EEG 信号中提取有区分度的相关特征从而达到筛选通道的目的。

5.2.3 共空间模式（CSP）

假设$Y_1 \in R^{M\times N}$ 和 $Y_2 \in R^{M\times N}$分别是两类运动想象任务下的多通道诱发响应时空信号矩阵， M为脑电通道数，N为每个通道所采集的样本点数。现假设 $K<N$，计算其归一化协方差矩阵如下：

$$R_2=\frac{E\left(Y_2Y_2^T\right)}{trace\left\{E\left(Y_2Y_2^T\right)\right\}} \quad , \tag{5.4}$$

$$R_1=\frac{E\left(Y_1Y_1^T\right)}{trace\left\{E\left(Y_1Y_1^T\right)\right\}} \quad , \tag{5.5}$$

其中$trace(\cdot)$ 表示矩阵的迹运算。根据协方差矩阵为实对称矩阵的性质，故存在正交矩阵W使下列等式成立：

$$W^TR_1W=Q=diag\left(\lambda_1,\lambda_2,...,\lambda_N\right) \quad , \tag{5.6}$$

$$W^TR_2W=I-Q \quad . \tag{5.7}$$

其中I为N维单位矩阵，$\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_N$为协方差矩阵$R_1$，取$Q$中最大最小的$N_f$个特征值对应的特征向量构建两个投影矩阵$W_1,W_2$，分别作为两类任务间相应的空间滤波器对数据进行分类。

在此方法的基础行，我们引入 2-范数来进行通道筛选。利用共空间模式法中求得的投影矩阵W_1和W_2，组成新的空间滤波器$W_{CSP}\in R^{2N_f\times N}$，则得出滤波后的信号为：

$$S=W_{CSP}X=\begin{bmatrix} w_{11} & \cdots & w_{1N} \\ \vdots & \ddots & \vdots \\ w_{2N_P1} & \cdots & w_{2N_PN} \end{bmatrix}\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}, \tag{5.8}$$

令$W_{CSP}=[w_1,w_2,...,w_N]$，其中$w_i\ (i=1,2,...,N)$表示矩阵$W_{CSP}$中的第$i$列向量，代表了各个通道信号$x_l\ (l=1,2,...,N)$在投影空间的权重，体现了原始通道 EEG 信号对投影后信号的影响程度。由此，通过 2-范数给各个通道l的权重LW；

$$LW(l)=\frac{\|w_l\|_2}{\|W_{CSP}\|_2}, \tag{5.9}$$

通过对权重从高到底排序，权重越大说明通道观测信号对投影后的信号影响较大，从而可选出L个包含信息量较多的有效通道。

5.3 模型求解

基于 2-范数下的共空间模式通道筛选模型。经过软件编程得出 5 个被试的各个通道的权重，如表 5.1：

表 5.1　5 个被试者各通道权重

通道名称	S1	S2	S3	S4	S5
1(FZ)	0.074	0.067	0.036	0.071	0.065
2(F3)	0.025	0.004	0.040	0.006	0.009
3(F4)	0.050	0.086	0.033	0.083	0.008
4(Cz)	0.069	0.095	0.069	0.090	0.080
5(C3)	0.088	0.069	0.056	0.095	0.093
6(C4)	0.095	0.077	0.069	0.083	0.055
7(T7)	0.054	0.075	0.084	0.076	0.011
8(T8)	0.014	0.040	0.087	0.050	0.085
9(CP3)	0.015	0.067	0.017	0.017	0.035
10(CP4)	0.025	0.017	0.013	0.039	0.030
11(CP5)	0.083	0.072	0.063	0.013	0.077
12(CP6)	0.025	0.003	0.008	0.003	0.001
13(Pz)	0.080	0.028	0.047	0.091	0.005
14(P3)	0.024	0.005	0.048	0.029	0.069
15(P4)	0.092	0.010	0.077	0.029	0.062
16(P7)	0.035	0.084	0.044	0.032	0.054
17(P8)	0.019	0.071	0.035	0.045	0.075
18(Oz)	0.025	0.032	0.060	0.063	0.073
19(O1)	0.061	0.097	0.067	0.002	0.081
20(02)	0.047	0.004	0.047	0.082	0.030

根据累计贡献率达到 85%能反应原来变量信息的原则，我们得到了 5 个被试者的最优通道数及最优通道组合如下表 5.2：

表 5.2　5 个被试者最优通道数及最优通道组合

被试者(S)	最优通道组合(L)-按权重顺序排列	通道数
S1	C4,P4,C3,CP5,Pz,Fz,Cz,O1,T7.F4,O2,P7,CP4	13
S2	O1,Cz,F4,P7,C4,T7,CP5,P8,C3,Fz,CP3	11
S3	T8,T7,P4,C4,Cz,O1,CP5,Oz,C3,P3,Pz,O2,P7,F3	14
S4	C3,Pz,Cz,F4,C4,O2,T7,Fz,Oz,T8,P8,CP4	12
S5	C3,T8,O1,Cz,CP5,P8,Oz,P3,Fz,P4,C4,P7	12

对于所选的 5 组最优通道组合，与问题一的全通道进行分类精度对比，得到如图 5.3 结果：

表 5.3　5 个被试者全通道与最优通道分类精度对比

被试者(S)	通道组合	F1-score
S1	全部通道	0.6877
	1,3,4,5,6,7,10,11,13,15,16,19,20	0.7642
S2	全部通道	0.7133
	1,3,4,5,6,7,9,11,16,17,19	0.7788
S3	全部通道	0.7222
	2,4,5,6,7,8,11,13,14,15,16,18,19,20	0.7795
S4	全部通道	0.6694
	1,3,4,5,6,7,8,10,13,17,18,20	0.7495
S5	全部通道	0.6847
	1,4,5,6,8,11,14,15,16,17,18,19	0.7834

通过表 的结果发现，对于每一位被试，优化后的通道组合比全部通道在分类效果上要好，$F1-score$指标都出现了一定程度的增加，由此可以证明共空间模式-2 范数通道选择算法是有效的。

根据问题一的模型，第二部分我们将选出的最优通道组合（L1-L5）分别对被试（S1-S5）进行 5 折交叉验证如图 1，根据不同通道对不同被试的$F1-score$来检验通道选择算法的合理性与有效性。

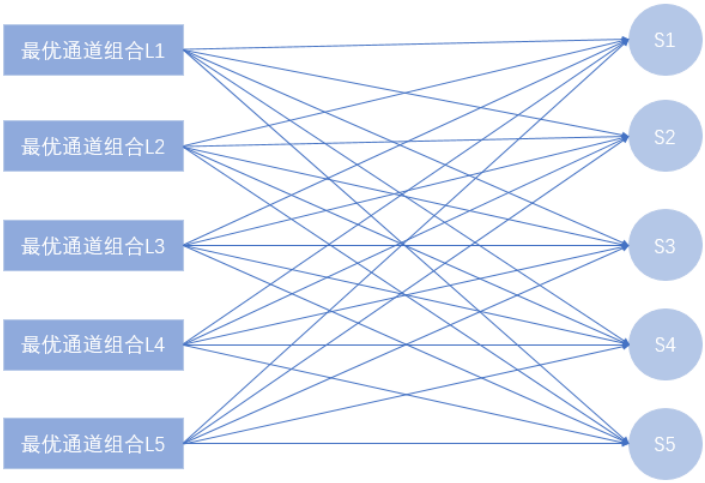


图 5.1　5 折交叉验证示意图

经求解，我们得到各最优通道组合对不同被试者在不同观测字符的$F1-score$如下表 5.4：

表 5.4 最优通道组合对不同被试者的$F1-score$

被试 / 通道组合	S1	S2	S3	S4	S5	平均 $F1-score$
L1	0.7642	0.7182	0.7052	0.6884	0.7759	0.7304
L2	0.7413	0.7788	0.7334	0.7124	0.7632	0.7458
L3	0.7584	0.6899	0.7795	0.7339	0.7194	0.7362
L4	0.6933	0.7486	0.7310	0.7495	0.7222	0.7289
L5	0.7119	0.7411	0.7259	0.7407	0.7834	0.7406

通过上表，我们可以发现 L2 的通道组合对各个被试的平均$F1\text{-}score$最大为 0.7458，故选择 L2 通道作为一组最优通道组合，其组合为通道 1, 3, 4, 5, 6, 7, 9, 11, 16, 17, 19。

6. 问题三模型建立与求解

6.1 问题分析

在 P300 脑-机接口系统中，往往需要花费很长时间获取有标签样本来训练模型。为了减少训练时间，设计一种半监督学习方法，选择适量的样本作为有标签样本，其余训练样本作为无标签样本，训练出 P300 脑电信号的分类模型并验证其效果。

P300 事件相关点位是诱发脑电信号的一种，在小概率刺激发生后 300 毫秒范围左右会出现一个正向波峰（相对基线来说呈现向上趋势的波）。P300 脑电信号有如下特点[4]：

- 任务效应：P300 潜伏期和波幅与靶刺激有关，即与任务有关，是普通诱发信号所没有的。
- 任务难度可以影响 P300。难度大，P300 明显，反之则不明显。
- 概率效应：刺激序列中的任务相关刺激（Task Stimulus , TS）和无关刺激即 NTS，二者的概率要事先设定。任务相关刺激为低概率，任务无关刺激是高概率，二者的概率互补，如目标字符所在行/列闪烁的概率是 0.167，任务无关刺激的概率是 0.833。加大 TS 概率则振幅变小、潜伏期延迟，此即概率效应。
- 刺激性质的影响：人脑接收外界信息是经由不同感受器在脑的不同部位参与下进行初级认知加工的。刺激性质不同，P300 的潜伏期和波幅也会有一定区别：视觉 P300 潜伏期略长于听觉，躯体感觉者最短。Cz 点记录 P300 波幅以听觉最小，视觉和体感为大。

根据 P300 信号数据的特点我们可以确定，在字符识别的任务中，P300 信号有明显的区别于无关信号的特征。所以，在理想的情况下，只需要一条任务相关刺激数据即可提取到 P300 脑电信号的特征。

半监督学习的基本思想是利用数据分布上的模型假设建立学习器对未标签样例进行标签。它的形式化描述是给定一个来自某未知分布的样例集 $S = L\ U$，其中 L 是已标签样例集 $L = \{ (x1, y1) , (x2, y2) \ldots (x|L|, y|L|) \}$，$U$ 是一个未标签样例集 $U = \{ xc1, xc2. ..xc|U| \}$，希望得到函数 $f(x)$ 可以准确地对样例 x 预测其标签 y。其中 x_i , xc_1 均为 d 维向量，y_i 为样例 x_i 的标签，$|L|$ 和 $|U|$ 分别为 L 和 U 的大小，即所包含的样例数，半监督学习就是在样例集 S 上寻找最优的学习器。如果 $S = L$，那么问题就转化为传统的有监督学习；反之,如果 $S = U$,那么问题是转化为传统的无监督学习。如何综合利用已标签样例和未标签样例,是半监督学习需要解决的问题。

半监督学习可以有效的利用 P300 信号的形态特点，通过少量的有标签样本提取信号特征，训练其余的无标签样本。以此为出发点，设计出一种基于 K-Shape 半监督学习的信号序列聚类方法。

6.2 模型建立

6.2.1 P300 信号相似度衡量

本文提出的基于脑电信号形态重心的有效 P300 脑电信号聚类方法根据 P300 信号于重心序列的相似度来实现，因此需要一种适用于 P300 信号的相似度衡量方法。对于传统衡量方法，如欧式距离、动态时间弯曲及 Hausdorff 距离，虽然被广泛应用于时间序列相似度计算，但并不适合用来衡量特征更复杂的 P300 信号的相似度。P300 信号是一种非线性、高纬度、低信噪比的诱发信号，而且对于不同采集时间、不同采集被试得到的 P300 还存在信号分布不一致的问题。本文采用了一种具有尺度不变于相位不变特性的 P300 相似度（距离）衡量方法。

为了衡量两条 P300 信号 e_x 与 e_y 的相似度，首先提出一种尺度-偏移一致性的 P300 距离衡量方法，具体定义为：

$$d\left(e_x,e_y\right)=\min_{\alpha,s}\frac{\left\|e_x-\alpha e_{y(s)}\right\|}{\left\|e_x\right\|},\qquad(6.1)$$

其中，α 与 $e_{y(s)}$ 分别表示尺度系数与相位偏移了 s 个数据点的 P300 信号 e_y。诚然，e_x 与 e_y 之间的距离越小，表明他们之间的相似度越高，反之则越低。相应的，e_x 与 e_y 之间的相似度则可以定义为：

$$s\left(e_x,e_y\right)=1-d\left(e_x,e_y\right),\qquad(6.2)$$

利用公式（6.1）的距离衡量方法可以搜寻到最优的 P300 信号尺度与偏移长度并得到他们之间的最小距离，最终根据公式（6.2）得到 P300 信号 e_x 与 e_y 的相似度。

为了得到最优偏移 P300 信号 $e_{y(s)}$，本文采用了互相关方法[5]进行计算。当得到 P300 信号 e_y 的偏移序列 $e_{y(s)}$ 之后，最优的尺度系数 α 则可以通过一个 $\frac{\left\|e_x-\alpha e_{y(s)}\right\|_2}{\left\|e_x\right\|_2}$ 得到。不难看出，该表达式是关于尺度系数 α 的一个凸函数，所以，设定 $\frac{\partial d\left(e_x,e_y\right)}{\partial\alpha}=0$，最优的尺度系数 α 则可由如下式子计算所得，即：

$$\alpha=\frac{e^T e_{y(s)}}{\left\|e_{y(s)}\right\|^2},\qquad(6.3)$$

其中 $d(\mathrm{e}_x,e_y)$ 为两条 P300 信号的相似度。这种方法有效得解决了 P300 信号存在的尺度变

换和相位变换的问题。

6.2.2 Kshape 聚类算法

时间序列分析中的许多任务依赖于通过一个序列有效地总结一组时间序列的方法。这个概要序列通常被称为平均序列，或者在聚类的情况下，称为重心。有意义的重心的提取是一项具有挑战性的任务，它在很大程度上取决于距离度量的选择。从一组序列中提取平均序列的最简单方法是计算平均序列的每个坐标，作为所有序列对应坐标的算术平均值。该方法采用最常用的聚类方法 k-means。为了避免这些问题,我们把重心计算作为一个优化问题，问题的目标是最小化其他信号序列到重心序列的距离平方和。

$$c^* = \arg\min \sum d(e_i, c)^2, \qquad (6.4)$$

其中，c^* 为 P300 信号的重心序列，e_i 为 P300 信号。以下的伪代码演示了重心提取过程，以捕获底层数据的共享特征。

算法 6.1： $c^* = Centroidextraction(E, c)$

输入： P300 脑电信号集合 $E_{m\times n}$ ，n 条长度为 m 的 P300 脑电信号集合；δ ：相似度阈值（即距离阈值 $1-\delta$ ）

输出： c^*：P300 脑电信号的重心序列

$E' \leftarrow []$ ；

for $i = 1$ *to* n **do**

  $e_i \leftarrow Shift(c, e_i)$ [6]；

  $d \leftarrow d(e_i, c)$ ； $s \leftarrow 1-d$ ；

  if $s \geq \delta$ （即 $d \leq 1-\delta$ ） **then**

    $E' \leftarrow [E'; e_i]$ ；

  end if

end for

$U \leftarrow \sum_{e_i \in E} (I - \frac{e_i e_i^T}{\|e_i\|^2})$ ；

$c^* \leftarrow Eigenvector(U, thesmallstone)$ ；

---

kshape 聚类算法是一种基于迭代优化过程的局部聚类方法，类似于 k-means 中使用的算法。通过这个迭代过程，kshape 最小化了距离的平方和，并假设:

（1）产生齐次且分离良好的重心

（2）与时间序列的数量集个数线性缩放。我们的算法比较序列有效和计算中心有效下，分切不变，平移不变，位移不变。Kshape 是 k-means 的一个重要实例，它的相似度度量和重心计算方法使 Kshape 成为唯一显著优于 k-means 的时间序列聚类算法。

在每次迭代中，Kshape 执行两个步骤:

（1）在分配步骤中，算法通过将每个时间序列与所有计算出的重心进行比较，并将每个时间序列分配给最接近重心的簇来更新簇的成员关系。

（2）在优化步骤中，更新聚类中心以反映前一步中聚类成员的变化。算法重复这两个步骤，直到没有集群成员发生变化，或者达到允许的最大迭代次数。在赋值步骤中，Kshape 依赖于信号的相似度算法，而在赋值步骤中，Kshape 依赖于如上所示的重心心计算方法。首先，我们将数据集中的信号序列随机分配给簇。然后，我们使用最大话相似度的方法计算每个簇重心。一旦计算出了重心心，我们通过 P300 信号相似度的衡量来优化聚类的成员关系。重复这个过程，直到算法收敛或达到最大迭代次数(通常是很小的次数，比如 100 次)。该算法的输出是将序列分配的聚类以及每个聚类的重心。

6.3 模型求解

选取第二问所求的最优通道的组合做为实验样本数据。采用受到闪烁刺激后的 75 个样本点到 150 个样本点（300ms）之间脑电信号序列作为 P300 信号，再将最优通道的信号列向叠加，聚类结果如下图 6.1 所示。横坐标表示样本点的特征信号，纵坐标表示正（负）样本聚类后按特征排列的簇的重心度量，蓝色折线 Cluster1 表示负样本信号的重心分布，红色折线 Cluster2 表示负样本的重心分布。可以观测到在 0~400 的维度区间正负样的重心分布存在差异，在 400~500 区间重心重合度较高，在 500~900 的区间聚类效果明显。

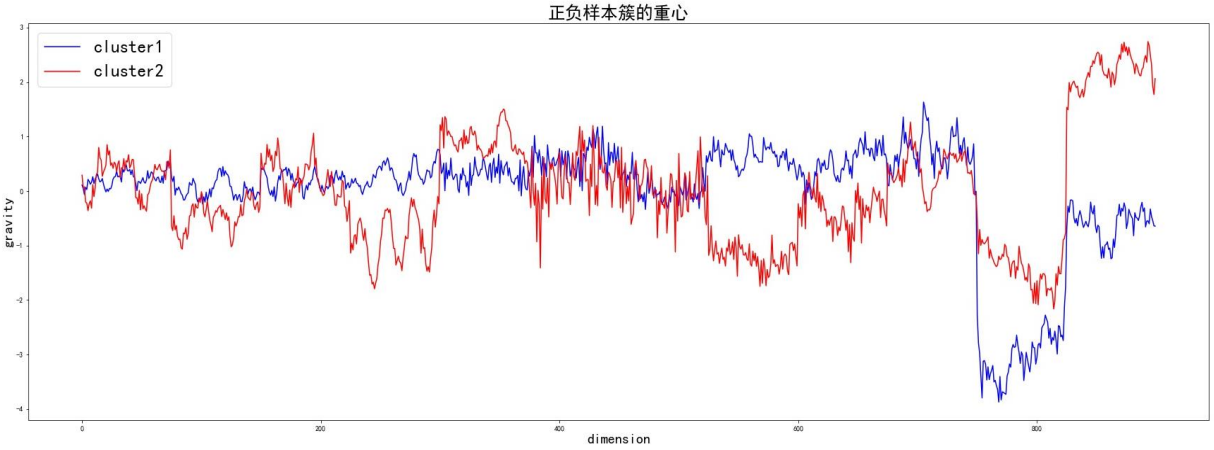


图 6.1　正负样本信号重心序列

分别对 S1 至 S5 5 位被试进行多次实验后，得到的最高准确率分别为 73.3% ，65.2% ，63.7% ，77.8% ，69.4% 。本文也实现了 KSC 聚类算法[7]，但由于聚类效果不明显不做分析。

P300 信号会因所受刺激强度不同产生尺度的变换，也会因被试生理条件的不同等原因产生相位的偏差。本文提出的基于 Kshaped 的信号序列聚类方法解决了小样本问题的同时，

还考虑了 P300 信号的尺度变换性和相位变换性。

7. 问题四模型建立与求解

7.1 问题分析

首先对于附件 2 的数据分析，“睡眠脑电数据”中子表分别代表了睡眠的 5 个时期，并在数据表格第一列中备注了所属的标签；第二列至第五列分别代表了不同频率范围内的能量占比，特征参数单位为百分比。

问题定义：根据题目所给条件可以得出，这是一个多分类问题，具体类别为 5 类，分别为清醒期、睡眠 1 期、睡眠 2 期、深睡眠期、快速眼动期，问题 4 则是需要我们根据所给数据训练一个多分类模型，使得该模型在接受一个指定输入时，能够自动判断该输入数据的类别是属于以上哪一个睡眠前期，与此同时需要我们手动划分所给数据，将其分为训练数据和测试数据，说明选取方式以及分配的比例，并对模型性能做了要求，要求在尽可能少的样本作为训练集的情况下取得较高的准确率，在模型训练完成后分析预测效果。

其中有几个要点，分别是：

（1）对数据的预处理。我们需要深入分析所给数据，观察每一维特征之间的关联度，以及对于不同类别的同一特征的数据分布，初步得出数据分布规律，并以此为依据选择最为合适的模型来拟合该多分类问题。

（2）特征提取。从所给数据中我们可以挖掘出一些重要的隐藏特征，这需要我们深入理解问题背景和数据的实际含义。

（3）模型选择。模型选择需要考虑题目中约束条件，即在选择尽量少的样本前提下如何获得较高的准确率，此时我们需要充分分析各种多分类模型的特点以及缺陷，通过实验划分训练集大小来验证选择的模型在小训练集情况下依旧具有良好的多分类性能。

（4）数据集划分。在本题中，所给出的样本数据长度为 4 维，其对应类别标签由子表名称给出，而没有划分为训练集和测试集，此时，需要我们选择一个合适的划分方案，并确定分配比例，划分的关键在于样本类别是否均衡，是否具有普遍性。

（5）预测效果分析。预测效果分析的关键在于合理定义一个模型性能评估指标，以该指标的值来衡量预测效果。

7.2 模型建立

7.2.1 模型输入输出定义

由问题分析可知，我们需要建立一个多分类模型。

其中，模型输入为：一个 N 维的特征向量(N 为大于 0 的整数)，模型输出为：标签类别，输出类型为整数，其中标签取值与睡眠分期对应关系为如表 7.1 所示，由于所给数据中标签最小值为 2，最大值为 6，因此需要对标签进行映射，转换为从 0 开始，得到标签类别的取值集合为{0,1,2,3,4}。

表 7.1　标签与睡眠分期映射关系

模型输出（标签	表格中实际数值	睡眠分期
0	2	清醒期
1	3	睡眠 1 期
2	4	睡眠 2 期
3	5	深睡眠期
4	6	快速眼动期

7.2.2 数据预处理

为了使得模型算法具有更好的学习能力，我们对所给数据进行了一定的预处理，具体工作包括数据合并、标签变换、标准化、缺失值处理、异常值处理。

不同的特征指标其度量单位以及类别往往具有一定差异，因此在数据分析环节，由于量纲不同而带来的分析误差直接影响特征的选取，从而影响模型学习能力，为了消除指标之间的量纲影响，需要进行数据标准化处理，以解决数据指标之间的可比性。在经过标准化后，所有特征指标均处于同一取值范围内，为同一数量级，便于综合评价特征指标的重要性，有利于特征提取，从而有效提高模型学习能力，加快模型收敛速度。因此，我们对所给数据做如下标准化变换：

$$x' = \frac{x - \min x}{\max x - \min x}, \qquad (7.1)$$

而缺失值、异常值处理则是对所给数据中缺省值或数据大小明显异常的值用所在列的各种统计指标进行代替，如平均数、众数以及中位数，若该特征指标中缺失值或异常值大于一定数值则舍弃该特征。经过对所给数据的检查分析，发现并没有此类数据，因此可以跳过这一环节。

7.2.3 特征提取

在完成数据的预处理后，进一步挖掘其隐藏特征以及合并一些呈线性关系的特征。我们在这一步所做工作如下：

（1）分析不同特征指标之间关联程度。分析不同特征指标的关联程度有利于剔除冗余特征，以及发现新的隐藏特征。首先，题目给出的原始数据具有 4 个不同的特征指标，名称分别为 Alpha、Beta、Theta、Delta，我们对其中每一个类别中 4 个不同特征进行相关性检验，如图 7.1-7.5 所示，分别对应了 5 个不同睡眠分期每一个特征的大致分布，横坐标代表样本个数，纵坐标代表特征指标对应数值，对此观察到以下特征：

- 清醒期 alpha（蓝色折线，第一维特征）以及 theta(红色折线，第三位特征)特征指标数值方差波动巨大，但二者相减后方差起伏大大减小。由此可见，清醒期 alpha 以及 theta 特征指标具有很强关联性，因此对于清醒期，可以构造一个新的特征，值为 alpha 与 theta 的差值，来代替这两个波动方差大的特征，从而减小方差波动。
- 睡眠 1 期 theta 与 alpha 特征指标数据分布重合，因此，这两个特征具有强关联性，可以剔除一个特征，用其余三个特征来对睡眠一期进行分类。
- 睡眠 2 期 delta 特征基本比其余特征数值大，beta 特征数值基本比其他特征数值小
- 深睡眠期和快速眼动期各个特征之间数值分布断层明显

由此可见对于不同的类，各特征之间数值分布都具有各自的规律，其中清醒期各特征指标混合程度最高，而快速眼动期混合程度最低，对于不同类别，能表征该类的特征组合数都各不相同。

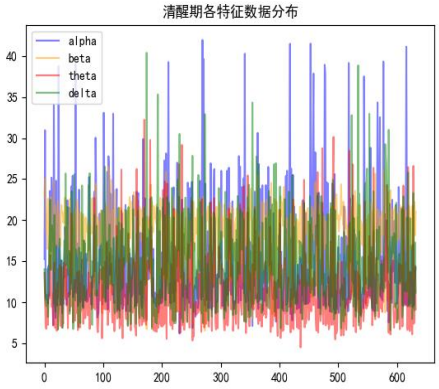


图 7.1 清醒期各特征数据分布

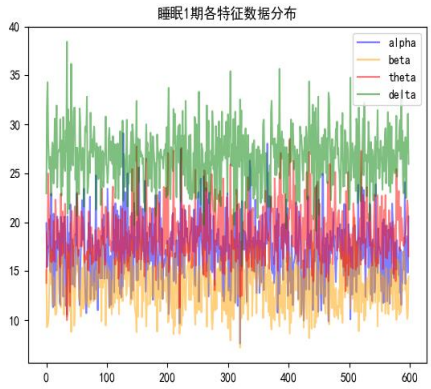


图 7.2 睡眠 1 期各特征数据分布

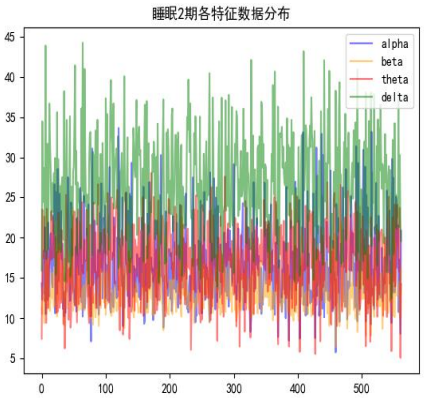


图 7.3 睡眠 2 期各特征数据分布

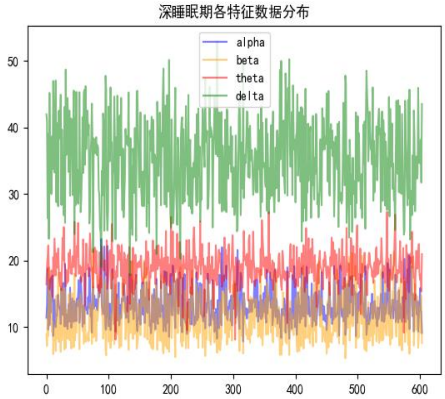


图 7.4 深睡眠期各特征数据分布

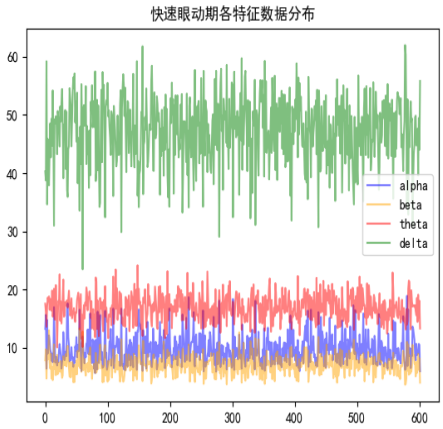


图 7.5 快速眼动期各特征数据分布

（2）分析不同类别对于同一特征指标的数据分布。除此之外，我们还分析了对于不同类别的同一特征的数据分布规律，如图 7.1-7.5：

- 对于 delta 特征（绿色曲线）可以很明显看出，除了清醒期，其余各睡眠分期的数值基本都在 25-40 范围内，而清醒期则基本处于 15-25 范围内，因此 delta 特征能够很

好的区分清醒期与其他睡眠分期。

- 对于 beta 特征（橙色曲线）可以很明显看出，除了清醒期，其余各睡眠分期的数值基本都在 5-15 范围内，而清醒期则基本处于 15-25 范围内，因此 beta 特征能够很好的区分清醒期与其他睡眠分期。
- 对于 alpha 特征（蓝色曲线）可以看出，清醒期，与睡眠 1 期重合较大，且由分析（1）得出，方差波动太大，因此 alpha 无法有效的表征清醒期。

同理，通过对比各个类同一特征数值分布，我们得出：

- 对于清醒期（对应类别标签 0），能够有效表征该分期的特征为：beta, theta, delta, 同时，由分析（1）得出，可以构造一个新的特征 alpha’,数值为 theta 与 alpha 的差值。
- 对于睡眠 1 期(对应类别标签 1),能够有效表征该分期的特征为:alpha，beta, theta’，数值为：beta 与 theta 的差值
- 对于睡眠 2 期(对应类别标签 2)，能够有效表征该分期的特征为：alpha, beta, theta
- 对于深睡眠期（对应类别标签 3），能够有效表征该分期的特征为：alpha,beta,theta,delta’(alpha 与 theta 的差值)
- 对于快速眼动期（对应类别标签 4），能够有效表征该分期的特征为：alpha,beta,theta,delta’(alpha 与 theta 的差值)

7.2.4 模型选择

根据以上分析，我们发现对于不同类别，能有效表征其类别的特征组合都有一定差异，因此，我们针对每一个类别设计一个二分类模型，该二分类模型只负责识一个确定的睡眠分期，例如，用于识别清醒期的二分类模型，我们假设清醒期对应标签为 1，其余睡眠分期标签全为 0，其余睡眠分期二分类模型同理。

图 7.6 给出了本题中五个二分类模型混合结构示意图：

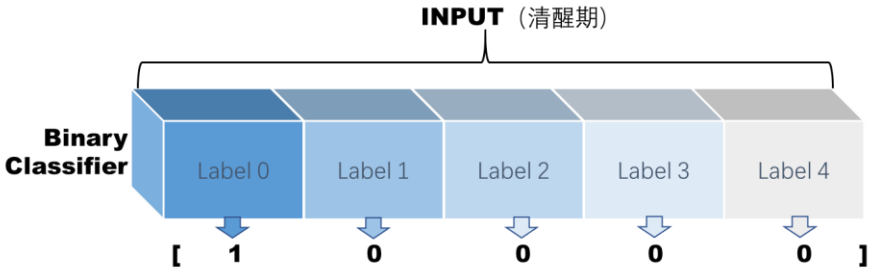


图 7.6　五个二分类模型混合结构示意图

对于每一个二分类模型，如问题（1）解决方案中所述，SVM 用于二分类问题具有很良好的性能，所以我们决定选取问题（1）中 SVM 模型进行二分类。

7.2.5 数据集划分

对于本题提出的由 5 个二分类模型组成的多分类混合模型，每一个二分类子模型所需数据集按如下步骤选取与划分：

首先将该二分类子模型所需识别的睡眠分期对应的表格数据集中数据作为正样本，然后从其余睡眠分期表格数据每一个表格随机采样相同个数的样本作为负样本，直到采集到

的负样本数量与正样本数量基本一致则停止采样，再对所有得到的样本随机打乱，避免偶然性因素，使得后续数据集划分结果具有普遍性，由此得到二分类子模型所需数据集。

其次，对于搜集到的用于二分类子模型的数据集，按照比例 7：2：1 划分为数据集，测试集，验证集。验证集作用在于：可以通过在验证集上进行 K 折交叉验证后的精确率高低来评判模型最优参数，另一方面，验证集还可以用于模型训练时判断是否过拟合，方法为：当训练集上的精确率还在升高，而验证集上的精确率却趋于不变甚至减小，则可以判断模型过拟合，便于模型训练早期停止。

7.3 结果分析

7.3.1 评估指标

由于我们设计的混合模型本质上属于由二分类模型叠加后形成的多分类模型，因此问题（1）中提出的性能评估指标——精确率(precision)、召回率(recall)以及 F1-score 同样适合本题，最终本题混合模型的精确率、召回率以及 F1-score 值为五个二分类模型的加权平均，如公式 5-7 所示。

7.3.2 对比实验

为了更好的衡量我们设计的模型的预测效果，我们选取几个传统的多分类模型与本题提出的由 5 个二分类器组成的混合模型进行了几组对比试验，分别统计在相同迭代次数以及相同测试集上的评估指标值，如表 5 所示，从表中可以很明显看出，本题提出的混合模型相比于传统的多分类模型——BP 神经网络多分类模型、随机森林，具有较高的精确率、召回率以及 F1-score 值，且具有较高的迭代收敛速度，图 18 给出了本题中提出的混合模型测试集上损失随迭代次数的变化情况。

表 7.2　模型性能对比实验数据统计

模型类别	精确率(precision)	召回率(recall)	F1-score
BP 神经网络多分类模型	75.1%	74.2%	0.746
随机森林	72.1%	69.8%	0.709
本题中混合模型	85.3%	86.7%	0.86

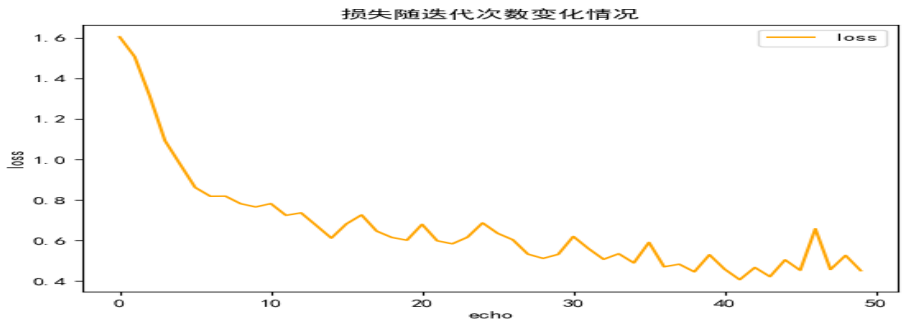


图 7.7　损失随迭代次数变化情况

最后，我们对于本题中要求的尽可能少的数据作为训练样本同时需要有较高的精确率这一性能要求，分别在训练集：测试集：验证集为 7：2：1、6：2：2、5：2：3 的比例下做了一个精确率比较实验，结果如表 10 所示，从表中看出，在比例为 5：2：3 时，本题中设计的混合模型依旧能够取得较高的精确率，符合题目所给要求。

表 7.3　数据集划分比例精确率测试结果

模型类别	数据集划分比例	精确率
BP 神经网络多分类模型	7：2：1	75.1%
	6：2：2	72.6%
	5：2：3	68.3%
随机森林	7：2：1	72.1%
	6：2：2	71.4%
	5：2：3	67.9%
本题中混合模型	7：2：1	85.3%
	6：2：2	84.9%
	5：2：3	82.2%

8. 模型评价

8.1 模型优点

问题一对原始数据进行 EMD 去噪和 FastICA 去伪迹的预处理，准确的定位了 P300 信号的电位。采用思想简单 SVM 分类算法，支持向量样本集具有一定的鲁棒性，可以解决非线性分类问题。

问题二采用的共空间模式-2 范数通道筛选方法，有效的去除了信息冗余的通道，极大的缩小了样本数据量，同时提高了分类的精度。

问题三采用了一种具有尺度偏移不变的 P300 脑电信号相似度（距离）衡量办法，该方法与常用方法（如欧式距离、动态时间）相比，具有更高的灵活性，更适用于衡量 P300 信号的相似度（距离）。同时，设计一种基于 Kshape 的形态重心筛选方法，能有效区分不同类标脑电信号的特征，从而为不同的分类器提供了更有辨别度、代表性的脑电信号特征，更有利于分类器的模型学习与分类。

问题四采用了混合模型，对于每一个二分类器，都专注于一个睡眠分期的识别，能够捕捉到一个睡眠分期里更细微的特征，同时可以过滤掉一些干扰特征，因此，识别性能相比于传统的单一多分类模型性能有较大提升。

8.2 模型缺点

对于问题一，对数据预处理的方法测试较少，提取的特征不够明显，用 SVM 解决多分类问题存在困难。

对于问题二，通过 5 折交叉实验后，仅根据平均 *F1-score* 来选择最优通道组合，会存

在各被试*F1-score*方差较大的情况，导致选择不合理。

对于问题四，存在正负样本不均衡问题，例如，原多分类数据集中由 5 类样本数据集构成，此时若选择其中一类样本作为正样本，那么负样本的数量是正样本的 4 倍，此时为了使得正负样本均衡，避免模型倾向于识别成样本数多的类别，则需要将负样本的数目减少至和正样本数量一致，由此不能完全利用所给数据集，造成一定性能损失。

参考文献

[1] 马也, 姜光萍. P300 脑电信号的特征提取及分类研究. 山东工业技术, (10):202-204，2017.

[2] 周蚌艳, 吴小培, 吕钊, 张磊, 郭晓静, 张超. 基于共空间模式方法的多类运动想象脑电的导联选择. 生物医学工程学杂志, 32(3):520-525, 2015.

[3] 汲继跃, 佘青山, 张启忠, 孟明. 最优区域共空间模式的运动想象脑电信号分类方法. 传感技术学报, 33(1):34-39, 2020.

[4] 杨文俊.事件相关电位简述.中华神经精神科杂志,24(5):309-312, 1991.

[5] Chenglong D , Dechang P , Lin C , et al. MTEEGC: A novel approach for multi-trial EEG clustering[J]. Applied Soft Computing,71:255-267, 2018.

[6] Dai, Chenglong, Pi, Dechang, Becker, Stefanie, I, Wu, Jia, Cui, Lin, Johnson, Blake. CenEEGs: Valid EEG Selection for Classification[J] . ACM transactions on knowledge discovery from data,14(2):18.1-18.25,2020.

[7] Yang, J. and J. Leskovec. “Patterns of temporal variation in online media.” *WSDM '11*,2011.

[8] https://github.com/sieve-microservices/kshape,2020-9-16.

附录

```
# coding=utf-8

"""
聚类算法找出 p300 电信号
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, DBSCAN
from math_match.load_data import load_one_people_data_for_char
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import FastICA
from math_match.load_data import load_solution1_data
from sklearn import svm
import tensorflow as tf
import pandas as pd
from math_match.file_path import ATTACH1_ROOT_PATH
from sklearn.ensemble import RandomForestClassifier
from math_match import wave_change
import re
import random
train_sheet_names = ['char01(B)', 'char02(D)', 'char03(G)', 'char04(L)', 'char05(O)',
'char06(Q)', 'char07(S)',
                     'char08(V)', 'char09(Z)', 'char10(4)', 'char11(7)', 'char12(9)']
train_sheet_names1 = ['char01(B)']
train_sheet_name_to_char = {'char01(B)':'B','char02(D)':'D','char03(G)':'G','char04(L)':
'L','char05(O)':'O','char06(Q)':'Q','char07(S)':'S','char08(V)':'V','char09(Z)':'Z',
'char10(4)': '4', 'char11(7)': '7', 'char12(9)': '9'}
train_chars_to_sheet_names = {'B': 'char01(B)', 'D':'char02(D)', 'G':'char03(G)',
'L':'char04(L)','O':'char05(O)', 'Q':'char06(Q)', 'S':'char07(S)', 'V':'char08(V)',
'Z':'char09(Z)' , '4':'char10(4)', '7':'char11(7)', '9':'char12(9)'}
test_sheet_names = ['char13', 'char14', 'char15', 'char16', 'char17', 'char18', 'char19',
                    'char20', 'char21', 'char22']
test_sheet_names2 = ['char13', 'char14', 'char15', 'char16', 'char17', 'char18', 'char19',
                    'char20', 'char21']
test_sheet_names1 = ['char13', 'char14', 'char15']
key_to_index = {'1':0, '2':1, '3':2, '4':3, '5':4, '6':5,
                '7':0, '8':1, '9':2, '10':3, '11':4, '12':5}  # 将电信号标识符转为索引
index_table = [
    ['A','B','C','D','E', 'F'],
    ['G','H','I','J','K', 'L'],
    ['M','N','O','P','Q', 'R'],
```

```
    ['S','T','U','V','W', 'X'],
    ['Y','Z','1','2','3', '4'],
    ['5','6','7','8','9', '0']
]
index_dic = {'A':(1, 7),'B':(1, 8),'C':(1, 9),'D':(1, 10),'E':(1, 11), 'F':(1, 12),
             'G':(2,7),'H':(2,8),'I':(2,9),'J':(2,10),'K':(2,11), 'L':(2,12),
             'M':(3,7),'N':(3,8),'O':(3,9),'P':(3,10),'Q':(3,11), 'R':(3,12),
             'S':(4,7),'T':(4,8),'U':(4,9),'V':(4,10),'W':(4,11), 'X':(4,12),
             'Y':(5,7),'Z':(5,8),'0':(5,9),'1':(5,10),'2':(5,11), '3':(5,12),
             '4':(6,7),'5':(6,8),'6':(6,9),'7':(6,10),'8':(6,11), '9':(6,12)
              }


def function1(labels):
    """
    返回出现次数最少的那个类别
    :param labels:
    :return:
    """
    counts = {}
    small = []
    length = 10000
    for i in range(len(labels)):
        if str(labels[i]) not in counts.keys():
            counts[str(labels[i])] = []
            counts[str(labels[i])].append(i)
        else:
            counts[str(labels[i])].append(i)
    for item in counts:
        if len(counts[item]) < length:
            small = counts[item]
            length = len(counts[item])
    return small


def find_res(input_datas, input_labels, res, iter=1):
    """

    :param input_datas:
    :param input_labels:
    :param res:
    :param k: 选择几轮作为聚类数据
    :return:
    """
```

```
    for i in range(len(input_labels)):
        for sheet_name in test_sheet_names:
            label_frame = input_labels[i][sheet_name]
            data_frame = input_datas[i][sheet_name]
            begin = 0
            count = 0
            list_row = []  # (sample_size, feature_size) 装行电波数据
            list_col = []  # 装列电波数据
            row_dic = []
            col_dic = []
            normal_state = normal_state = data_frame.iloc[label_frame.iloc[begin][1] - 1, :]
            for j in range(label_frame.shape[0]):
                if j % 13 == 0:  # 表示已经到本轮实验末尾了
                    begin = j
                    normal_state = data_frame.iloc[label_frame.iloc[begin][1] - 1, :]
                    count += 1
                    if count > iter:
                        break
                    continue
                if label_frame.iloc[j, 0] != 666 and label_frame.iloc[j][0] != 100:
                    if label_frame.iloc[j, 0] <= 6:  # 说明为列

list_row.append(data_frame.iloc[label_frame.iloc[j][1]-1, :].sub(normal_state.tolist()).t
olist())
                        row_dic.append(j)
                    else:
                        list_col.append(data_frame.iloc[label_frame.iloc[j][1] -
1, :].sub(normal_state.tolist()).tolist())
                        col_dic.append(j)
            # 得到一轮实验样本的向量组成的矩阵
            array_row = np.array(list_row)  # 行电波数据
            array_col = np.array(list_col)  # 列电波数据
            row_s, col_s = StandardScaler(), StandardScaler()
            # print('行电波数据集大小：', array_row.shape)
            array_row = row_s.fit_transform(array_row)
            array_col = col_s.fit_transform(array_col)
            print(array_row, array_col)
            km_row = DBSCAN(eps=4.4, min_samples=1)
            km_col = DBSCAN(eps=4.4, min_samples=1)
            km_row.fit(array_row)
            km_col.fit(array_col)
            labels_row, labels_col = km_row.labels_, km_col.labels_
            print(labels_row, labels_col)
            index_row, index_col = function1(labels_row), function1(labels_col)  # 找到
```

p300 电信号所在的索引

```
            key_row, key_col = [], []  # 统计行闪和列闪标识符
            for index in index_row:
                key_row.append(label_frame.iloc[row_dic[index]][0])
            for index in index_col:
                key_col.append(label_frame.iloc[col_dic[index]][0])
            row_final_key, col_final_key = -1, -1
            if len(key_row) != 0:
                print('行电波：', key_row)
                row_final_key = key_to_index[str(key_row[0])]
            if len(key_col) != 0:
                print('列电波：', key_col)
                col_final_key = key_to_index[str(key_col[0])]

            if row_final_key != -1 and col_final_key != -1:
                c = index_table[row_final_key][col_final_key]
            else:
                c='NAN'
            res[i].append(c)

    return res


def normal_vector(vector, max_length=600):
    if len(vector) >= max_length:
        return vector[:max_length]
    else:
        for i in range(max_length - len(vector)):
            vector.append(0)
        return vector


def get_inputs_for_one_people_cnn(train_frame, event_frame, keys=None, iter=5, train=True):
    """
    用于得到输入分类器的标准输入 shape:(sample_size, max_length)
    每一个表格有 60 个有效数据（其中 50 个负样本，10 个正样本）sample_size = 12 * 60 = 720
    :param train_frame:3000 * 12
    :param event_frame:
    :param keys: (行标识符,  列标识符)
    :param iter:默认加载 5 轮实验
    :param train:加载训练集还是测试集，默认为加载训练集
    :return:
    """
    # print('event_frame:', event_frame)
```

```
    if train:
        inputs = []  # 用于存储输入数据
        labels = []
        count = 0  # 记录第几轮
        # count_s = 0  # 记录采集的样本序号
        for i in range(0, event_frame.shape[0]):
            if event_frame.iloc[i][0] == 100:
                count += 1
                if count == iter:
                    break
                continue
            # 获取特征向量
            # 该闪烁信号的特征向量 shape 为:(50 , 12)，拉伸为一列
            temp_frame =
train_frame.iloc[event_frame.iloc[i][1]+74:event_frame.iloc[i][1]+149, :]
            # feature_vector = temp_frame.iloc[:, 3:6].sum(axis=1).tolist()
            feature_vector = []
            for j in range(temp_frame.shape[0]):
                feature_vector.append(temp_frame.iloc[j, :].tolist())
            # # 统一维度
            # feature_vector = normal_vector(feature_vector, max_length=max_length)
            # print('特征向量统一维度后长度为：', len(feature_vector))
            print('特征向量维度为：', np.array(feature_vector).shape)
            inputs.append(feature_vector)
            if event_frame.iloc[i][0] not in keys:  # 如果不是该字符的 p300 闪烁信号,样本标
记为负
                labels.append(0)
            else:  # 如果是该字符的 p300 闪烁信号,样本标记为正
                labels.append(1)
        return inputs, labels
    else:
        inputs = []  # 用于存储输入数据
        key_s = []  # 用于存储该电信号相应的行或列的标识符
        count = 0  # 记录第几轮
        # count_s = 0  # 记录采集的样本序号
        for i in range(0, event_frame.shape[0]):
            if event_frame.iloc[i][0] == 100:
                count += 1
                if count == iter:
                    break
                continue
            # 获取特征向量
            # 该闪烁信号的特征向量 shape 为:(50 , 12)，拉伸为一列
            temp_frame                                                    =
```

```
train_frame.iloc[event_frame.iloc[i][1]+74:event_frame.iloc[i][1]+149, :]
            feature_vector = []
            for j in range(temp_frame.shape[0]):
                feature_vector.append(temp_frame.iloc[j, :].tolist())
            # # 统一维度
            # feature_vector = normal_vector(feature_vector, max_length=max_length)
            # print('特征向量统一维度后长度为：', len(feature_vector))
            print('特征向量长度为：', np.array(feature_vector).shape)
            inputs.append(feature_vector)
            key_s.append(event_frame.iloc[i][0])
        # inputs 长度和 key_s 长度相同，一个 input 对应一个行或列标识符
        # print('test rows：', len(inputs), 'test cols:', len(inputs[0]))
        # print('key_s rows:', len(key_s))
        return inputs, key_s


def get_inputs_for_one_people(train_frame, event_frame, keys=None, max_length=12*75, iter=5,
train=True):
    """
    用于得到输入分类器的标准输入 shape:(sample_size, max_length)
    每一个表格有 60 个有效数据（其中 50 个负样本，10 个正样本）sample_size = 12 * 60 = 720
    :param train_frame:3000 * 12
    :param event_frame:
    :param keys: (行标识符， 列标识符)
    :param iter:默认加载 5 轮实验
    :param train:加载训练集还是测试集，默认为加载训练集
    :return:
    """
    # print('event_frame:', event_frame)
    if train:
        inputs = []  # 用于存储输入数据
        labels = []
        count = 0  # 记录第几轮
        # count_s = 0  # 记录采集的样本序号
        print('event_frame:', event_frame)
        for i in range(0, event_frame.shape[0]):
            if event_frame.iloc[i][0] == 100:
                count += 1
                if count == iter:
                    break
                continue
            # 获取特征向量
            # 该闪烁信号的特征向量 shape 为:(50 , 12)，拉伸为一列
            temp_frame                                                              =
```

```
train_frame.iloc[event_frame.iloc[i][1]+74:event_frame.iloc[i][1]+149, :]
            # feature_vector = temp_frame.iloc[:, 3:6].sum(axis=1).tolist()
            feature_vector = []
            for j in range(temp_frame.shape[1]):
                feature_vector.extend(temp_frame.iloc[:, j].tolist())
            # # 统一维度
            feature_vector = normal_vector(feature_vector, max_length=max_length)
            print('特征向量统一维度后长度为：', len(feature_vector))
            inputs.append(feature_vector)
            if event_frame.iloc[i][0] not in keys:  # 如果不是该字符的 p300 闪烁信号,样本标
记为负
                labels.append(0)
            else:  # 如果是该字符的 p300 闪烁信号,样本标记为正
                labels.append(1)
        return inputs, labels
    else:
        inputs = []  # 用于存储输入数据
        key_s = []  # 用于存储该电信号相应的行或列的标识符
        count = 0  # 记录第几轮
        # count_s = 0  # 记录采集的样本序号
        for i in range(0, event_frame.shape[0]):
            if event_frame.iloc[i][0] == 100:
                count += 1
                if count == iter:
                    break
                continue
            # 获取特征向量
            # 该闪烁信号的特征向量 shape 为:(50 , 12)，拉伸为一列
            temp_frame                                                                    =
train_frame.iloc[event_frame.iloc[i][1]+74:event_frame.iloc[i][1]+149, :]
            feature_vector = []
            for j in range(temp_frame.shape[1]):
                feature_vector.extend(temp_frame.iloc[:, j].tolist())
            # 统一维度
            # feature_vector = temp_frame.iloc[:, 3:6].sum(axis=1).tolist()
            # feature_vector = normal_vector(feature_vector, max_length=max_length)
            print('特征向量统一维度后长度为：', len(feature_vector))
            inputs.append(feature_vector)
            key_s.append(event_frame.iloc[i][0])
        # inputs 长度和 key_s 长度相同，一个 input 对应一个行或列标识符
        # print('test rows：', len(inputs), 'test cols:', len(inputs[0]))
        # print('key_s rows:', len(key_s))
        return inputs, key_s
```

```
def tst1():
    test_data_dic1, test_data_label_dic1 = load_one_people_data_for_char(people_name='S1',

sheet_names2=test_sheet_names)
    res1 = []
    res = [res1]
    input_datas = [test_data_dic1]
    input_labels = [test_data_label_dic1]  # 包含 5 个人的测试集
    res = find_res(input_datas, input_labels, res)
    print(res)


def get_all_chars_data_for_one_people_cnn(people_name='S1',
sheet_names=train_sheet_names1):
    train_data_dic, train_event_dic =
load_one_people_data_for_char(people_name=people_name,
                                                          sheet_names=sheet_names)
    train_x, train_y = [], []
    for sheet_name in sheet_names:
        # 获取每一个字符的实验数据，合并到一起
        data_frame, event_frame = train_data_dic[sheet_name], train_event_dic[sheet_name]
        # sd = StandardScaler()
        # data_frame = pd.DataFrame(sd.fit_transform(data_frame))
        sd = StandardScaler()
        data_array = sd.fit_transform(data_frame)
        # f = FastICA(n_components=12)
        # data_array = f.fit_transform(data_array)
        data_frame = pd.DataFrame(data_array)

        # data_array = data_frame.values
        # f = FastICA(n_components=12)
        # data_array = f.fit_transform(data_array)
        # data_frame = pd.DataFrame(data_array)
        # print('经过 ICA 后，train_data_frame 维度：', data_array.shape)  # 结果为
(sample_size, 12)
        inputs_for_one, labels_for_one = get_inputs_for_one_people_cnn(data_frame,
event_frame, keys=index_dic[train_sheet_name_to_char[sheet_name]], iter=5)
        # 所有字符的数据加到一起
        train_x.extend(inputs_for_one)
        train_y.extend(labels_for_one)
    # # train_x, train_y = np.array(train_x), np.array(train_y)
    # # train_x_n, train_y_n = np.zeros(train_x.shape), np.zeros(train_y.shape)
    positives_x, positives_y = [], []
```

```
    nagtives_x, nagtives_y = [], []
    for i in range(len(train_x)):
        if train_y[i] == 1:
            positives_x.append(train_x[i])
            positives_y.append(train_y[i])
        else:
            nagtives_x.append(train_x[i])
            nagtives_y.append(train_y[i])
    splits = [1, 1.2]
    nagtives_size = int(len(positives_y) * (splits[1] / splits[0]))
    if nagtives_size > len(train_y) - len(positives_y):
        nagtives_size = len(train_y) - len(positives_y)
    random_sort = random.sample([i for i in range(len(nagtives_y))], nagtives_size)
    n_x, n_y = [], []
    for i in range(len(random_sort)):
        n_x.append(nagtives_x[random_sort[i]])
        n_y.append(nagtives_y[random_sort[i]])
    positives_x.extend(n_x)
    positives_y.extend(n_y)
    train_x, train_y = [], []
    random_sort = random.sample([i for i in range(len(positives_y))], len(positives_y))
    for i in range(len(random_sort)):
        train_x.append(positives_x[random_sort[i]])
        train_y.append(positives_y[random_sort[i]])
    return np.array(train_x), np.array(train_y)


def get_all_chars_data_for_one_people(people_name='S1', sheet_names=train_sheet_names1):
    train_data_dic, train_event_dic =
load_one_people_data_for_char(people_name=people_name,
                                                          sheet_names=sheet_names)
    train_x, train_y = [], []
    for sheet_name in sheet_names:
        # 获取每一个字符的实验数据，合并到一起
        data_frame, event_frame = train_data_dic[sheet_name], train_event_dic[sheet_name]
        # sd = StandardScaler()
        # data_frame = pd.DataFrame(sd.fit_transform(data_frame))
        sd = StandardScaler()
        data_array = sd.fit_transform(data_frame)
        # f = FastICA(n_components=12)
        # data_array = f.fit_transform(data_array)
        data_frame = pd.DataFrame(data_array)

        # data_array = data_frame.values
```

```
        # f = FastICA(n_components=12)
        # data_array = f.fit_transform(data_array)
        # data_frame = pd.DataFrame(data_array)
        # print('经过 ICA 后，train_data_frame 维度：', data_array.shape)  # 结果为
(sample_size, 12)
        inputs_for_one, labels_for_one = get_inputs_for_one_people(data_frame, event_frame,
keys=index_dic[train_sheet_name_to_char[sheet_name]], iter=5)
        # 所有字符的数据加到一起
        train_x.extend(inputs_for_one)
        train_y.extend(labels_for_one)
    # # train_x, train_y = np.array(train_x), np.array(train_y)
    # # train_x_n, train_y_n = np.zeros(train_x.shape), np.zeros(train_y.shape)
    positives_x, positives_y = [], []
    nagtives_x, nagtives_y = [], []
    for i in range(len(train_x)):
        if train_y[i] == 1:
            positives_x.append(train_x[i])
            positives_y.append(train_y[i])
        else:
            nagtives_x.append(train_x[i])
            nagtives_y.append(train_y[i])
    splits = [1, 1.0]
    nagtives_size = int(len(positives_y) * (splits[1] / splits[0]))
    if nagtives_size > len(train_y) - len(positives_y):
        nagtives_size = len(train_y) - len(positives_y)
    random_sort = random.sample([i for i in range(len(nagtives_y))], nagtives_size)
    n_x, n_y = [], []
    for i in range(len(random_sort)):
        n_x.append(nagtives_x[random_sort[i]])
        n_y.append(nagtives_y[random_sort[i]])
    positives_x.extend(n_x)
    positives_y.extend(n_y)
    train_x, train_y = [], []
    random_sort = random.sample([i for i in range(len(positives_y))], len(positives_y))
    for i in range(len(random_sort)):
        train_x.append(positives_x[random_sort[i]])
        train_y.append(positives_y[random_sort[i]])
    train_x.extend(train_x)
    train_y.extend(train_y)
    return np.array(train_x), np.array(train_y)


def get_all_chars_data_for_one_people_test_cnn(people_name='S1',
sheet_names=test_sheet_names1):
```

```
    test_data_dic, test_event_dic = load_one_people_data_for_char(people_name=people_name,
                                                                  sheet_names2=sheet_names)
    test_x, key_s = [], []
    for sheet_name in sheet_names:
        # 获取每一个字符的实验数据，合并到一起
        data_frame, event_frame = test_data_dic[sheet_name], test_event_dic[sheet_name]
        # data_array = data_frame.values
        # f = FastICA(n_components=12)
        # data_array = f.fit_transform(data_array)
        # data_frame = pd.DataFrame(data_array)
        # # print('经过 ICA 后，train_data_frame 维度：', data_array.shape)  # 结果为
(sample_size, 12)
        sd = StandardScaler()
        data_array = sd.fit_transform(data_frame)
        # f = FastICA(n_components=12)
        # data_array = f.fit_transform(data_array)
        data_frame = pd.DataFrame(data_array)
        inputs_for_one, keys_for_one = get_inputs_for_one_people_cnn(data_frame,
event_frame, train=False, iter=5)
        # 所有字符的数据加到一起
        test_x.extend(inputs_for_one)
        key_s.extend(keys_for_one)
    test_x, key_s = np.array(test_x), np.array(key_s)
    return test_x, key_s


def get_all_chars_data_for_one_people_test(people_name='S1',
sheet_names=test_sheet_names1):
    test_data_dic, test_event_dic = load_one_people_data_for_char(people_name=people_name,
                                                                  sheet_names2=sheet_names)
    test_x, key_s = [], []
    for sheet_name in sheet_names:
        # 获取每一个字符的实验数据，合并到一起
        data_frame, event_frame = test_data_dic[sheet_name], test_event_dic[sheet_name]
        # data_array = data_frame.values
        # f = FastICA(n_components=12)
        # data_array = f.fit_transform(data_array)
        # data_frame = pd.DataFrame(data_array)
        # # print('经过 ICA 后，train_data_frame 维度：', data_array.shape)  # 结果为
(sample_size, 12)
        sd = StandardScaler()
        data_array = sd.fit_transform(data_frame)
        f = FastICA(n_components=12)
        data_array = f.fit_transform(data_array)
```

```
        data_frame = pd.DataFrame(data_array)
        inputs_for_one, keys_for_one = get_inputs_for_one_people(data_frame, event_frame,
train=False, iter=1)
        # 所有字符的数据加到一起
        test_x.extend(inputs_for_one)
        key_s.extend(keys_for_one)
    test_x, key_s = np.array(test_x), np.array(key_s)
    return test_x, key_s


def save_input_train_data(people_name, train_path, sheet_names):
    """
    存储用于二分类的脑电信号数据
    :param save_path:
    :return:
    """
    train_x1, train_y1 = get_all_chars_data_for_one_people(people_name,
sheet_names=sheet_names)
    train_x1 = pd.DataFrame(train_x1)
    train_y1 = pd.Series(train_y1)
    train_x1['label'] = train_y1
    train_x1.to_csv(train_path)
    print(train_x1)


def save_input_test_data(people_name, test_path, sheet_names):
    """
    存储用于二分类的脑电信号数据
    :param save_path:
    :return:
    """
    test_x1, key_s = get_all_chars_data_for_one_people_test(people_name,
sheet_names=sheet_names)
    test_x1 = pd.DataFrame(test_x1)
    key_s = pd.Series(key_s)
    test_x1['row_or_col_key'] = key_s
    test_x1.to_csv(test_path)
    print(test_x1)


def find_res2():
    train_x, train_y = get_all_chars_data_for_one_people('S1',
sheet_names=train_sheet_names)
    # data_frame = load_solution1_data(ATTACH1_ROOT_PATH + '\\train_data_s1.csv')
```

```
    # train_x, train_y = data_frame.iloc[:, 1:-1], data_frame.iloc[:, -1]
    # train_x, train_y = train_x.values, train_y.values
    train_x_c = []
    for i in range(train_x.shape[0]):
        train_x_c.append(list(wave_change.plot_signal_decomp(train_x[i, :])))
    train_x = np.array(train_x_c)
    print('train_x.shape:', train_x.shape)
    test_x, key_s = get_all_chars_data_for_one_people_test('S1',
sheet_names=test_sheet_names1)
    test_x_c = []
    for i in range(test_x.shape[0]):
        test_x_c.append(list(wave_change.plot_signal_decomp(test_x[i, :])))
    test_x = np.array(test_x_c)
    print('test_x.shape:', test_x.shape)
    # # svm
    # print(train_x.shape)
    # # class_weights = {0: 1, 1:5}  # 权重向量
    # SVMmodel = svm.SVC()  # ovr:一对多策略
    # # model = RandomForestClassifier(n_estimators=50, max_features=24, bootstrap=True)
    # SVMmodel.fit(train_x, train_y)  # ravel 函数在降维时默认是行序优先
    # # """
    # #     用 tensorflow 深度学习框架
    # #     :return:
    # #     """
    inputs = tf.keras.Input((train_x.shape[1],))
    x = tf.keras.layers.Dense(units=1000, activation=tf.nn.relu)(inputs)
    x = tf.keras.layers.Dense(units=500, activation=tf.nn.relu)(x)
    # x = tf.keras.layers.Dense(units=100, activation=tf.nn.relu)(x)
    x = tf.keras.layers.Dense(units=300, activation=tf.nn.relu)(x)

    # x = tf.keras.layers.Dropout(0.1)(x)
    outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(x)
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    model.summary()
    model.compile(optimizer=tf.keras.optimizers.Adam(),
                  loss=tf.keras.losses.binary_crossentropy,
                  metrics=[tf.keras.metrics.BinaryAccuracy()])
    num_epochs, batch_size = 5, 128
    history = model.fit(x=train_x, y=train_y, epochs=num_epochs, batch_size=batch_size)
    y_pred = model.predict(test_x)
    # print(y_pred)
    for i in range(y_pred.shape[0]):
        # print('???')
        if i % 12 == 0:
```

```
            print('-----------------------------------------------')
        if y_pred[i] > 0.5:
            print('标识符：', key_s[i])
        else:
            print('Nan')


def function():
    text1 = """
    """
    res = re.findall('loss: (.*?) ', text1)
    for i in range(len(res)):
        res[i] = float(res[i])
    res_new = [res[i] for i in range(len(res)) if i %2 == 0]
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
    plt.title('损失随迭代次数变化情况')
    plt.xlabel("echo")
    plt.ylabel("loss")
    plt.plot(res_new, label='loss', c='orange')
    plt.legend()
    plt.savefig(ATTACH1_ROOT_PATH + '\\' + '损失随迭代次数变化情况' + '.png')
    plt.show()



def find_res_cnn():
    train_x1, train_y1 = get_all_chars_data_for_one_people_cnn('S1',
sheet_names=train_sheet_names)
    # print('x1, y1:', train_x1.shape, train_y1.shape)
    train_x2, train_y2 = get_all_chars_data_for_one_people_cnn('S2',
sheet_names=train_sheet_names)
    # print('x2, y2:', train_x2.shape, train_y2.shape)
    train_x3, train_y3 = get_all_chars_data_for_one_people_cnn('S3',
sheet_names=train_sheet_names)
    train_x4, train_y4 = get_all_chars_data_for_one_people_cnn('S4',
sheet_names=train_sheet_names)
    train_x5, train_y5 = get_all_chars_data_for_one_people_cnn('S5',
sheet_names=train_sheet_names)
    train_x = np.vstack((train_x1, train_x2))
    train_x = np.vstack((train_x, train_x3))
    train_x = np.vstack((train_x, train_x4))
    train_x = np.vstack((train_x, train_x5))
    train_y = list(train_y1)
    train_y.extend(list(train_y2))
```

```
    train_y.extend(list(train_y3))
    train_y.extend(list(train_y4))
    train_y.extend(list(train_y5))
    train_y = np.array(train_y)
    # train_x, train_y = train_x1, train_y1
    test_x, key_s = get_all_chars_data_for_one_people_test_cnn('S1',
sheet_names=test_sheet_names)
    # svm
    print('x, y:', train_x.shape, train_y.shape)
    # class_weights = {0: 1, 1:5}  # 权重向量
    # SVMmodel = svm.SVC()  # ovr:一对多策略
    # # model = RandomForestClassifier(n_estimators=50, max_features=24, bootstrap=True)
    # SVMmodel.fit(train_x, train_y)  # ravel 函数在降维时默认是行序优先
    """
        用 tensorflow 深度学习框架
        :return:
        """
    inputs = tf.keras.Input((train_x.shape[1], train_x.shape[2]))
    cnn1 = tf.keras.layers.Conv1D(75, 3, padding='same', activation='relu')(inputs)
    # print('after cnn1')
    cnn1 = tf.keras.layers.MaxPooling1D(padding='same')(cnn1)
    cnn2 = tf.keras.layers.Conv1D(75, 4, padding='same', activation='relu')(inputs)
    cnn2 = tf.keras.layers.MaxPooling1D(padding='same')(cnn2)
    cnn3 = tf.keras.layers.Conv1D(75, 5, padding='same', activation='relu')(inputs)
    cnn3 = tf.keras.layers.MaxPooling1D(padding='same')(cnn3)
    # print('after cnn3')
    cnn = tf.concat([cnn1, cnn2, cnn3], axis=-1)
    flat = tf.keras.layers.Flatten()(cnn)
    drop = tf.keras.layers.Dropout(0.1)(flat)
    full = tf.keras.layers.BatchNormalization()(drop)
    full1 = tf.keras.layers.Dense(128, activation='relu')(full)
    dense = tf.keras.layers.Dropout(0.2)(full1)
    outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(dense)
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    model.summary()
    model.compile(optimizer=tf.keras.optimizers.Adam(),
                  loss=tf.keras.losses.binary_crossentropy,
                  metrics=[tf.keras.metrics.BinaryAccuracy()])
    num_epochs, batch_size = 16, 64
    history = model.fit(x=train_x, y=train_y, epochs=num_epochs, batch_size=batch_size)
    # loss, accuracy = model.evaluate(x=test_x, y=test_y)
    # print('test loss: ', loss, 'test accuracy:', accuracy)
    # print(history)
    # print(test_x.shape())
```

```
    y_pred = model.predict(test_x)
    # print(y_pred)
    print(y_pred)
    for i in range(y_pred.shape[0]):
        # print('???')
        if i % 12 == 0:
            print('-----------------------------------------------')
        if y_pred[i] > 0.5:
            print('标识符：', key_s[i])
        else:
            print('NAN')



if __name__ == '__main__':

    # print(test_x.shape, key_s.shape

    # find_res2()

    # find_res_cnn()

    # save_input_train_data('S1', ATTACH1_ROOT_PATH + r'\train_data_s1.csv',
sheet_names=train_sheet_names1)

    function()
```