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	西安交通大学
参赛队号	20106980009
队员姓名	1. 陈栩栩
	2. 李珂
	3. 刘硕

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和判别模型**

摘　　要：

大脑是人体中高级神经活动的中枢，以外围神经为媒介向身体各部分发出指令，脑机接口技术旨在不依赖正常的由外围神经或肌肉组织组成的输出通路的通讯系统，实现大脑与外部辅助设备之间的交流沟通。P300 事件相关电位是诱发脑电信号的一种，人在受到某种刺激时通常在 300-500ms 能诱发 P300 电位的产生。

针对问题一，本文首先对数据集中的脑机信号进行了预处理，包括使用 ICA 去除眼电信号的干扰、使用滤波器去除噪声，基于小波变换和能量熵对信号进行特征提取，以及对信号进行数据分段。在训练模型的选择上，本文使用了动态卷积神经网络、SVM、KNN 三种模型，经比较选择识别效果最好的 SVM 模型来确定被试测试集中的 10 个待识别目标。通过训练支持向量机得到可以识别 P300 电位的模型，从而根据 P300 出现的位置确定识别目标，识别结果依次为：M、F、5、2、I、T、K、X、A、0。

针对问题二，本文沿用了问题一中的 SVM 分类模型，同时加入递归特征消除，通过不断消除当前特征子集中对目标函数贡献最小的特征，得到基于特征的重要程度排序表。首先使用 SVR-RFE 算法对每个被试者进行通道选择，得到每个通道的特征权重，选取前 10 个权重排名高的通道作为每个被试者的最优通道组。再对所有被试者进行通道选择，得到一组泛化性较强的适用所有被试者的一组最优通道名称组合，最优通道组合选择为：Fz、F3、C3、C4、CP4、Pz、P4、P7、Oz、O2。

针对问题三，由于半监督学习带来的训练样本不足问题，本文首先使用了生成对抗网络 GAN 扩增样本，然后使用原型网络进行训练。原型网络能通过训练映射函数，将样本特征映射到高维空间，使得类别相同的样本处于相近的区域，从而实现通过少量样本数据进行分类的目标。通过对问题二给出的数据测试检验，能证明该方法的有效性。将模型用于预测其余待识别目标，可得其余识别目标为: T、K、X、A、0。

第四部分使用 SVM 和 KNN 模型预测睡眠分期，可以在较少的训练样本中得到较高的预测准确率，当训练集占 15.0%，测试集占 85.0%时，模型测试准确率达 71.50%。在对 5 个睡眠分期逐个进行准确率分析时，发现模型在检测清醒期、快速眼动期和深睡眠期有着更好的分类效果。

关键字：P300 电位，ICA，SVM，通道选择，半监督学习，GAN

目录

1. 问题重述

大脑是人体中高级神经活动的中枢，通过数以亿计的神经元之间的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，人的大脑控制着感知、思维、运动及语言等功能，并且将外围神经为媒介向身体各部分发出指令。研究发现，在外围神经失去作用的情况下，人的大脑依旧在正常运行，并且其发出指令的部分信息使可以通过一些路径表征出来，脑-机接口技术[1]旨在不依赖正常的由外围神经或肌肉组织组成的输出通路的通讯系统，实现大脑与外部辅助设备之间的交流沟通。P300 事件相关电位是诱发脑电信号的一种，它的意义在于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，相对基线来说呈现着向上趋势的波。由于个体之间存在差异性，每个个体 P300 的发生时间也有所不同，存在 300~500ms 的刺激延时。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。

现阶段，如何提高睡眠质量，减少睡眠相关疾病对健康的影响，在社会上受到广发关注，因而需要研究自发脑电信号（即睡眠过程中采集的脑电信号），它能够反映身体状态的自身变化，是用来诊断和治疗相关疾病的重要依据。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。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。

问题描述为：

问题一：在脑-机接口系统中既要考虑目标的分类准确率，同时又要保证一定的信息传输速率。根据 P300 脑机接口实验数据，设计或采用一个方法，在尽可能使用较少轮次（要求轮次数小于等于 5）的测试数据的情况下，找出 P300 脑机接口实验数据中 5 个被试测试集中的 10 个待识别目标，并给出具体的分类识别过程，可与几种方法进行对比，来说明设计方法的合理性。

问题二：由于采集的原始脑电数据量较大，这样的信号势必包含较多的冗余信息。在 20 个脑电信号采集通道中，无关或冗余的通道数据不仅会增加系统的复杂度，且影响分类识别的准确率和性能。分析 P300 脑机接口实验数据，并设计一个通道选择算法，给出针对每个被试的、更有利于分类的通道名称组合（要求通道组合的数量小于 20 大于等于 10，每个被试所选的通道可以不相同）。基于通道选择的结果，进一步分析对于所有被试都较适用的一组最优通道名称组合，并给出具体分析过程。

问题三：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，请根据所给 P300 脑机接口实验数据，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，在问题二所得一组最优通道组合的基础上，设计一种学习的方法，并利用问题二的测试数据（char13-char17）检验方法的有效性，同时利用所设计的学习方法找出测试集中的其余待识别目标（char18-char22）。

问题四：根据睡眠脑电数据中特征样本，设计一个睡眠分期预测模型，在尽可能少的训练样本的基础上，得到相对较高的预测准确率，给出训练数据和测试数据的选取方式和分配比例，说明具体的分类识别过程，并结合分类性能指标对预测的效果进行分析。

2. 模型假设

大脑有着数以亿计的神经元，通过相互连接来传递和处理人体信息，因此，脑电信号容易受其他信号干扰，如：心电、肌电、眼电等信号，为了分析其本质，我们对模型进行假定：

假定 P300 脑机接口实验数据的脑电信号滤除了心电、肌电等信号影响，不包括眼电信号

3. 符号说明

符号	意义
EEG	脑电
EOG	眼电
EMG	肌电
ECG	心电
ICA	独立成分分析
SVM	支持向量机
CNN	卷积神经网络
KNN	K 近邻算法
SVR	支持向量回归机
RBF	高斯径向基核函数
GAN	对抗学习网络

4. 问题的分析

4.1 问题一的分析与建模

4.1.1 问题描述与分析

P300 电位是诱发脑电信号的一种，在发生刺激后 300 毫秒范围左右会出现一个正向波峰。使用 P300 脑-机接口对被试者进行测试的时候，如果字符矩阵中某行或某列出现目标字符的时候，被试者的脑电信号中会出现 P300 电位。本题目标是根据已有的被试者脑际实验数据，设计方法，找出被试测试集中的 10 个待识别目标。

本题可以通过判断 P300 出现的位置从而确定目标字符所在的行和列，从而标志出目标字符。具体实验时，由于脑电信号原始数据中含有噪声，因此首先要对数据进行预处理，随后建立二分类神经网络模型并训练出可以识别 P300 电位的网络[2][3][4]，测试数据经过神经网络可以得到 P300 电位出现的位置，最终得到待识别字符。

4.1.2 预处理

1) 去 EOG

脑电（EEG）信号是一种微弱的电生理信号，脑电信号容易受到其他生理信号诸如眼电（EOG）、心电（ECG）、肌电（EMG）的干扰，其中眼电伪迹是一种最主要的干扰成分。当人眨眼或者眼球移动的时候，测量电极处会产生较大的电位变化，从而形成眼电。在采集脑电信号的时候，眼电从其源发出，并弥散到整个头皮，导致采集到的脑电信号产生畸变，形成伪迹。一般来说，心电信号和肌电信号产生的干扰较弱，因此在处理脑电信号时应当去除眼电信号的干扰。目前常见的去除眼电信号干扰的方法有线性回归法和独立成分分析法（ICA），前者在处理时需要 EOG 通道，而后者无需任何参考信号即可工作。

本题的脑电信号采集通道没有说明 EOG 通道，因此这里采用 ICA 方法来消除 EEG 中的 EOG。ICA 是一种基于信号高阶统计量的盲源分离方法，也是应用最为广泛的盲源分离方法。ICA 在假设各个源信号是统计独立的前提下，将观察到的多通道信号分解为若干个独立分量，从而把源信号从混合信号中分离出来。利用 ICA 进行脑电信号去噪时，先对 EEG 信号进行独立分量分析，之后去除其中含干扰伪迹的分量再进行 ICA 重构，即可得到纯净的脑电信号。定义

$$s = (s_1, s_2, \dots, s_m)^T$$

是 m 个相互独立的源信号，在经过未知矩阵 A 线性混合产生 N 维观测信号

$$x = (x_1, x_2, \dots, x_N)^T$$

即

$$x = As$$

ICA 分离的目的就是在分离结果相互独立的情况下，寻找一解混矩阵 W，使得经分离矩阵线性处理后得到的各分量

$$u = (u_1, u_2, \dots, u_n)^T$$

最大限度的逼近各源信号，即

$$u = Wx = WAs$$

原理框图如图 4-1：

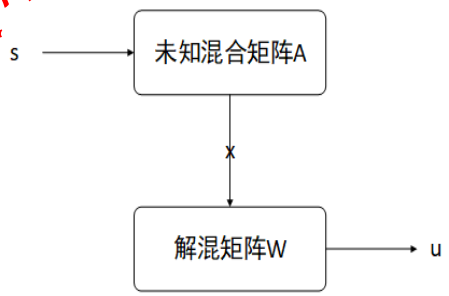


图 4-1 ICA 原理框图

将 P300 脑机接口数据进行 ICA 后可以得到 20 个独立分量。将被试者 1 的脑机数据经过 ICA 后得到的独立分量绘制成头皮地形图、组件图以及 ICA 分解后各分量波形图见图 4-2、4-3、4-4。

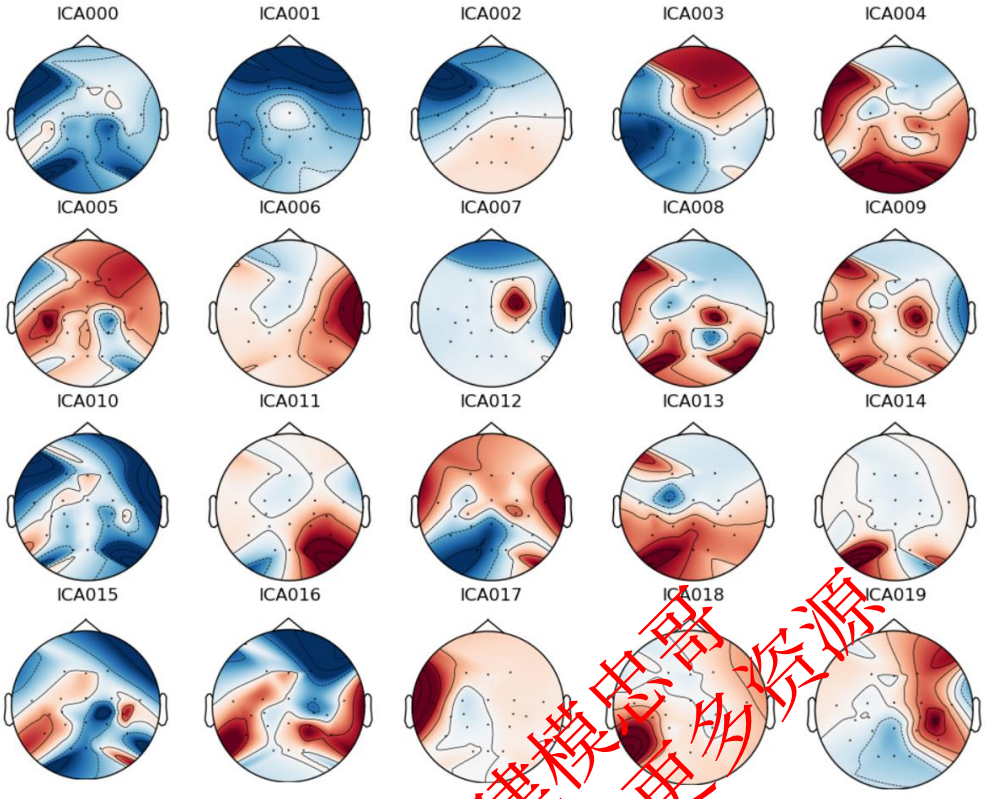


图 4-2 独立成分头皮地形图

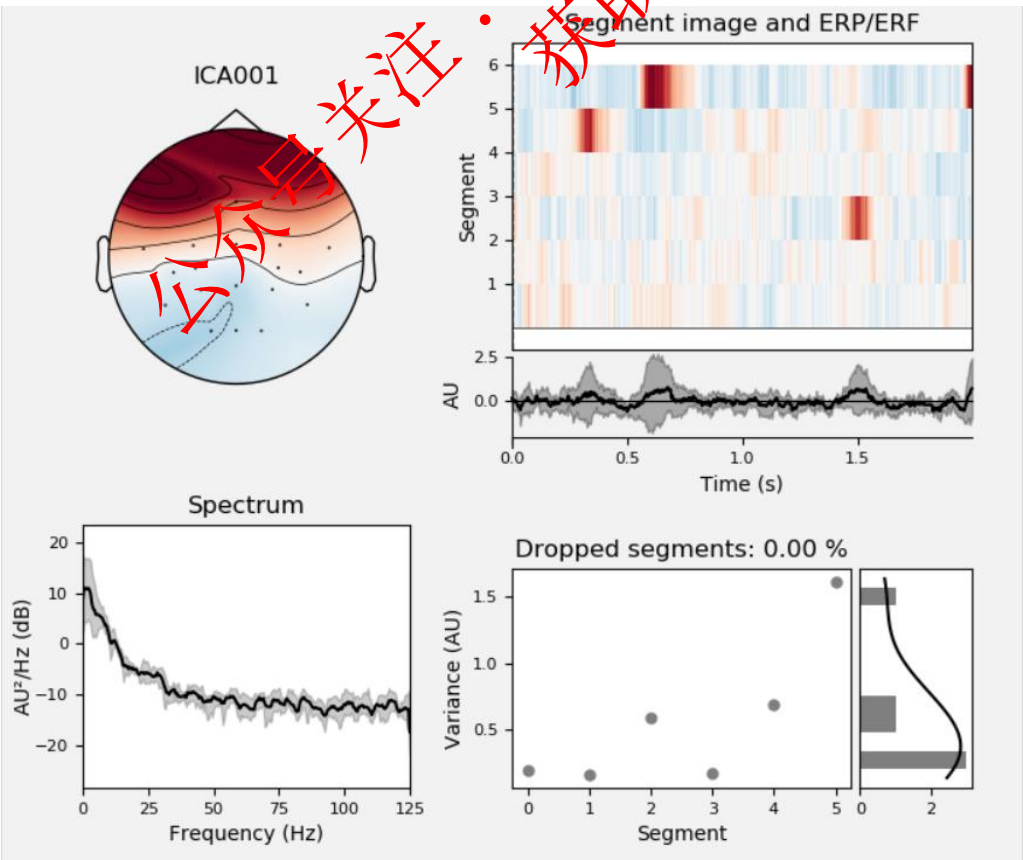


图 4-3 组件图

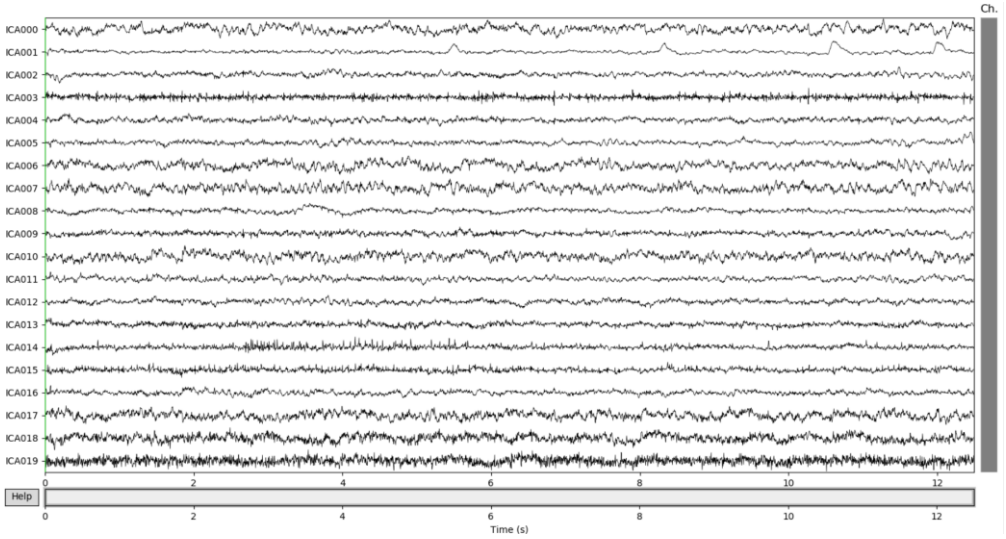
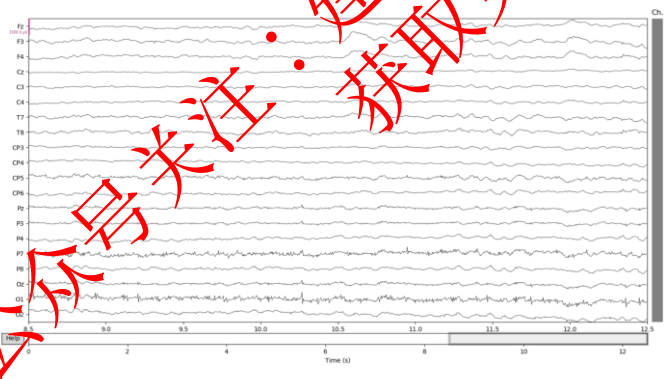


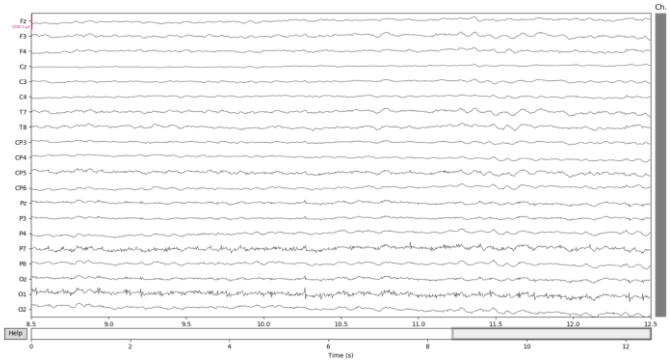
图 4-4 ICA 分解后各分量波形图形

可以观察到独立分量 ICA001 的头皮地形图红色区域分布于头的前部且向脑后部逐渐衰减。组件图中的 ERP 图有小方块出现，且功率频谱图中低频能量高，能量随着频率的增加而降低。同时 ICA001 波形不定时的会出现波动，符合人眨眼的特征。因此可以判断分量 ICA001 为眼动成分。

将脑机信号各通道去除 ICA001 分量后结果如图 4-5，可以观察到在 10.6s 和 12.0s 处基本消除了 EOG 干扰。



(a)去除 ICA001 分量前



(b)去除 ICA001 分量后

图 4-5 去除 ICA001 分量前后对比图

本实验在数据预处理中考虑到被试者存在个体差异，因此需要对每个被试者的训练集和测试集逐个进行 ICA。其中被试者训练集的 ICA 分解后各分量波形图见图 4-6。

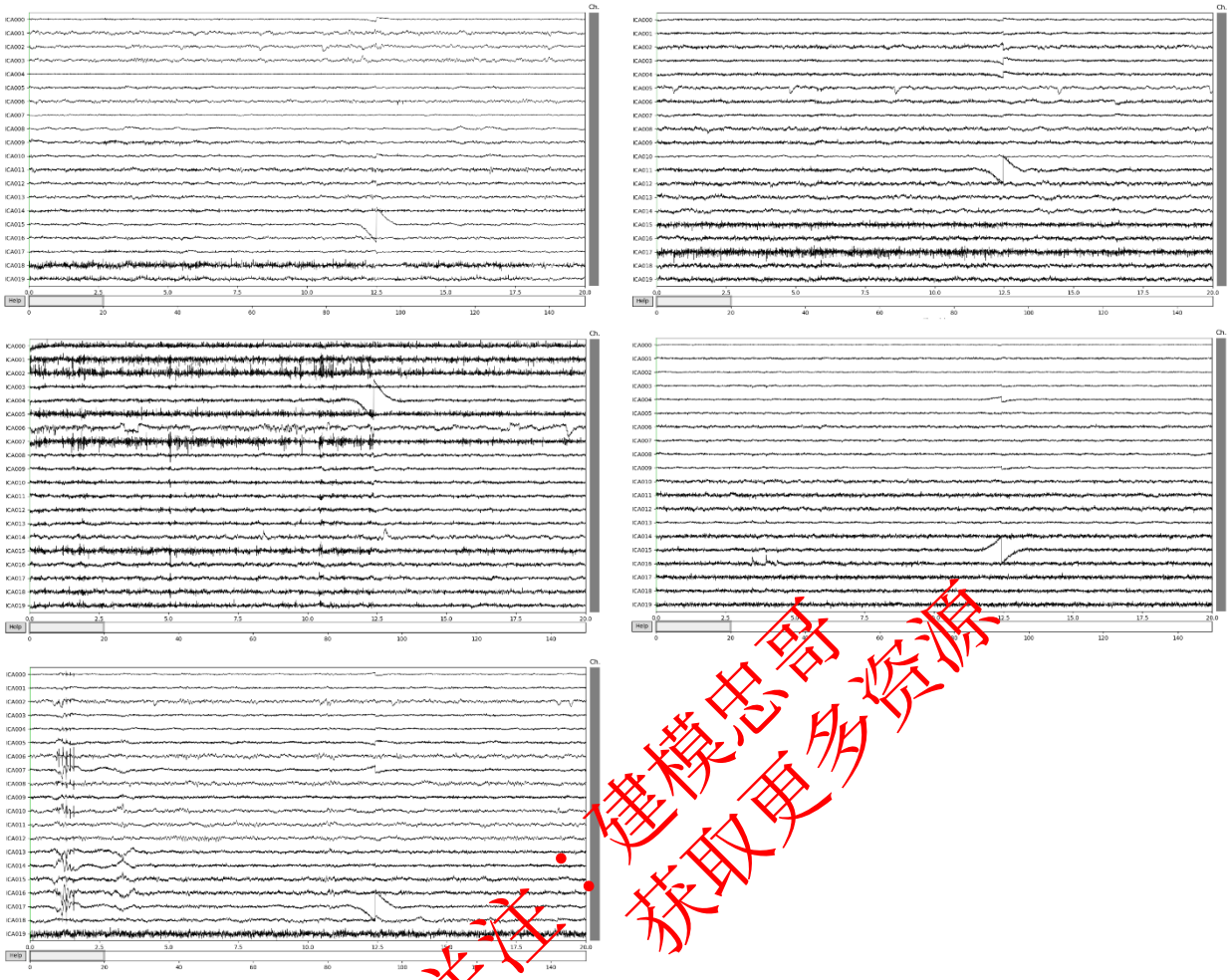


图 4-6 被试者训练集的 ICA 分解后各分量波形图

经过分析筛选出被试者训练集 ICA 分解后的眼动分量分别为 ICA002、ICA005、ICA014、ICA016、ICA002，测试集 ICA 分解后的眼动分量分别为 ICA010、ICA010、ICA013、ICA013、ICA010，将这些 EOG 分量去除最终得到实验预处理数据集。

2) 滤波

上一步去除了 EOG 之后得到的信号由于外界噪声等的存在依旧有较大的震荡，因此这里使用 Butterworth 滤波器对信号进行滤波处理。

滤波器可以对信号中特定频率的频点或该频点以外的频率进行有效滤除，得到一个特定频率的信号，利用滤波器的选频作用，可以滤除干扰噪声或进行频谱分析。本次实验中诱发的 P300 电位是低频信号，频率范围在 2～8Hz 之间，因此选择 0.5～10Hz 的 Butterworth 滤波器对信号进行滤波，去除高频干扰和低频基线漂移。Butterworth 滤波器的特点是通频带内的频率响应曲线最大限度平坦，没有起伏，而在阻频带则逐渐下降为零。在振幅的对数对角频率的波特图上，从某一边界角频率开始，振幅随着角频率的增加而逐渐减少，趋向负无穷大。Butterworth 滤波器可用如下振幅的平方对频率的公式表示：

$$|H(\omega)|^2 = \frac{1}{1+(\frac{\omega}{\omega_c})^{2n}} = \frac{1}{1+\epsilon^2(\frac{\omega}{\omega_p})^{2n}}$$

其中，n为滤波器的阶数，ω_c为截止频率，ω_p为通频带边缘频率，本实验设置 Butterworth 滤波器截止频率为 8Hz，滤波前后的波形图见图 4-7。

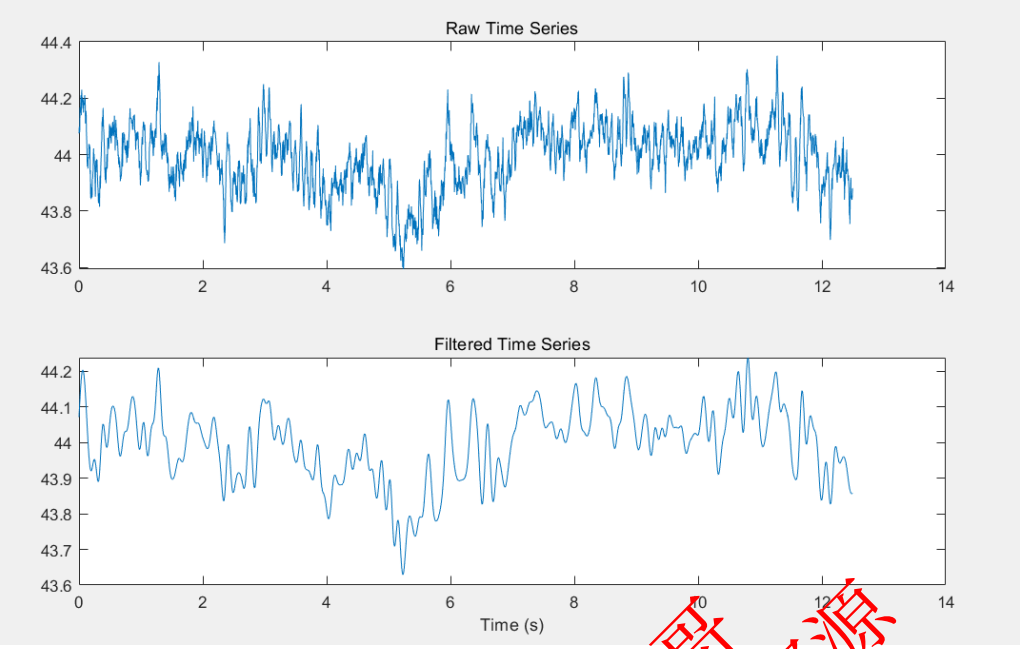


图 4-7 滤波前后波形图对比

3) 标准化

脑电信号数据经过 ICA 去除 EOG 以及滤波后采集通道对应的值差别较大，在后续使用神经网络的过程中可能出现网络收敛过慢和数值问题等，因此需要对数据进行标准化，将数据的平均值设置为 0，方差设置为 1。方法如下：

对序列$x_1, x_2, \ldots, x_n$进行变换：

$$y_i = \frac{x_i - \bar{x}}{s}$$

其中$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$，$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$，标准化前后的脑电信号各通道波形图见图 4-8：

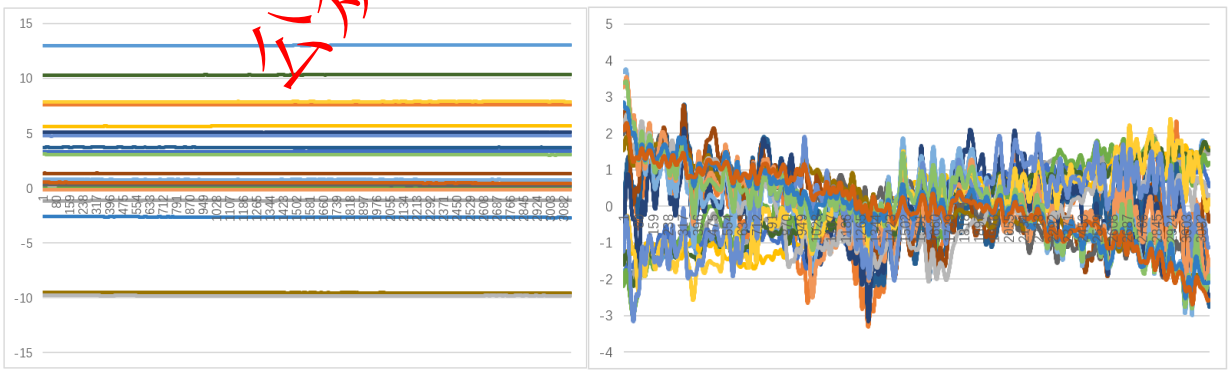


图 4-8 标准化前后脑电信号各通道波形图

4) 基于小波变换的特征提取

将问题中的脑电数据经过小波分解之后得到一组小波细节系数和一个近似系数，这些系数表征了信号在特定时频段的特性，可以作为分类的特征集合来区分 P300 和非 P300 信号。按照一定的层数进行小波分解之后，选择不同尺度下的小波系数和尺度系数可以降低数据的维数，减少运算量，提高分类器的处理速度。对信号进行小波分解之后，小波系数的平方反映的是信号能量的变化情况，因此可以将特定尺度下小波分解的近似系数和细节系数的平方作为特征。

5) 基于时域的能量熵的特征提取

熵是一种热力学参量，用以表征物质分子状态的混乱程度。将热力学中的“熵”引用到信息上，就是“信息熵”，其定义为离散随机事件的出现概率，用来描述信源X的不确定度。信息熵的值越大，表明信号源越不稳定，随机事件出现该状态的概率也越小。

可以分别从时域、频域、时频域、空域来表述信息熵。由于 P300 的特征主要表现在时域，因此本文采用时域能量熵对采集到的脑电信号进行分析，提取出含有 P300 的脑电信号和不含 P300 的脑电信号的差异特征。把信号平均分为n段，第i段信号的能量为Q_i，n段信号的总能量为Q，那么第i段信号的时域能量熵E_i定义如下：

$$E_i = -log_2(\frac{Q_i}{Q})$$

$$\sum_{i=1}^{n} Q_i = Q$$

时域能量熵反映的是信号的能量在时域上的分布情况，某段信号的能量在时域上所占信号总能量的比值越大，那么该段信号的时域能量熵值E_i就越小。反之，某段信号的能量在时域上所占信号总能量的比值越小，那么该段信号的时域能量熵值E_i就越大。在时域中，信号的能量和其波幅的大小呈正相关关系。观察含有 P300 的脑电信号和不含 P300 的脑电信号，很明显地发现，诱发出 P300 电位时段信号的波幅明显升高，所占整个信号能量的比重大，而没有诱发出 P300 电位的脑电信号在整个数据段上的波幅没有太大变化，能量的分布较为均匀。

4.1.3 模型建立与求解

本问题目的是确定被试测试集中的待识别目标，而待识别目标由出现 P300 点的位置确定，因此该问题的关键是判断被试者脑机信号中何处出现 P300 点。对于某个时间段，该时间段要么存在 P300 电位，要么不存在，因此该问题可以转化为二分类问题，即判断测试集样本是否存在 P300 电位。本题在数据预处理后分别使用动态卷积（Dynamic Convolutions）神经网络模型[5]、支持向量机（SVM）线性分类器以及最邻近节点算法（KNN）来训练样本，之后对测试集用各方法进行预测，并比较得到的结果。SVM 和 KNN 模型在后面的章节有具体讲解，本节着重介绍动态卷积神经网路模型。

动态卷积(dynamic convolutions)建立在轻量级卷积的基础上，能够在不增加网络深度或宽度的情况下增加模型的表达能力。动态卷积的基本思路就是根据输入图像，自适应地调整卷积参数，计算公式如下：

$$DynamicConv(X, i, c) = LightConv(X, f(X_i)_{h,i}, i, c)$$

动态卷积内核只是当前时间步长的函数，模型要学习的就是f(·): $R^d \rightarrow R^{H\times k}$的映射关系，具体实现如下：

$$W^Q \in R^{H\times k\times d}, f(X_i) = \sum_{c=1}^{d} W_{h,j,c}^Q X_{i,c}$$

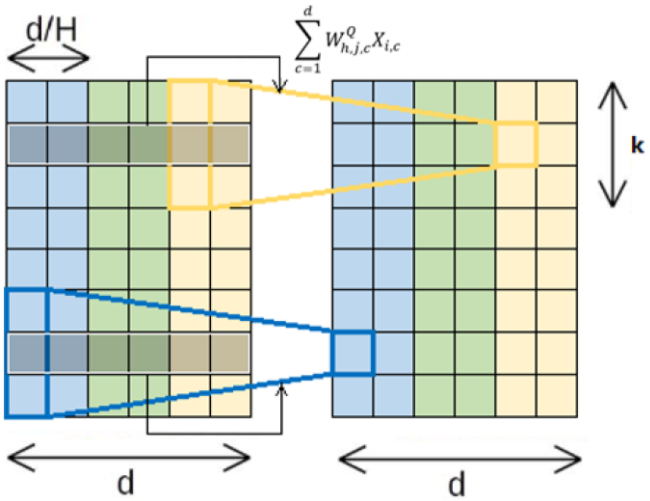


图 4-9 动态卷积模型

另外，产生的$K \times H$个权重，在每一个分割子通道上h上对参数进行归一化以及dropout 操作：

$$\text{softmax}(W)_{h,j} = \frac{\exp W_{h,j}}{\sum_{j'=1}^{k} \exp W_{h,j'}}$$

动态卷积模型整体流程见图 4-10 所示，输入经过输入映射后，通过 GLU，分成两个通道：一个直接通过去往 LConv，另一个用 Dynamic CNN 计算权重，两者结合得到 LConv 的输出，最后经过线性映射得到最终输出。

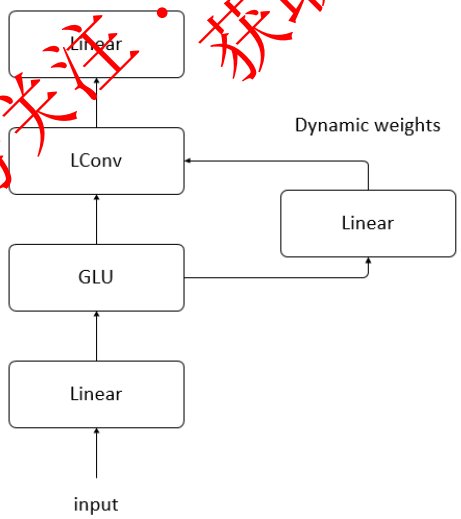


图 4-10 动态卷积模型整体流程图

本题由于脑机信号数据是持续采样的结果，因此需要对信号进行数据分段。P300 是刺激事件出现后约 300-500ms 间记录到的正成分，本实验字符闪烁时长为 80ms，间隔 80ms，两次刺激的时间间隔是 160ms，因此截取每次点亮后 700ms 数据段即得到数据集，如图 4-11 所示。

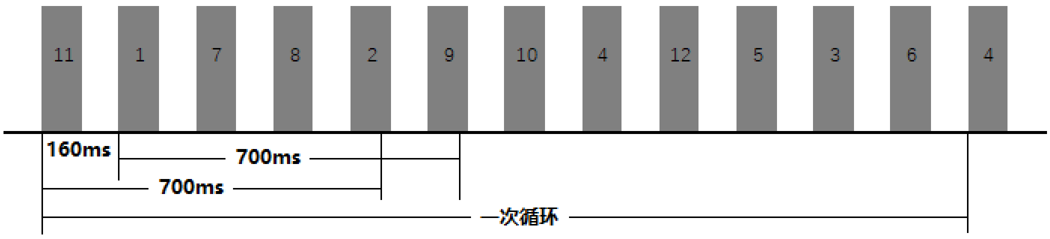


图 4-11 脑电数据分段示意图

脑电数据采样频率为 250Hz，即采样间隔为 4ms，因此每个数据段含有 175 组采样数据，每组采样数据有 20 个脑机通道，所以每个数据集含有 175*20=3500 维数据，将数据集使用动态卷积网络训练，随后对测试集进行预测，最终得到 P300 电位的位置。

4.1.4 实验结果分析

本实验使用动态卷积神经网络、支持向量机以及最邻近节点算法构建模型，训练时随机选取 80%样本作为训练集，20%样本作为测试集，得到的三种模型预测结果及运行速率如表 4-1，其中 SVM 准确率最高，且运行速度快。由于选取时间序列长度较短，动态卷积神经网络的准确率较低，并且速率较慢。KNN 算法准确率最低而速率较快。故 SVM 模型在本实验中的效果最佳。

表 4-1 三种模型的预测结果及运行速率

模型	准确率	速率
动态卷积神经网络	71.8%	较慢
支持向量机（SVM）	86.7%	快
最邻近节点算法（KNN）	68.4%	较快

鉴于以上分析，本实验采用 SVM 模型来确定被试测试集中的 10 个待识别目标，得到的识别结果如表 4-2。

表 4-2 识别结果

被试者	识别结果
S1	M F 5 2 I T K X A 0
S2	M F 5 2 I T Q X A
S3	M F Y 2 I T K X A
S4	M E 5 V I H K X A 0
S5	M F 5 2 I T K V A 0
综合	M F 5 2 I T K X A 0

经过多次计算对比，综合分析最终得出被试测试集中的 10 个待识别目标为 M，F，5，2，I，T，K，X，A，0。

4.2 问题二的分析与建模

4.2.1 问题描述与分析

问题二的展开是承接问题一进行的，由于采集的原始脑电数据量较大，这样的信号势必包含较多的冗余信息，在 20 个脑电信号采集通道中，无关或冗余的通道数据不仅会增加系统的复杂度，而且会影响分类识别的准确率和性能。基于以上分析，问题二要求设计一个通道选择算法，通道组合的数量小于 20 大于等于 10。由于不同被试间会存在个体差异，

可以首先给出针对每个被试的、更有利于分类的通道名称组合，再基于通道选择的结果，进一步分析对于所有被试都较适用的一组最优通道名称组合，并给出具体分析过程。

为了对问题有一个直观的认识，以被试一的 char01 字符实验为例，分别绘制出其 20 个通道的电位变化图，如图 4-12 所示。从图中可以明显的看出一些通道具有相似的电位变化趋势，这也势必包含有较多的冗余信息，因此，对通道进行选择便是要保留那些对特征确定有显著影响的通道。

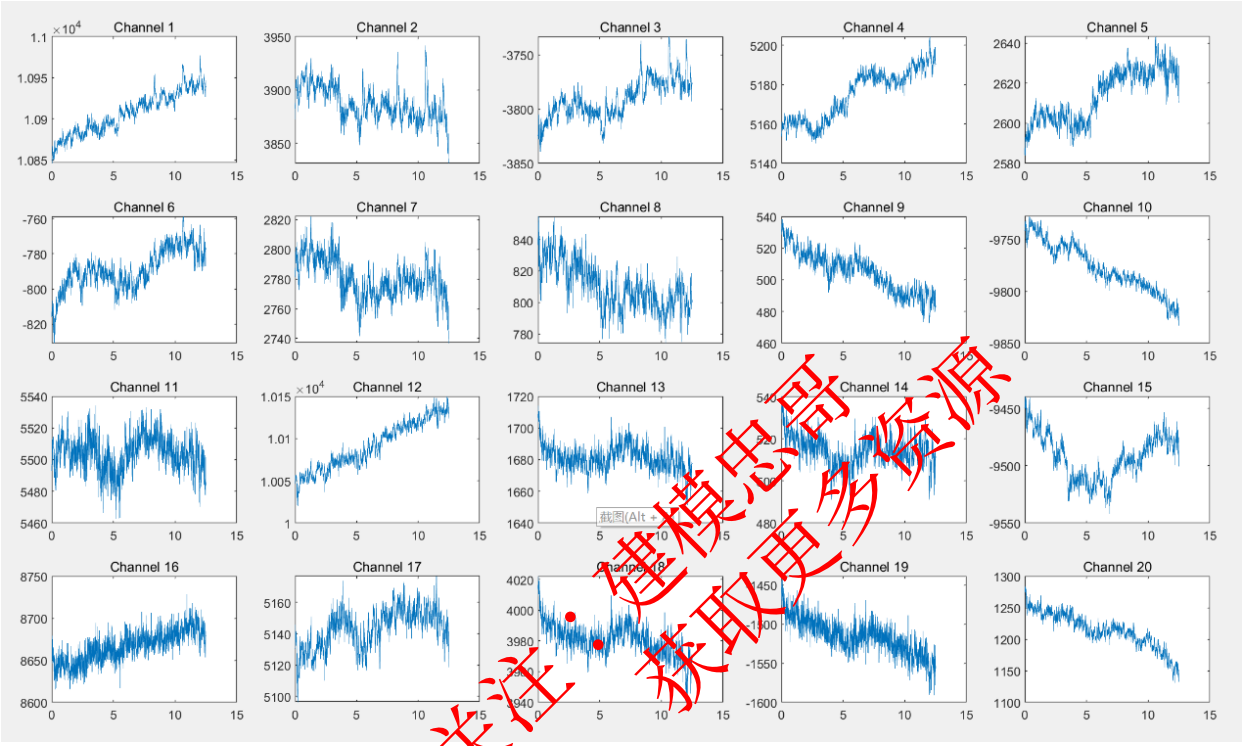


图 4-12 被试一 20 个通道的电位变化图

4.2.2 模型建立

这里我们沿用问题一中的 SVM 分类模型，同时为了实现通道选择的功能，再加入递归特征消除步骤，建立 SVR-RFE 模型进行通道选择。支持向量回归机（Support Vector Regression，SVR）是以结构风险最小化原则为基础的新型机器学习方法，其目标是构建一个最优的超平面，使训练样本距离最优超平面误差最小，以达到对未知样本具有更好的拟合性能和泛化能力。其主要思路是将已知的训练样本数据集由低维转向到高维空间的非线性映射，从而达到在低维非线性的问题转化为高维线性的目的，最后在高维空间完成线性回归，随后通过逆映射将回归结果返输入到空间，直至完成整个样本数据集的回归结果。SVR 的实现过程是依托于核函数完成的，其决策函数为：

$$f(x)=\sum_{i=1}^{D}\omega_i\varphi_i(x)+b$$

式中，$\varphi_i(x)$为输入空间到高维空间的非线性特征变量，ω_i为权系数，b为偏置量。为求解权系数和偏置量，引入最小化式如下：

$$R(f)=\frac{1}{2}\|\omega\|^2+c\frac{1}{n}\sum_{i=1}^{n}L_\varepsilon(y_i,f(x_i))$$

式中，L_ε为损失函数，c为惩罚因子，不敏感损失函数ε作为结构最小化风险的估计问

题。此外，引入 Lagrange 算子，通过对相关参数做偏导数处理后得到 SVR 的回归估计函数如下：

$$f(x) = \sum_{i=1}^{N}(\alpha_i - \alpha_i{}^*)K(x_i, x_j) + b$$

式中，$\alpha_i, \alpha_i{}^* \geq 0(i = 1,2,\ldots,N)$为 Lagrange 算子，$K(x_i, x_j)$为核函数，$N$为输入参数的数目。

SVR 的构建中最主要是核函数的选取，一般情况下选择高斯径向基核函数（Radial Basis Function，RBF）作为 SVR 模型中的核函数，其表达式如下，其中σ为数据标准差。

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right)$$

一个典型的特征选择过程包括 4 个步骤：生成特征子集、特征子集评价、终止条件判定和结果验证。生成特征子集的过程实际上是一个搜索过程，常用的特征选择搜索策略有：随机搜索策略、启发式搜索策略（如序列前向选择策略、序列后向选择策略）等，然后使用评价准则对每个候选子集进行比较分析。

SVR-RFE 方法使用的是序列后向选择策略，利用选择后的特征构成的样本训练得到的分类器的性能指标作为评判准则，目前已应用于多个领域中。SVR-RFE 方法通过不断消除当前特征子集中对最小化目标函数贡献最小的特征，直到特征子集为空，输出结果是基于所有特征的重要程度排序表。该方法排序在前面的特征组合的效果总是最好的，可有效去除冗余特征。

算法 1：线性 SVR-RFE 特征选择算法：

输入　　训练样本集$X_0 = [x_1, x_2, \ldots, x_k, \ldots, x_N]^T$，

$y = [y_1, y_2, \ldots, y_k, \ldots, y_N]^T$，特征指标集$S = [1,2,\ldots,d]$，空集$r$

输出　　经过排序的特征指标集r

步骤 1：获得当前训练样本$X = X_0(:, S(-i))$。

步骤 2：给定参数后训练 SVR 回归器$SVRtrain(X, y, C, \gamma)$。其中，$C$为惩罚因子，$\gamma$为核函数参数。

步骤 3：计算权重向量，$\omega = \sum_{i=1}^{N}(\alpha_i - \alpha_i{}^*)\varphi(x_i)$。

步骤 4：计算排序系数，$rank(i) = (\omega_i)^2$。

步骤 5：找出具有最小排序系数的特征，$f = argmin(rank)$。

步骤 6：更新排序特征指标集，$r = [S(f), r]$。

步骤 7：消去最小得分特征属性，$S = S(1{:}f-1, f+1{:}length(S))$。

步骤 8：如果$S = [\,]$，结束，否则转入步骤 1。

在线性 SVR-RFE 特征选择算法的迭代过程中，具有最小排序系数的特征将被移除，然后 SVR 对剩下特征构成的样本重新训练以获取新的排序系数。SVR-RFE 算法通过迭代执行该过程，得到一个特征排序集合。在特征集合中，排在前面的单个特征不一定使 SVR 回归器获得最好的学习性能，需要多个特征组合在一起才会使回归器获得最佳的泛化性能。

4.2.3 模型求解及结果分析

由于被试者存在个体差异，因此针对每个被试者，无关或冗余的通道数据存在差异，从而影响分类识别的准确率和性能。从而得到对每个被试者的一个最优通道选择结果。

针对每个被试者，通过线性 SVR-RFE 特征选择算法得到通道选择的特征权重，然后

进行排序，取前 10 个作为该被试者的通道选择。得到的结果如表 4-3。

表 4-3 各个被试者的通道选择组

被试者	通道选择
S1	Fz、F3、C3、CP3、CP4、Pz、P4、P7、P8、O2
S2	Fz、C4、T7、Pz、P3、P4、P7、Oz、O1、O2
S3	Fz、F3、C4、T8、CP3、Pz、P4、P7、Oz、O2
S4	Fz、F3、T7、P3、P4、P7、P8、Oz、O1、O2
S5	Fz、F3、T8、CP4、Pz、P4、P7、P8、Oz、O2

针对所有被试者，通过线性 SVR-RFE 特征选择算法得到一组泛化性较强的适用所有被试者的一组最优通道名称组合，即 Fz、F3、C3、C4、CP4、Pz、P4、P7、Oz、O2。

从原始数据的波形图上分析可以印证我们选择的通道具有一定的科学性，删除了一些冗余通道的信号，并通过赛题给出的测试数据（char13-char17）的结果印证，选择的通道是存在其合理性。

4.3 问题三的分析与建模

4.3.1 问题描述与分析

问题三承接问题一和问题二，将监督学习分类问题转化为半监督学习分类问题。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，题目要求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，在问题二所得一组最优通道组合的基础上，设计一种学习的方法，并利用问题二的测试数据（char13-char17）检验方法的有效性，同时利用所设计的学习方法找出测试集中的其余待识别目标（char18-char22）。

4.3.2 模型建立

由于训练样本中只有适量样本为有标签样本，这样的半监督学习问题会带来训练样本不足以致欠拟合的问题，因此首先构建生成对抗网络扩增样本。GAN 的核心思想来源于博弈论的纳什均衡，它设定双方分别为生成器和判别器，生成器的目的是尽量学习真实的数据分布，而判别器的目的是尽量正确判别输入数据是来自真实数据还是来自生成器。GAN 中的生成器和判别器需要不断优化，各自提高生成能力和判别能力，其学习优化过程就是寻找二者之间的一个纳什均衡[6]。

GAN 的系统结构主要包括生成器和判别器。首先，在给定生成器G的情况下，最优化判别器D。采用基于 Sigmoid 的二分类模型的训练方式，判别器D的训练是最小化交叉熵的过程，其损失函数表示为：

$$O^D(\theta_D,\theta_G)=-\frac{1}{2}E_{x-P_{data}(x)}lgD(x)-\frac{1}{2}E_{z-P_z(z)}\lg\left(1-D\left(g(z)\right)\right)$$

式中，x采样于真实数据分布$P_{data}(x)$，z采样于先验分布$P_z(z)$，例如高斯噪声分布。

给定生成器G，最小化上式可以得到最优解。对于任意的非零实数m和n，且实数值$y\in[0,1]$，表达式为：

$$\emptyset=-mlgy-nlg(1-y)$$

上式在$\frac{m}{m+n}$处得到最小值。因此，给定生成器G的情况下，上述损失函数最小值为判别

器最优解。

$$D_G^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)}$$

$D(x)$代表x来源于真实数据而非生成数据的概率。当输入数据采样自真实数据x时，D的目标是使得输出概率$D(x)$趋近于 1，而当输入来自生成数据$G(z)$时，D的目标是正确判断数据来源，使得$D\big(G(z)\big)$趋近于 0，同时G的目标是使得其趋近于 1。生成器G损失函数可表示为：

$$O^G(\theta_G) = -O^D(\theta_D, \theta_G)$$

其优化问题是一个极值问题，GAN 的目标函数可以描述为：

$$\min(G)\max(D)\ \{f(D,G) = E_{x-P_{data}(x)} lgD(x) + E_{z-P_z(z)} \lg(1 - D\big(g(z)\big))\}$$

GAN 模型需要训练模型D最大化判别数据来源于真实数据或者伪数据分布$G(z)$的准确率，同时，需要训练模型G最小化$\lg(1 - D\big(g(z)\big))$。

GAN 学习优化的方法为：先固定生成器G，优化判别器D，使得D的判别准确率最大化；然后固定判别器D，优化生成器G，使得D的判别准确率最小化。当且仅当$P_{data} = P_g$时达到全局最优解。

扩增样本之后，使用原型网络进行分类训练。原型网络利用支撑集S为每个类别计算原型向量c_k,然后根据查询集样本x^*与原型向量的距离来对样本进行分类。原型网络最早应用于图像处理领域，在小样本图像分类任务[7]中取得了很好的效果。原型网络的原理如图 4-13 所示。

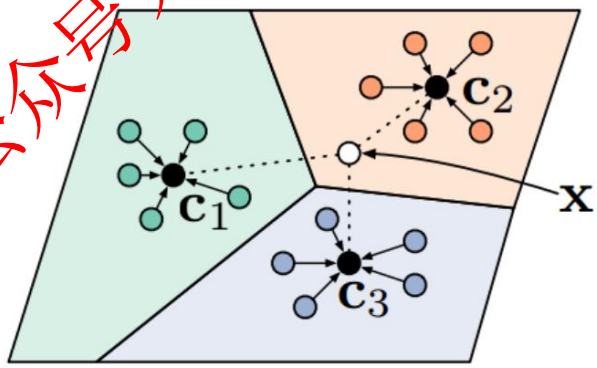


图 4-13 原型网络的原理

原型网络(Prototypical Networks)的思想是：在嵌入空间中,每个类别都存在一个特殊的点，称为类的原型。利用神经网络的非线性映射将输入图像映射到嵌入空间中，此时嵌入空间中训练集的加权平均值就是类的原型。该类别中每个样本的嵌入空间表示都会围绕类原型进行聚类。预测分类的时候，将测试图像也映射到嵌入空间中，计算与训练集类别的各个类原型间的距离，即可进行分类。原型网络将复杂的分类问题转化成了在特征向量空间中的最近邻问题，对于处理小样本数据集下的分类问题可以得到较好的效果。

假设我们有训练集：

$$D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

假设我们一共有K个类别，那么我们就从D中为每个类别分出K个子集，每一次迭代就

对一个子集随机抽样得到支持集（Support Set）和查询集（Query Set），需注意支持集和查询集不能有重复，利用支持集计算原型：

$$c_k = \frac{1}{S_k} \sum_{(x_i, y_i) \in S_k}$$

得到每个类的原型之后，再用查询集计算每个样本对应每个类的概率：

$$P_\emptyset(y = k|x) = \frac{\exp\left(-d(f_\emptyset(x), c_k)\right)}{\sum_{k'} \exp\left(-d(f_\emptyset(x), c_{k'})\right)}$$

假设样本对应的真实类别为k'，那么就可以定义损失函数：

$$J(\emptyset) = -\log\left(P_\emptyset(y = k'|x)\right)$$

网络的训练目标时希望最小化损失，也就是最大化x正确分类的概率。通过最小化损失函数，从而优化最初的 Embedding 网络，以实现网络训练的目的。整个模型比较特别的地方就是训练过程中把数据划分成支持集和查询集。

4.3.3 模型求解及结果分析

网络结构由两部分组成，第一部分为基于生成对抗网络的数据倍增模块，第二部分为基于原型网络的小样本学习分类器。训练过程如下：首先使用 GAN 倍增训练样本，以弥补训练中样本数的不足，然后使用原型网络将样本特征映射到高维空间内，对类别相同的样本点映射到相邻近的区域，第三步，基于训练的映射函数对测试数据和样本进行验证，将样本归为离之最近的原型向量的类别。

采用问题二所得的一组最优通道组合:Fz、F3、C3、C4、CP4、Pz、P4、P7、Oz、O2,将其他通道信号数据去除，保留所选取通道组合的脑电信号数据，通过给定的问题二中的测试数据(char13-char17)检验，可以验证该方法的有效性，将模型用于预测其余待识别目标，得到表 4-4，可知其余识别目标为：T、K、X、A、0。

表 4-4 其余识别目标识别结果

被试者	识别结果
S1	T K F A 0
S2	T Q X S
S3	T E X A
S4	H K X A 0
S5	T K V A 0
综合	T K X A 0

4.4 问题四的分析与建模

4.4.1 问题描述与分析

问题四的题目描述为：根据睡眠脑电数据所给出的特征样本，设计一个睡眠分期预测模型，在尽可能少的训练样本的基础上，得到相对较高的预测准确率，给出训练数据和测试数据的选取方式和分配比例，说明具体的分类识别过程，并结合分类性能指标对预测的效果进行分析。

在睡眠脑电数据中提供了 3000 个睡眠脑电特征样本及其标签，分为 5 个类别，即清醒期（6）、快速眼动期（5）、睡眠 I 期（4）、睡眠 II 期（3）、深睡眠期（2），根据所给出的 4 个特征参数（分别对应了脑电信号在“8-13Hz”，“14-25Hz”，“4-7Hz”和“0.5-4Hz”频率范围内的能量占比），即“Alpha”、“Beta”、“Theta”、“Delta”，用于分类模型的输入，训练得

到训练模型，将其用于预测睡眠分期。

问题的求解的目标在于找出训练数据和测试数据的选取方式和分配比例，使得训练样本尽可能少的基础上得到较高的预测准确率。

4.4.2 模型建立

1) 支持向量机分类模型

支持向量机（support vector machines，SVM）[9]是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，这一特点使它有别于感知机。通过设计 SVM 的核函数，也可以使得它变为实质上的非线性分类器，该模型的学习策略使使得间隔最大化，可以形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。

支持向量机分类模型的优势在于它在小样本训练集上能够得到比其他算法好很多的结果，并且该模型算法本身的优化目标使结构化风险最小，而不是经验风险最小，因此模型本身具有优秀的泛化能力。同时通过对数据分布的结构化描述，也减低了对数据规模和数据分布的要求。

支持向量机学习的基本思想是求解能够正确划分训练数据集并且几何间隔最大的一个分离超平面：$w \cdot x + b = 0$，对于线性可分的数据集来说，这样的超平面有无穷多个，但对支持向量机来说，几何间隔最大的分离超平面只有一个。

假定一个特征空间上的训练数据集为

$$T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

其中，$x_i \epsilon R^n, y_i \epsilon \{+1, -1\}, i = 1,2, \ldots, N$，$x_i$为第$i$个特征向量，$y_i$为类标记，当$y_i = +1$时为正例，当$y_i = -1$时为负例，假定训练数据集时线性可分的。

将超平面关于样本点(x_i, y_i)的几何间隔定义为：

$$\gamma_i = y_i\left(\frac{w}{||w||} \cdot x_i + \frac{b}{||w||}\right)$$

由此可以得到超平面关于所有样本点的几何间隔的最小值，即支持向量到超平面的距离为：

$$\gamma = \min_{i=1,2,\ldots,N} \gamma_i$$

模型的优化目标是最大化几何间隔，可以形式化为最大化最小问题，目标转化为：

$$\max_{w,b} \gamma = \max_{w,b} \min_{i=1,2,\ldots,N} \gamma_i$$

$$s.t.\ y_i\left(\frac{w}{||w||\gamma} \cdot x_i + \frac{b}{||w||\gamma}\right) \geq 1$$

令

$$\boldsymbol{w} = \frac{w}{||w||\gamma}$$

$$\boldsymbol{b} = \frac{b}{||w||\gamma}$$

最终目标是最大化γ，实际上等价于最大化$\frac{1}{||\boldsymbol{w}||}$，也就等价于最小化$\frac{1}{2}||\boldsymbol{w}||^2$，因此 SVM 模型的求解目标归纳为：

$$\min_{w,b}\frac{1}{2}||w||^2$$

$$s.t.\ y_i(w\cdot x_i+b)\geq 1$$

将带有不等式约束的上述目标函数转换为无约束的拉格朗日目标函数：

$$L(w,b,\alpha)=\frac{1}{2}||w||^2-\sum_{i=1}^{N}\alpha_i(y_i(w\cdot x_i+b)-1)$$

其中，$\alpha_i\geq 0$为拉格朗日乘子，令

$$\theta(w)=\max_{\alpha_i\geq 0}L(w,b,\alpha)$$

由样本点的约束易知：

$$y_i(w\cdot x_i+b)\begin{cases}<1,\ \text{样本点在可行解区域外}\\ \geq 1,\ \text{样本点在可行解区域内}\end{cases}$$

由此可以得到新的目标函数：

$$\theta(w)=\begin{cases}\frac{1}{2}||w||^2,\ \text{x 在可行解区域外}\\ +\infty,\ \text{x 在可行解区域内}\end{cases}$$

于是约束问题就转化为：

$$\max_{w,b}\theta(w)=\min_{w,b}\max_{\alpha_i\geq 0}L(w,b,\alpha)$$

利用拉格朗日函数的对偶性，将最小和最大的位置交换一下得到：

$$\max_{\alpha_i\geq 0}\min_{w,b}L(w,b,\alpha)$$

要使得$\min_{w,b}\max_{\alpha_i\geq 0}L(w,b,\alpha)=\max_{\alpha_i\geq 0}\min_{w,b}L(w,b,\alpha)$，则需要满足 KKT 条件，即要求满足：

$$\begin{cases}\alpha_i\geq 0\\ y_i(w_i\cdot x_i+b)-1\geq 0\\ \alpha_i(y_i(w_i\cdot x_i+b)-1)=0\end{cases}$$

令$\frac{\partial L(w,b,\alpha)}{\partial w}=0,\frac{\partial L(w,b,\alpha)}{\partial b}=0$可得：

$$\min_{w,b}L(w,b,\alpha)=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i\cdot x_j)+\sum_{i=1}^{N}\alpha_i$$

同理，根据拉格朗日的对偶问题，将求解极大转换为求解极小，可以得到：

$$\min_{\alpha}\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(x_i\cdot x_j)-\sum_{i=1}^{N}\alpha_i$$

$$s.t.\ \sum_{i=1}^{N}\alpha_i y_i=0$$

$$\alpha_i\geq 0,i=1,2,\dots,N$$

根据前述推导，可以得到下列式子成立：

$$\begin{cases} \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i x_i \\ \sum_{i=1}^{N} \alpha_i y_i = 0 \end{cases}$$

由此，可以知道对于任意训练样本(x_i, y_i)，总有$\alpha_i = 0$或者$y_i(\boldsymbol{w} \cdot x_i + \boldsymbol{b}) = 1$。若$\alpha_i = 0$，则该样本不会再最后求解模型参数的式子中出现；若$\alpha_i > 0$，则必有$y_i(\boldsymbol{w} \cdot x_i + \boldsymbol{b}) = 1$，所对应的样本点位于最大间隔边界上，即支持向量。

对于输入空间中的非线性分类问题，可以通过非线性变换将它转化为某个维度的特征空间中的线性分类问题，通常采用核函数替换线性支持向量机中的内积，核函数表示，通过一个非线性转换后的两个实例间的内积。具体地，$K(x,z)$是一个函数，或正定核，意味着存在一个从输入空间到特征空间的映射$\phi(x)$，对任意输入空间中的x,z，有

$$K(x,z) = \phi(x) \cdot \phi(z)$$

在线性支持向量机学习的对偶问题中，用核函数$K(x,z)$替代内积，求解得到的就是非线性支持向量机

$$f(x) = sign(\sum_{i=1}^{N} \alpha_i^* y_i K(x, x_i) + b)$$

2) K 近邻算法分类模型

K 近邻算法(k-Nearest Neighbor, KNN)[10]是一种基本分类与回归方法，是一种监督学习算法。该算法的思想在于计算已知类别中数据集的点与当前点的距离，按照距离递增次序排序，然后选取与当前点距离最小的 K 个点，确定前 K 个点所在类别的出现频率，返回前 K 个点出现频率最高的类别作为当前点的预测分类。该算法的优点在于精度高，对异常值敏感，没有假定输入数据。由于脑电信号存在较多冗余的信息，存在眼电、心电、肌电等干扰信号，很容易出现异常值，因此，K 近邻算法分类模型很适合对脑电信号的 4 个特征预测睡眠分期，存在它的优势性。

K 近邻算法分类模型的三个基本要素有三点：K 值的选择、距离度量、分类决策规则。

(1) K 值的选择

K 值是 K 近邻算法中的一个超参数，它的选取直接影响着模型的性能，如果 K 值设置的比较小，那么得到的模型更加复杂更加精确，且更加容易过拟合；如果 K 值设置的较大，那么得到的模型趋于简单化，此时较远的训练数据点也会起到预测作用，容易出现欠拟合。

(2) 距离度量

距离的度量描述了测试样本与训练样本的邻近程度，这个邻近程度就是 K 个样本选择的依据，在 K 邻近算法中，如果特征是连续的，那么距离函数一般用曼哈顿距离或欧式距离，如果特征是离散的，一般距离函数选用汉明距离。

曼哈顿距离的本质是在 KNN 中样本特征每一个维度上的差值的和：

$$d(I_1, I_2) = \sum_{P} |I_1^P - I_2^P|$$

欧式距离的本质是在 KNN 中样本特征每一个维度上的差值的平方和开根号

$$d(I_1, I_2) = \sqrt{\sum_{P} (I_1^P - I_2^P|)^2}$$

汉明距离的式子如下：

$$d(I_1, I_2) = \sum_P |I_1 - I_2|$$

$$\begin{cases} I_1 = I_2, & |I_1 - I_2| = 0 \\ I_1 \neq I_2, & |I_1 - I_2| = 1 \end{cases}$$

(3) 分类决策规则

常用的分类决策规则是选取 K 个近邻训练数据中类别出现次数最多的作为输入新实例的类别，即要确认前 K 个点所在类别的出现概率，然后离散分类返回前 K 个点出现频率最多的类别作为预测的类别。

4.4.3 模型求解及结果分析

1) 支持向量机分类模型

支持向量机分类模型具有一个超参数$gamma$，将睡眠脑电数据分割成训练集和测试集，需要满足训练集和测试集的样本分别是均匀的，即满足训练集和测试集的 5 个类别数据都是均匀分布的。其中测试集占全部数据集的 80%，此时得到模型测试准确率与$gamma$超参数的一个折线图，见图 4-14。

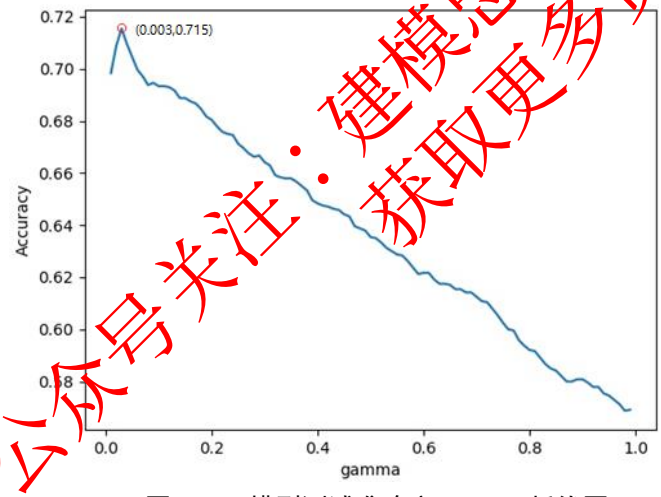


图 4-14 模型测试准确率-gamma 折线图

从图 2-1 中可以得出，当$gamma$参数设定为 0.03 时，准确率可以达到 0.715。因此，我们选用$gamma$参数为 0.03 的 SVM 分类模型用于预测。

为了尽可能在少的的训练样本的基础上得到相对较高的预测准确率，我们需要同时考虑训练样本的占比和预测准确率，给定参数$gamma = 0.03$的基础上，讨论在每一种测试数据占比的情况下的模型测试准确率，得到模型测试准确率与测试数据占比的一个折线图，见图 4-15。

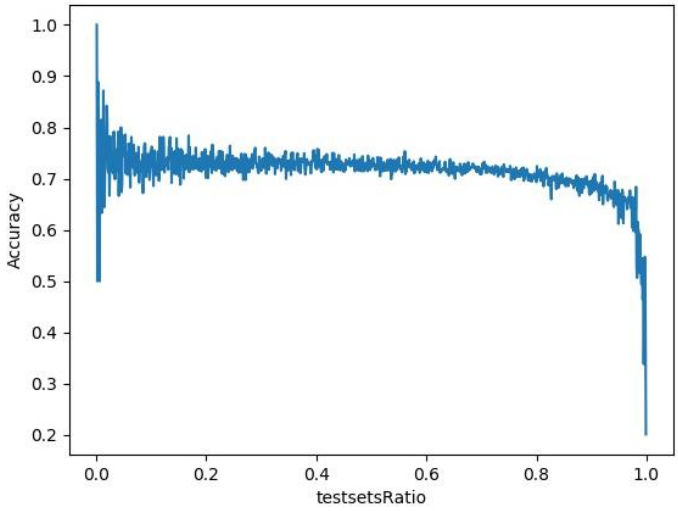


图 4-15 模型测试准确率-测试集比例折线图

从图 4-15 中可以得到，当测试集占比越大时，模型的准确率缓慢下降，考虑到测试存在一定的偶然性的因素，且要满足在尽可能少的训练样本的基础上得到较高的准确率，这个模型在训练样本占比为 15.0%、测试样本占比为 85.0%时，模型的测试准确率达到 71.50%。

当训练样本占比为 15.0%，测试样本占比为 85.0%、超参数$gamma = 0.03$的模型分类准确率到 71.50%，分别画出特征空间的二维散点图(见图 4-16)和三维散点图(见图 4-17)。

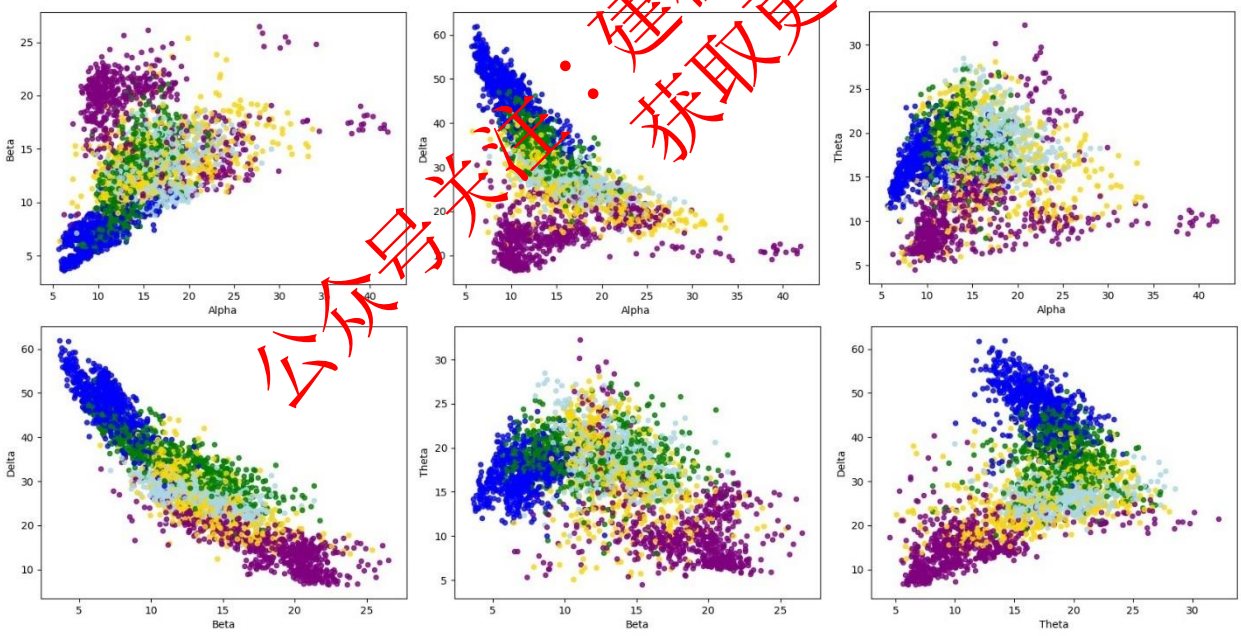


图 4-16 SVM 模型二维散点分类图

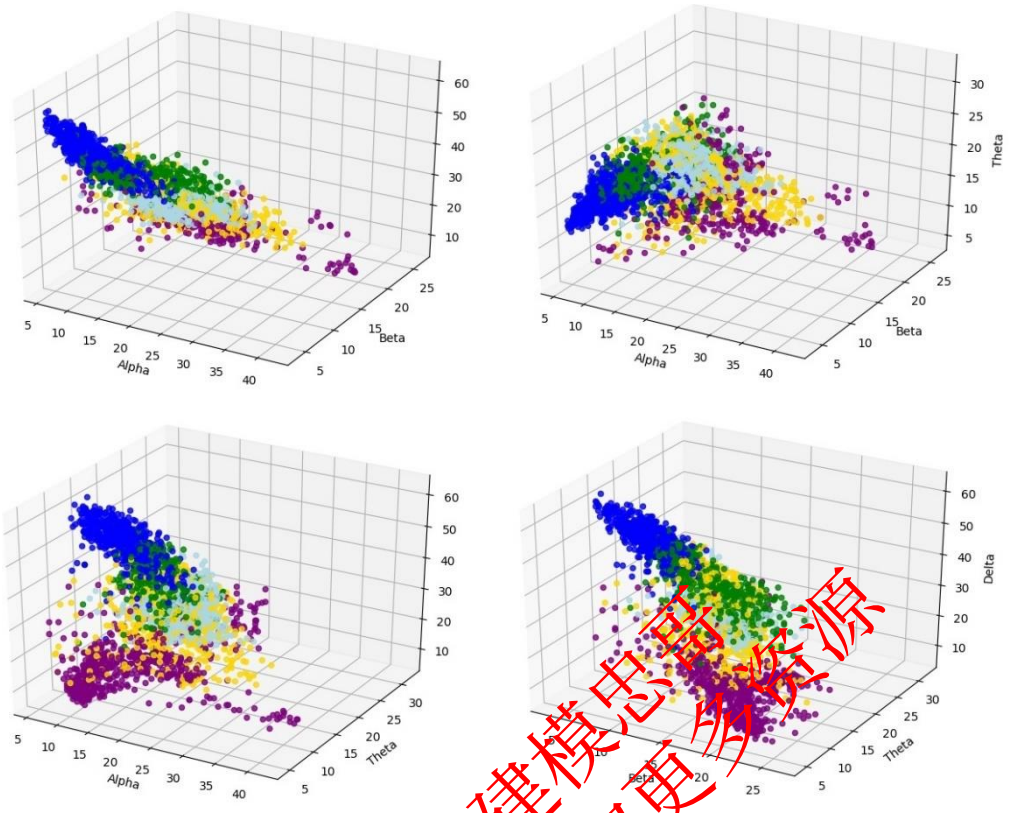


图 4-17 SVM 模型三维散点分类图

由图 4-16 的二维散点分类图和图 4-17 的三维散点分类图可以看出分类的效果具有一定的成效性，其中每个特征维度都对分类模型有一定的影响程度。

2) K 近邻算法分类模型

K 近邻算法分类模型的关键点在于K值的选取，与 SVM 模型输入一致，将睡眠脑电数据分割成训练集和测试集，其中测试集占全部数据集的 80%，此时得到模型测试准确率与K值的一个折线图，见图 4-18。

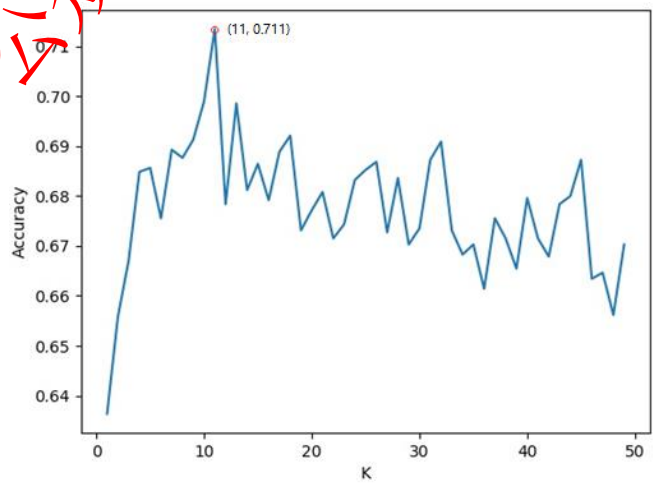


图 4-18 模型测试准确率-K 折线图

从图 4-18 中可以得出，当 K 值取 11 的时候，模型准确率可以达到 0.711。因此，我们选用K值为 11 的 K 近邻算法分类模型用于预测。

为了尽可能在较少的训练样本的基础上得到相对较高的预测准确率，我们需要同时考虑训练样本的占比和预测准确率，给定$K = 11$的基础上，设定 KNN 模型，讨论在每一种

测试数据占比的情况下的模型测试准确率，得到测试数据占比与模型测试准确率的一个折线图，见图 4-19。

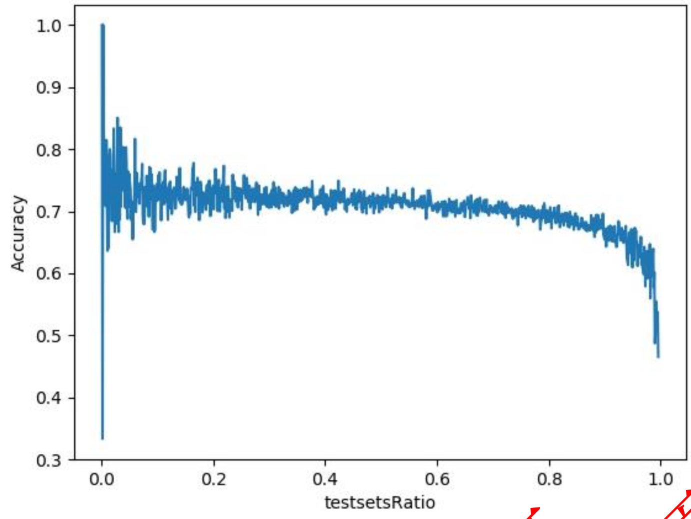


图 4-19 模型测试准确率-测试集比例折线图

从图 4-19 中可以得到，当测试集占比越大时，模型的准确率缓慢下降，考虑到测试存在一定的偶然性的因素，且要满足在尽可能少的训练样本的基础上得到较高的准确率，这个模型在训练样本占比为 19.9%、测试样本占比为 80.1%时，模型的测试准确率达到 71.14%。

当训练样本占比为 19.9%，测试样本占比为 80.1%，$K = 11$ 的模型分类准确率到 71.14%，分别画出特征空间的二维散点图(见图 4-20)和三维散点图(见图 4-21)。

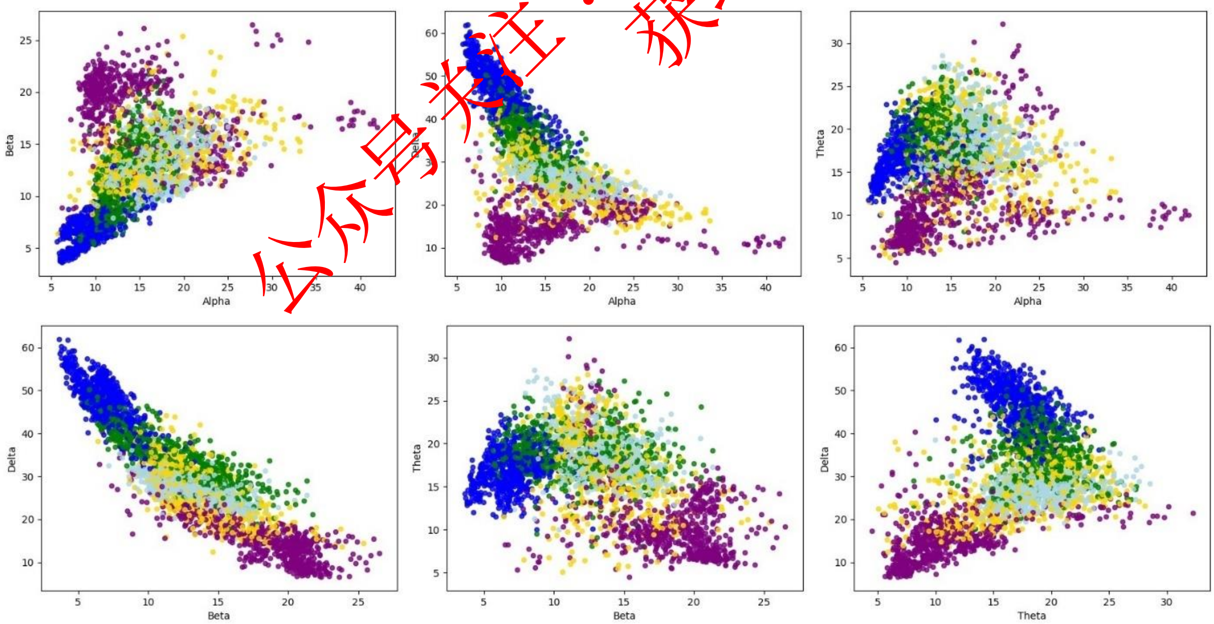


图 4-20 KNN 模型二维散点分类图

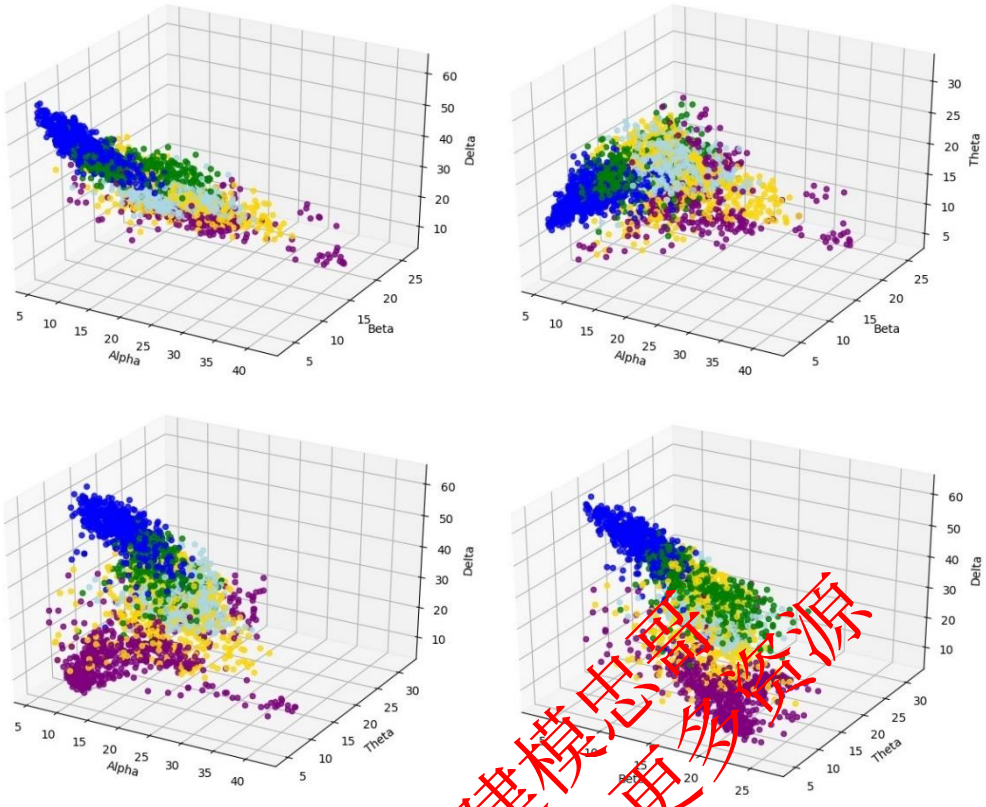


图 4-21 KNN 模型三维散点分类图

由图 4-20 的二维散点分类图和图 4-21 的三维散点分类图可以看出分类的效果具有一定的成效性，其中每个特征维度都对分类模型有一定的影响程度。

3) 模型结果对比

表 4-5 模型对比

	SVM 模型	KNN 模型
测试集比例	85.0%	80.1%
模型测试准确率	71.50%	71.14%

根据表 4-5 可以看出，在测试集占比大的情况下，SVM 模型的测试准确率比 KNN 模型更具有优势。

睡眠脑电数据分为 5 类：清醒期、快速眼动期、睡眠 I 期、睡眠 II 期和深睡眠期，通过对每一个睡眠分期的测试准确率发现，我们的模型在清醒期、快速眼动期和深睡眠期有着很好的分类效果，见表 4-6。

表 4-6 睡眠分期准确率

	清醒期	快速眼动期	深睡眠期
SVM 模型	90.36%(572/633)	87.81%(526/599)	87.54%(527/602)
KNN 模型	88.63%(561/633)	82.30%(493/599)	88.87%(535/602)

从表 4-6 可以看出，SVM 和 KNN 模型对清醒期、快速眼动期和深睡眠期有着很好的分类效果，并且，在对各个睡眠分期分类测试下，SVM 模型略好于 KNN 模型。

综合以上所诉，基于本题考虑，SVM 模型更适用于本问题的预测模型。

5. 模型的评价

5.1 模型的优势

1.在滤除眼电信号上，ICA 方法较其他方法更有效。
2.在基于少量样本的训练集上，SVM 模型较其他分类方法能够得到较高的准确率。

5.2 模型的改进

在睡眠分期预测任务中，模型在睡眠Ⅰ期的准确率有一定的改进空间。

参考文献

[1] 王毅军. 基于节律调制的脑机接口系统——从离线到在线的跨越[D]. 北京：清华大学，2007.

[2] Hubert C，Axel G．Conolutional neural networks for P300 detection with application to brain-computer interfaces [J]．IEEE Transactions on Pattern Analysis and Machine Intelligence，2011，33(3)：433-444.

[3] 刘志勇,孙金玮,卜宪庚.单通道脑电信号眼电伪迹去除算法研究[J].自动化学报,2017,43(10):1726-1735.

[4] 王丽娟. 视觉诱发 P300 电位的去噪与提取算法研究[D]. 济南: 山东大学, 2016.

[5] TheLongGoodbye ， Pay less attention with light-weight &dynamic CNN ，https://zhuanlan.zhihu.com/p/60482693，2020/9/21.

[6] 高强, 姜忠昊. 基于 GAN 等效模型的小样本扩增研究[J]. 电测与仪表, 2019, 56(5): 76-81.

[7] 樊笛, 巨志勇. 基于原型网络的小样本图像识别方法[J]. 计算机与现代化, 2020, 3: 103-107.

[8] Shieh M D, Yang C C. Multiclass SVM-RFE for product form feature selection[J]. Expert Systems with Applications, 2009, 35(1/2): 531-541.

[9] 野风，支持向量机（SVM）——原理篇，https://zhuanlan.zhihu.com/p/31886934，2020/09/21.

[10] xufabing1993，knn 原理介绍以及构建一个 KNN 分类器来进行图像分类，https://blog.csdn.net/qq_35206320/article/details/81946246，2020/09/21

附录(代码)

```
ICA 去除 EOG（Python）
from scipy.io import loadmat
import mne
import os
import xlrd
import xlwt
import numpy as np

def GetFileFromThisRootDir(dir, ext=None):
    allfiles = []
    needExtFilter = (ext != None)
    for root, dirs, files in os.walk(dir):
        for filespath in files:
            filepath = os.path.join(root, filespath)
            extension = os.path.splitext(filepath)[1][1:]
            if needExtFilter and extension in ext:
                allfiles.append(filepath)
            elif not needExtFilter:
                allfiles.append(filepath)
    return allfiles


filename_train = GetFileFromThisRootDir('C:/Users/dell/Desktop/2020 年中国研究生数学建模竞赛赛题/2020 年 C 题/try/newdata/traindata', '.xlsx')

for i in range(10):
    filename_train[i] = filename_train[i].replace("\\", "/")

for i in range(0, 10, 2):
    X1 = xlrd.open_workbook(filename_train[i])

    sheet_name1 = X1.sheet_names()
    workbook1 = xlwt.Workbook(encoding = 'utf-8')

    length = 0
    for j in range(len(sheet_name1)):
        length = length+X1.sheet_by_name(sheet_name1[j]).nrows


    train_data = np.zeros((length, 20), dtype=np.float)
    print(length)
```

```
k = 0
for j in range(len(sheet_name1)):
    for row in X1.sheet_by_name(sheet_name1[j]).get_rows():
        for column in range(20):
            train_data[k][column] = row[column].value
        k = k + 1

eeg = train_data * 10e-4
ch_names = ["Fz", "F3", "F4", "Cz", "C3", "C4", "T7", "T8", "CP3", "CP4",
            "CP5", "CP6", "Pz", "P3", "P4", "P7", "P8", "Oz", "O1", "O2"]
info = mne.create_info(ch_names, 250, ch_types=["eeg"] * 20)
raw = mne.io.RawArray(eeg.T, info)


raw.set_montage("standard_1020")
raw_tmp = raw.copy()
raw_tmp.filter(1, None)
ica = mne.preprocessing.ICA(method="infomax",
                            fit_params={"extended": True},
                            random_state=1)

ica.fit(raw_tmp)
ica.plot_components(inst=raw_tmp, picks=range(20))
ica.plot_sources(inst=raw_tmp)

if i==0:
    ica.exclude = [10]
elif i==2:
    ica.exclude = [10]
elif i==4:
    ica.exclude = [13]
elif i==6:
    ica.exclude = [13]
elif i==8:
    ica.exclude = [10]

raw_corrected = raw.copy()
ica.apply(raw_corrected)
raw.plot(n_channels=20, start=0, duration=4, scalings=dict(eeg=500e-6))
raw_corrected.plot(n_channels=20, start=0, duration=4, scalings=dict(eeg=500e-6))

end_index = 0
for j in range(len(sheet_name1)):
```

```
        worksheet = workbook1.add_sheet(sheet_name1[j])
        start_index = end_index
        end_index = end_index + X1.sheet_by_name(sheet_name1[j]).nrows
        p = 0
        for k in range(start_index, end_index):
            for q in range(20):
                worksheet.write(p,q,raw_corrected[q][0][0][k])
            p = p+1
        print(sheet_name1[j])
    workbook1.save(str(i)+'.xls')
```

巴特沃斯滤波器(MATLAB)

```
data1 = importdata('C:\Users\93690\Desktop\testdata_eogprocess\8.xlsx');
for i = 1:10
    sheetnames_nf=fieldnames(data1);
    name=sheetnames_nf{i};
    A = getfield(data1, name);
    [m,n] = size(A)

    S = [1:m];
    A(:,:) = A(:,:);
    % Butterworth IIR Filter
    srate=250; % sample rate
    npts=m; %npts in signal
    Nyquist=srate/2; %Nyquist frequency
    lpf=8; %low-pass frequency
    hpf=0.3; %-pass frequency
    x=A;

    order=5; %filter order
    t=[0:npts-1]/srate; %time scale for plot
    [b,a]=butter(order,lpf/Nyquist); %create filter coefficients

    filtered_data=filtfilt(b,a,x); % filter using 'b' and 'a' coefficients
    xlswrite('C:\Users\93690\Desktop\testdata_eogprocess_filter\8.xlsx', filtered_data,
name)

    figure;
    subplot(2,1,1)
    plot(t,A);
    title('Raw Time Series');
    subplot(2,1,2)
    %filtered_data = sum(filtered_data,2);
```

```
    plot(t,filtered_data);
    title('Filtered Time Series');
    xlabel('Time (s)');
end
```

SVM-RFE 通道选择(Python)

```
import os
import xlrd
import numpy as np
from sklearn.feature_selection import RFE
import torch.nn as nn
import torch
from sklearn.svm import SVR


# GetFileFromThisRootDir 函数得到某路径下文件后缀的所有文件名

def GetFileFromThisRootDir(dir, ext=None):
    allfiles = []
    needExtFilter = (ext != None)
    for root, dirs, files in os.walk(dir):
        for filespath in files:
            filepath = os.path.join(root, filespath)
            extension = os.path.splitext(filepath)[1][1:]
            if needExtFilter and extension in ext:
                allfiles.append(filepath)
            elif not needExtFilter:
                allfiles.append(filepath)
    return allfiles


filename_train = GetFileFromThisRootDir('C:/Users/15836/Desktop/2020 研究生数学建模/Code/Question_2/Data/filter_eog_8hz', '.xlsx')
filename_test = GetFileFromThisRootDir('C:/Users/15836/Desktop/2020 研究生数学建模/Code/Question_2/Data/testdata', '.xlsx')
print(filename_train)
print("---  loading data begin  ---")
list = []
XX_traindata = []
y_label= []
total_pos = 0
total_t = 0
for i in range(0, 10, 2):
```

```
    # print(i)
    X1 = xlrd.open_workbook(filename_train[i])
    X2 = xlrd.open_workbook(filename_train[i+1])
    sheet_name1 = X1.sheet_names()
    sheet_name2 = X2.sheet_names()
    # print(sheet_name)
    for j in range(len(sheet_name1)):
        train_data = np.zeros((X1.sheet_by_name(sheet_name1[j]).nrows, 20),
dtype=np.float)
        #  train_data1 = np.zeros((X1.sheet_by_name(sheet_name1[j]).nrows, 1),
dtype=np.float)
        test_data = np.zeros((X2.sheet_by_name(sheet_name2[j]).nrows, 2),
dtype=np.float)
        k = 0
        list_node = []
        list_value = []
        for row in X2.sheet_by_name(sheet_name2[j]).get_rows():
            for column in range(2):
                test_data[k][column] = row[column].value
            k = k + 1
        # print(type(test_data[0][0]))
        # print(test_data)
        list_node.append(int((int(test_data[0][0]) - 101) / int(6)) + 1)
        list_node.append(int((int(test_data[0][0]) - 101) % int(6)) + 7)
        print(list_node)
        for row in X2.sheet_by_name(sheet_name2[j]).get_rows():
            if row[0].value == list_node[0] or row[0].value == list_node[1]:
                list_value.append(row[1].value)
        print(list_value)
        k = 0
        for row in X1.sheet_by_name(sheet_name1[j]).get_rows():
            for column in range(20):
                train_data[k][column] = row[column].value
            k = k + 1


        for t in range(0, 2881, 40):
            data = train_data[t:t+175, :]
            flag = 0
            for i in range(10):
                if list_value[i] >= t and list_value[i] < (t + 50):
                    flag = 1
                    total_pos = total_pos + 1
            total_t = total_t + 1
```

```
            y_label.append(flag)
            XX = np.expand_dims(data, axis=0)
            XX_traindata.append(XX)
tup = tuple(XX_traindata)
X = np.vstack(tup)

print(y_label)
print("总数量：", total_t)
print("正样本：", total_pos)

print("---  loading data succeeded  ----")


# 全连接层
net = nn.Linear(175, 1)
net.weight.data.normal_(0, 0.01)
net.bias.data.zero_()
XY_Linear = []
for i in range(total_t):
    X_change = np.zeros(20)
    for j in range(20):
        data = X[i][:, j]
        data = np.squeeze(data)
        data = torch.tensor(data, dtype=torch.float32)
        X_change[j] = net(data)

    XY_change1 = np.expand_dims(X_change, axis=0)
    XY_Linear.append(XY_change1)
tup = tuple(XY_Linear)
X = np.vstack(tup)

names = ["cha1", "cha2", "cha3", "cha4", "cha5", "cha6", "cha7", "cha8", "cha9", "cha10",
         "cha11", "cha12", "cha13", "cha14", "cha15", "cha16", "cha17", "cha18",
"cha19", "cha20"]
print(names)
print('----------3----------')

lr = SVR(kernel = "linear")
rfe = RFE(lr, n_features_to_select=1)
rfe.fit(X, y_label)

print( "Features sorted by their rank:")
print( sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), names)))
```

```
SVM 及 KNN 分类算法
from sklearn import svm
import os
import xlrd
from sklearn.model_selection import train_test_split
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as p3d
from sklearn.neighbors import KNeighborsClassifier as kNN


# GetFileFromThisRootDir 函数得到某路径下文件后缀的所有文件名
def GetFileFromThisRootDir(dir, ext=None):
    allfiles = []
    needExtFilter = (ext != None)
    for root, dirs, files in os.walk(dir):
        for filespath in files:
            filepath = os.path.join(root, filespath)
            extension = os.path.splitext(filepath)[1][1:]
            if needExtFilter and extension in ext:
                allfiles.append(filepath)
            elif not needExtFilter:
                allfiles.append(filepath)
    return allfiles


# 导入数据
def loadData():
    trainData = []
    filename_train = []
    filename_train.append('C:/Users/93690/Desktop/2020 年中国研究生数学建模竞赛赛题
/2020 年 C 题/附件 2-睡眠脑电数据.xlsx')
    print("---  loading data begin  ---")
    for i in range(0, 1):
        X1 = xlrd.open_workbook(filename_train[i])
        sheet_name1 = X1.sheet_names()
        for j in range(len(sheet_name1)):
            train_data    =    np.zeros((X1.sheet_by_name(sheet_name1[j]).nrows-1,    5),
dtype=np.float)
            k = 0
            for row in X1.sheet_by_name(sheet_name1[j]).get_rows():
```

```
                if k == 0:
                    k = k + 1
                    continue
                for column in range(5):
                    train_data[k-1][column] = row[column].value
                k = k + 1
            trainData.append(train_data)
    tup = tuple(trainData)
    allData = np.vstack(tup)
    return allData


# 获取数据
def getdata(alpha):
    allData = loadData()
    # test_Xdata，test_Ylabels 为睡眠某一期的数据和标签
    # 0:633 为清醒期，633:1232 为快速眼动期，1232:1794 为睡眠 I 期，1794:2398 为睡眠 II 期，
2398:3000 为深睡眠期
    test_data1 = allData[2398:3000]
    test_Xdata = np.zeros((len(test_data1), 4), dtype=np.float)
    test_Ylabels = np.zeros(len(test_data1), dtype=np.float)
    for i in range(len(test_data1)):
        for j in range(5):
            if j == 0:
                test_Ylabels[i] = test_data1[i][j] - 2
            else:
                test_Xdata[i][j - 1] = test_data1[i][j]
    # raw_data 为训练数据
    # labels 为训练数据对应的标签
    np.random.shuffle(allData)
    raw_data = np.zeros((len(allData), 4), dtype=np.float)
    labels = np.zeros(len(allData), dtype=np.float)
    for i in range(len(allData)):
        for j in range(5):
            if j == 0:
                labels[i] = allData[i][j] - 2
            else:
                raw_data[i][j-1] = allData[i][j]
    train, test, train_label, test_label = train_test_split(raw_data, labels,
test_size=alpha, random_state=20)
    print(raw_data)
    print(np.size(raw_data, 0))
    print(np.size(raw_data, 1))
    print('-------1-------')
```

```
    print(train)
    print('-------3-------')
    print(train_label)
    print(np.size(train_label, 0))
    print('-------4-------')
    return raw_data, labels, train, test, train_label, test_label, test_Xdata,
test_Ylabels


class Svm(nn.Module):
    def __init__(self, in_dim, hidden_dim, n_layer, n_classes):
        super(Svm, self).__init__()
        self.n_layer = n_layer
        self.hidden_dim = hidden_dim
        self.lstm = nn.LSTM(in_dim, hidden_dim, n_layer, batch_first=True)
        self.classifier = nn.Linear(hidden_dim, n_classes)

    def forward(self, x):
        out, (h_n, c_n) = self.lstm(x)
        # 此时可以从 out 中获得最终输出的状态 h
        # x = out[:, -1, :]
        x = h_n[-1, :, :]
        x = self.classifier(x)
        return x


# 绘制二维散点图
def plot_point(dataArr, labelArr, str, a, b, alpha):
    for i in range(np.shape(dataArr)[0]):
        if labelArr[i] == 0:
            plt.scatter(dataArr[i][a], dataArr[i][b], c='#0000FF', s=20, alpha=alpha)
        elif labelArr[i] == 1:
            plt.scatter(dataArr[i][a], dataArr[i][b], c='#008000', s=20, alpha=alpha)
        elif labelArr[i] == 2:
            plt.scatter(dataArr[i][a], dataArr[i][b], c='#FFD700', s=20, alpha=alpha)
        elif labelArr[i] == 3:
            plt.scatter(dataArr[i][a], dataArr[i][b], c='#ADD8E6', s=20, alpha=alpha)
        elif labelArr[i] == 4:
            plt.scatter(dataArr[i][a], dataArr[i][b], c='#800080', s=20, alpha=alpha)
    str1 = ""
    str2 = ""
    if a == 0:
        str1 = "Alpha"
    elif a == 1:
```

```
        str1 = "Beta"
    elif a == 2:
        str1 = "Theta"
    elif a == 3:
        str1 = "Delta"
    if b == 0:
        str2 = "Alpha"
    elif b == 1:
        str2 = "Beta"
    elif b == 2:
        str2 = "Theta"
    elif b == 3:
        str2 = "Delta"
    string = "./test/" + str + "-" + str1 + "-" + str2 + ".jpg"
    plt.xlabel(str1)
    plt.ylabel(str2)
    plt.savefig(string)
    plt.show()


# 绘制三维散点图
def plot_point_3D(dataArr, labelArr, str, a, b, c, alpha):
    # 创建一个三维的绘图工程
    fig = plt.figure()
    ax = p3d.Axes3D(fig)
    ax.set_zlabel('Z')
    ax.set_ylabel('Y')
    ax.set_xlabel('X')
    for i in range(np.shape(dataArr)[0]):
        if labelArr[i] == 0:
            ax.scatter(dataArr[i][a], dataArr[i][b], dataArr[i][c], s=20, c='#0000FF',
alpha=alpha)
        elif labelArr[i] == 1:
            ax.scatter(dataArr[i][a], dataArr[i][b], dataArr[i][c], s=20, c='#008000',
alpha=alpha)
        elif labelArr[i] == 2:
            ax.scatter(dataArr[i][a], dataArr[i][b], dataArr[i][c], s=20, c='#FFD700',
alpha=alpha)
        elif labelArr[i] == 3:
            ax.scatter(dataArr[i][a], dataArr[i][b], dataArr[i][c], s=20, c='#ADD8E6',
alpha=alpha)
        elif labelArr[i] == 4:
            ax.scatter(dataArr[i][a], dataArr[i][b], dataArr[i][c], s=20, c='#800080',
alpha=alpha)
```

```
    str1 = ""
    str2 = ""
    str3 = ""
    if a == 0:
        str1 = "Alpha"
    elif a == 1:
        str1 = "Beta"
    elif a == 2:
        str1 = "Theta"
    elif a == 3:
        str1 = "Delta"
    if b == 0:
        str2 = "Alpha"
    elif b == 1:
        str2 = "Beta"
    elif b == 2:
        str2 = "Theta"
    elif b == 3:
        str2 = "Delta"
    if c == 0:
        str3 = "Alpha"
    elif c == 1:
        str3 = "Beta"
    elif c == 2:
        str3 = "Theta"
    elif c == 3:
        str3 = "Delta"
    ax.set_xlabel(str1)
    ax.set_ylabel(str2)
    ax.set_zlabel(str3)
    string = "./test/" + str + "-" + str1 + "-" + str2 + "-" + str3 + ".jpg"
    plt.savefig(string)
    plt.show()


# testsetsRatio - testAcc
'''
# test 占比
x_list = []
# 相应测试准确率
acc_list = []
```

```
for k in range(1, 1000):
    x = float(k) / 1000.0
    data, labels, train, test, train_label, test_label = getdata(x)
    x_list.append(x)


    # 设置 SVM 模型  gamma=0.03,
    clf = svm.SVC(gamma=0.03, decision_function_shape='ovo')


    # 训练集
    train_data = train
    train_data_label = train_label


    # 训练模型
    clf.fit(train_data, train_data_label)
    # 训练准确率
    print('Train ACC: : ', clf.score(train_data, train_data_label))


    # 测试集
    test_data = test
    test_data_label = test_label


    # 测试
    dec = clf.decision_function(test_data)
    # 此时维度为 5*4/2 = 10
    # print(dec.shape[1])
    clf.decision_function_shape = "ovr"
    dec = clf.decision_function(test)
    # 此时维度为 5

    # 预测
    predicted_test = clf.predict(test_data)
    acc_num = 0


    for i in range(len(predicted_test)):
        if predicted_test[i] == test_data_label[i]:
            acc_num = acc_num + 1
    print("Test  ACC:  ",  clf.score(test,  test_label),  "  |  (  ",  acc_num,  "  /  ",
len(predicted_test), " ) | ", "Predicted Labels: ", predicted_test, " | True Labels: ",
```

```
test_label)
    acc_list.append(clf.score(test, test_label))


fig = plt.figure()  # figsize=(10, 10)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x_list, acc_list)
plt.xlabel(u"testsetsRatio")
plt.ylabel(u"Accuracy")
plt.savefig("./picture/testsetRatio-testAcc.jpg")
plt.show()
'''


'''
data, labels, train, test, train_label, test_label = getData(0.85)


# 设置 SVM 模型  gamma=0.1,
clf = svm.SVC(gamma=0.03, decision_function_shape='ovr')


# 训练集
train_data = train
train_data_label = train_label


# 训练模型
clf.fit(train_data, train_data_label)
# 训练准确率
print('Train ACC: : ', clf.score(train_data, train_data_label))


# 测试集
test_data = test
test_data_label = test_label


# 测试
dec = clf.decision_function(test_data)
# 此时维度为 5*4/2 = 10
# print(dec.shape[1])
clf.decision_function_shape = "ovr"
dec = clf.decision_function(test)
```

```
# 此时维度为 5


# 预测
predicted_test = clf.predict(test_data)
acc_num = 0


for i in range(len(predicted_test)):
    if predicted_test[i] == test_data_label[i]:
        acc_num = acc_num + 1
print("Test ACC: ", clf.score(test, test_label), " | ( ", acc_num, " / ", len(predicted_test), " ) | ", "Predicted Labels: ", predicted_test, " | True Labels: ", test_label)
'''



# gamma - testAcc   gamma = 0.03
'''
data, labels, train, test, train_label, test_label = getdata(0.8)
acc_list = []
x_list = []


for i in range(1, 100):
    alpha = float(i) / 100
    # 设置 SVM 模型  gamma=0.1,
    clf = svm.SVC(gamma=alpha, decision_function_shape='ovo')
    x_list.append(alpha)


    # 训练集
    train_data = train
    train_data_label = train_label


    # 训练模型
    clf.fit(train_data, train_data_label)
    # 训练准确率
    print('Train ACC: : ', clf.score(train_data, train_data_label))


    # 测试集
    test_data = test
```

```
    test_data_label = test_label


    # 测试
    dec = clf.decision_function(test_data)
    # 此时维度为 5*4/2 = 10
    # print(dec.shape[1])
    clf.decision_function_shape = "ovr"
    dec = clf.decision_function(test)
    # 此时维度为 5

    # 预测
    predicted_test = clf.predict(test_data)
    acc_num = 0


    for i in range(len(predicted_test)):
        if predicted_test[i] == test_data_label[i]:
            acc_num = acc_num + 1
    print("Test  ACC:  ",  clf.score(test,  test_label),"  |  (  ",  acc_num,  "  /  ",
len(predicted_test), " ) | ", "Predicted Labels: ", predicted_test, " | True Labels: ",
test_label)
    acc_list.append(clf.score(test, test_label))


fig = plt.figure()  # figsize=(10, 10)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x_list, acc_list)
plt.xlabel(u"gamma")
plt.ylabel(u"Accuracy")
plt.savefig("./picture/gamma-testAcc.jpg")
plt.show()
'''


# KNN

'''
data, labels, train, test, train_label, test_label = getdata(0.801)


# 设置 KNN 模型  gamma=0.1,
clf = kNN(n_neighbors=11, algorithm='auto', weights='distance', n_jobs=1)
```

```
# 训练集
train_data = train
train_data_label = train_label


# 训练模型
clf.fit(train_data, train_data_label)
# 训练准确率
print('Train ACC: : ', clf.score(train_data, train_data_label))


# 测试集
test_data = test
test_data_label = test_label


# 预测
predicted_test = clf.predict(test_data)
acc_num = 0


for i in range(len(predicted_test)):
    if predicted_test[i] == test_data_label[i]:
        acc_num = acc_num + 1
print("Test ACC: ", clf.score(test, test_label), " | ( ", acc_num, " / ",
len(predicted_test), ")\n", "Predicted Labels: ", predicted_test, " | True Labels: ",
test_label)
'''


# KNN-n_neighbors-testAcc n_neighbors = 11
'''
# n_neighbors
x_list = []
# 相应测试准确率
acc_list = []
for i in range(1, 50):
    data, labels, train, test, train_label, test_label = getdata(0.825)
    x_list.append(i)
    # 设置 KNN 模型
    clf = kNN(n_neighbors=i, algorithm='auto', weights='distance', n_jobs=1)

    # 训练集
```

```
    train_data = train
    train_data_label = train_label

    # 训练模型
    clf.fit(train_data, train_data_label)
    # 训练准确率
    print('Train ACC: : ', clf.score(train_data, train_data_label))

    # 测试集
    test_data = test
    test_data_label = test_label

    # 预测
    predicted_test = clf.predict(test_data)
    acc_num = 0

    for i in range(len(predicted_test)):
        if predicted_test[i] == test_data_label[i]:
            acc_num = acc_num + 1
    print("Test  ACC:  ",  clf.score(test,  test_label),"  |  (  ",  acc_num,  "  /  ",
len(predicted_test), " ) | ",
          "Predicted Labels: ", predicted_test," | True Labels: ", test_label)
    acc_list.append(clf.score(test, test_label))
fig = plt.figure()  # figsize=(10, 10)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x_list, acc_list)
plt.xlabel(u'K')
plt.ylabel(u'Accuracy')
plt.savefig("./picture/KNN-n_neighbors-testAcc.jpg")
plt.show()
'''



# KNN-testsetRatio-testAcc  testsetRadio = 0.849

'''
# test 占比
x_list = []
# 相应测试准确率
acc_list = []
for i in range(1, 997):
    x = float(i) / 1000.0
    data, labels, train, test, train_label, test_label = getdata(x)
```

```
    x_list.append(x)
    # 设置 KNN 模型  gamma=0.1,
    clf = kNN(n_neighbors=11, algorithm='auto', weights='distance', n_jobs=1)

    # 训练集
    train_data = train
    train_data_label = train_label

    # 训练模型
    clf.fit(train_data, train_data_label)
    # 训练准确率
    print('Train ACC: : ', clf.score(train_data, train_data_label))

    # 测试集
    test_data = test
    test_data_label = test_label

    # 预测
    predicted_test = clf.predict(test_data)
    acc_num = 0

    for i in range(len(predicted_test)):
        if predicted_test[i] == test_data_label[i]:
            acc_num = acc_num + 1
    print("Test  ACC:  ",  clf.score(test,  test_label),  "  |  (  ",  acc_num,  "  /  ",
len(predicted_test), " ) |",
          "Predicted Labels: ", predicted_test, " | True Labels: ", test_label)
    acc_list.append(clf.score(test, test_label))
fig = plt.figure()  # figsize=(10, 10)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x_list, acc_list)
plt.xlabel(u'testsetsRatio')
plt.ylabel(u'Accuracy')
plt.savefig("./picture/KNN-testsetRatio-testAcc.jpg")
plt.show()
y = []
# max = 0
# max_k = 0
# for j in range(len(x_list)):
#     y.append(0.77 * acc_list[j] + 0.23 * x_list[j])
# for k in range(len(y)):
#     if max < y[k]:
#         max = y[k]
#         max_k = k
```

```
# print(y)
# print(max)
# print("最优目标对应 testsetsRadio： ", x_list[max_k], " | testAcc： ", acc_list[max_k])
'''



# plot_point(data, labels, "KNN", 0, 1, 0.8)
# plot_point(data, labels, "KNN", 0, 2, 0.8)
# plot_point(data, labels, "KNN", 0, 3, 0.8)
# plot_point(data, labels, "KNN", 1, 2, 0.8)
# plot_point(data, labels, "KNN", 1, 3, 0.8)
# plot_point(data, labels, "KNN", 2, 3, 0.8)
# plot_point_3D(data, labels, "KNN", 0, 1, 2, 0.8)
# plot_point_3D(data, labels, "KNN", 0, 1, 3, 0.8)
# plot_point_3D(data, labels, "KNN", 0, 2, 3, 0.8)
# plot_point_3D(data, labels, "KNN", 1, 2, 3, 0.8)



# plot_point(data, labels, "SVM", 0, 1, 0.8)
# plot_point(data, labels, "SVM", 0, 2, 0.8)
# plot_point(data, labels, "SVM", 0, 3, 0.8)
# plot_point(data, labels, "SVM", 1, 2, 0.8)
# plot_point(data, labels, "SVM", 1, 3, 0.8)
# plot_point(data, labels, "SVM", 2, 3, 0.8)
# plot_point_3D(data, labels, "SVM", 0, 1, 2, 0.8)
# plot_point_3D(data, labels, "SVM", 0, 1, 3, 0.8)
# plot_point_3D(data, labels, "SVM", 0, 2, 3, 0.8)
# plot_point_3D(data, labels, "SVM", 1, 2, 3, 0.8)



# plot_point(test_data, predicted_test, "SVM", 0, 1, 0.8)
# plot_point(test_data, predicted_test, "SVM", 0, 2, 0.8)
# plot_point(test_data, predicted_test, "SVM", 0, 3, 0.8)
# plot_point(test_data, predicted_test, "SVM", 1, 2, 0.8)
# plot_point(test_data, predicted_test, "SVM", 1, 3, 0.8)
# plot_point(test_data, predicted_test, "SVM", 2, 3, 0.8)
# plot_point_3D(test_data, predicted_test, "SVM", 0, 1, 2, 0.8)
# plot_point_3D(test_data, predicted_test, "SVM", 0, 1, 3, 0.8)
# plot_point_3D(test_data, predicted_test, "SVM", 0, 2, 3, 0.8)
# plot_point_3D(test_data, predicted_test, "SVM", 1, 2, 3, 0.8)



# plot_point(test_data, predicted_test, "KNN", 0, 1, 0.8)
# plot_point(test_data, predicted_test, "KNN", 0, 2, 0.8)
```

```
# plot_point(test_data, predicted_test, "KNN", 0, 3, 0.8)
# plot_point(test_data, predicted_test, "KNN", 1, 2, 0.8)
# plot_point(test_data, predicted_test, "KNN", 1, 3, 0.8)
# plot_point(test_data, predicted_test, "KNN", 2, 3, 0.8)
# plot_point_3D(test_data, predicted_test, "KNN", 0, 1, 2, 0.8)
# plot_point_3D(test_data, predicted_test, "KNN", 0, 1, 3, 0.8)
# plot_point_3D(test_data, predicted_test, "KNN", 0, 2, 3, 0.8)
# plot_point_3D(test_data, predicted_test, "KNN", 1, 2, 3, 0.8)


'''
data, labels, train, test, train_label, test_label, test_Xdata, test_Ylabels = getdata(0.801)


# 设置 KNN 模型  gamma=0.1,
clf = kNN(n_neighbors=11, algorithm='auto', weights='distance', n_jobs=1)


# 训练集
train_data = train
train_data_label = train_label


# 训练模型
clf.fit(train_data, train_data_label)
# 训练准确率
print('Train ACC: :', clf.score(train_data, train_data_label))


# 测试集
test_data = test_Xdata
test_data_label = test_Ylabels


# 预测
predicted_test = clf.predict(test_data)
acc_num = 0


for i in range(len(predicted_test)):
    if predicted_test[i] == test_data_label[i]:
        acc_num = acc_num + 1
print("Test ACC: ", clf.score(test_data, test_data_label), " | ( ", acc_num, " / ",
```

```
len(predicted_test), " ) | ", "Predicted Labels: ", predicted_test, " | True Labels: ",
test_data_label)
'''


data, labels, train, test, train_label, test_label, test_Xdata, test_Ylabels =
getdata(0.85)


# 设置 SVM 模型  gamma=0.03
clf = svm.SVC(gamma=0.03, decision_function_shape='ovo')


# 训练集
train_data = train
train_data_label = train_label


# 训练模型
clf.fit(train_data, train_data_label)
# 训练准确率
print('Train ACC: : ', clf.score(train_data, train_data_label))


# 测试集
test_data = test_Xdata
test_data_label = test_Ylabels


# # 测试
# dec = clf.decision_function(test_data)
# # 此时维度为 5*4/2 = 10
# # print(dec.shape[1])
# clf.decision_function_shape = "ovr"
# dec = clf.decision_function(test)
# # 此时维度为 5


# 预测
predicted_test = clf.predict(test_data)
acc_num = 0


for i in range(len(predicted_test)):
```

```
    if predicted_test[i] == test_data_label[i]:
        acc_num = acc_num + 1
print("Test ACC: ", clf.score(test_data, test_data_label), " | ( ", acc_num, " / ",
len(predicted_test), " ) | ", "Predicted Labels: ", predicted_test, " | True Labels: ",
test_data_label)
```