


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	上海交通大学
参赛队号	**No.20102480095**
队员姓名	**1.** 眭泽
	2. 刁丽
	3. 刘琦

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　脑电信号分析与判别模型研究

摘　　要:

对脑电信号的准确有效分析，有助于把握大脑的活动状态，是实现脑机接口技术和评估睡眠质量的重要前提。本文分别考虑了目标字符识别问题、最优组合通道的选取问题，使用部分无标签数据进行半监督学习问题、数据集选取方式和分配比例问题、睡眠分期问题，并分别建立模型进行求解。

针对问题一，考虑原始诱发脑电包含干扰信号，利用经验模态分解方法得到 8 个 IMF 分量，分析不同频率下的信号特征。采用四阶低通 Butterworth 滤波器和快速傅里叶变换进行滤波和去噪。将原始训练集按 5:1 划分为训练集和测试集，利用重采样方法实现正负样本平衡，通过识别目标字符所在的行和列，将多分类问题转化为二分类问题。建立了基于 SVM、DNN、LSTM 和 1D-CNN 的目标字符识别模型，验证集上的识别准确率分别为 68.17%、71.12%、73.44% 和 77.87%。仅通过 2 轮实验，就可实现测试集字符的准确预测，char13～char22 的识别结果依次为 **M、F、5、2、I、T、K、X、A、0**（被试者 S2 和 S3 由于原始数据的问题仅能识别 char13～char21，结果与其余三个被试者相同）。

针对问题二，考虑各个通道重要程度不同，建立了带注意力机制的 1D-CNN 通道筛选模型，得到了每个被试者 20 个通道的注意力权重。分别选取权重最大的 15 个通道作为每个被试者的最优组合通道，被试者 S1 为 {Fz、F3、F4、Cz、C4、T7、CP3、CP4、CP5、CP6、Pz、P3、P4、O1、O2}，S2 为 {Fz、F3、Cz、C3、T7、CP3、CP5、CP6、P3、P4、P7、P8、Oz、O1、O2}，S3 为 {Fz、F4、C3、C4、T8、CP3、CP4、CP6、Pz、P3、P4、P7、P8、Oz、O1}，S4 为 {Fz、F3、F4、Cz、C4、T7、T8、CP3、Pz、P3、P4、P7、Oz、O1、O2 }，S5 为 {Fz、F3、Cz、C3、T7、T8、CP3、CP5、Pz、P3、P4、P7、P8、Oz、O1} 将其取交集得到适用于所有被试者的最优组合通道 {Fz、CP3、P3、P4、O1}，且每个被试者基于 15 通道和 5 通道的测试集识别结果均为 **M、F、5、2、I、T、K、X、A、0**。五名被试者基于 5 通道的 1D-CNN 验证集平均识别准确率为 75.20%，基于原有 20 通道为 77.87%，引入注意力机制后小幅提升为 77.93%，基于 15 个最优组合通道时进一步提升为 78.06%。

针对问题三，考虑部分数据无标签，利用问题一中的 1D-CNN 预测网络、问题二的最优通道组合和现有标签数据预训练得到了初始分类模型。按是否使用先验信息，设计了两种伪标签添加方式。无先验信息时，利用初始分类模型，对无标签数据添加伪标签，选出 softmax 输出概率大于 0.8 的数据，将其添加到标签数据中，并更新初始分类模型；使用先验信息时，考虑到每轮测试中每行和每列都会各出现一次正样本，其余为负样本，选取所有行中输出概率最高的行作为行预测结果，列预测同理，并将其作为无标签数据的伪标签。实验结果表明，上述过程重复 10 次后，所有无标签数据都会被添加到标签数据中，也即得到了稳定的半监督学习分类模型。无先验和有先验方式最终对 char13～char22 的识别结果均为 **M、F、5、2、I、T、K、X、A、0**，在验证集上的分类准确率分别从初始的 63% 提升至 67.23% 和 71.48%，接近有监督训练的分类准确率 75.2%，验证了半监督学习模型的有效性。

针对问题四，考虑原始睡眠脑电数据仅包含四个特征，从能量角度出发构造了 9 个新特征，扩充了原有数据集，并按训练集和测试集 4:1 的比例随机划分了 100 次，SVM、随机森林和 CatBoost 在 100 个测试集上睡眠分期的平均分类正确率分别为 65%、94.3%、95.06%。改变训练集占总样本的比例分别为 0.8、0.7、0.6、0.5 和 0.4，得到 CatBoost 在 100 个测试集上睡眠分期的平均分类正确率分别为 95.06%、91.22%、88.42%、85.73% 和 82.06%，验证了所建立的模型可以在较少训练样本情况下实现较高的分类正确率。

关键字：脑电信号　傅里叶变换　1D-CNN　LSTM　CatBoost

目录

1. 问题重述 5
1.1 问题背景 5
1.2 问题解析 5
1.2.1 问题一解析 5
1.2.2 问题二解析 5
1.2.3 问题三解析 6
1.2.4 问题四解析 6

2. 模型假设 7

3. 符号说明 7

4. 问题一的分析与求解 8
4.1 问题一的分析 8
4.2 数据预处理 8
4.2.1 经验模态分解（EMD） 8
4.2.2 Butterworth 滤波器 9
4.2.3 快速傅里叶变换（FFT） 15
4.3 数据集构建与划分 16
4.4 模型建立 18
4.4.1 支持向量机（SVM） 18
4.4.2 全连接神经网络（DNN） 19
4.4.3 长短时记忆网络（LSTM） 20
4.4.4 一维卷积神经网络（1D-CNN) 21
4.4.5 模型训练 22
4.5 模型性能评估 22
4.6 结果预测 23
4.6.1 测试集结果预测 23
4.6.2 基于预测结果的性能分析 24

5. 问题二的分析与求解 25
5.1 问题二的分析 25
5.2 注意力机制模型 25
5.2.1 注意力机制算法原理 26
5.2.2 模型搭建 26
5.3 基于注意力权重的通道选择 27
5.4 性能评估与结果预测 28
5.4.1 性能评估 28
5.4.2 结果预测 29

6. 问题三的分析与求解 30
6.1 问题三的分析 30
6.2 基本半监督学习算法 30
6.3 半监督模型搭建 31
6.3.1 伪标签方案设计 31
6.3.2 半监督训练流程 31

6.4 性能评估与结果预测 32
6.4.1 模型训练与评估 32
6.4.2 结果预测 32
7. 问题四的分析与求解 35
7.1 问题四的分析 35
7.2 脑电特征构造与数据处理 35
7.2.1 脑电波特征分析 35
7.2.2 脑电波能量特征构造 35
7.2.3 脑电数据集划分 36
7.3 分类器的构建 36
7.3.1 决策树和随机森林 36
7.3.2 GBDT 和 CatBoost 37
7.4 实验性能对比 39
7.4.1 不同分类器的性能对比 39
7.4.2 脑电特征有效性分析 39
7.4.3 数据集划分方式的性能对比 40
8. 模型的评价 42
8.1 模型的优点 42
8.2 模型的缺点 42
9. 参考文献 43
附录 A 补充对比实验（消融实验） 44
1.1 采样时段长度 S 对模型精度的影响实验 44
1.2 不同优化器对模型精度的影响实验 44
1.3 滤波算法预处理数据对模型精度的影响实验 44
附录 B 程序代码 45

1. 问题重述

1.1 问题背景

脑机接口技术（BCI）是当前的一个研究热点，有着巨大的实际应用价值。BCI 是在人或动物脑（或者脑细胞的培养物）与外部设备间建立的直接连接通路。BCI 在医疗领域具有广阔的运用前景，它可以用于恢复损伤的听觉、视觉以及肢体运动能力，为残障人士带来了福音。

事件相关信号（ERP）是 BCI 系统中的常用脑电信号，经典的 ERP 包含多种电位，其中 P300 电位是目前研究最广泛的一种 ERP，受试者在受到刺激大约 300ms 后会产生一个正向的峰值，这个峰值就是 P300 电位。由于 P300 电位与注意、记忆等高级心理活动密切相关, 因此从脑电信号中准确有效地提取出 P300 电位信息具有十分重要的意义。

1.2 问题解析

1.2.1 问题一解析

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。请根据附件 1 所给数据，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，找出附件 1 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。本题解题步骤如下：

步骤一：数据读取，读取 excel 中的数据，找到每个刺激信号对应的固定序列长度的响应数据，根据刺激与字符所在行列是否相符作为正负样本依据，建立二分类数据集，并进行正负样本平衡 (1:1)；

步骤二：数据预处理，通过 Butterworth 滤波器进行低通滤波；此外，对于提供的训练集数据，按 5:1 随机划分训练集和验证集，用于评估方法的性能。

步骤三：模型建立，基于处理过的数据，分别建立 SVM 分类器、全连接神经网络、一维 CNN 网络和 LSTM 时序网络在训练集进行训练，评估各种方法在验证集上的准确率并进行对比；同时，设计实验评估数据预处理方法对性能的影响；

步骤四：目标字符识别，通过步骤三中训练好的网络，在尽可能少的轮数下预测结果。同时，为了获得尽可能准确的数据，对识别的行、列结果进行统计和投票，选择票数最多的结果作为字符识别结果。

1.2.2 问题二解析

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。根据附件 1 所给数据，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（通道组合的数量小于 20 大于等于 10）。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。

步骤一：基于问题一的模型，搭建通道注意力机制模块；

步骤二：训练带注意力机制的神经网络，并在已知验证集上评估性能；

步骤三：导出训练好的通道注意力机制模块的权重，找到平均权重最高的 k 个通道 ($10 \leq k < 20$)，删去其他通道数据并调整神经网络；

步骤四：基于所选通道，重新训练神经网络，并在验证集上评估性能；

步骤五：基于所选通道，实验测试识别已知测试数据的正确性，并计算剩余未知测试数据的识别结果。

1.2.3 问题三解析

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，请根据附件 1 所给数据，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）。

步骤一：选择一定比例的训练数据作为有标签数据，按照问题二中选择的通道，训练神经网络。

步骤二：根据步骤一中训练好的网络（多次训练最终取平均值），根据情况采用两种方法为无标签数据贴上伪标签：（1）无任何先验信息：直接利用二分类网络预测出来的结果，选择置信度高的部分标签，作为无标签数据的伪标签，并进行正负样本平衡；（2）基于先验信息：每轮测试中每行和每列会各出现一次正样本，其余为负样本。因此，直接用 1～6 中预测得分最高的数据作为行预测结果，列预测同理，将这个预测结果作为无标签数据的伪标签；

步骤三：将步骤二中打上伪标签的数据与有标签的数据混合，重新训练网络并评估性能，比较两种伪标签方法带来的性能提升。

步骤四：重复上述过程；训练过程中，伪标签数据的 Loss 计算权重随重复轮次逐渐增大；

步骤五：在验证集上测试半监督训练模型的性能，并给出测试数据识别结果。

1.2.4 问题四解析

问题四主要需要解决的是一个根据少量特征进行睡眠分期五分类的问题。本文的解决思路基本为构建特征工程与选择多种分类器进行实验，对比得到最为适合的特征构造方法与分类器模型。

步骤一：数据预处理，检验是否存在异常数据；

步骤二：特征构造，调研脑电波能量特征与睡眠分期的关联或潜在关系，构建新特征以扩充数据集；

步骤三：在 3000 条数据中按 4:1 的比例随机抽取训练数据与测试数据（待步骤四确定最佳分类器后，可以适当减小训练集与测试集的样本比例）；

步骤四：分别建立 CatBoost、随机森林、SVM 等分类器训练模型，并对比其在测试集上的准确率等的指标性能，同时进行特征重要性分析与特征筛选。

2. 模型假设

1. 假设题目提供的大部分数据的采集准确，标签记录无误；
2. 假设被试者只在受到正确刺激后才会出现 P300 信号；
3. 假设所有被试者的 P300 信号只出现在受到刺激后的 1000ms 内。

3. 符号说明

符号	意义
S	每个用于训练、验证或测试样本序列的长度
$H(j\omega)$	Butterworth 低通滤波器频率响应
$F(\omega)$	时间信号 f(t) 的傅里叶变换
$X(k)$	时间序列 x(t) 的离散傅里叶变换
f_t	LSTM 的遗忘门
i_t	LSTM 的输入门
O_t	LSTM 输出门
Z_c	压缩后的注意力机制模块的输入信号
L	自训练模型损失函数
$\alpha(t)$	自训练模型的平衡系数
$Ratio_{\theta+\delta}$	θ 波和 δ 波能量占脑电总能量的比例
$Ratio_{\alpha+\beta}$	α 波和 β 波能量占脑电总能量的比例
$MeanEnergy$	4 种波的能量占脑电总能量比例的均值
$\ln E_\alpha$	α 波能量取对数
$\ln E_\beta$	β 波能量取对数
$\ln E_\theta$	θ 波能量取对数
$\ln E_\delta$	δ 波能量取对数
$\hat{x}_k^i$	CatBoost 类别特征值的转换结果

4. 问题一的分析与求解

4.1 问题一的分析

一方面，原始的脑电波信号中包含许多干扰噪声，为了准确有效地提取出P300 诱发电位的信号，需要对数据进行预处理，也即通过滤波去噪等方法对数据进行清洗，从而提升数据质量。

另一方面，考虑到字符总数是 36 个，而训练集中的已知字符只有 12 个，训练集标签数小于目标分类数，因此需要将原有的 36 分类问题转化为判断目标字符所在行或列的 2 分类问题。综合分析 5 个被试者在 5 轮测试中分别判断出的待测字符最优行和最优列，利用统计学原理确定待测字符最终所在的行和列，从而锁定待测字符。

同时，为了评估不同预测模型的性能，需要采用相同的数据分别训练 SVM、DNN、LSTM 和 CNN 四个预测模型，并利用相同的验证集对模型预测准确率进行测试。

最后，通过减少被试者的测试轮数，观察预测模型在验证集上的准确率变化情况，分析测试轮次与预测准确率之间的关系。

4.2 数据预处理

P300 诱发电位具有非平稳、非线性的特点，而且信号强度十分微弱，在信号采集的过程中往往会掺杂人体其它部位的生理信号，例如眼电信号（EOG）、肌电信号（EMG）、心电信号（EEG）等。同时，信号采集设备会不可避免地收到环境因素的干扰，例如工频干扰、电磁噪声等。

这些伪迹干扰信号与 P300 诱发电位的信号在频率和幅值上都具有一定的差异，往往会将 P300 信号掩盖掉，不利于真实脑电信号的特征提取和识别。因此，需要选择合适的数据预处理方法对原始脑电信号进行滤波和去噪，为后续模型建立提供高质量的数据。

4.2.1 经验模态分解（EMD）

本文首先采用 EMD 分析原始脑电信号，将其按频率大小分解成不同的本征模函数（IMFs），主要目的是观察原始脑电信号在不同频率下的特征，为后续选择滤波和去噪方法提供参考。

经验模态分解方法被认为是 2000 年来以傅立叶变换为基础的线性和稳态频谱分析的一个重大突破，该方法是依据数据自身的时间尺度特征来进行信号分解，无须预先设定任何基函数。这一点与建立在先验性的谐波基函数和小波基函数上的傅里叶分解与小波分解方法具有本质性的差别。正是由于这样的特点，EMD 方法在理论上可以应用于任何类型的信号的分解，因而在处理非平稳及非线性数据上，具有非常明显的优势, 适合于分析非线性、非平稳信号序列，具有很高的信噪比。

经验模态分解（EMD）被认为是 Hilbert-Huang 变换（HHT）的基础部分。Hilbert-Huang 变换可以分为两个阶段进行。第一阶段，利用 EMD 算使复杂信号分解为有限个本征模函数（IMFs），所分解出来的各 IMF 分量包含了原信号的不同时间尺度的局部特征信号。在第二阶段，将 Hilbert 变换应用于上述步骤的结果（分解得到的 IMFs），得到初始序列有物理意义的瞬时频谱。与短时傅立叶变换、小波分解等方法相比，这种方法是直观的、直接的、后验的和自适应的，因为基函数是由数据本身所分解得到。由于分解是基于信号序列时间尺度的局部特性，因此具有自适应性。

EMD 方法是基于以下假设条件：

（1）数据至少有 2 个极值，一个极大值和一个极小值。

（2）数据的局部时域特性是由极值点间的时间尺度唯一确定。

（3）如果数据没有极值点，但是有拐点，则可以通过对数据微分一次或者多次求得极值，然后通过积分来获得分解结果。

利用 EMD 分解获取 IMFs 的主要步骤如下：

1、利用式1计算第一个分量 h_1：

$$h_1 = X(t) - m_1 \tag{1}$$

2、在计算第二个 IMF 分量时，h_1 被当作已知数据，m_{11} 是 h_1 上包络线和下包络线的平均值，如式3所示：

$$h_{11} = h_1 - m_{11} \tag{2}$$

3、该筛选程序重复 k 次，直到 h_{1k} 为 IMF，即：

$$h_{1(k-1)} - m_{1k} = h_{1k} \tag{3}$$

4、得到的 h_{1k} 就是数据中的第一个 IMF 分量，它包含信号的最短周期分量：

$$c_1 = h_{1k} \tag{4}$$

5 将第一个 IMF 分量与其他数据分离开：

$$X(t) - c_1 = r_1 \tag{5}$$

6、重复上述这个过程，直至得到所有 IMF 分量：

$$r_1 - c_2 = r_2, \ldots, r_{n-1} - c_n = r_n \tag{6}$$

可以看出，所描述的 EMD 分解过程不是基于严格的数学计算，而是真正的经验模式分解过程。

按照上述步骤，本文利用 EMD 对五个被试者在各个字符 20 个通道的上的训练数据和测试数据进行了 IMFs 分解。以第一个被试者（S1）在训练集和测试集上的 EMD 分解情况进行说明。

被试者 S1 在训练集中字符 char01(B) 通道 1 的原始脑电信号如图4-1所示。利用 EMD 方法对字符 char01(B) 通道 1 的数据进行分解，得到八个 IMF 分量，结果如图4-2所示。八个 IMF 分量的瞬时频率如图4-3所示。被试者 S1 在测试集中字符 char13 通道 1 的原始脑电信号如图4-4所示。利用 EMD 方法对字符 char13 通道 1 的数据进行分解，得到八个 IMF 分量，结果如图4-5所示。八个 IMF 分量的瞬时频率如图4-6所示。不难发现，被试者 S1 在训练集中字符 char01(B) 和测试集中 char13 通道 1 的第 6 和第 7 个 IMF 分量呈现出了较为明显波峰，这进一步验证了 P300 诱发脑波信号的存在，同时表明当信号频率接近 P300 脑电信号频率时，可以更为方便地提取出信号中有效信息。

4.2.2 Butterworth 滤波器

上述 EMD 方法只能按一定的频率分离原始信号, 所得到的八个 IMF 分量并不一定完全落在 P300 诱发信号的频率范围内，而且还可能还包含大脑、肌肉和眼部持续活动产生的干扰电信号，导致数据的信噪比很低，分类识别的准确率很低。为此，本文选用 Butterworth 滤波器，通过设置 0.5～25Hz 的通频带，对原始

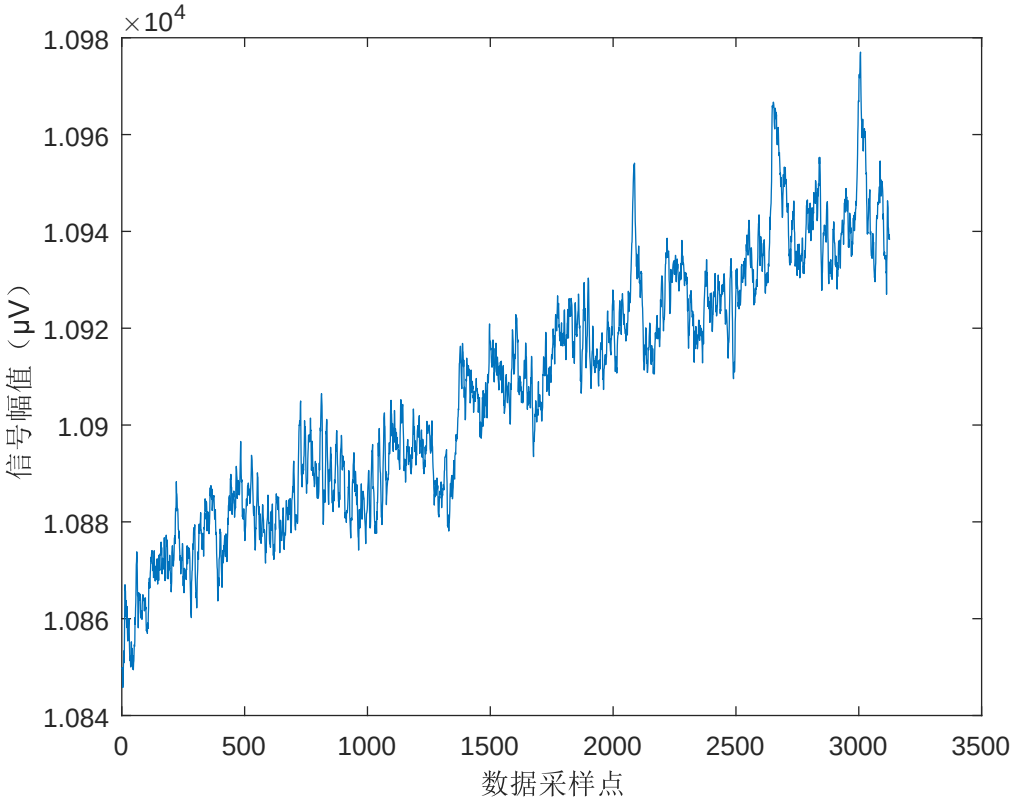


图 4-1　字符 char01(B) 通道 1 上的原始脑电信号

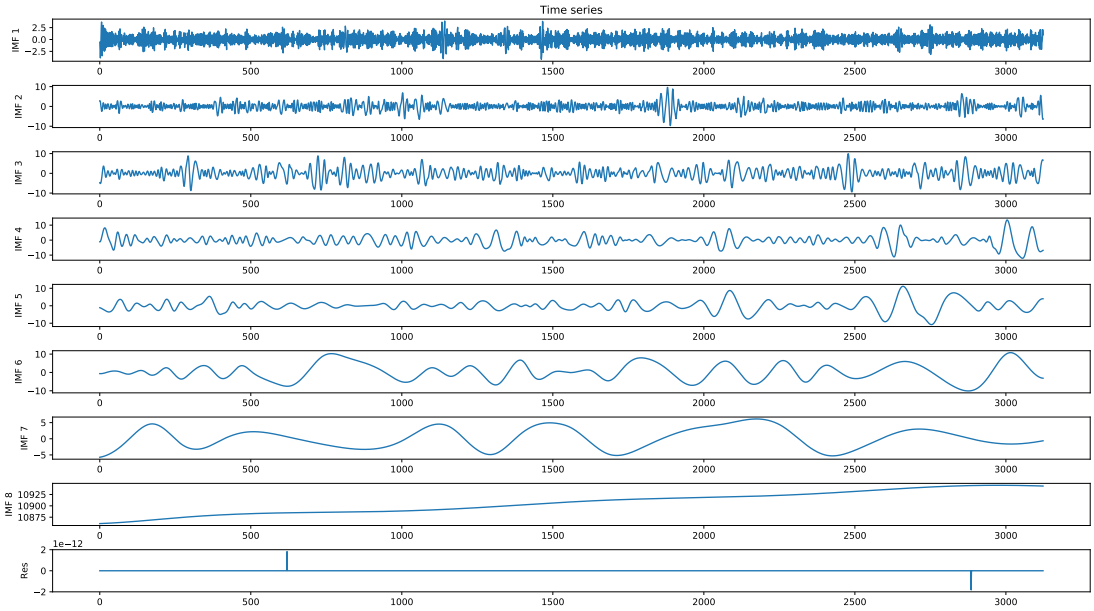


图 4-2　EMD 分解字符 char01(B) 通道 1 得到的 IMF 分量

信号进行滤波处理，将不属于 P300 诱发脑电波频率范围内的信号滤除，从而提升信号质量。

Butterworth 滤波器是一种信号处理滤波器，它的设计目的是让通频带内的频率响应尽可能平坦。因此，Butterworth 滤波器也被称为“最大平坦幅度滤波

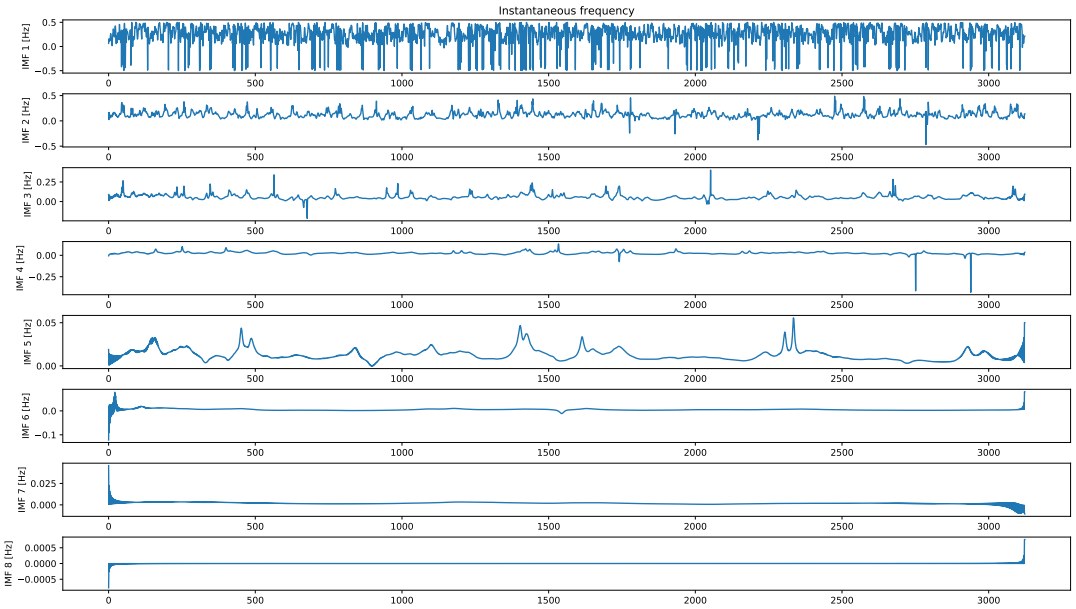


图 4-3　字符 char01(B) 八个 IMF 分量的瞬时功率

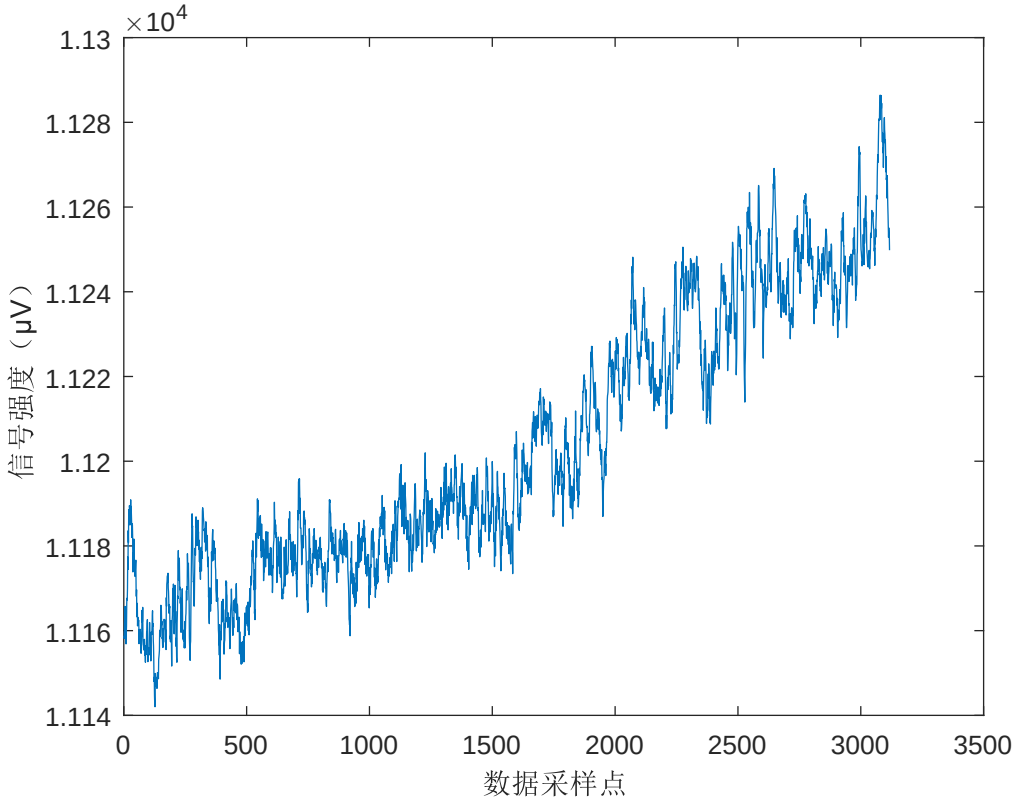


图 4-4　字符 char13 通道 1 上的原始脑电信号

器”。

Butterworth 滤波器的频率响应在通频带（即带通滤波器）中是平坦的，在阻带中衰减到零，响应速率取决于滤波器的阶数，滤波器的阶数由电路中使用的无功元件数量决定。一般来说，电感器和电容器是滤波器中使用的无功元件，但

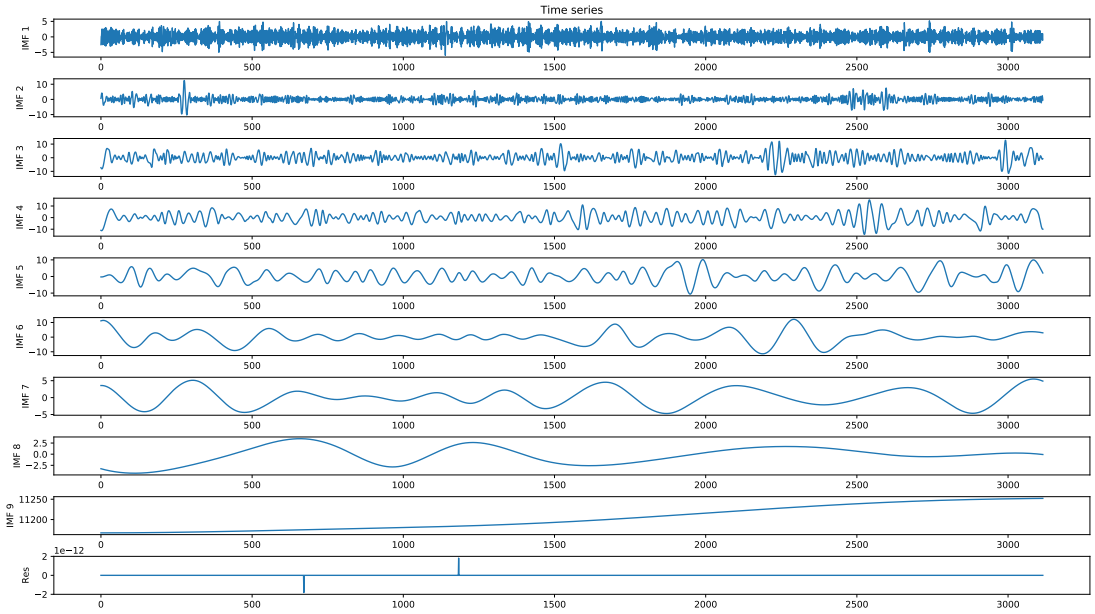


图 4-5　EMD 分解字符 char13 通道 1 得到的 IMF 分量

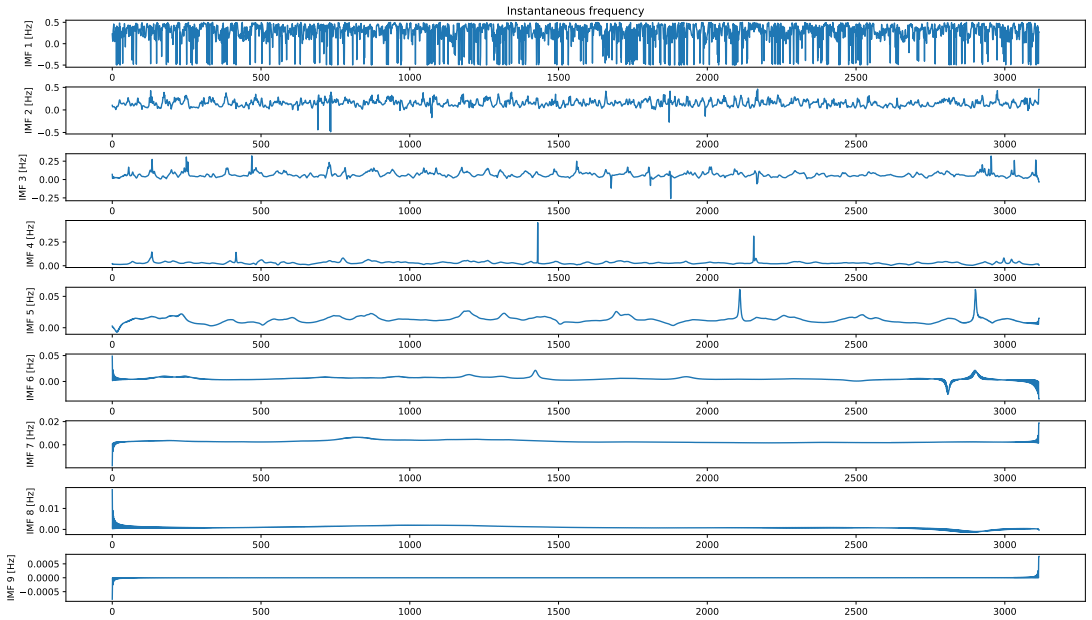


图 4-6　字符 char13 八个 IMF 分量的瞬时功率

在 Butterworth 滤波器中，只使用电容器。因此，电容器的数量将决定滤波器的阶数。考虑到 P300 诱发脑电波信号的频率范围为 0.5~25Hz，频率较低，因此选用带低通滤波器的 Butterworth 滤波器对原始信号进行处理。

理论上，复杂的高阶滤波器可以获得接近理想特性的特性。但实际设计过程中，增加滤波器的阶数会导致级联级数也会增加，会在通频带产生过多的纹波，难以到 Butterworth 滤波器的理想频率响应。图4-7展示了不同阶数的 Butterworth 滤波器的频率响应：

n 阶 Butterworth 低通滤波器频率响应可由式7计算得到。

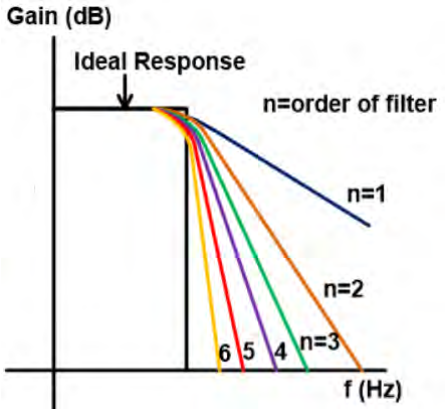


图 4-7　不同阶数 Butterworth 滤波器的频率响应

$$H(j\omega)=\frac{1}{\sqrt{1+\varepsilon^2\left(\frac{\omega}{\omega_C}\right)^{2n}}} \tag{7}$$

其中，n 表示滤波器的阶数，ω 表示通频带频率，ω_C 表示截止频率，ε 表示最大通频带增益。

ε 的值可由式8计算得到。

$$H_1=\frac{H_0}{\sqrt{1+\varepsilon^2}} \tag{8}$$

其中，H_1 表示最小通频带增益，H_2 表示最大通频带增益。

本文选用的是四阶低通 Butterworth 滤波器，它由两个二阶低通数 Butterworth 滤波器级联而成。

四阶低通 Butterworth 滤波器的频率响应如图4-8所示：

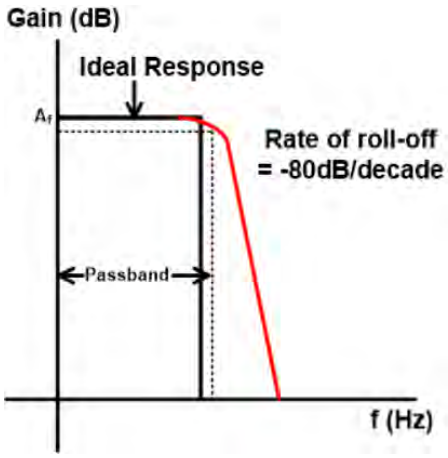


图 4-8　四阶低通 Butterworth 滤波器频率响应

按照上述步骤，本文利用四阶低通 Butterworth 滤波器对五个被试者在各个字符 20 个通道的上的训练数据和测试数据进行滤波处理。以第一个被试者（S1）在训练集和测试集上的滤波情况进行说明。

被试者 S1 在训练集中字符 char02(D) 通道 17 的原始脑电信号以及利用四阶低通 Butterworth 滤波器滤波处理后的结果如图4-9所示。进一步分析滤波后数据

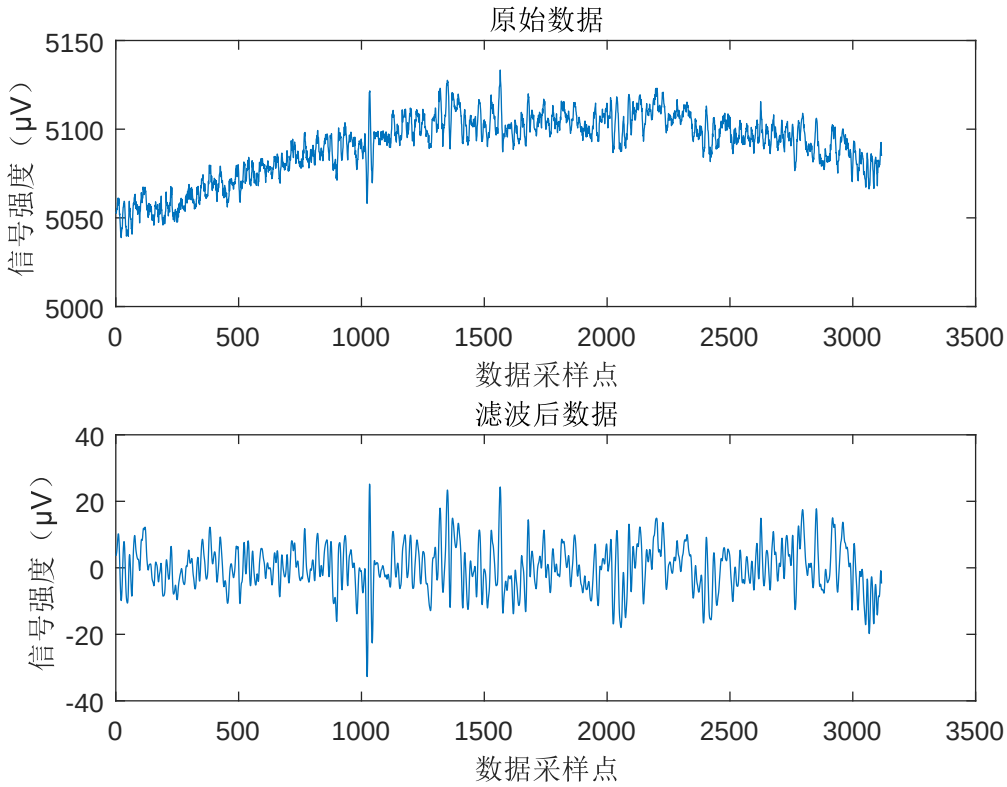


图 4-9　字符 char02(D) 通道 17 滤波前后信号对比

的频谱，结果如图4-10所示。被试者 S1 在测试集中字符 char15 通道 17 的原始

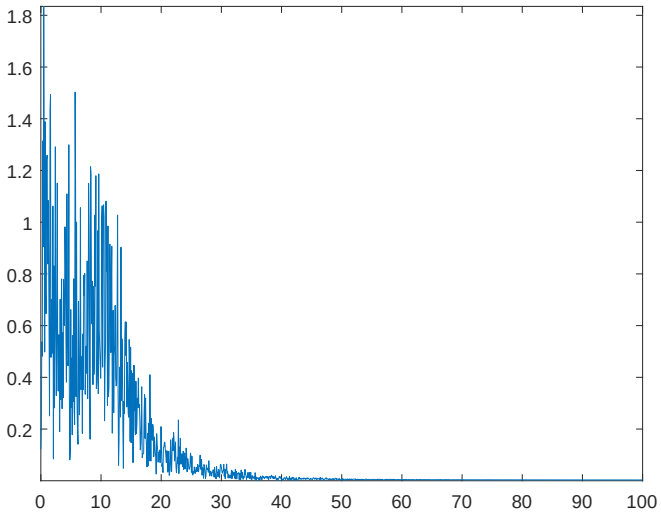


图 4-10　滤波后字符 char02(D) 通道 17 的数据频谱

脑电信号以及利用四阶低通 Butterworth 滤波器滤波处理后的结果如图4-11所示。进一步分析滤波后数据的频谱，结果如图4-12所示。利用四阶低通 Butterworth 滤波器处理后的数据，其频率均位于 0.3～25Hz 之间，符合 P300 诱发脑电信号的

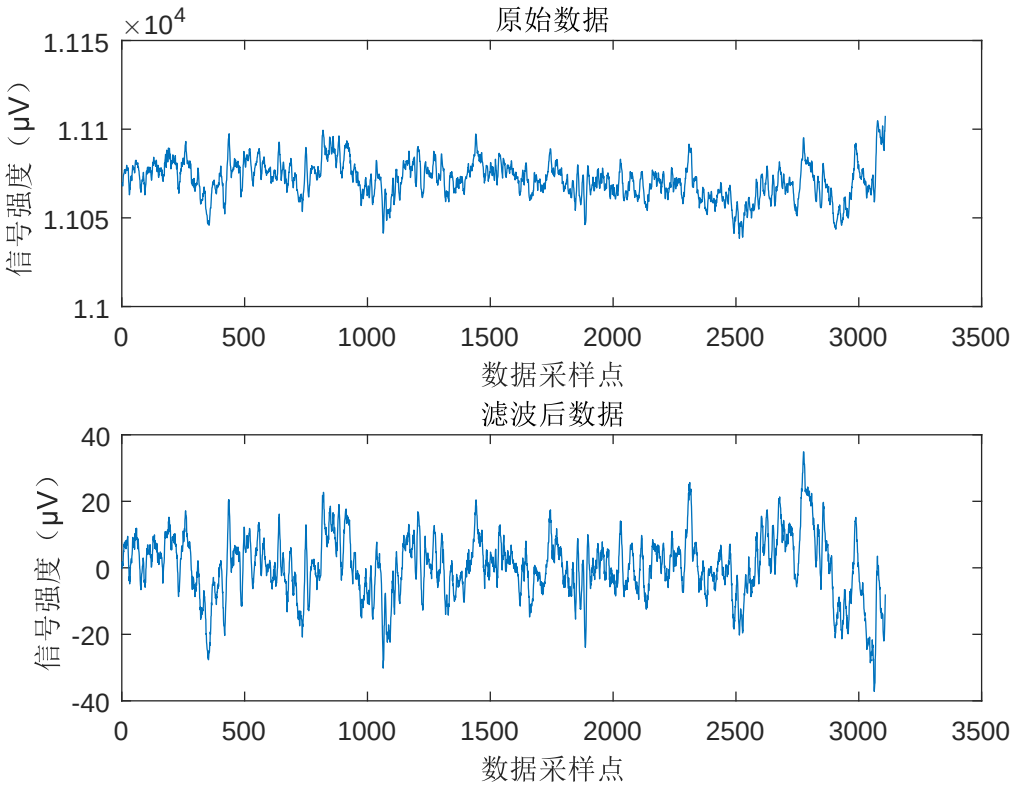


图 4-11　字符 char14 通道 1 滤波前后信号对比

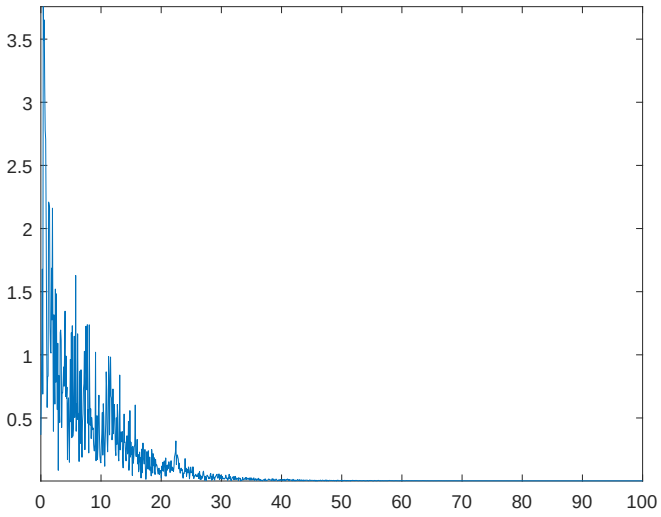


图 4-12　滤波后字符 char14 通道 1 的数据频谱

频率范围，有效去除了处于其它频段的干扰信号，有效提升了数据质量。

4.2.3 快速傅里叶变换（FFT）

原始脑电信号经过四阶低通 Butterworth 滤波器滤波后，仅保留了频率在 0.5~25Hz 的信号。虽然该过程已经滤除了大部分的干扰信号，但还存在眼电等

伪迹信号，难以准确有效地提取出 P300 信号的特征。为此，本文进一步采用了快速傅里叶变换，进一步处理滤波后的信号，剔除与 P300 诱发脑电信号无关的信息。

傅里叶变换（FT）允许用简单的正弦波表示任意函数，任意函数 $f(t)$ 可以表示为：

$$F(\omega)=\int_{-\infty}^{\infty}f(t)e^{-\mathrm{i}\omega t}dt \tag{9}$$

要使这个积分存在，f 必须是绝对可积的，也即：

$$\int_{-\infty}^{\infty}|f(t)|dt<\infty \tag{10}$$

然而，使用 δ 函数可以表示不完全可积的函数（例如周期函数）的变换。当 f 是周期为 T 的周期函数时，它的变换在通频带中不是连续的，而是由周期为 $1/T$ 的分隔脉冲组成。

离散傅里叶变换（DFT）是傅里叶变换的离散化形式。在有限时间内采样的函数由时间序列 $\{x(0),x(1),\ldots,x(N-1)\}$ 定义，$x(t)$ 的离散傅里叶变换形式为：

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-\mathrm{i}2\pi nk/N},\quad k=0,1,\ldots,N-1 \tag{11}$$

为了简化表达式，定义常数 W:

$$W=e^{-\mathrm{i}2\pi/N} \tag{12}$$

对各个信号加和可以得到：

$$X(k)=\sum_{n=0}^{N-1}x(n)W^{nk} \tag{13}$$

对应的逆函数是逆离散傅里叶变换（IDFT）：

$$x(t)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W^{-nk} \tag{14}$$

快速傅里叶变换充分利用了离散傅里叶变换（DFT）计算式中指数因子所具有的对称性质和周期性质，对所求出这些短序列相应的 DFT 进行适当组合，将 2 点 DFT 和 4 点 DFT 推广到 8 点，16 点.....，得到 FFT 算法，从而达到删除重复计算，减少乘法运算和简化结构的目的，提升了计算速度。

利用 FFT 及其逆变换 IFFT 对上述滤波后的数据进行处理，可以进一步提升数据质量。以上述四阶低通 Butterworth 滤波器对字符 char14 通道 1 滤波后的数据为例，经过 FFT 处理后可以进一步去除部分干扰噪声，数据更加平坦质量更高，如图4-13所示。

4.3 数据集构建与划分

经过上述分析，原始脑波信号经过 Butterworth 滤波以及 FFT 处理后，数据质量得到了明显提升。接下来，本文利用预处理后的数据建立模型分类所需的数据集。

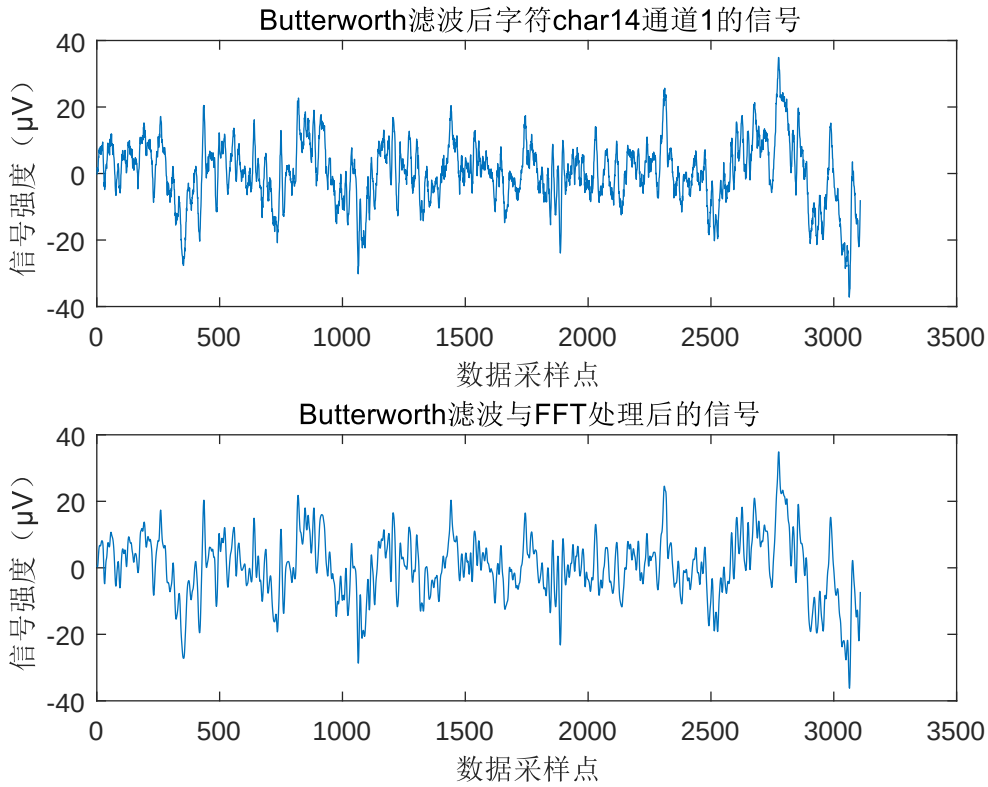


图 4-13　FFT 处理前后字符 char14 通道 1 的信号对比

题目提供了 5 个健康成年被试（S1-S5）的 P300 脑机接口实验数据，包括 5 轮训练数据和 5 轮测试数据。由于问题一不允许使用测试数据，因此下面仅对训练数据进行分析。每个人每轮的训练数据包含 12 个已知字符的电刺激信号（char01～char12）。具体以 char01 为例，被试者会连续观察 12 次行列信息，行数 1～6 以及列数 7～12 的顺序是随机出现的。字符 char01 对应字母 B，如图4-14所示，字符 B 对应第 1 行和第 8 列。当被试者观察到第 1 行或第 8 列时，会产生 P300 诱发脑波信号（这个信号的生成会有一定程度的延迟），当看到其余 10 个行或列时则不会。经过反复测试，本文最终选取看到对应行列时刻之后 160 个采样点时间段内的脑电波信号作为 P300 有效信息。由于原始的字符共有 36 个，若直接采用数据进行分类，则需要处理一个 36 分类问题，然而我们只有其中十个分类的训练数据，且每个分类的训练数据都很有限，无法直接求解这个 36 分类问题。

考虑到不同个体受到刺激的延迟时间不尽相同，且训练数据有限，将不同被试者的数据混合训练可能会个体差异导致模型训练失败或者难以把握总体规律。因此，本文将不同被试者的数据作分别的训练。为此，本文将被试者看到行或列后是否会产生 P300 脑电波信号作为数据标签，将原始问题转化为一个二分类问题。对于字符 char01 而言，被试者观察第 1 行和第 8 列时所产生的脑波信号的标签对应 1，其余 10 个行和列的脑波信号的标签对应 0，因此每轮测试中，每个字符包括 2 个为 1 的标签和 10 个为 0 的标签，分别对应 12 个 P300 有效信息

为了能够有效评估不同模型的性能，对于训练集中提供的 12 个已知字符的数据，本文按 5:1 随机划分训练集和验证集集，选择了 2 个已知字符（char01(B)，char07(S)）作为验证集数据，剩余 10 个字母作为训练集数据，总共可以得到 $10 \times 5 \times 12 = 600$ 个训练样本，以及 $2 \times 5 \times 12 = 120$ 个验证样本。对于每一个

图 4-14　行/列的标识符

训练集而言，每轮测试中只有 2 个 1 标签（已知字符对应行和对应列）和 10 个 0 标签（其余 10 个行和列），正负样本比例为 5:1。由于此时的数据正负样本比例不是 1:1，如果直接利用这些数据进行模型分类训练，模型难以真实准确地学习到输入输出之间的映射关系。

为此，本文采用了样本标签平衡策略，将标签为 1 的样本重复采样五次，使得 0 标签和 1 标签的比例为 1:1。以字符 char01(B) 为例，采用样本标签平衡策略后，会向原始 12 个样本数据（1 个第 1 行样本，1 个第 8 列样本以及 10 个其余行列的样本）中再添加 4 个第 1 行样本和 4 个第 8 列样本（对应标签为 1），此时样本数据由 12 个扩充为 20 个（5 个第 1 行样本，5 个第 8 列样本以及 10 个其余行列的样本）,0 标签和 1 标签数据比例为 1:1。

4.4 模型建立

基于上一小节建立的二分类数据集，可以使用不同的方法建立二分类模型，包括支持向量机（SVM）、全连接神经网络（DNN）、长短时记忆网络（LSTM）、一维卷积神经网络（1D CNN）。

4.4.1 支持向量机（SVM）

考虑到脑电信号是非线性的，本文选用非线性 SVM 进行数据分类。首先，利用非线性算 $\Phi(\cdot)$ 将输入模式 $\mathbf{X}$ 映射到高维空间 $\mathcal{H}$，从而将线性 SVM 推广到非线性 SVM。得到的非线性 SVM 分类器可以用下式表示：

$$f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b \tag{15}$$

其中，原始数据 $\mathbf{X}$ 是非线性的，但经过非线性算 $\Phi(\cdot)$ 处理后得到的 $\Phi(\mathbf{x})$ 是线性的。

经过非线性变换后，换句话说，SVM 找到了导致这两个类的“边界”示例的决策函数值之间最大分隔的超平面。从数学上讲，可以通过最小化以下成本函数来找到该超平面：

$$\min J(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \tag{16}$$

需要满足约束条件：

$$y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \qquad \xi_i \geq 0;\ i = 1, 2, \ldots, l \tag{17}$$

al networks with
due to difficulty
rovements made
efficient training
tail later in this
er of computers
orithms on large
phical processor
rk is successfully
owever it is rarely
chanical systems
k.

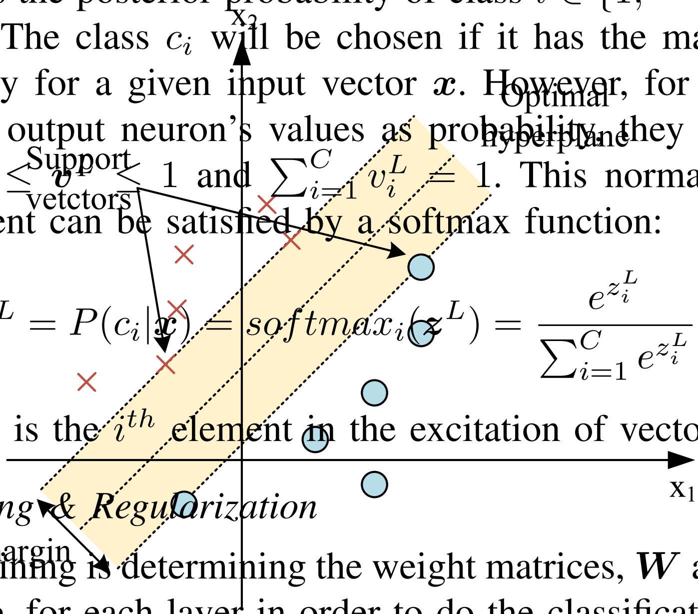
n of deep neural
e their eligibility
ical system from
sis platform, the
acoustic pressure
domain. Then, a
minative features
network, a fea-
is robust against
te diagnostic sys-
nosis engineering
s the interference
runs as real time
sing platform. A
se 250W squirrel
one-stage helical
ed to validate the

llowing: Section
III introduces the
test rig. Obtained
uding remarks are

L

ational multilayer
wo) hidden layers
network with L
put vector of the
s are denoted by

The output layer activation function needs to be chosen differently for accomplishing a classification task. In a multi-class problem with C classes, the value of i^{th} output neuron represents the posterior probability of class $i \in \{1, \cdots, C\}$ as $P(c_i|\boldsymbol{x})$. The class c_i will be chosen if it has the maximum probability for a given input vector $\boldsymbol{x}$. However, for considering the output neuron's values as probability, they need to satisfy $0 \leq v^L \leq 1$ and $\sum_{i=1}^{C} v_i^L = 1$. This normalization requirement can be satisfied by a softmax function:



$$\boldsymbol{v}^L = P(c_i|\boldsymbol{x}) = softmax_i(\boldsymbol{z}^L) = \frac{e^{z_i^L}}{\sum_{i=1}^{C} e^{z_i^L}} \quad (3)$$

where z_i^L is the i^{th} element in the excitation of vector $\boldsymbol{z}^L$.

B. Training & Regularization

The training is determining the weight matrices, $\boldsymbol{W}$ and bias vectors, $\boldsymbol{b}$, for each layer in order to do the classification job. Suppose, a training set with M sample pairs $\mathbb{S} = \{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}$ is given where the vector $\boldsymbol{y}_k$ is posterior probability vector associated with the k^{th} input vector $\boldsymbol{x}_k$. Usually, hard class labels are used for $\boldsymbol{y}_k$ in training set which means $\boldsymbol{y}_k$ has zero elements except just an element equal to 1 corresponding to the class that $\boldsymbol{x}_k$ belongs to it. The training can be done by minimizing the following cost function:

$$J(\boldsymbol{W}, \boldsymbol{b}; \mathbb{S}) = \frac{1}{M} \sum_{k=1}^{M} J_{CE}(\boldsymbol{W}, \boldsymbol{b}; \boldsymbol{x}_k, \boldsymbol{y}_k) + \lambda \|\boldsymbol{W}\|_2^2 \quad (4)$$

图 4-15 SVM 分类示意图[6]

本文中 SVM 分类器使用 sklearn 库中的 svm 方法，参数设置为：C=10（错误项的惩罚系数），kernel='linear'（核函数选用线性核函数），其余参数为默认参数。

4.4.2 全连接神经网络（DNN）

全连接神经网络(DNN)是最朴素的神经网络，它的网络参数最多，计算量最大。DNN 的结构不固定，一般神经网络包括输入层、隐藏层和输出层，一个 DNN 结构只有一个输入层，一个输出层，输入层和输出层之间的都是隐藏层。每一层神经网络有若干个神经元，层与层之间神经元相互连接，层内神经元互不连接，而且下一层神经元连接上一层所有的神经元。图4-16展示了一个 3 层的 DNN 网络。

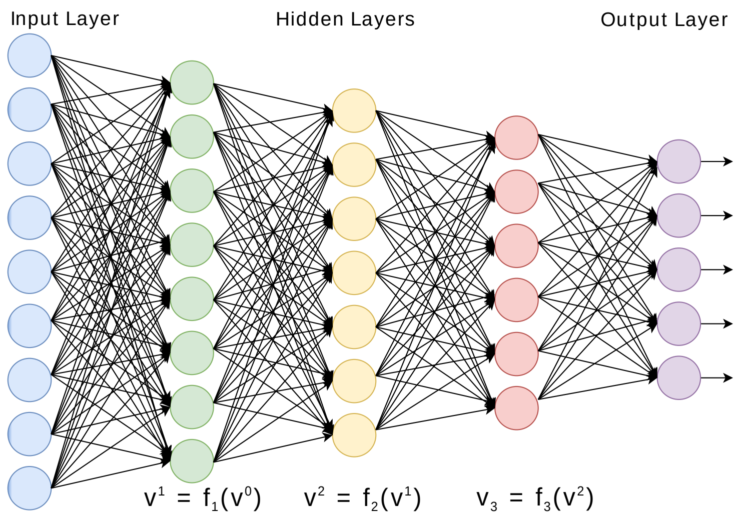


图 4-16 DNN 网络示意图[7]

Fig. 1. A DNN with three hidden layers (L=5). Input and output layers are counted separately. Each layer's output is obtained by applying a nonlinear map to its predecessor layer output.

考虑到训练数据较少，不适合搭建很深的 DNN 网络，因此搭建一个三层的

DNN 网络如4-17所示。

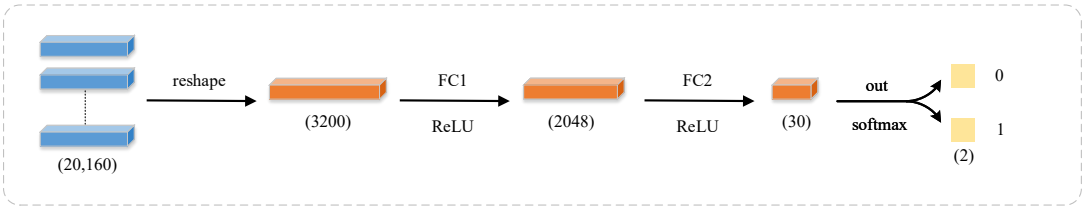


图 4-17　DNN 网络结构图

输入尺寸为 (20,160) 的数据，160 是采样的序列长度，20 为通道数目；将输入 reshape 成一个一维的张量，张量尺寸为 (3200)；将 reshape 后的张量输入到 2 个串联的全连接层中，全连接层的输出尺寸分别为 (2048) 和 (30)，激活函数采用 ReLU 非线性激活函数。最后，特征经过输出层和 softmax 激活函数，输出一个二分类结果。

4.4.3 长短时记忆网络（LSTM）

每个被试者在单次实验中所记录的脑电波数据信号，是一个按固定采样频率采样的时间序列。为此，本文选用 LSTM 网络提取数据之间包含的隐藏时序性信息。

LSTM 是一种特殊的循环神经网络。除了短时记忆外，还可以学习数据间的长期依赖信息 [5]。LSTM 通过特殊的机制设计来避免长期依赖问题，从而解决长序列训练过程中的梯度消失和梯度爆炸问题。

LSTM 包含有四个不同的模块，以一种非常特殊的方式进行交互，如图4-18所示。

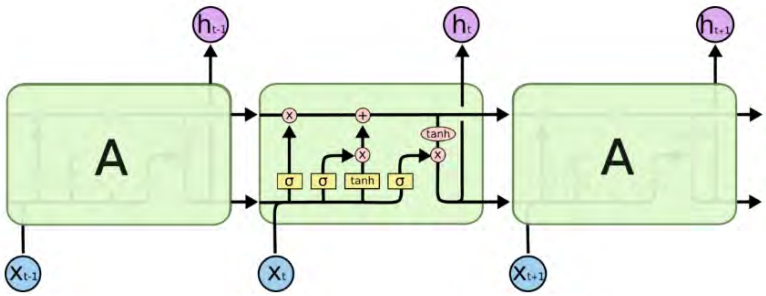


图 4-18　LSTM 中的重复模块

LSTM 的关键就是细胞状态，水平线在图上方贯穿运行。细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。LSTM 通过精心设计的被称为“门”的结构来去除或者增加信息到细胞状态中。门是一种决定信息是否通过的方法，包含一个 sigmoid 神经网络层和一个按位的乘法操作。Sigmoid 层输出 0 到 1 之间的数值，0 代表“不许任何量通过”，1 就指“允许任意量通过”。

LSTM 中的第一步是决定需要从细胞状态中丢弃什么信息。这个决定是通过遗忘门完成的。该门会读取 $h_(t-1)$ 和 x_t，输出一个在 0 到 1 之间的数值给细胞状态 $C_(t-1)$。

$$f_t = \sigma\left(W_f \times [C_{t-1}, h_{t-1}, x_t] + b_f\right) \tag{18}$$

接着，LSTM 决定要往 cell 中存储哪些新信息。一方面，通过构建一个输入门 (input gate)，决定要更新哪些信息。

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{19}$$

另一方面，构建一个候选值向量 (cell)：$C_(t-1)$，之后会用输入门点乘这个候选值向量，来选出要更新的信息。

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{20}$$

之后，LSTM 更新细胞状态 $C_(t-1)$ 为 C_t，f_t 点乘 $C_(t-1)$ 代表掉要丢弃遗忘的信息。$\tilde{C}$ 点乘 i_t 代表候选值向量中要更新记住的信息。

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{21}$$

最后，就是 LSTM 决定输出什么。该步骤需要构建一个输出门 (output gate)，决定要输出哪些信息。

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{22}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{23}$$

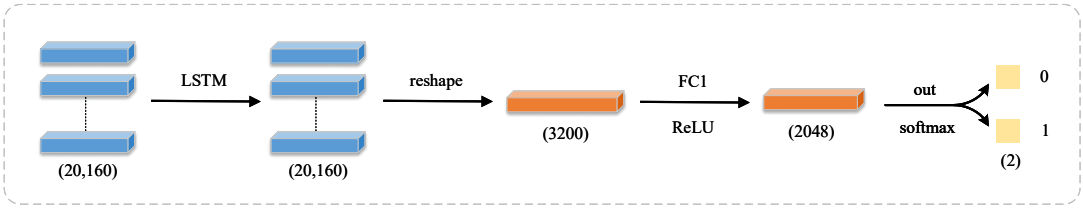


图 4-19　LSTM 网络结构

与 DNN 网络一样，考虑到训练用的数据较少，搭建的 LSTM 网络层数也不多。如图4-19所示。输入尺寸为 (20,160) 的数据，160 是采样的序列长度，20 为通道数目；将输入经过一个 LSTM 层提取时序信息后，将输出 reshape 成一个一维的张量，张量尺寸为 (3200)；将 reshape 后的张量输入到全连接层中，全连接层的输出尺寸为 (2048)，激活函数采用 ReLU 非线性激活函数。最后，特征经过输出层和 softmax 激活函数，输出一个二分类结果。

4.4.4 一维卷积神经网络（1D-CNN)

卷积神经网络（Convolutional Neural Networks, CNN）是一类包含卷积计算且具有深度结构的前馈神经网络（Feedforward Neural Networks），是深度学习（deep learning）的代表算法之一。1D CNN 可以很好地应用于传感器数据的时间序列分析（比如陀螺仪或加速度计数据），同样也可以很好地用于分析具有固定长度周期的信号数据（比如音频信号、脑电波信号），图4-20展示了一个具有 3 个 CNN 隐藏层和 2 个和 MLP 隐藏层的 1D CNN 网络。

1D CNN 的结构由以下超参数构成：

（1）CNN 和 MLP 层的隐藏层数和神经元数量，用于设计网络的具体结构。

（2）每层 CNN 中的过滤器（kernel）大小，用于实现卷积运算。

（3）池化层以及激活函数的选取。

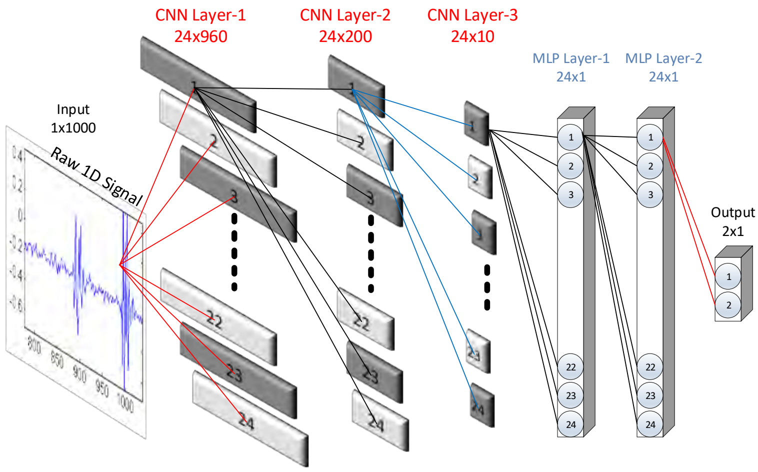


Figure 5: A sample 1D CNN configuration with 3 CNN and 2 MLP layers.

图 4-20　1D CNN 网络示意图 [4]

As in the conventional 2D CNNs, the input layer is a passive layer that receives the raw 1D signal and the output layer is a MLP layer with the number of neurons equal to the number of classes. Three consecutive CNN layers of a 1D CNN are presented in Figure 6. As shown in this figure, the 1D filter kernels have size 3 and the sub-sampling factor is 2 where the k^{th} neuron in the hidden CNN layer, l, first performs a sequence of convolutions, the sum of which is passed through the activation function, f, followed by the sub-sampling operation. This is indeed the main difference between 1D and 2D CNNs, where 1D arrays replace 2D matrices for both kernels and feature maps. As a next step, the CNN layers process the raw 1D data and "learn to extract" such features which are used in the classification task performed by the MLP-layers. As a consequence, both feature extraction and classification operations are *fused* into one process that can be optimized to maximize the classification performance. This is the major advantage of 1D CNNs which can also result in a low computational complexity since the only operation with a significant cost is a sequence of 1D convolutions which are simply linear weighted sums of two 1D arrays. Such a linear operation during the Forward and Back-Propagation operations can effectively be executed in parallel.

本文中的 1D-CNN 网络搭建如图 4-21 所示。输入尺寸为 (20,160) 的数据，160 是采样的序列长度，20 为通道数目；将输入经过一个 1-D CNN 层提取时序信息后，经过池化层降采样到 (5,160)；再将输出 reshape 成一个一维的张量，张量尺寸为 (400)；将 reshape 后的张量输入到全连接层中，全连接层的输出尺寸为 (30)，激活函数采用 ReLU 非线性激活函数。最后，特征经过输出层和 softmax 激活函数，输出一个二分类结果。

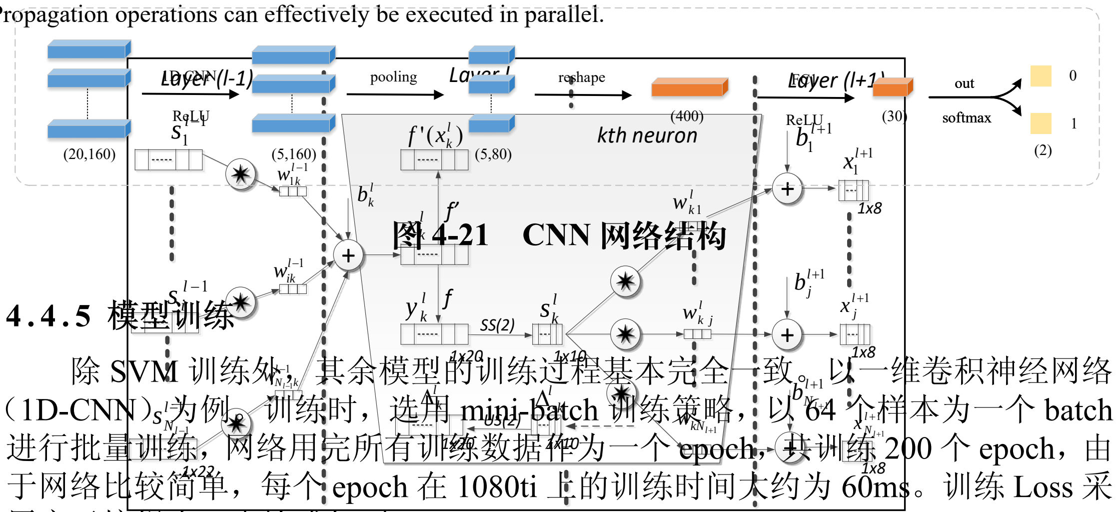


图 4-21　CNN 网络结构

Figure 6: Three consecutive hidden CNN layers of a 1D CNN [56].

4.4.5 模型训练

除 SVM 训练外，其余模型的训练过程基本完全一致，以一维卷积神经网络（1D-CNN）为例。训练时，选用 mini-batch 训练策略，以 64 个样本为一个 batch 进行批量训练，网络用完所有训练数据作为一个 epoch，共训练 200 个 epoch，由于网络比较简单，每个 epoch 在 1080ti 上的训练时间大约为 60ms。训练 Loss 采用交叉熵损失，表达式如下：

$$H(p,q) = -\sum_{x} p(x)\log q(x) \tag{24}$$

式中，p 表示理想结果即正确标签向量和 q 表示神经网络输出结果即经过 softmax 转换后的结果向量。

模型训练采用 Adam 算法进行优化，通过反向传播更新模型权重。每训练完一个 epoch，使用验证集数据对模型准确率进行评估。

This is also an adaptive implementation since the CNN topology will allow the variations in the input layer dimension in such a way that the sub-sampling factor of the output CNN layer is tuned adaptively. The details related to Forward and Back-Propagation in CNN layers are presented in the next sub-section.

2.3　Forward and Back-Propagation in CNN-layers

In each CNN-layer, 1D forward propagation (1D-FP) is expressed as follows:

$$x_k^l = b_k^l + \sum_{i=1}^{N_{l-1}} conv1D(w_{ik}^{l-1}, s_i^{l-1}) \tag{2}$$

4.5 模型性能评估

为了评估所建立的不同预测模型间的性能差异，本文选用相同的数据分别训练和测试了 SVM、DNN、LSTM 以及 CNN 四个预测模型，模型的预测精度通过其在验证集上的识别准确率进行评估。

值得注意的是，训练数据额测试数据均来源于训练集中的 12 个已知字符数据，其中测试数据对应验证集上的 2 个已知字符数据，训练数据对应训练集上其余 10 个已知字符的数据。在验证集上，预测模型的目标输出是二分类，也即判断当前所在的行或者列是否是待测试字符所在的行或者列, 同时，训练数据会按照数据集构建小节中的步骤，利用重采样实现正负样本比例为 1:1，确保预测模型评估的准确性。

SVM、DNN、LSTM 以及 CNN 四个预测模型在验证集上的识别准确率（二分类准确率）如表1所示。

表 1　不同分类模型在验证集上的二分类准确率

预测模型	CNN	LSTM	DNN	SVM
二分类准确率	77.87%	73.44%	71.12%	68.17%

4.6 结果预测

4.6.1 测试集结果预测

综合上述分析，不难发现本文所建立的预测模型的输出是目标字符所在的行或者列的得分。这里的得分可以看作是预型预测目标字符位于某一行或某一列的概率，得分越高，表明字符位于某一行或某一列的概率越高，该得分是通过神经网络的 softmax 输出得到的。

以被试者 S1 预测字符 char13 为例，在第 1 轮测试中，他看到图4-20中行或列的顺序依次为 8,10,6,12,11,4,5,7,9,2,3,1，其中 1∼6 代表行，7∼12 代表列。利用之前数据及构建与划分中的方法，我们可以将第一轮测试中采样到的信号划分为 12 个 P300 信息，分别与这 12 个行或列数相对应。被试者 S1 首先看到的是第 8 列，此时将与之对应的 P300 有效信息（从刺激开始往后 160 次采样时间段内的脑波信号）作为预测模型的输入，由于所建立的预测模型是 2 分类的，因此 softmax 层会输出一个 0∼1 之间的浮点数，记为当前行或者列的预测得分。同理，被试者 S1 之后看到剩余 11 个行或者列时，重复这一过程，就可以获得此时行或者列的预测得分。每个被试者在测试集上实验一轮，我们总共可以得到 12 个介于 0∼1 之间的预测得分。

进一步对这 12 个预测得分按行列进行分类排序，也即将 1∼6 行对应的预测得分按大小排序，从中挑选出得分最高的行，将其作为本轮测试中被试者预测的目标字符所在的行。同理，将 7∼12 列对应的预测得分按大小进行排序，并从中挑选出得分最高的列，将其视为本轮测试中被试者预测的目标字符所在的列。如题所述，每个被试者对同一个待预测的字符都需要测试 5 轮，因此 5 轮测试下来，我们会得到 5 个得分最高的行和 5 个得分最高的列。由于不同被试者之间由于反应延迟以及个体差异等因素，所产生的 P300 有效信号会存在差异，上述过程需要对每个被试者单独分析，也即不能将 5 个被试者的测试数据混合在一起后再进行测试。因此，当 5 名被试者对同一个待预测字符分别做完 5 轮实验后，我们总共会得到 $5 \times 5 = 25$ 个得分最高的行，以及 $5 \times 5 = 25$ 个得分最高的列。

基于统计学原理，我们分别从 25 个得分最高的行和列中选取出现频次最高的行和列作为当前待预测字符所在的行和列。由于二维矩阵中，只需要知道行和列就可以锁定对应的元素，因此，我们可以利用挑选出的最优行和最优列锁定目标预测字符。例如，对于字符 char13，在 25 个得分最高的行中出现频率最高的是第 3 行，在 25 个得分最高的列中出现频率最高的列是第 7 列，由图4-20可知，待预测的目标字符为 M。

通过对网络进行 20 次重复训练和预测，我们可以获得测试集上每个字符的 20 个预测结果。值得注意的是，模型 20 次反复预测的结果具有高度的一致性，也即对于每个待预测的字符，20 次都会得到相同的预测结果，只会在极个别情况下会出现不同的结果（例如 18 次或者 19 次预测结果均为同一个字符，只有 1~2 次预测结果为其它字符），因此基于统计学原理，我们可以确定测试集上的 01 个待测字符。**例如，我们最终预测出测试集 char13~char22 的结果分别"M，F，5，2，I，T，K，X，A，0"**，如表2所示。因此，我们将这个预测结果作为参考结果，用于探究实验轮次对预测准确率的影响，从而在保证预测准确率的情况下确定最少的观测轮次。

表 2　预测结果

预测结果 / 待测字符 / 被试者序号	char13	char14	char15	char16	char17	char18	char19	char20	char21	char22
member 1	M	F	5	2	I	T	K	X	A	0
member 2 预测结果	M	F	5	2	I	T	K	X	A	
member 3 预测结果	M	F	5	2	I	T	K	X	A	
member 4 预测结果	M	F	5	2	I	T	K	X	A	0
member 5 预测结果	M	F	5	2	I	T	K	X	A	0

4.6.2 基于预测结果的性能分析

在上一小节分析的基础上，本文进一步研究了测试轮次与预测准确率之间的关系。具体来说，本文分别测试了实验轮次为 1~5 时，预测模型在测试集上的预测结果相较于参考结果的准确率。为了避免单次实验造成随机误差，对于每个实验轮次，本文都重复测试了 20 次，并计算了平均预测准确率，所得到的测试轮次与预测准确率之间的关系如表3所示。不难发现，当观测轮次为 5 时，网络的平均预测准确率高达 97%。然而，如果只观测一轮时，网络的平均预测准确率仅有 41.5%。值得注意的是，即使只进行 2 轮测试，本文所建立的预测模型仍能达到 82.5% 的预测准确率，这个结果再次验证了本文所提出预测模型的有效性和准确性。

表 3　使用不同轮次测试数据下的平均字符判断准确率

测试轮次	1	2	3	4	5
字符预测准确率	41.5%	82.5%	85%	86%	97%

为了更加直观地比较不同实验轮次下的结果，下面用图4-22和图4-23分别展示试验轮次 1~5 时，20 次重复实验下的行和列的预测准确率与平均预测准确率。其中，20 条折线分别代表 20 次重复实验，柱状图的横坐标表示实验轮次，纵坐标表示 20 次重复实验下行和列的平均预测准确率。图4-24表示 20 次重复实验下的目标字符的预测准确率和平均预测准确率。

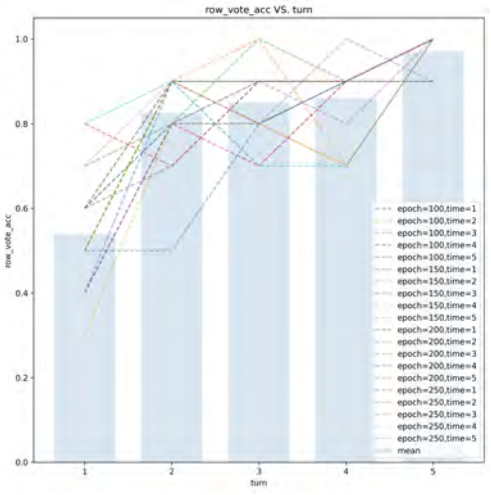


图 4-22　行预测准确率

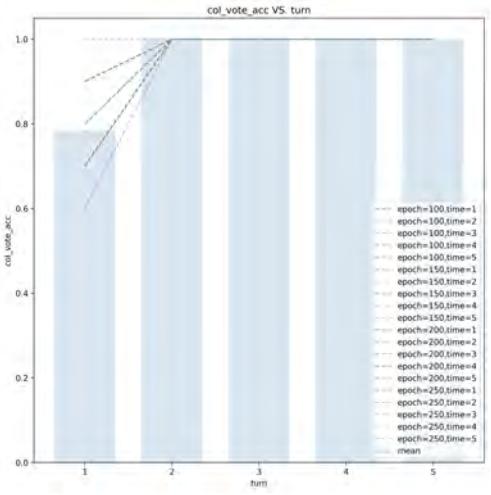


图 4-23　列预测准确率

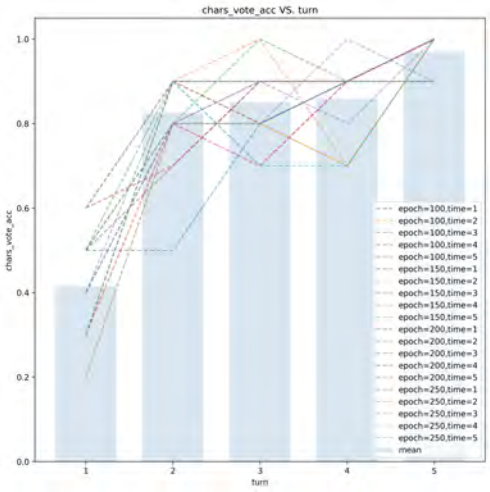


图 4-24　字符预测准确率

5. 问题二的分析与求解

5.1 问题二的分析

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。

为了找到最优的组合通道数，本文将注意力机制 [9] 算法引入 1D-CNN 网络中，构建如5-2所示的神经网络。

5.2 注意力机制模型

注意力机制的引入可以使得神经网络具备专注于其输入（或特征）子集的能力，也即能够从多个输入中优先选择特定的输入。在计算能力有限或者输入信息冗余的情况下情况下，注意力机制是解决信息超载问题的有效方案。

5.2.1 注意力机制算法原理

通道注意力机制通过通过对各通道的依赖性进行建模，可以区分不同特征通道的重要程度，然后针对不同的任务增强或者抑制不同的通道，也即对特征进行逐通道调整，这样网络就可以学习通过全局信息来有选择性地加强包含有用信息的特征并抑制无用特征。

图5-1展示了一个基于数据压缩和特征提取的注意力通道机制模型，其中注意力机制模块主要操作包括以下个步骤：

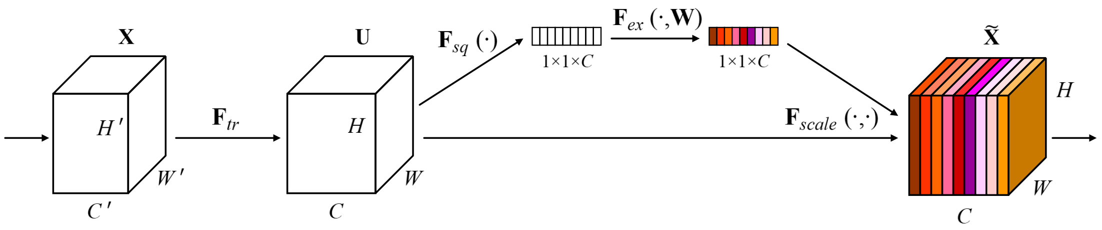


Fig. 1. A Squeeze-and-Excitation block. **图 5-1 通道注意力机制模型示意图 [10]**

（1）Squeeze 操作：将各通道的全局空间特征作为该通道的标识，形成一个通道描述符，具体的计算如下式所示：

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W}\sum_{i=1}^{H}\sum_{j=1}^{W} u_c(i,j) \qquad (25)$$

很明显这个函数做了一个全局平均值，把每个通道内所有的特征值相加再平均，即全局平均池化（global average pooling）。其中 H 和 W 表示输入特征的空间维度，u_c 表示压缩前的原始输入信号，z_c 表示压缩后的注意力机制模块的输入信号。这部分在本文中没有涉及到，因为本文中的研究数据是脑电波信号，不包含空间特征。

（2）Excitation 操作：本文重点采用的是这一步操作的方法，具体计算如下：

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2\delta(\mathbf{W}_1\mathbf{z})) \qquad (26)$$

其中 δ 函数是 ReLU，而 σ 是一个 sigmoid 激活函数，用于激活每一层通道。得到了每一个特征通道的权重之后，就将该权重应用于原来的每个特征通道，之后通过参数 W 调整每个特征通道生成权重，就可以实现有选择性地加强包含有用信息的通道并抑制无效通道。

5.2.2 模型搭建

为了从 20 个通道中选取最优的组合通道数，本文搭建了如图5-2所示的带通道注意力机制的 1D-CNN 神经网络模型。该模型的运行流程主要包括以下几个步骤：

（1）网络初始输入维度的选取：原始训练集中的每一个已知字符都包含 20 个通道，以及3000 个左右的采样点，利用问题一中的数据集构建和划分方法，我们选取每个 event 事件发生后的 160 个采样点作为一个有效区间，因此网络的初始输入维度为 20*160，其中 20 代表 20 个不同的通道。

（2）注意力权重的获取：注意力权重通过一个全连接网络生成，我们将初始数据输入到全连接层，全连接层共有 20 输出，考虑到 softmax 选择通道的权重含

ture (Section 6.4). While the template for the building block is generic, the role it performs at different depths differs throughout the network. In earlier layers, it excites informative features in a class-agnostic manner, strengthening the shared low-level representations. In later layers, the SE blocks become increasingly specialised, and respond to different inputs in a highly class-specific manner (Section 7.2). As a consequence, the benefits of the feature recalibration performed by SE blocks can be accumulated through the network.

The design and development of new CNN architectures is a difficult engineering task, typically requiring the selection of many new hyperparameters and layer configurations. By contrast, the structure of the SE block is simple and can be used directly in existing state-of-the-art architectures by replacing components with their SE counterparts, where the performance can be effectively enhanced. SE blocks are also computationally lightweight and impose only a slight increase in model complexity and computational burden.

To provide evidence for these claims, we develop several SENets and conduct an extensive evaluation on the ImageNet dataset [10]. We also present results beyond ImageNet that indicate that the benefits of our approach are not restricted to a specific dataset or task. By making use of SENets, we ranked first in the ILSVRC 2017 classification competition. Our best model ensemble achieves a 2.251% top-5 error on the test set. This represents roughly a 25% relative improvement when compared to the winner entry of the previous year (top-5 error of 2.991%).

2 RELATED WORK

Deeper architectures. VGGNets [11] and Inception models [5] showed that increasing the depth of a network could significantly increase the quality of representations that it was capable of learning. By regulating the distribution of the inputs to each layer, Batch Normalization (BN) [6] added stability to the learning process in deep networks and produced smoother optimisation surfaces [12]. Building on these works, ResNets demonstrated that it was possible to learn considerably deeper and stronger networks through the use of identity-based skip connections [13], [14]. Highway networks [15] introduced a gating mechanism to regulate the flow of information along shortcut connections. Following these works, there have been further reformula-

improvements in the learning and representational properties of deep networks.

An alternative, but closely related line of research has focused on methods to improve the functional form of the computational elements contained within a network. Grouped convolutions have proven to be a popular approach for increasing the cardinality of learned transformations [18], [19]. More flexible compositions of operators can be achieved with multi-branch convolutions [5], [6], [20], [21], which can be viewed as a natural extension of the grouping operator. In prior work, cross-channel correlations are typically mapped as new combinations of features, either independently of spatial structure [22], [23] or jointly by using standard convolutional filters [24] with 1×1 convolutions. Much of this research has concentrated on the objective of reducing model and computational complexity, reflecting an assumption that channel relationships can be formulated as a composition of instance-agnostic functions with local receptive fields. In contrast, we claim that providing the unit with a mechanism to explicitly model dynamic, non-linear dependencies between channels using global information can ease the learning process, and significantly enhance the representational power of the network.

Algorithmic Architecture Search. Alongside the works described above, there is also a rich history of research that aims to forgo manual architecture design and instead seeks to learn the structure of the network automatically. Much of the early work in this domain was conducted in the neuro-evolution community, which established methods for searching across network topologies with evolutionary methods [25], [26]. While often computationally demanding, evolutionary search has had notable successes which include finding good memory cells for sequence models [27], [28] and learning sophisticated architectures for large-scale image classification [29], [30], [31]. With the goal of reducing the computational burden of these methods, efficient alternatives to this approach have been proposed based on Lamarckian inheritance [32] and differentiable architecture search [33].

By formulating architecture search as hyperparameter optimisation, random search [34] and other more sophisticated model-based optimisation techniques [35], [36] can also be used to tackle the problem. Topology selection as a path through a fabric of possible designs [37] and direct architecture prediction [38], [39] have been proposed

易只集中在一个通道，本文选用 sigmoid 函数作为每个输出的激活函数，从而可以得到 20 个 0~1 之间的输出值，分别对应 20 个通道的权重。

（3）输入数据重构：将所获取的 20 个通道的注意力权重分别与其对应的通道相乘，并将其作为后续 1D-CNN 的输入特征，此时输入特征的维度与初始输入维度相同。

由于注意力权重生成的整个过程是可微的，可以通过对网络进行端到端的训练，自动生成合适的通道注意力权重，无需人为设置。

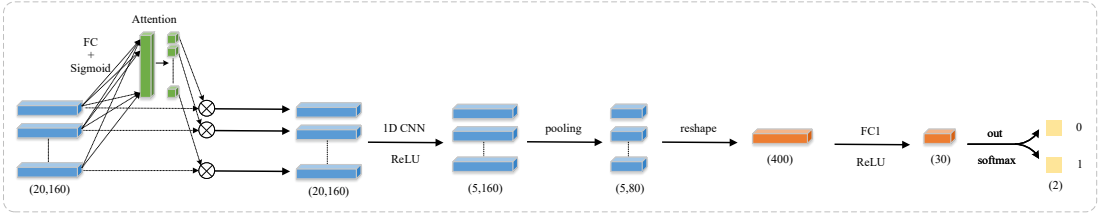


图 5-2　带注意力机制的 1D-CNN 网络图

5.3 基于注意力权重的通道选择

为了选出最优的组合通道数，我们分别获取了每个被试者对 20 个数据通道的平均注意力权重，之后通过取通道交集的方式确定了整体最优的 5 个数据通道，具体的步骤包括：

（1）获取每个被试者一轮实验的注意力权重：在每一轮实验中，我们设置网络训练周期为 100 个 epoch，每次迭代后注意力机制模块都会输出 20 个数据通道的注意力权重。考虑到网络参数会随着迭代次数自动优化更新，我们仅选取最后 10 次迭代的结果求平均值，得到每个被试者一轮实验的平均注意力权重。

（2）获取每个被试者的 5 轮平均注意力权重：重复 5 次（1）中的步骤，将获得的 5 轮注意力权重求平均值，得到每个被试者的 5 轮平均注意力权重，如图5-3所示。

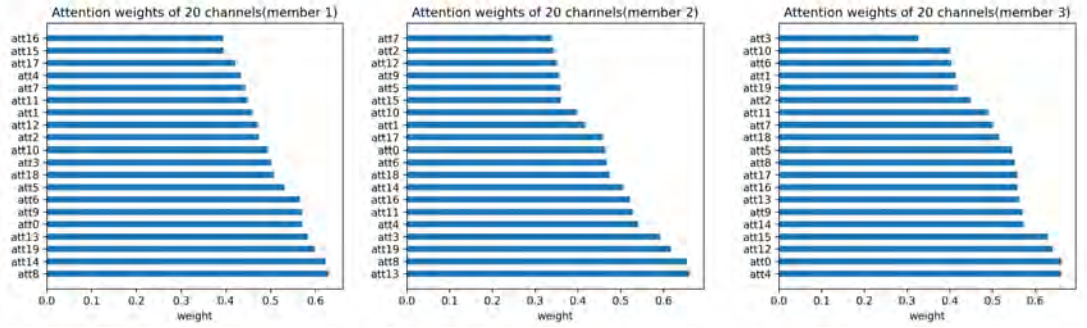
（3）选取每个被试者的最优组合通道：对于每个被试者，我们将 20 个通道按（2）中得到 5 轮平均注意力权重从大到小排序，选取前 15 个通道作为最优组合通道。

（4）选取所有被试者的最优组合通道：为了确定所有被试者的最优组合通道，我们将每个被试者的 15 个最优组合通道取交集，交集的通道数量有 5 个，这 5 个通道就是最终确定的整体最优通道组合。

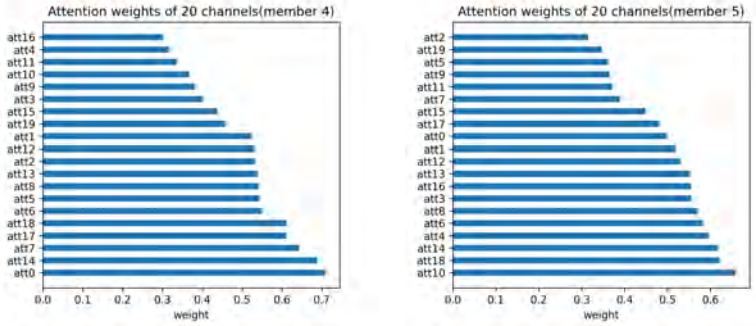
表4展示了每个被试者经过 5 轮测试后各自选取的 15 个最优组合通道，以及取交集后确定的适用于所有被试者的 5 个最优通道组合。

表 4　通道选择结果

被试者序号 \ 通道名称	Fz	F3	F4	Cz	C3	C4	T7	T8	CP3	CP4	CP5	CP6	Pz	P3	P4	P7	P8	Oz	O1	O2
member 1	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓				✓	✓
member 2	✓	✓		✓	✓		✓		✓		✓	✓		✓	✓	✓	✓	✓	✓	✓
member 3	✓		✓		✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
member 4	✓	✓	✓	✓		✓	✓	✓	✓				✓	✓	✓	✓		✓	✓	✓
member 5	✓	✓		✓	✓		✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	
公共通道	✓								✓					✓	✓				✓	



(a) (b) (c)



(d) (e)

图 5-3　5 个被试者的 5 轮平均注意力权重

5.4 性能评估与结果预测

5.4.1 性能评估

基于上一小节的分析和建模内容，本文设计了四个对比实验。第一个实验选用完整的 20 个通道数据进行训练和验证；第二列在原有数据基础上引入注意力机制，比较注意力机制对网络性能的影响；第三个实验分别采用上一小节中各个被试者对应的 15 个最佳通道进行训练和验证；第四个实验采用上一小节得到的适用于所有被试者的五个最优通道。实验结果如表5所示。

表 5　注意力机制与不同通道选择方案性能对比

验证集准确率 / 通道选择方式 / 被试者序号	20 通道	注意力机制	15 通道	5 通道
member 1	78.02	**80.10**	79.86	78.52
member 2	**78.94**	78.07	78.48	75.30
member 3	**79.11**	77.96	78.82	76.75
member 4	78.01	79.28	**80.62**	77.35
member 5	**75.26**	74.25	72.52	68.09
平均	77.87	77.93	**78.06**	75.20

从表中可以看出：（1）引入注意力机制后，网络性能有所提高，但提高不明

显，原因是注意力机制带来的模型参数量增加，使得网络训练难以收敛；（2）选用注意力机制筛选得到的 5 个被试者不同的 15 个通道，性能在仅仅使用注意力机制的基础上得到了进一步提高，由此可见注意力机制筛选得到的通道相较于使用 20 个通道，冗余数据更少，网络提取到的有效信息更多；（3）选用适合所有被试者的 5 个通道后，每个被试者最终的表现都比 15 通道略有下降，但是总体性能依然很高。并且，由于减少通道带来的模型参数量减少，可以缩短训练时间，减少所需的训练样本数量。

5.4.2 结果预测

表6展示了 5 个被试者利用所选取的 5 个最优组合通道在测试集上的字符预测结果。可以看出，通过所选取的 5 个最优组合通道可以准确预测出测试集的前五个字符 char13-char17，验证了所选取的 5 个通道的有效性。

表 6　预测结果

预测结果 / 待测字符 / 被试者序号	char13	char14	char15	char16	char17	char18	char19	char20	char21	char22
member 1	M	F	5	2	I	T	K	X	A	0
member 2 预测结果	M	F	5	2	I	T	K	X	A	
member 3 预测结果	M	F	5	2	I	T	K	X	A	
member 4 预测结果	M	F	5	2	I	T	K	X	A	0
member 5 预测结果	M	F	5	2	I	T	K	X	A	0
参考结果	M	F	5	2	I					

6. 问题三的分析与求解

6.1 问题三的分析

针对问题三，考虑到在 P300 脑-机接口系统中，获取训练模型所需的标签样往往需要花费很长时间，本文设计了一种半监督学习算法来预测待识别字符目标。本文在问题一的基础上，从原始的训练集划分出一半的样本作为有标签样本，将另一半样本作为无标签样本。同时，在问题二得到的最优通道组合基础上，采用简单自训练（simple self-training）的思想，设计半监督学习算法。经验证，该半监督学习算法在问题二的测试数据集（char13-char17）上，能够获得 100%的预测准确率。并且，采用该模型预测的其余识别目标（char18-char22）与第一问和第二问所得到的结果完全相同，由此验证了该半监督学习算法的有效性与准确性。本文使用的半监督学习算法主要包括以下步骤：

步骤一：在问题一的基础上，从原始的训练集上划分出一半的数据作为有标签样本，并按照问题二中的方法选择的最优组合通道，用于训练神经网络。

步骤二：利用步骤一中训练好的网络（为了减少单次实验造成的随机误差，本文重复进行了多次训练，选取网络的平均预测值作为最终的预测结果）为剩下的一半无标签数据贴上伪标签。具体来说，需要根据是否具有先验信息，分别采用两种方法添加标签: （1）无任何先验信息：直接将二分类网络预测出来的结果作为无标签数据的伪标签；（2）基于先验信息：每轮测试中每行和每列会各出现一次正样本，其余为负样本。因此，可以将预测得分最高的行作为行预测结果，将预测得分最高的列作为行预测结果，利用预测的行和列锁定预测的字符，并基于这个预测结果为无标签数据添加伪标签。

步骤三：将步骤二中添加伪标签的数据与有标签的数据混合，重新训练网络并评估性能，重复执行上述步骤二与步骤三，从而得到最终的分类模型；

步骤四：比较两种伪标签方法带来的性能提升。

6.2 基本半监督学习算法

一般，常见的半监督学习算法包括：self-training（自训练算法）、Graph-based Semi-supervised Learning（基于图的半监督算法）、Semi-supervised supported vector machine（半监督支持向量机，S3VM）。简单介绍如下：

（1）简单自训练（simple self-training）：首先用有标签数据训练一个分类器，然后用这个分类器对无标签数据进行分类，这样就会产生伪标签。之后，设计一个伪标签筛选准则，挑选出认为分类正确的无标签样本，用于训练分类器。

（2）协同训练（co-training）：其实也是 self-training 的一种。假设每个数据可以从不同的角度进行分类，而且不同角度可以训练出不同的分类器，然后利用这些从不同角度训练出来的分类器对无标签样本进行分类，再从中筛选出认为可信的无标签样本加入训练集中。由于这些分类器是从不同角度训练出来的，因此可以形成一种互补，从而而提高分类精度。

（3）半监督字典学习：其实也是 self-training 的一种，先是用有标签数据作为字典，对无标签数据进行分类，之后挑选出认为分类正确的无标签样本，加入字典中（此时的字典就变成了半监督字典了）

（4）标签传播算法（Label Propagation Algorithm）：是一种基于图的半监督算法，通过构造图结构（数据点为顶点，点之间的相似性为边）来寻找训练数据中标签数据和无标签数据之间的关系。标签传播算法是一种直推式的半监督算法，即只对训练集中的无标签数据进行分类。

（5）半监督支持向量机：监督支持向量机是利用结构风险最小化来进行分类的。半监督支持向量机利用了无标签数据的空间分布信息，即决策超平面应该与

无标签数据的分布一致。

考虑到上述的五种常见半监督学习算法中，方法 2、3、4 适用于数据样本量较大的情形，并且本文在问题一中已经尝试过方法 5 中使用的支持向量机，但实际效果不及卷积网络和 LSTM 网络。综合分析，方法 1 适用于问题三的求解，因为该问题的数据量较小，且本文在问题一中已经建立了预测效果较好的神经网络端到端模型，便于移植到问题三中建立半监督学习算法的基本架构网络。

本文针对问题三的半监督学习模型的构建思路参考文献 [8]，本文将在卷积神经网络的基础上，构建一种融合标签数据和无标签数据同时训练的监督方式。对于未标记的数据，本文为其提供伪标签，即将无标签的数据输入有标签数据训练所得模型后，选取具有最大预测概率的分类，将其作为无标签数据的伪标签，之后将这些带有伪标签的数据与有标签的数据混合在一起重新训练预测模型，可以得到新的分类模型。这实际上相当于熵正则化，也是半监督学习通常假定的前提。

6.3 半监督模型搭建

对于问题一中已经建立好的基于滤波去噪与卷积神经网络的信号分类器，本文仅需要在此基础上将原始的训练数据集划分成比例为 1:1 的标签样本和无标签样本，之后使用自训练半监督模型架构对其循环训练即可。

6.3.1 伪标签方案设计

其中本文设计了两种不同的伪标签设计方案：

（1）无先验伪标签生成方案：伪标签的设计不参考任何先验信息，直接利用上一轮训练所得的模型对无标签样本进行预测，并将预测得到的结果作为无标签数据的伪标签，其中，正样本只选择 softmax 预测概率大于 0.8 的结果，负样本只选择预测结果小于 0.2 的结果。并且为了不破坏正负样本的平衡，以正样本少于负样本为例，重采样带有正伪标签的样本，使得正负样本比例依然为 1:1；

（2）基于先验的伪标签生成方案：伪签的设计参考先验信息，考虑到每轮测试中每行和每列会各出现一次正样本，其余为负样本。因此，可使用在所有行中预测得分最高的行作为行预测的结果，在所有列中预测得分最高的列作为列预测的结果，将其作为无标签数据的伪标签。

6.3.2 半监督训练流程

基于以上两种伪标签生成方案，分别按照如图6-1所示流程训练模型。

对于无标签数据，我们选择预测概率最大（softmax 某类输出概率大于 0.8 时才将该无标签数据与训练集数据混合进入下一轮训练）的类别作为该样本的伪标签，如下式：

$$y' = \begin{cases} 1 & \text{if } i = argmax_{i'} f'(x) \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

我们选用的损失函数如下式：

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L\left(y_i^m, f_i^m\right) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L\left(y_i'^m, f_i'^m\right) \tag{28}$$

其中 n 和 n' 为 mini batch 的大小，C 为类别个数，f 为网络输出，y 和 y' 分别标签和伪标签。

α 为平衡系数，是关于训练时间 t 的函数，在一开始,α 的值为 0，模型只训练有标记的样本，从而尽快学习，之后 α 开始缓慢增加，迭代次数达到 10 之后，停止增加，并不再增加伪标签样本，从而实现半监督学习，其表达式如下：

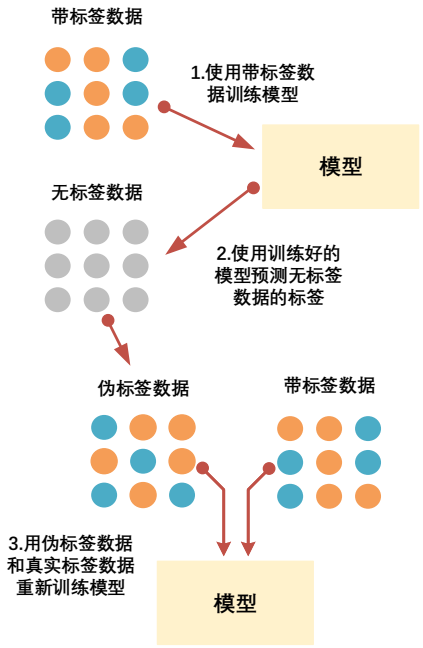


图 6-1　半监督学习模型训练流程图

$$\alpha(t)=\begin{cases}0 & t<T_1\\ \frac{t-T_1}{T_2-T_1}\alpha_f & T_1\le t\le T_2\\ \alpha_f & T_2\le t\end{cases} \tag{29}$$

6.4 性能评估与结果预测

6.4.1 模型训练与评估

图6-2表示了无先验伪标签生成方案下无监督训练的数据分布情况和准确率变化。图中横坐标表示训练的轮次（每轮都完成了 100 个 epoch 的独立训练）。柱状图纵坐标表示加入训练的样本数，折线图表示模型在验证集上的准确率。在第一轮中，没有加入无标签的数据，只有 500 个有标签的样本加入训练。随着伪标签样本加入，参与训练的样本数量越来越多，准确率也随之增加。当然，伪标签的数据并不完全是正确的数据，所以最终的准确率只从 63% 提高到了 67.23%，距离有监督训练准确率依然有一定距离。

图6-3表示基于先验的伪标签生成方案下无监督模型训练的数据分布情况和缺勤率变化。横坐标表示训练的轮次（每轮都完成了 100 个 epoch 的独立训练）。柱状图纵坐标表示加入训练的样本数，折线图表示模型在验证集上的准确率。相比于无先验方法的结果，由于先验带来的信息使得伪标签的正确率得到的大幅提高，因此最终准确率也从 63% 提高到 71.48%，接近有监督训练的准确率（75.2%）。

6.4.2 结果预测

本小节设计了两种添加伪标签的方法，实际测试表明两种方法在测试集上的预测结果是相同的（采用与问题一相同的结果预测流程）。表7展示了所建立的半监督学习模型在测试集上的字符预测结果。同时可以看出，该方法可以准确预测出测试集的前五个字符 char13-char17，验证了预测模型的有效性。

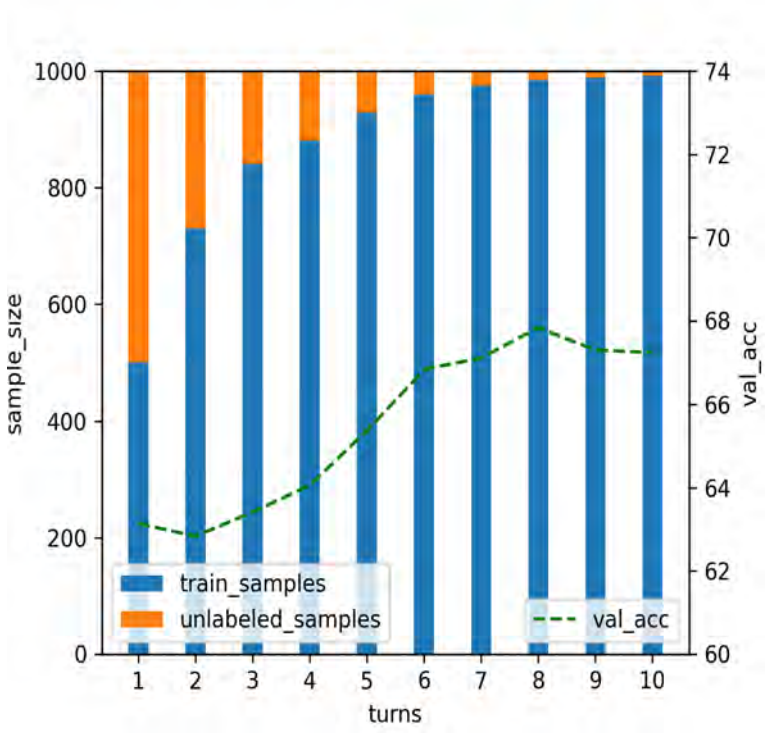


图 6-2　半监督训练结果（无先验伪标签生成方案）

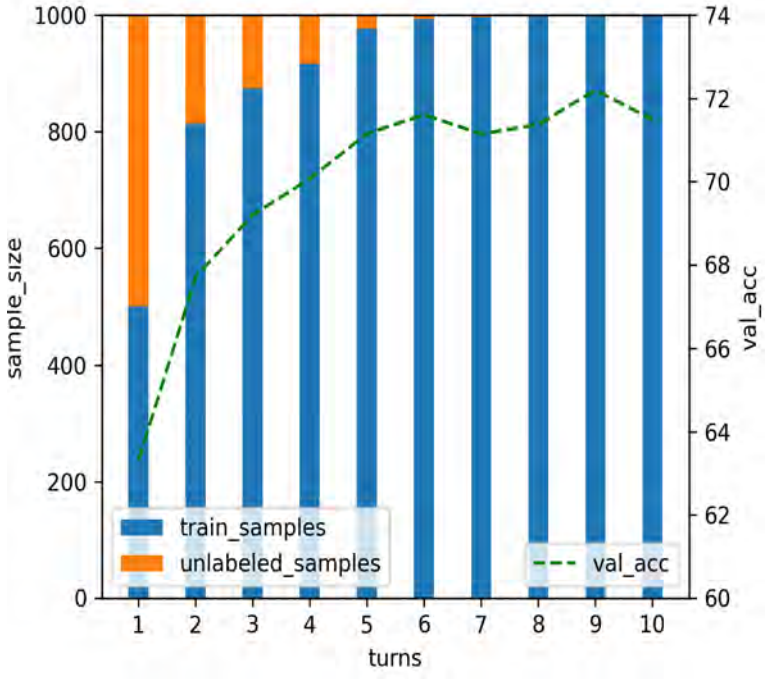


图 6-3　半监督训练结果（基于先验的伪标签生成方案）

表 7　预测结果

预测结果 / 待测字符 / 被试者序号	char13	char14	char15	char16	char17	char18	char19	char20	char21	char22
member 1	M	F	5	2	I	T	K	X	A	0
member 2 预测结果	M	F	5	2	I	T	K	X	A	
member 3 预测结果	M	F	5	2	I	T	K	X	A	
member 4 预测结果	M	F	5	2	I	T	K	X	A	0
member 5 预测结果	M	F	5	2	I	T	K	X	A	0
参考结果	M	F	5	2	I					

7. 问题四的分析与求解

7.1 问题四的分析

人脑在睡眠过程中会产生的自发脑信号（EEG）。这些信号能够在一定程度上反映身体状态的变化，也是用来诊断和治疗睡眠相关疾病的重要依据。准确划分睡眠阶段对于评估睡眠质量十分重要。由于睡眠是一个动态过程，通常情况下脑电信号在不同的睡眠分期会呈现不同的特征。基于脑电信号的自动睡眠分期，可以较为准确地反映睡眠质量，为医学诊断和治疗提供依据 [1]。

问题四需要解决的是一个根据少量特征进行睡眠分期五分类的问题。本文的解决思路是构建新的有效脑电特征，扩充原有数据集，并选择多种分类器进行实验。通过对比分类结果，确定最优的特征构造方法与分类器模型。问题四的求解主要包括以下步骤：

（1）数据预处理，检验是否存在异常数据；

（2）特征构造，调研脑电波能量特征与睡眠分期的关联或潜在关系，构建新特征以扩充数据集；

（3）在 3000 条数据中按 4：1 的比例随机抽取训练数据与测试数据（待步骤四确定最佳分类器后，可以适当减小训练集与测试集的样本比例）；

（4）分别建立 CatBoost、随机森林、SVM 等分类器训练模型，并对比其在测试集上的准确率等的指标性能，同时进行特征重要性分析与特征筛选。

7.2 脑电特征构造与数据处理

为了提高睡眠分期的准确性，本小节利用 5 种睡眠期的睡眠脑电数据构造了一部分新的脑电特征，扩充了原有的数据集，并将 3000 多条数据按 4:1 的比例进行划分，构建了后续模型所需的训练集和测试集。

7.2.1 脑电波特征分析

自发脑电的频率范围通常在 0.5～30Hz 之间，EEG 信号主要由四种节律组成，即 α 波、β 波、θ 波、δ 波。下面对这四种主要节律波形分别进行简要说明。

α 波: 频率范围 8～13Hz，是正常人脑的基本节律。当人保持清醒、闭眼、周围环境安静且身体放松时，大脑频率处于 α 波尤为明显。当受到外界刺激或睁开眼睛时，α 波会马上消失并以 β 波替代，这种现象称为 α 波阻断。

β 波：频率范围为 14～25Hz，大部分时间大脑频率处于这一状态。β 波被普遍认为是大脑处于兴奋状态、精神紧张或者从睡梦中惊醒时的脑电活动。

θ 波：频率范围为 4～7Hz，它的出现是中枢神经系统受到抑制的表现。当大脑频率处于 θ 波时，人们一般处于困倦状态，意识中断，身体深沉放松。

δ 波：频率范围为 0.5～4Hz，它只有在人进入深度睡眠状态或者某些特殊情况下出现。当大脑频率处于 δ 波时，人们处于无意识状态。

7.2.2 脑电波能量特征构造

题目附件 2 提供了 5 种睡眠期（清醒期（6）、快速眼动期（5）、睡眠 I 期（4）、睡眠 II 期（3）和深睡眠期（2））的睡眠脑电数据，每一种睡眠期都包含 α 波、β 波、θ 波、δ 波的相关数据信息以及对应的标签，样本点个数在 600 左右。以清醒期（6）为例，数据共有 5 列，分别对应数据标签、α 波、β 波、θ 波以及 δ 波，共有 634 行，对应 634 个样本点。

原有数据集仅有 α 波、β 波、θ 波以及 δ 波这 4 个脑电特征，且各个样本是离散的，彼此之间没有隐含的时序性信息，分类器难以准确有效地提取出其中包含的有效信息，因此需要构造新的有效脑电特征来提升睡眠分期的精度。

考虑到脑电信号本质上是一种能量波，且 α 波、β 波、θ 波、δ 波具有不同的频率（能量不同）。本文从能量角度出发，将原有数据集中 α 波、β 波、θ 波、δ 波对应的频率视为 4 种能量特征，分别记为 $E_\alpha, E_\beta, E_\theta, E_\delta$，并构造了以下新的脑电特征：

$$Ratio_{\theta+\delta} = \frac{E_\theta + E_\delta}{100} \tag{30}$$

$$Ratio_{\alpha+\beta} = \frac{E_\alpha + E_\beta}{100} \tag{31}$$

$$Product_{\theta\&\delta} = \frac{E_\theta * E_\delta}{1000} \tag{32}$$

$$MeanEnergy = \frac{E_\alpha + E_\beta + E_\theta + E_\delta}{4} \tag{33}$$

$$\text{Product}_{\alpha\&\beta} = \frac{E_\alpha * E_\beta}{1000} \tag{34}$$

$$\ln E_\alpha = \ln\left(E_\alpha + 1\right) \tag{35}$$

$$\ln E_\beta = \ln\left(E_\beta + 1\right) \tag{36}$$

$$\ln E_\gamma = \ln\left(E_\gamma + 1\right) \tag{37}$$

$$\ln E_\delta = \ln\left(E_\delta + 1\right) \tag{38}$$

本文将新构造的 9 种脑电波能量特征与原始数据集中的 $E_\alpha, E_\beta, E_\theta, E_\delta$ 进行整合，实现了数据集的特征扩充，得到的新数据集总共包含 3000 多个样本（每种睡眠期大约有 600 个左右的样本点），其中每个样本点包含 13 个脑电特征。

7.2.3 脑电数据集划分

在对脑电数据集进行划分前，本文首先对数据集的质量进行了检查。通过数据分析，5 种睡眠期的数据样本没有出现数据缺失的情况，且 α 波、β 波、θ 波、δ 波均分别落在 8∼13Hz,14∼25Hz,4∼7Hz,0.5∼4Hz 的标准频段范围内，也即扩充后的数据集具有较高的数据质量，可以用于脑电数据集的划分。

考虑到每种睡眠期的样本数量均在 600 左右，也即数据集中 5 种标签的比例近似 1:1:1:1:1，因此不存在问题一中样本标签不平衡的问题。之后，本文按 4：1 的比例将原始数据集中的 3000 多个样本随机划分成训练集与测试集。

为了避免单次划分带来的随机误差，同时充分验证各个分类模型性能的有效性，本文对 3000 多个相同的样本按 4：1 的比例随机划分了 100 次，总共生成了 100 组不同的训练集与测试集划分情况，并分别测试了各个模型在不同划分情况下的分类准确率，最终各个模型的分类准确率是通过 100 次重复实验的平均结果得到的。

7.3 分类器的构建

7.3.1 决策树和随机森林

机器学习中，决策树（DT）是目前运用最广泛的预测模型之一，代表的是对象属性与对象值之间的一种映射关系。树中每个节点表示某个对象，每个分叉路径则代表的某个可能的属性值，每个叶结点则对应从根节点到该叶节点所经历的路径所表示的对象的值。决策树算法与其它神经网络相比，模型训练时不必花费大量的时间进行迭代，因此当数据集规模不大时，决策树算法具有明显的优

势。同时，除了训练数据外决策树算法不需要其他条件信息，具有较好的分类精度。

决策树的构造过程一般分为 3 个部分，分别是特征选择、决策树生产和决策树裁剪:

（1）特征选择：特征选择表示从众多的特征中选择一个特征作为当前节点分裂的标准，如何选择特征有不同的量化评估方法，从而衍生出不同的决策树，如 ID3（通过信息增益选择特征）、C4.5（通过信息增益比选择特征）、CART（通过 Gini 指数选择特征）等。目的是使用某特征对数据集划分之后，各数据子集的纯度要比划分前的数据集的纯度高，也即降低不确定性。

（2）决策树的生成：根据选择的特征评估标准，从上至下递归地生成子节点，直到数据集不可分则停止决策树停止生长。这个过程实际上就是使用满足划分准则的特征不断的将数据集划分成纯度更高，不确定行更小的子集的过程。对于当前数据集的每一次划分，都希望根据某个特征划分之后的各个子集的纯度更高，不确定性更小。

（3）决策树的裁剪：决策树容易过拟合，因此一般需要剪枝来缩小树结构规模、缓解过拟合。

决策树的流程图如7-1所示：

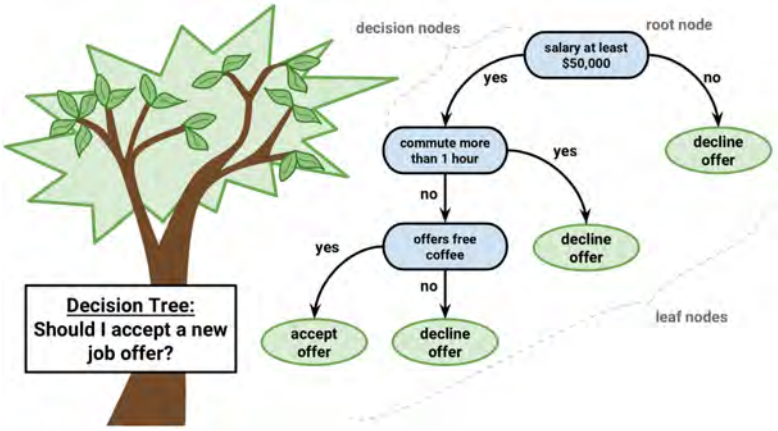


图 7-1　决策树示意图

在 GBM 流行前，研究者常用随机森林（Random forests）这种机器学习方法。将若干个弱分类器的分类结果进行投票选择，从而组成一个强分类器，这就是随机森林主要的思想。随机森林中树的生成规则如下说明 [3]：

（1）如果训练集大小为 M，对于每棵树随机且有放回地抽取训练样本。

（2）若每个样本的特征数量为 N，随机地从中选 n 个特征，每次对树进行分裂时，从刚选定的特征中选最优的特征。

（3）每棵树生长时并且没有剪枝过程。

本文中，随机森林分类器使用 sklearn.ensemble 库中的 RandomForestClassifier，参数设置为：n_estimators=500（森林里决策树的数目为 500），random_state=0（随机树种子为 0），其余参数为默认参数。

7.3.2 GBDT 和 CatBoost

梯度提升树 (GBDT)，是一个以回归树为基学习器，以 boost 为框架的加法模型的集成学习。在机器学习中，损失函数用于评价模型性能。一般而言，损失函数越小模型性能越好。因此，最好的方法就是使损失函数沿着梯度方向下降，从而不断调整提升模型性能。GBDT 在此基础上，基于负梯度 (当损失函数为均

方误差的时候，可以看作是残差) 进行学习。在分类问题中，GBDT 的损失函数跟逻辑回归一样，采用对数似然函数。在回归问题中，GBDT 采用最小化误差平方 (LS)。GBDT 的每一个叶子节点都会得到一个预测值，该预测值等于属于这个节点的所有标签的均值。分枝时通过穷举每一个 feature 的所有阈值找最好的分割点。

2014 年，XGBoost 诞生，被广泛应用于机器学习竞赛，直至 2016 年它都是 GBM（梯度提升树算法）界的最好算法。2017 年，微软推出了 LightGBM——一种训练精度比 XGBoost 更高的 GBM 算法。同年，俄罗斯互联网搜索引擎公司 Yandex 提出了 CatBoost——也是一种训练及测试时间少、调节参数时间少、训练精度高的表现通常比 XGBoost 优越的 GBM 算法。

CatBoost 在类别特征和特征组合方面有独特的处理方式：（1）类别特征：首先，对输入的数据集合进行随机排列以生成多种随机排列顺序。然后给定一个序列，对相同类别的例子计算平均样本值。再使用以下公式将类别特征值转换为数值。

$$x_k^i = \frac{\sum_{j=1}^{n}[x_j^i = x_k^i]\cdot y_j + aP}{\sum_{j=1}^{n}[x_j^i = x_k^i] + a} \quad (39)$$

其中，其中，序列数据集是 $D = \{(x_k, y_k)\}_{k=1..n}$，$x_k = (x_k^1, \ldots, x_k^m)$ 是一个具有 m 个特征的向量，y_k 是预测目标，$[x_j^i = x_k^i]$ 代表着和 x_k^i 类别相同的 x_j^i。P 是先验值，a 是大于 0 的参数，aP 代表先验值的权重，添加 aP 是为了降低低频噪声。CatBoost 使用的这种更有效的类别特征策略，减少了过度拟合，并允许使用整个数据集进行训练。

（2）特征组合：随着分类特征个数增加，可能的组合数量也呈指数增长。算法中不太可能考虑所有组合的情况。当试图构造树上的一个新的分割时，CatBoost 以贪婪的方式考虑组合。第一次分裂不考虑树上任何的组合。在接下来的分割中，CatBoost 将当前树中的所有组合和类别特征与数据集中的类别特征相结合。CatBoost 还通过以下方式生成数值和分类特征的组合：在树中选择的所有分割都被认为是具有两个值的分类，并以与分类值相同的方式组合使用[2]。

在 GBDT 中，构建下一棵树包含两步，即选择树的结构以及树结构固定之后设置叶子节点的值。在 CatBoost 中，对于每个样本 Sample，都单独构建一个利用该样本之前的样本点的梯度估计得到的模型 Model，针对这些模型 Model，估计该样本 Sample 的梯度，然后利用新模型重新对样本 Sample 打分。算法的伪代码如图 7-2 所示。

3 Fighting Gradient Bias

CatBoost, as well as all standard gradient boosting implementations, builds each new tree to approximate the gradients of the current model. However, all classical boosting algorithms suffer from overfitting caused by the problem of biased pointwise gradient estimates. Gradients used at each step are estimated using the same data points the current model was built on. This leads to a shift of the distribution of estimated gradients in any domain of feature space in comparison with the true distribution of gradients in this domain, which leads to overfitting. The idea of biased gradients was discussed in previous literature [1] [9]. We have provided a formal analysis of this problem in the paper [5]. The paper also contains modifications of classical gradient boosting algorithm that try to solve this problem. CatBoost implements one of those modifications, briefly described below.

In many GBDTs (e.g., XGBoost, LightGBM) building next tree comprises two steps: choosing the tree structure and setting values in leafs after the tree structure is fixed. To choose the best tree structure, the algorithm enumerates through different splits, builds trees with these splits, sets values in the obtained leafs, scores the trees and selects the best split. Leaf values in both phases are calculated as approximations for gradients [8] or for Newton steps. In CatBoost the second phase is performed using traditional GBDT scheme and for the first phase we use the modified version.

According to intuition obtained from our empirical results and our theoretical analysis in [5], it is highly desirable to use unbiased estimates of the gradient step. Let F^i be the model constructed after building first i trees, $g^i(\mathbf{X}_k, Y_k)$ be the gradient value on k-th training sample after building i trees. To make the gradient $g^i(\mathbf{X}_k, Y_k)$ unbiased w.r.t. the model F^i, we need to have F^i trained without the observation $\mathbf{X}_k$. Since we need unbiased gradients for all training examples, no observations may be used for training F^i, which at first glance makes the training process impossible. We consider the following trick to deal with this problem: for each example $\mathbf{X}_k$, we train a separate model M_k that is never updated using a gradient estimate for this example. With M_k, we estimate the gradient on $\mathbf{X}_k$ and use this estimate to score the resulting tree. Let us present the pseudo-code that explains how this trick can be performed. Let $\text{Loss}(y, a)$ be the optimizing loss function, where y is the label value and a is the formula value.

Algorithm 1: Updating the models and calculating model values for gradient estimation

```
input : {(X_k, Y_k)}_{k=1}^n ordered according to σ, the number of trees I;
M_i ← 0 for i = 1..n;
for iter ← 1 to I do
    for i ← 1 to n do
        for j ← 1 to i − 1 do
            g_j ← d/da Loss(y_j, a)|_{a=M_i(X_j)};
        M ← LearnOneTree((X_j, g_j) for j = 1..i − 1);
        M_i ← M_i + M;
return M_1 ... M_n; M_1(X_1), M_2(X_2) M_n(X_n)
```

Note that M_i is trained without using the example $\mathbf{X}_i$. CatBoost implementation uses the following relaxation of this idea: all M_i share the same tree structures.

In CatBoost we generate s random permutations of our training dataset. We use several permutations to enhance the robustness of the algorithm: we sample a random permutation and obtain gradients on its basis. These are the same permutations as ones used for calculating statistics for categorical features. We use different permutations for training distinct models, thus using several permutations does not lead to overfitting. For each permutation σ, we train n different models M_i, as shown above. That means that for building one tree we need to store and recalculate $O(n^2)$ approximations for each permutation σ: for each model M_i, we have to update $M_i(\mathbf{X}_1), \ldots, M_i(\mathbf{X}_i)$. Thus, the resulting complexity of this operation is $O(s\,n^2)$. In our practical implementation, we use one important trick which reduces the complexity of one tree construction to $O(s\,n)$: for each permutation, instead of storing and updating $O(n^2)$ values $M_i(X_j)$, we maintain values $M'_i(\mathbf{X}_j), i = 1, \ldots, \lceil\log_2(n)\rceil, j <$

图 7-2 CatBoost 算法伪代码

本文中，CatBoost 分类器的建立使用 catboost 库中的 CatBoostClassifier。参

数设置为：loss_function='MultiClass',learning_rate=0.1,depth=6（树的深度）, iterations=1000（数的数量），其余参数使用默认参数。

7.4 实验性能对比

本小节比较了不同分类器的睡眠分期准确率，并进一步分析了扩充后的 13 个脑电特征在不同模型中对分类准确率的贡献度。同时，本小节还研究了训练集和测试集划分比例对分类识别准确率的影响。

7.4.1 不同分类器的性能对比

针对问题四，为了确定最优的睡眠分期分类器，本文设计并进行了 100 轮重复实验。具体来说，本文对 3000 多个相同的样本按 4：1 的比例随机划分了 100 次，总共生成了 100 组不同的训练集与测试集划分情况，并分别测试了各睡眠分期分类器在 100 组测试集上的平均正确率（Accuracy）、准确率（Precision）和召回率（Recall），结果分别如表8、表9所示。图7-3展示了不同分类器的平均睡眠分期准确率。

表 8　各分类器的平均准确率、正确率

指标 / 模型	深睡眠期（2）准确率	睡眠 II 期（3）准确率	睡眠 I 期（4）准确率	快速眼动期（5）准确率	清醒期（6）准确率	5 种分期的平均正确率
SVM	0.7826	0.6131	0.5454	0.5526	0.7105	0.65
随机森林	0.9749	0.9289	0.9396	0.906	0.9643	0.943
CatBoost	0.9624	0.9451	0.921	0.9399	0.9818	0.9506

表 9　各分类器的平均召回率

指标 / 模型	深睡眠期（2）召回率	睡眠 II 期（3）召回率	睡眠 I 期（4）召回率	快速眼动期（5）召回率	清醒期（6）召回率
SVM	0.7894	0.6774	0.1791	0.7777	0.9
随机森林	0.9638	0.9338	0.8597	0.959	0.9882
CatBoost	0.9766	0.9171	0.9252	0.9579	0.9733

7.4.2 脑电特征有效性分析

为了验证本文新构建的脑电特征的有效性，本文进一步分析了扩充后的 13 个脑电特征在不同模型中对分类准确率的贡献度。图7-4展示了 CatBoost 分类器训练得到的 13 个脑电特征的重要性条形图。可以得到，在 CatBoost 分类器下，原有数据数据集中对模型预测结果有较高贡献的特征有 $E_\alpha, E_\beta, E_\theta, E_\delta$, 以及新构建的特征 MeanEnergy 和 $\ln E_\delta$。

图7-5展示了随机森林分类器训练得到的 13 个脑电特征的重要性条形图。可以得到，在随机森林分类器下，原有数据集中对模型预测结果有较高贡献的特征为 E_θ, E_β 以及新构建的特征 $\ln E_\alpha, \ln E_\delta, Product_{\theta\&\delta}, Ratio_{\theta+\delta}$。

此外，本文还使用递归式特征消除法分析了 SVM 分类器中 13 个脑电特征的重要性，其中对模型预测结果有较高贡献的特征为 $Product_{\theta\&\delta}$,$Product_{\alpha\&\beta}$,E_θ,E_δ, E_β,$\ln E_{alpha}$。

图7-6展示了不同脑电特征下 CatBoost 的平均睡眠分期准确率。

不难发现，在不同的分类模型中，各个脑电特征的重要程度不同。然而，无论是 CatBoost、随机森林还是 SVM，都存在一部分新构建的脑电特征具有较高的贡献度，这一结果验证了所构建的脑电特征的有效性。

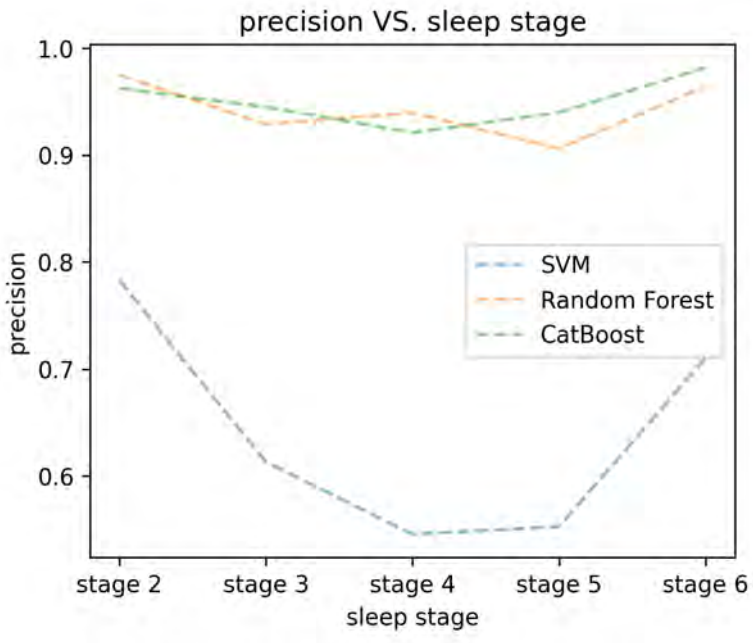


图 7-3　不同分类器的平均睡眠分期准确率

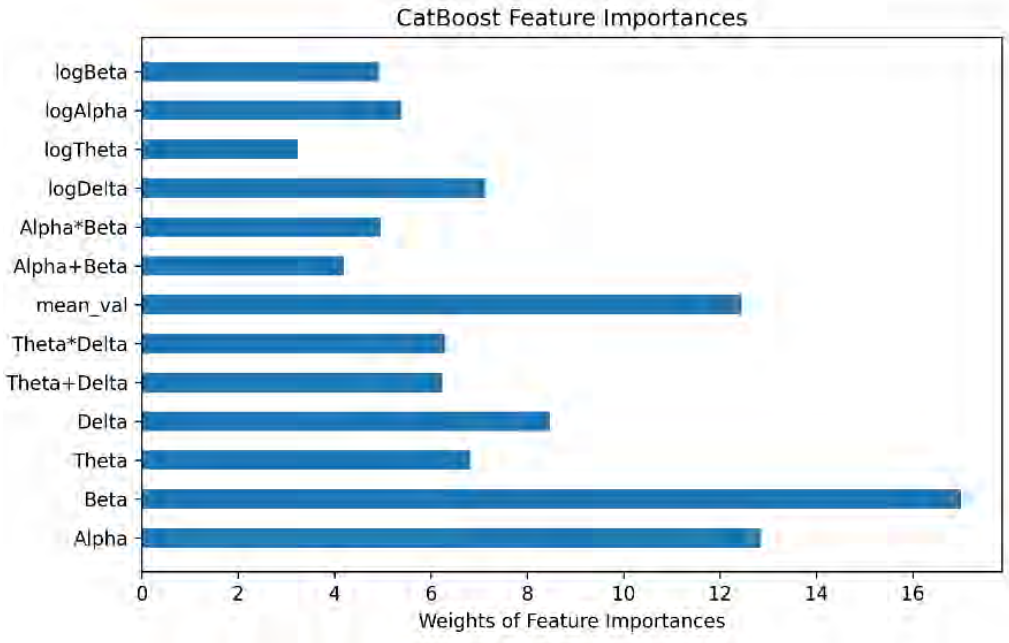


图 7-4　CatBoost 分类器中各个脑电特征的贡献度

7.4.3 数据集划分方式的性能对比

在上述实验基础上，为了进一步分析不同的测试集和训练集划分比例对睡眠分期准确率的影响，选择 CatBoost 分类器，观测在不同的训练集与测试集分割比例下模型预测效果的差异，结果如表10和图7-7所示。不难发现，当测试样本占总样本比例越小时，5 种睡眠分期预测的准确率越高，正确率也越高。值得注意的是，即使是测试样本占总样本比例高达 0.6，也即训练样本占总样本比例

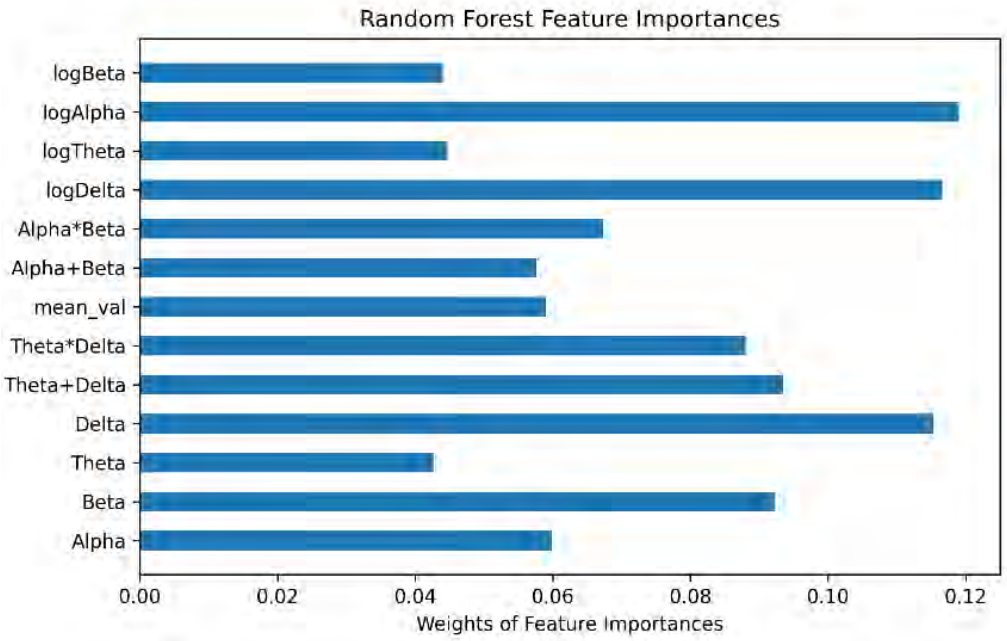


图 7-5　随机森林分类器中各个脑电特征的贡献度

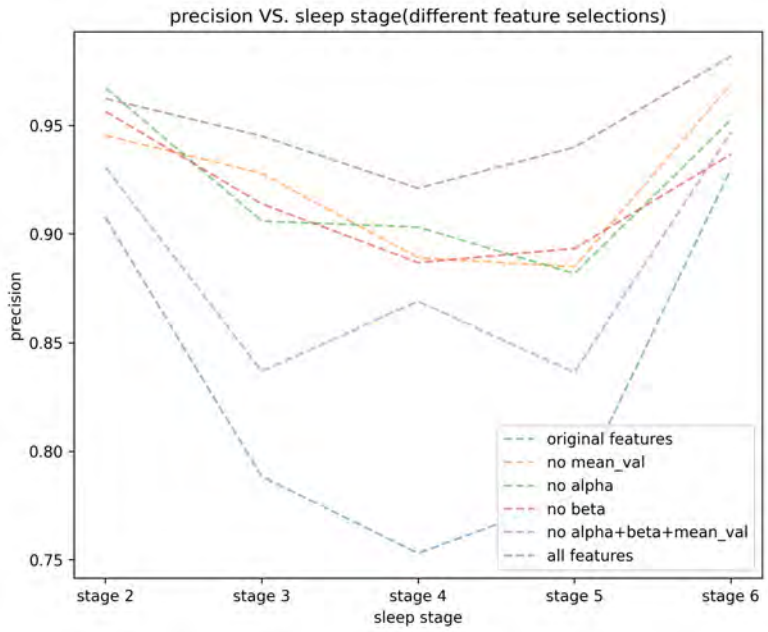


图 7-6　不同脑电特征下 CatBoost 的平均睡眠分期准确率

低至 0.4 时，该分类器依然能够保持大于 0.8 的平均正确率水平，这一结果再次验证了本文睡眠分期模型的优秀性能。

表 10　不同数据集划分比例下 CatBoost 的预测效果

测试集所占比例 \ 指标	深睡眠期（2）准确率	睡眠 II 期（3）准确率	睡眠 I 期（4）准确率	快速眼动期（5）准确率	清醒期（6）准确率	5 种分期的平均正确率
0.2	0.9624	0.9451	0.921	0.9399	0.9818	0.9506
0.3	0.9486	0.9009	0.8476	0.8878	0.9694	0.9122
0.4	0.9325	0.8354	0.819	0.8798	0.9395	0.8842
0.5	0.9132	0.8227	0.7808	0.8106	0.9493	0.8573
0.6	0.8887	0.7771	0.7365	0.7678	0.9214	0.8206

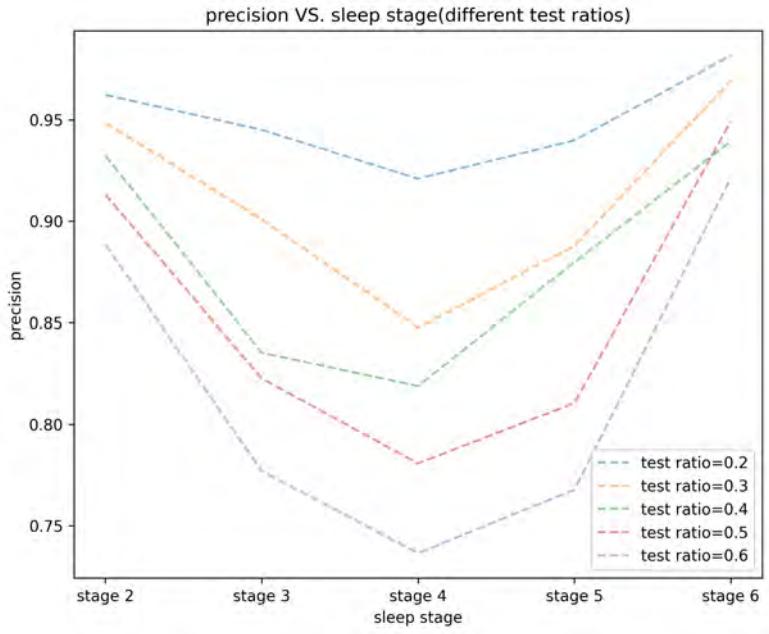


图 7-7　不同数据集划分比例下 CatBoost 的预测效果

8. 模型的评价

8.1 模型的优点

（1）本文针对问题一至问题三提出的模型构建了端到端的深度学习架构，无需进行更多的特征提取与数据分析过程，该模型节省了很多时间与人力成本，为脑电波信号分析与判别任务提供了相对简易的分类模型。

（2）本文提出的模型与其他多种机器学习方法或模型进行了对比，该模型在验证集上表现出优越的性能。与此同时，在实验轮次低于 3 次的条件下，本文的模型依然能够保持稳定的、良好的预测性能。

（3）本文对脑电信号的滤波去噪方法简单易行，能够准确提取出其中包含的有效特征信息，相应的补充实验见附录 A1.3。

（4）本文提出的模型具有较强的迁移能力，适用于其他的信号检测识别任务。

8.2 模型的缺点

（1）本文针对问题一至问题三提出的模型在行的预测性能上略逊于在列的预测性能，需要后期采用统计学中经典的投票法则以确保最终的字符预测精度。

（2）本文针对问题一至问题三提供的模型为深度学习模型，可解释性较差。

（3）由于时间限制，本文针对问题四使用的模型为 GBDT 模型，没有使用深度网络，因此取得的分类准确率没有达到 98% 以上。

9. 参考文献

[1] K Wulff, S Gatti, JG Wettstein and RG Foster, ”Sleep and circadian rhythm disruption in psychiatric and neurodegenerative disease”, Nature Reviews Neuroscience, vol. 11, no. 8, pp. 589-599, 2010.

[2] Dorogush A V, Ershov V, Gulin A. CatBoost: gradient boosting with categorical features support[J]. arXiv preprint arXiv:1810.11363, 2018.

[3] Cutler D R, Edwards Jr T C, Beard K H, et al. Random forests for classification in ecology[J]. Ecology, 2007, 88(11): 2783-2792.

[4] Kiranyaz S , Avci O , Abdeljaber O , et al. 1D Convolutional Neural Networks and Applications: A Survey[J]. 2019.

[5] J. Ospina, A. Newaz and M. O. Faruque, ”Forecasting of PV plant output using hybrid wavelet-based LSTM-DNN structure model,” in IET Renewable Power Generation, vol. 13, no. 7, pp. 1087-1095, 20 5 2019, doi: 10.1049/iet-rpg.2018.5779.

[6] I. El-Naqa, Yongyi Yang, M. N. Wernick, N. P. Galatsanos and R. M. Nishikawa, ”A support vector machine approach for detection of microcalcifications,” in IEEE Transactions on Medical Imaging, vol. 21, no. 12, pp. 1552-1563, Dec. 2002, doi: 10.1109/TMI.2002.806569.

[7] Heydarzadeh M , Hedayati S , Nourani M , et al. Gear fault diagnosis using discrete wavelet transform and deep neural networks[C]// Conference of the IEEE Industrial Electronics Society. IEEE, 2016.

[8] Lee D H. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks[C]//Workshop on challenges in representation learning, ICML. 2013, 3(2).

[9] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Advances in neural information processing systems. 2017: 5998-6008.

[10] Jie, Shen, Samuel, et al. Squeeze-and-Excitation Networks.[J]. IEEE transactions on pattern analysis and machine intelligence, 2019.

附录 A 补充对比实验（消融实验）

1.1 采样时段长度 S 对模型精度的影响实验

为了找到合适的采样时长 S，本文设计采样时段长度 S 对模型精度的影响实验如下。采样时段长度以 40 的步长从 80 依次增加到 240，网络采用 1D-CNN，优化器选用 Adam 优化器，数据选用经过 Butterworth 和 FFT 处理后的数据。实验结果如表11所示。从表中结果可以看出，采样时段选择受刺激后的 160 个采样点为最佳（即观测时长为 640ms）。

表 11　不同采样时长对模型精度的影响

采样时段长度	80	120	160	200	240
验证集准确率	55.60%	71.74%	**77.87%**	75.14	73.89

1.2 不同优化器对模型精度的影响实验

神经网络的优化器对于网络的训练有一定的影响。合适的优化器可以加快网络收敛，节省训练时间并提高模型性能。本文分别采用 SGD 算法、Momentum 算法和 Adam 算法作为网络的优化器，采样长度固定 160 个采样点，数据选用经过 Butterworth 和 FFT 处理后的数据。实验结果如表12所示。从表中结果可以看出，Adam 优化器的性能最佳。

表 12　不同优化器对模型精度的影响

滤波方法	Sgd	Momentum	Adam
验证集准确率	77.32%	76.93%	**77.87%**

1.3 滤波算法预处理数据对模型精度的影响实验

脑电原始数据中包含眼电、伪迹等干扰信号，需要使用合适的滤波方法处理。本文比较了滤波算法对模型精度的影响，如表13对模型精度的影响。可以看到，将数据滤波后再加入网络训练大幅提高的网络的性能。

表 13　不同滤波处理对模型精度的影响

滤波方法	无滤波	Butterworth	Butterworth+FFT
验证集准确率	66.27%	74.64%	**77.87%**

附录 B 程序代码

（1）主函数代码 main.py

```
from trainer import Trainer
import numpy as np
from tqdm import tqdm
char = [['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']]

def get_ave_result(p,skip22_12 = False):
    global char
    a = np.array(p)
    b = a.reshape(5,10,5,2)
    b = b.transpose(0,2,1,3)
    count_i = np.zeros([6,10])
    count_j = np.zeros([6,10])

    for i,b1 in enumerate(b):
        for j,b11 in enumerate(b1):
            for char_i in range(10):
                if i in [1,2] and char_i==9 and skip22_12:
                    continue
                count_i[b11[char_i,0],char_i]+=1
                count_j[b11[char_i,1],char_i]+=1
    index_i = count_i.argmax(axis=0)
    index_j = count_j.argmax(axis=0)
    for i,j in zip(index_i,index_j):
        print(char[i][j])


if __name__ == '__main__':
    #data_root:'data_mat' 'data_mat_revised/'
    s = []
    p = []
    r = []
    epoch = 100
    attention = []
    for member in range(1,6):
        model =
            Trainer(member,data_root='data_mat_revised/',opti='adam')
        mean_acc = []
        mean_loss = []
        with tqdm(total=epoch) as t:
            acc = []
            loss = []
            for i in range(epoch):
                train_loss,train_acc = model.trainA(i)
                val_loss,val_acc = model.valA(i)
                t.set_description('Member %i' % member)
```

```
            t.set_postfix(loss='{:.3f}'.format(val_loss),\
                acc='{:.2f}%'.format(100*val_acc))
            t.update(1)
            if epoch - i<10:
                loss.append(val_loss.cpu().item())
                acc.append(val_acc)
        acc = np.mean(np.array(acc))
        loss = np.mean(np.array(loss))
        attention.append(\
            np.array(model.model.atten.cpu().detach()))
    mean_acc.append(acc)
    mean_loss.append(loss)
    series,predicts,result = model.predictA()
    r.append(result)
    p.append(predicts)
loss = np.mean(np.array(mean_loss))
acc = np.mean(np.array(mean_acc))
r = np.mean(np.array(r),axis= 0)
print('test:\nchar acc:{:.2f}% \n row acc:{:.2f}% \n col
    acc:{:.2f}%'.format(*list(r*100)))
print('mean val acc:{:.2f}%'.format(float(acc*100)))
print('mean val loss:',loss)
get_ave_result(p,True)
# import pickle
# with open('atten.pkl','wb') as f:
#    pickle.dump(attention,f)
```

（2）网络代码 net.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np


class FullyConnected(nn.Module):
    def __init__(self,seq_length=None):
        super(FullyConnected, self).__init__()
        if seq_length == None:
            seq_length = 160
        self.seq_length = seq_length
        self.fc1 = nn.Sequential(
            nn.Linear(20*seq_length,2048),
            nn.Dropout(0.5),
            nn.ReLU()
        )

        self.fc2 = nn.Sequential(
            nn.Linear(2048,30),
            nn.Dropout(0.5),
            nn.ReLU()
        )
        self.out = nn.Sequential(
            nn.Linear(30, 2),
```

```
            nn.Dropout(0.5),
            nn.Softmax()
        )

        self.atten = torch.tensor([0])


    def forward(self, x):
        b,c,s = x.shape
        x = x.view(b,-1)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.out(x)

        return x, 0

class CNN(nn.Module):
    def __init__(self,seq_length=None):
        super(CNN, self).__init__()
        self.atten = torch.zeros([1])
        if seq_length == None:
            seq_length = 240

        self.attention = nn.Sequential(
            nn.Linear(seq_length*20,20),
            nn.Sigmoid()
        )

        self.conv = nn.Sequential(
            nn.Conv1d(20, 5, 5, padding=2),
            nn.MaxPool1d(2),
            nn.ReLU()
        )

        self.fc = nn.Sequential(
            nn.Linear(5*seq_length//2, 30),
            nn.Dropout(0.5),
            nn.ReLU()
        )

        self.out = nn.Sequential(
            nn.Linear(30, 2),
            nn.Dropout(0.5),
            nn.Softmax(dim = 1)
        )

    def forward(self, x,use_attention = 0):
        b,c,s = x.shape
        if use_attention==1:
            x_atten = x.view(b,c*s)
            self.atten = self.attention(x_atten) #(b,20)
            # x = x.view(b,s,c)
            atten = self.atten.repeat(240,1,1)
            # print(atten.shape)
```

```
            atten = atten.permute(1,2,0)
            # print(atten.shape)
            x = atten*x
        elif use_attention==2:
            self.atten = np.zeros([b,c,s])
            for i in [ 0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 16,
                19]:
                self.atten[:,i,:] = 1
            self.atten =
                torch.tensor(self.atten,dtype=torch.float32).cuda()
            x = self.atten*x

        elif use_attention==3:
            self.atten = np.zeros([b,c,s])
            for i in [1, 3, 4, 7, 8, 9, 10, 11, 13, 19]:
                self.atten[:,i,:] = 1
            self.atten =
                torch.tensor(self.atten,dtype=torch.float32).cuda()
            x = self.atten*x
        elif use_attention==4:
            self.atten = np.zeros([b,c,s])
            for i in [1,9,19]:
                self.atten[:,i,:] = 1
            self.atten =
                torch.tensor(self.atten,dtype=torch.float32).cuda()
            x = self.atten*x
        elif use_attention==5:
            self.atten = np.zeros([b,c,s])
            for i in [5,12,14]:
                self.atten[:,i,:] = 1
            self.atten =
                torch.tensor(self.atten,dtype=torch.float32).cuda()
            x = self.atten*x
        #(b,240,20)
        x = self.conv(x) #(b,240,20) -> (b,240,5) -> (b,120,5)
        fc = self.fc(x.view(x.size(0), -1)) #(b,600) -> (b,30)
        x = self.out(fc)#.squeeze() #(b,2)
        return x, fc


class LSTM(nn.Module):
    def __init__(self,seq_length=None):
        super(LSTM, self).__init__()
        if seq_length == None:
            seq_length = 160
        self.seq_length = seq_length
        self.rnn = nn.LSTM( # LSTM 效果要比 nn.RNN() 好多了
            input_size=20, # 图片每行的数据像素点
            hidden_size=64, # rnn hidden unit
            num_layers=1,  # 有几层 RNN layers
            batch_first=True, # input & output 会是以 batch size
                为第一维度的特征集 e.g. (batch, time_step, input_size)
        )
        self.atten = torch.tensor([0])
```

```
        self.fc = nn.Linear(seq_length*64,64)
        self.out = nn.Linear(64, 2) # 输出层

    def forward(self, x):

        x = x.permute(0,2,1)
        r_out, (h_n, h_c) = self.rnn(x, None) # None 表示 hidden
            state 会用全0的 state

        # 选取最后一个时间点的 r_out 输出
        # 这里 r_out[:, -1, :] 的值也是 h_n 的值
        x = self.fc(r_out.contiguous().view(-1,self.seq_length*64))
        out = self.out(x)
        return out, 0
```

（3）训练代码 trainer.py

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import numpy as np
import hiddenlayer as hl
import torch.onnx as onnx
from Dataset import Trainset, Testset
from train_valid_split import train_valid_split
from Model import CNN,LSTM,FullyConnected

class Trainer(object):

    def __init__(self,member = 1,data_root =
        'data_mat/',train_batch = 64,val_batch =
        80,opti='adam',seq_length = None,model='CNN'):
        self.member = member

        self.device = torch.device('cuda' if
            torch.cuda.is_available() else 'cpu')

        if model=='CNN':
            self.model = CNN(seq_length =
                seq_length).to(self.device)
        elif model=='LSTM':
            self.model = LSTM(seq_length =
                seq_length).to(self.device)
        else:
            self.model = FullyConnected(seq_length =
                seq_length).to(self.device)

        self.random_seed = np.random.seed(666)

        self.trainsetA, _ = train_valid_split(Trainset('A',\
        member,data_root=data_root,seq_length = seq_length),\
                        random_seed=self.random_seed)
```

```
        self.valsetA,_ = train_valid_split(Trainset('B',\
           member,data_root=data_root,seq_length = seq_length), \
                              random_seed=self.random_seed)
        # testset, valsetB = train_valid_split(Test('B',1),
           random_seed=random_seed)

        self.train_loaderA = DataLoader(
           dataset=self.trainsetA,
           batch_size=train_batch,
           shuffle=True,
        )

        self.valid_loaderA = DataLoader(
           dataset=self.valsetA,
           batch_size=val_batch,
           shuffle=True,
        )

        self.test_loaderA = DataLoader(
           dataset=Testset('A',member,data_root=data_root,seq_length
              = seq_length),
           shuffle=False
        )

        self.mse_criterion = nn.MSELoss()
        self.cross_entropy_criterion = nn.CrossEntropyLoss()
        if opti == 'sgd':
           self.optimizer = optim.SGD(self.model.parameters(),
              lr=5e-4, momentum=0.9,
                          weight_decay=1e-4)
        else:
           self.optimizer = optim.Adam(self.model.parameters(),
              lr=5e-4,weight_decay=1e-4)



    def trainA(self,ep):
        self.model.train()
        total_loss = 0
        total_acc = 0
        for step, (x, y) in enumerate(self.train_loaderA):
           data, target = x.to(self.device),
              y.to(self.device).long()
           output, _ = self.model(data)

           self.optimizer.zero_grad()
           loss = self.cross_entropy_criterion(output, target)
           total_loss += loss
           loss.backward()

           self.optimizer.step()

           predict = output.data.max(1)[1]
           acc = predict.eq(target.data).sum()
```

```
            total_acc += acc.item() / self.train_loaderA.batch_size

        avg_loss = total_loss / len(self.train_loaderA)
        avg_acc = total_acc / len(self.train_loaderA)
        # print("Epoch: {} Loss: {}, Acc: {}".format(ep, avg_loss,
        #                                           avg_acc))

        if ep == 499:
            torch.save({
                'epoch': ep,
                'model_state_dict': self.model.state_dict(),
                'optimizer_state_dict': self.optimizer.state_dict(),
                'loss': total_loss,
                'acc': total_acc
            }, './vallina_cnn_A')

        return avg_loss,avg_acc


    def valA(self,ep):
        self.model.eval()
        total_loss = 0
        total_acc = 0
        all_acc = []

        with torch.no_grad():
            ii=0
            for step, (x, y) in enumerate(self.valid_loaderA):
                data, target = x.to(self.device),
                    y.to(self.device).long()
                output, _ = self.model(data)

                loss = self.cross_entropy_criterion(output, target)
                total_loss += loss

                predict = output.data.max(1)[1]
                acc = predict.eq(target.data).sum()
                total_acc += acc.item() / len(predict)
                ii=step
                all_acc.append(float(acc.item() / len(predict)))

        # print("Valid Epoch: {} Loss: {}, Acc: {}".format(ep,
            total_loss / (ii+1),
        #                                           total_acc /
            (ii+1)))
        return total_loss/(ii+1),total_acc / (ii+1)

    def predictA(self):
        char = [['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']]
```

```
char2index = {'A': (0, 0),'B': (0, 1),'C': (0, 2),'D': (0,
   3),'E': (0, 4),'F': (0, 5),
         'G': (1, 0),'H': (1, 1),'I': (1, 2),'J': (1,
            3),'K': (1, 4),'L': (1, 5),
         'M': (2, 0),'N': (2, 1),'O': (2, 2),'P': (2,
            3),'Q': (2, 4),'R': (2, 5),
         'S': (3, 0),'T': (3, 1),'U': (3, 2),'V': (3,
            3),'W': (3, 4),'X': (3, 5),
         'Y': (4, 0),'Z': (4, 1),'1': (4, 2),'2': (4,
            3),'3': (4, 4),'4': (4, 5),
         '5': (5, 0),'6': (5, 1),'7': (5, 2),'8': (5,
            3),'9': (5, 4),'0': (5, 5)}

series = []
real_a = 'M'*5+'F'*5+'5'*5+'2'*5+'I'*5
real_index = [char2index[i] for i in real_a]
index_predicts = []
self.model.eval()
with torch.no_grad():
   for step, (cols, rows) in enumerate(self.test_loaderA):
      col_pred_set = []
      row_pred_set = []
      # print("  0 - 6")
      for line in range(6):
         data = cols[:, line, :, :].to(self.device)
         output, _ = self.model(data)
         col_pred_set.append(output.data.cpu().numpy())
      # print("  7 - 12")
      for line in range(6):
         data = rows[:, line, :, :].to(self.device)
         output, _ = self.model(data)
         row_pred_set.append(output.data.cpu().numpy())
      col_pred_set = np.array(col_pred_set).squeeze()
      row_pred_set = np.array(row_pred_set).squeeze()
      col_pred = np.argmax(col_pred_set, axis=0)[1]
      row_pred = np.argmax(row_pred_set, axis=0)[1]
      series.append(char[col_pred][row_pred])
      index_predicts.append([col_pred,row_pred])
series = ''.join(series)
for i in range(13,23):
   print("char({}):{}".format(i,series[5*(i-13):5*(i-12)]))
counter = 0
for i in range(len(real_a)):
   if real_a[i] == series[i]:
      counter += 1
i_counter = 0
j_counter = 0
for (i_t,j_t),(i,j) in zip(real_index,index_predicts):
   if i_t ==i:
      i_counter+=1
   if j_t ==j:
      j_counter+=1
print('char acc:',counter / len(real_a))
print('i acc:',i_counter / len(real_a))
```

```
        print('j acc:',j_counter/len(real_a))
        return series,index_predicts,[counter /
            len(real_a),i_counter /
            len(real_a),j_counter/len(real_a)]
```

（4）数据生成代码 gen_mat.py

```
import pandas as pd
import numpy as np
import torch
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
import torch
from torch import nn
import torch.nn.functional as F
import time
import sys
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_recall_curve
import os
import scipy.io as scio

sheet2label = {
    'char01(B)':[1,8],
    'char02(D)':[1,10],
    'char03(G)':[2,7],
    'char04(L)':[2,12],
    'char05(O)':[3,9],
    'char06(Q)':[3,11],
    'char07(S)':[4,7],
    'char08(V)':[4,10],
    'char09(Z)':[5,8],
    'char10(4)':[5,12],
    'char11(7)':[6,9],
    'char12(9)':[6,11]
}



def get_test_datas(test_data_files,test_label_files):
    test_datas = []
    test_labels = []
    for train_data_file,train_label_file in
        zip(test_data_files,test_label_files):
        trainval_data = pd.read_excel(train_data_file,sheet_name =
            None,header=None)
        trainval_label = pd.read_excel(train_label_file,sheet_name
            = None,header=None)

        test_data = dict()
        for i,key in enumerate(trainval_data.keys()):
```

```
            tmp = np.array(trainval_data[key])
            tmp_min = tmp.min(axis=0).reshape(1,-1)
            tmp_max = tmp.max(axis=0).reshape(1,-1)
            tmp = (tmp-tmp_min)/(tmp_max-tmp_min)
            test_data[key] = tmp

        test_label = dict()
        for i,key in enumerate(trainval_label.keys()):
            if i % 6 != 0:
                test_label[key] = np.array(trainval_label[key])[1:]
            else:
                test_label[key] = np.array(trainval_label[key])[1:]
        test_labels.append(test_label)
        test_datas.append(test_data)
    return test_datas,test_labels

def test_write_mat_like_norm_putinplace(datas,labels,file =
    'test_like_norm_place.mat',length = 50):
    data_out = np.zeros([12,240,20,length],dtype=np.float32)
    label_out = np.zeros([12,length],dtype=np.float32)
    jj = 0
    for key in datas.keys():
        for stimulate,line in labels[key]:
            if stimulate<=12:
                tmp = datas[key][line:line+240,:]

                tmp_min = tmp.min(axis=0).reshape(1,-1)
                tmp_max = tmp.max(axis=0).reshape(1,-1)
                tmp = (tmp-tmp_min)/(tmp_max-tmp_min)
                data_out[stimulate-1,:,:,jj] = tmp
                label_out[stimulate-1,jj] = 0
            else:
                jj+=1
    data_numpy = np.array(data_out)
    label_numpy = np.array(label_out)
    scio.savemat(file,\
        {"responses":data_numpy,"is_stimulate":label_numpy})
    return data_numpy,label_numpy


def get_trainval_datas(train_data_files,train_label_files):
    train_datas = []
    train_labels = []
    val_datas = []
    val_labels = []
    for train_data_file,train_label_file in
        zip(train_data_files,train_label_files):
        trainval_data = pd.read_excel(train_data_file,sheet_name =
            None,header=None)
        trainval_label = pd.read_excel(train_label_file,sheet_name
            = None,header=None)

        train_data = dict()
        val_data = dict()
```

```
        for i,key in enumerate(trainval_data.keys()):
            tmp = np.array(trainval_data[key])
            tmp_min = tmp.min(axis=0).reshape(1,-1)
            tmp_max = tmp.max(axis=0).reshape(1,-1)
            tmp = (tmp-tmp_min)/(tmp_max-tmp_min)
            if i % 6 != 0:
                train_data[key] = tmp
            else:
                val_data[key] = tmp

        train_label = dict()
        val_label = dict()
        for i,key in enumerate(trainval_label.keys()):
            if i % 6 != 0:
                train_label[key] = np.array(trainval_label[key])[1:]
            else:
                val_label[key] = np.array(trainval_label[key])[1:]
        train_labels.append(train_label)
        train_datas.append(train_data)
        val_labels.append(val_label)
        val_datas.append(val_data)
    return train_datas,train_labels,val_datas,val_labels

def write_mat_like_norm_putinplace(datas,labels,file =
    'train_like_norm_place.mat',length = 50):
    data_out = np.zeros([12,240,20,length],dtype=np.float32)
    label_out = np.zeros([12,length],dtype=np.float32)
    jj = 0
    for key in labels.keys():
        for stimulate,line in labels[key]:
            if stimulate<=12:
                tmp = datas[key][line:line+240,:]

                tmp_min = tmp.min(axis=0).reshape(1,-1)
                tmp_max = tmp.max(axis=0).reshape(1,-1)
                tmp = (tmp-tmp_min)/(tmp_max-tmp_min)
                data_out[stimulate-1,:,:,jj] = tmp
                label_out[stimulate-1,jj] = 1 if stimulate in
                    sheet2label[key] else 0
            else:
                jj+=1
    data_numpy = np.array(data_out)
    label_numpy = np.array(label_out)
    scio.savemat(file,\
        {"responses":data_numpy,"is_stimulate":label_numpy})
    return data_numpy,label_numpy


def main(root_in = './data/',root_out = './data_mat/',range1 =
    1,range2 = 6):
    test_data_files =
        [root_in+'/S{}/S{}_test_data.xlsx'.format(i,i) for i in
        range(range1,range2)]
    test_label_files =
```

```
        [root_in+'/S{}/S{}_test_event.xlsx'.format(i,i) for i in
        range(range1,range2)]
    train_data_files =
        [root_in+'/S{}/S{}_train_data.xlsx'.format(i,i) for i in
        range(range1,range2)]
    train_label_files =
        [root_in+'/S{}/S{}_train_event.xlsx'.format(i,i) for i in
        range(range1,range2)]
    test_datas,test_labels =
        get_test_datas(test_data_files,test_label_files)
    train_datas,train_labels,val_datas,val_labels =
        get_trainval_datas(train_data_files,train_label_files)

    for i in range(5):
        print(i+1)
        test_write_mat_like_norm_putinplace(
        test_datas[i],test_labels[i],\
        file = root_out+str(i+1)+'test_like_norm_place.mat',length
            = 50)
        write_mat_like_norm_putinplace(
        train_datas[i],train_labels[i],\
        file =
            root_out+str(i+1)+'train_like_norm_place.mat',length =
            50)
        write_mat_like_norm_putinplace(
        val_datas[i],val_labels[i],\
        file = root_out+str(i+1)+'val_like_norm_place.mat',length
            = 10)

#main('./origin_data/','./data_mat/')
main('./data_revised/','./data_mat_revised/')
```

（5）数据加载代码 Dataset.py

```
import torch
from torch.utils.data import Dataset
import numpy as np
from scipy.io import loadmat


class Trainset(Dataset):
    def __init__(self,subject_name, fileindex,data_root =
        'data_mat/',seq_length = None,halfdata = False):
        if seq_length==None:
            seq_length = 240
        if subject_name == 'A':
            raw_data = loadmat(data_root+str(fileindex)+\
                'train_like_norm_place.mat')
        else:
            raw_data = loadmat(data_root+str(fileindex)+\
                'val_like_norm_place.mat')

        signals = raw_data['responses']
```

```
        label = raw_data['is_stimulate']
        data = []
        target = []
        self.halfdata = halfdata
        for i in range(signals.shape[0]):
            for j in range(signals.shape[3]):
                if label[i, j] == 1:
                    data.append(signals[i, :, :, j].reshape(-1, 20))
                    target.append(label[i, j])
                    data.append(signals[i, :, :, j].reshape(-1, 20))
                    target.append(label[i, j])
                    data.append(signals[i, :, :, j].reshape(-1, 20))
                    target.append(label[i, j])
                    data.append(signals[i, :, :, j].reshape(-1, 20))
                    target.append(label[i, j])
                data.append(signals[i, :, :, j].reshape(-1, 20))
                target.append(label[i, j])

        self.data = np.array(data)
        self.target = np.array(target)
        self.seq_length = seq_length

    def __len__(self):
        num=1
        if self.halfdata:
            num = 2

        return self.target.shape[0]//num

    def __getitem__(self, index):
        return self.data[index, :self.seq_length,
            :].astype(np.float32).T,
            self.target[index].astype(np.float32)


class Testset(Dataset):


    def __init__(self,subject_name, fileindex,data_root =
        'data_mat/',seq_length = None):
        if seq_length==None:
            seq_length = 240
        self.seq_length = seq_length

        assert subject_name == 'A'
        raw_data = loadmat(
            data_root+str(fileindex)+'test_like_norm_place.mat')
        # else:

        self.signals = raw_data['responses']

    def __len__(self):
        return 50
```

```
    def __getitem__(self, index):
        col = self.signals[:6, :self.seq_length, :,
            index].astype(np.float32).transpose([0, 2, 1])
        row = self.signals[6:, :self.seq_length, :,
            index].astype(np.float32).transpose([0, 2, 1])
        return col, row
```

（6）经验模态分解代码 EMD.py

```
from PyEMD import EMD, Visualisation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as scio

f=open('train.csv',encoding='utf-8')
df = pd.read_csv(f)
S = np.array(df)

emd = EMD()
emd.emd(S[:,16])
imfs, res = emd.get_imfs_and_residue()

# 绘制 IMF
vis = Visualisation()
vis.plot_imfs(imfs=imfs, residue=res, t=np.arange(0,3117),
    include_residue=True)
vis.show()
vis.plot_instant_freq(t=np.arange(0,3117), imfs=imfs)
vis.show()
```

（7）快速傅里叶变换代码 FIFFT.m

```
function [ new_data ] = FiFFT( x )
fc = 250;      %采样频率250Hz
[a,b] = size(x);
N = a;         %采样点数
n = 0:N-1;
f = n*fc/N;     %频率序列
Wn = [0.3*2 25*2]/fc;%设置通带为0.5-25Hz
[k,l] = butter(2,Wn);%4阶IIR滤波器
new_data = zeros(N,20);
for i=1:20
    VarName1 = x(:,i);
    result = filtfilt(k,l,VarName1);
    tmp = fft(result);
    VarName1_IFFT = ifft(tmp);
    new_data(:,i) = VarName1_IFFT ;
    i=i+1;
end
end
```

（8）问题一 SVM 分类器代码 svm.py

```
from numpy import loadtxt
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import roc_auc_score, f1_score
from sklearn import metrics
from sklearn import svm
import random
from sklearn.utils import shuffle
from scipy.io import loadmat
import pickle
import heapq

sheet2label = {
   'char01(B)':[1,8],
   'char02(D)':[1,10],
   'char03(G)':[2,7],
   'char04(L)':[2,12],
   'char05(O)':[3,9],
   'char06(Q)':[3,11],
   'char07(S)':[4,7],
   'char08(V)':[4,10],
   'char09(Z)':[5,8],
   'char10(4)':[5,12],
   'char11(7)':[6,9],
   'char12(9)':[6,11]
}

def getDataLabel(random_seed=2,train_test_split = 0.2 ):
   X = []
   Y =[]
   test_X=[]
   test_Y=[]
   for charName,val in sheet2label.items():
      df =
         pd.read_excel('data/S1/S1_train_data.xlsx',sheet_name=charName,header=N
      df.index = df.index +1
      train_event =
         pd.read_excel('data/S1/S1_train_event.xlsx',sheet_name=charName,header=
      for i in train_event.index.values:
         if train_event.loc[i,0]<100:
            X.append(df.loc[
               (train_event.loc[i,1]):(train_event.loc[i,1]+125)
               , 0 ].values)
            if train_event.loc[i,0] in sheet2label[charName]:
               Y.append(1)
            else:
               Y.append(0)

   X = np.array(X)
   Y =np.array(Y)
   index = [i for i in range(len(X))]
   total_num = len(X)
   random.shuffle(index)
```

```
    X = X[index]
    Y = Y[index]
    test_X = X[ int((1-train_test_split)*total_num): ,:]
    test_Y = Y[ int((1-train_test_split)*total_num): ]
    X = X[ :int((1-train_test_split)*total_num) ,:]
    Y = Y[ :int((1-train_test_split)*total_num) ]
    print('hello')
    return X, Y ,test_X, test_Y

def calculate_metrics(test_Y,pred_Y):
    true_num = np.sum(test_Y == pred_Y)
    accuracy = true_num/ test_Y.shape[0]
    return accuracy

def getData_Label(feature_id=None):
    pkl_dir = 'sz_data/pkl'
    with open(pkl_dir+'/'+'train.pkl','rb') as f:
        trainData, trainLabel = pickle.load(f)
    with open(pkl_dir+'/'+'val.pkl','rb') as f:
        valData,valLabel = pickle.load(f)

    X = trainData
    Y = trainLabel
    test_X = valData
    test_Y = valLabel

    index = [i for i in range(len(X))]
    random.shuffle(index)
    X = X[index]
    Y = Y[index]
    index = [i for i in range(len(test_X))]
    random.shuffle(index)
    test_X = test_X[index]
    test_Y = test_Y[index]
    X = X.reshape(len(X), -1)
    test_X = test_X.reshape(len(test_X),-1)

    if not feature_id:
        print('hello')
        return X, Y, test_X, test_Y
    else:
        X = X[:, feature_id]
        test_X = test_X[:, feature_id]

        return X, Y, test_X, test_Y

if __name__ == '__main__':
    meanAUC=np.array([])
    meanAccuracy=np.array([])
    meanPrecision=np.array([])
    meanRecall = np.array([])

    for i in range(0,1):
        X, Y ,test_X, test_Y = getData_Label()
```

```
        #SVM
        model = svm.SVC(C=10,kernel='linear')
        model.fit(X , Y)

        # make predictions for test data
        pred_Y = model.predict(test_X)
        pred_Y = np.squeeze(pred_Y)
        accuracy = calculate_metrics(test_Y,pred_Y)
        meanAccuracy=np.append(meanAccuracy, accuracy)
        fpr, tpr, thresholds = metrics.roc_curve(test_Y , pred_Y)
        roc_auc=metrics.auc(fpr,tpr)
        meanAUC=np.append(meanAUC,roc_auc)
        meanPrecision=np.append(meanPrecision,metrics.precision_score(test_Y,pred_
        meanRecall=np.append(meanRecall,metrics.recall_score(test_Y,pred_Y))

    print('test meanAUC= %0.4f'% ( np.mean(meanAUC)))
    print('test meanAccuracy= %0.4f'% ( np.mean(meanAccuracy)))
    print('test meanPrecision= %0.4f'% ( np.mean(meanPrecision)))
    print('test meanRecall= %0.4f'% ( np.mean(meanAccuracy)))
```

（9）问题四睡眠分期代码 Q4.py

```
from numpy import loadtxt
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import roc_auc_score, f1_score
from sklearn import metrics
from xgboost import plot_importance
from sklearn import svm
from sklearn.externals import joblib
import random
from sklearn.utils import shuffle
from scipy.io import loadmat
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.feature_selection import RFE
from sklearn.svm import SVR

def getDataLabel(random_seed=2,train_test_split = 0.2 ):
    df = pd.read_csv('eegData2.csv')
    output = 'label'
    df['Theta+Delta'] = (df['Theta'] + df['Delta']) / 100.0
    df['Theta*Delta'] = (df['Theta'] * df['Delta']) / 1000.0
    df['mean_val'] = (df['Alpha'] + df['Beta'] + df['Theta'] +
        df['Delta']) / 4.0
    df['Alpha+Beta'] = (df['Alpha'] + df['Beta']) / 100.0
    df['Alpha*Beta'] = (df['Alpha'] * df['Beta']) / 1000.0
    df['logDelta'] = df['Delta'].apply(np.log)
    df['logTheta'] = df['Theta'].apply(np.log)
    df['logAlpha'] = df['Delta'].apply(np.log)
```

```
    df['logBeta'] = df['Theta'].apply(np.log)

    feature_columns = df.columns.values
    feature_columns = np.delete(feature_columns, 0).tolist()
    '''
    # 特征删除
    delete_fea = [feature_columns.index(name) for name in
        ['mean_val'] ]
    feature_columns = np.delete(feature_columns, delete_fea)'''

    random.seed(random_seed)
    df = shuffle(df)
    Index = df.index.values
    train_num = int((1-train_test_split) * Index.shape[0])
    train_Ind= Index[:train_num]
    test_Ind=Index[train_num:]

    X = df.loc[train_Ind,feature_columns].values
    Y = df.loc[train_Ind,output].values
    test_X = df.loc[test_Ind,feature_columns].values
    test_Y = df.loc[test_Ind,output].values

    return X, Y ,test_X, test_Y,feature_columns

def calculate_metrics(test_Y,pred_Y):
    true_num = np.sum(test_Y == pred_Y)
    accuracy = true_num/ test_Y.shape[0]
    precisionDict=dict()
    recallDict=dict()
    for i in range(2,7):
        true_num = np.sum( (test_Y==i) & (pred_Y==test_Y))
        precisionDict['precision_%d'%(i)]=true_num/(np.sum(pred_Y==i))
        recallDict['recall_%d'%(i)]=true_num/(np.sum(test_Y==i))

    return accuracy,precisionDict,recallDict

if __name__ == '__main__':
    meanAccuracy=np.array([])
    meanPrecision=[ [],[],[],[],[] ]
    meanRecall=[ [],[],[],[],[] ]
    for i in range(0,100):
        X, Y ,test_X, test_Y,feature_columns =
            getDataLabel(random_seed= i ,train_test_split = 0.2)

        # CatBoost
        model =
            CatBoostClassifier(loss_function='MultiClass',learning_rate=0.1,
            depth=6, iterations=1000)
        model.load_model('question4_model/' + 'CatBoost_.model')
        #model.fit(X, Y, verbose=True)
        #model.save_model('question4_model/' + 'CatBoost_.model')

        '''
        fea_ = model.feature_importances_
```

```
        fea_name = feature_columns
        print('hello')
        plt.figure(figsize=(8, 5))
        plt.xlabel('Weights of Feature Importances')
        plt.ylabel('Feature Names')
        plt.title('CatBoost Feature Importances')
        plt.barh(fea_name, fea_, height=0.5)
        plt.savefig('feature_importance.png',dpi=300)'''

        '''
        #RF
        model = RandomForestClassifier(n_estimators=500,
           random_state=0, n_jobs=-1)
        model.fit(X, Y)
        joblib.dump(model, 'question4_model/' + 'RF_.model')
        fea_ = model.feature_importances_
        fea_name = feature_columns
        plt.figure(figsize=(8, 5))
        plt.xlabel('Weights of Feature Importances')
        plt.ylabel('Feature Names')
        plt.title('Random Forest Feature Importances')
        plt.barh(fea_name, fea_, height=0.5)
        plt.savefig('RF_feature_importance.png',dpi=300)
        #model = joblib.load('question4_model/' + 'RF_.model')'''

        '''
        #SVM
        model = svm.SVC(C=10,kernel='linear')
        model.fit(X , Y)
        joblib.dump(model,'question4_model/' +'SVM_.model')
        #model=joblib.load('question4_model/' +'SVM_.model')
        estimator = SVR(kernel="linear")
        selector = RFE(estimator, n_features_to_select=1, step=1)
        selector = selector.fit(X, Y)
        print(selector.ranking_)'''

        # make predictions for test data
        pred_Y = model.predict(test_X)
        pred_Y = np.squeeze(pred_Y)
        accuracy,precisionDict,recallDcit =
           calculate_metrics(test_Y,pred_Y)
        meanAccuracy=np.append(meanAccuracy, accuracy)
        for i in range(2,7):
           meanPrecision[i-2].append(precisionDict['precision_%d'%(i)])
           meanRecall[i-2].append(recallDcit['recall_%d'%(i)])

    print('test meanPrecision= ', ( np.mean(meanPrecision,axis=1)
       ))
    print('test meanRecall= ', (np.mean(meanRecall,axis=1) ))
    print('test meanAccuracy= %0.4f'% ( np.mean(meanAccuracy)))
```

（10) 作图 drawPics.py

```
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.pyplot import MultipleLocator
import heapq

def drawPics():
   df = pd.read_csv('grid0920.csv')
   pic_dir = './pics/all_data'
   for y_name in ['chars_vote_acc', 'row_vote_acc',
      'col_vote_acc']:
      plt.figure(figsize=(10, 10))
      ax1 = plt.subplot(1, 1, 1, facecolor='white')
      ax1.set_title(y_name+' VS. turn')
      ax1.set_xlabel('turn')
      ax1.set_ylabel(y_name)
      x_major_locator=MultipleLocator(1)
      ax1.xaxis.set_major_locator(x_major_locator)
      for epoch in [100, 150, 200, 250]:
         for times in [0, 1, 2, 3, 4]:
            df_curve = df[(df['epoch'] == epoch) & (df['times']
               == times) ].loc[:,y_name].values
            turn_list = np.array([1, 2, 3, 4, 5])
            plot1 = ax1.plot(turn_list,df_curve ,
               linestyle='--', alpha=0.5,
               label='epoch=%d,time=%d'%(epoch,times+1))
      mean_list = []
      for turn in [1, 2, 3, 4,5]:
         df_bar = df[(df['turn'] == turn)].loc[:, y_name].values
         mean_list.append(np.mean(df_bar))
      ax1.bar(x=[1, 2, 3, 4, 5], height=mean_list, label='mean',
         color='steelblue', alpha=0.2,width=0.7)
      plt.legend(loc=4)
      plt.savefig(pic_dir+'/'+y_name+' VS.
         turn'+'.png',dpi=300,bbox_inches='tight')
      plt.clf()

      print('hello')

def drawPics_Q4_models():
   svm_acc = [0.7826, 0.6131, 0.5454, 0.5526, 0.7105]
   rf_acc = [0.9749, 0.9289, 0.9396, 0.9060, 0.9643]
   cat_acc = [0.9624, 0.9451, 0.9210, 0.9399, 0.9818]
   pic_dir = 'pics/Q4'

   plt.figure(figsize=(5, 4))
   ax1 = plt.subplot(1, 1, 1, facecolor='white')
   ax1.set_title('precision'+' VS. sleep stage')
   ax1.set_xlabel('sleep stage')
   ax1.set_ylabel('precision')
   turn_list=['stage 2','stage 3','stage 4','stage 5','stage 6']
   plot1 = ax1.plot(turn_list, svm_acc, linestyle='--',
      alpha=0.5, label='SVM')
   plot2 = ax1.plot(turn_list,rf_acc , linestyle='--',
      alpha=0.5, label='Random Forest')
```

```
    plot3 = ax1.plot(turn_list, cat_acc, linestyle='--',
        alpha=0.5, label='CatBoost')

    plt.legend(loc=7)
    plt.savefig(pic_dir+'/'+'precision'+' VS. sleep
        stage'+'.png',dpi=300,bbox_inches='tight')
    plt.clf()


def drawPics_Q4_splitTest():
    split_02 = [0.9624, 0.9451, 0.9210, 0.9399, 0.9818]
    split_03 = [0.9486, 0.9009, 0.8476, 0.8878, 0.9694]
    split_04 = [0.9325, 0.8354, 0.8190, 0.8798, 0.9395]
    split_05 = [0.9132, 0.8227, 0.7808, 0.8106, 0.9493]
    split_06 = [0.8887, 0.7771, 0.7365, 0.7678, 0.9214]
    pic_dir = 'pics/Q4'

    plt.figure(figsize=(8, 6.4))
    ax1 = plt.subplot(1, 1, 1, facecolor='white')
    ax1.set_title('precision'+' VS. sleep stage(different test
        ratios)')
    ax1.set_xlabel('sleep stage')
    ax1.set_ylabel('precision')
    turn_list=['stage 2','stage 3','stage 4','stage 5','stage 6']
    plot1 = ax1.plot(turn_list, split_02, linestyle='--',
        alpha=0.5, label='test ratio=0.2')
    plot2 = ax1.plot(turn_list, split_03, linestyle='--',
        alpha=0.5, label='test ratio=0.3')
    plot3 = ax1.plot(turn_list, split_04, linestyle='--',
        alpha=0.5, label='test ratio=0.4')
    plot4 = ax1.plot(turn_list, split_05, linestyle='--',
        alpha=0.5, label='test ratio=0.5')
    plot5 = ax1.plot(turn_list, split_06, linestyle='--',
        alpha=0.5, label='test ratio=0.6')

    plt.legend(loc=4)
    plt.savefig(pic_dir+'/'+'precision'+' VS. sleep
        stage(different test
        ratios)'+'.png',dpi=300,bbox_inches='tight')
    plt.clf()


def drawPics_Q4_featureSelection():
    only4 = [0.90757417, 0.78840346, 0.7529841, 0.77944082,
        0.92979261]
    nomean = [0.94534922, 0.92787586, 0.88894686, 0.88478347,
        0.96894546]
    noalpha = [0.96746961, 0.90585537, 0.90308763, 0.88163454,
        0.95255275]
    nobeta = [0.95644995, 0.91385205, 0.88671653, 0.89323518,
        0.93668056]
    nof3 = [0.93071798, 0.83696177, 0.86884802, 0.83645828,
        0.94683229]
    allfea=[0.9624, 0.9451, 0.9210, 0.9399, 0.9818]
```

```
    pic_dir = 'pics/Q4'

    plt.figure(figsize=(8, 6.4))
    ax1 = plt.subplot(1, 1, 1, facecolor='white')
    ax1.set_title('precision'+' VS. sleep stage(different feature
        selections)')
    ax1.set_xlabel('sleep stage')
    ax1.set_ylabel('precision')
    turn_list=['stage 2','stage 3','stage 4','stage 5','stage 6']
    plot1 = ax1.plot(turn_list, only4, linestyle='--', alpha=0.5,
        label='original features')
    plot2 = ax1.plot(turn_list, nomean, linestyle='--',
        alpha=0.5, label='no mean_val')
    plot3 = ax1.plot(turn_list, noalpha, linestyle='--',
        alpha=0.5, label='no alpha')
    plot4 = ax1.plot(turn_list, nobeta, linestyle='--',
        alpha=0.5, label='no beta')
    plot5 = ax1.plot(turn_list, nof3, linestyle='--', alpha=0.5,
        label='no alpha+beta+mean_val')
    plot6 = ax1.plot(turn_list, allfea, linestyle='--',
        alpha=0.5, label='all features')

    plt.legend(loc=4)
    plt.savefig(pic_dir+'/'+'precision'+' VS. sleep
        stage(different feature
        selections)'+'.png',dpi=300,bbox_inches='tight')
    plt.clf()

def drawAtten(topK_num=15):
    df = pd.read_csv('atten.csv')
    att_name = ['att' + str(x) for x in range(0, 20)]
    pic_dir = './pics/Q2/'
    topK_list=list(range(0,20))
    for member in [0, 1, 2, 3, 4]:
        '''
        plt.figure(figsize=(5, 4.5))
        ax1 = plt.subplot(1, 1, 1, facecolor='white')
        ax1.set_title('Attention weights of 20 channels(member
            %d)' % (member + 1))
        ax1.set_xlabel('weight') #设置x轴名称,plt.xlabel
        df_use = df[(df['member'] == member)].loc[:, att_name]
        data = df_use.values
        mean_data = np.mean(data, axis=0)
        topK_index = heapq.nlargest(topK_num,
            range(len(mean_data)), mean_data.take)
        plt.barh(np.array(att_name)[topK_index],
            mean_data[topK_index], height=0.5)
        plt.savefig(pic_dir+'Attention weights of 20
            channels(member %d)' % (member + 1)+'.png',dpi=300)
        '''

        df_use = df[(df['member'] == member)].loc[:, att_name]
        data = df_use.values
        mean_data = np.mean(data, axis=0)
```

```
        topK_index = heapq.nlargest(topK_num,
            range(len(mean_data)), mean_data.take)
        print('topK_index_%d'%(member+1),topK_index)
        topK_list =
            list(set(topK_list).intersection(set(topK_index)))

    print('topK_list: ', np.array(att_name)[topK_list])

def drawPics_Q3():
    pic_dir = 'pics/Q3/'
    '''
    train_num = [500, 813, 874, 916, 977, 993, 996, 1000, 1000,
        1000]
    all_num = [1000-x for x in train_num]
    val_acc = [63.30113306059693, 67.70241900287103,
        69.19546324122186, 70.06258898725791,
        71.14816301881214, 71.60553802582892, 71.1421054921608,
            71.36844175798016,
        72.2004581246644, 71.48270246266668] '''
    train_num = [500, 730, 840, 881, 929, 960, 975, 985, 988, 992]
    all_num = [1000-x for x in train_num]
    val_acc = [63.13570226074297, 62.82996290564528,
        63.41236107701262,
        64.05747541680154, 65.36986864901041, 66.82857890163497,
        67.10069139956038, 67.82799748945725, 67.29152755884778,
            67.23224108653888]

    plt.figure(figsize=(5, 4.5))
    ax1 = plt.subplot(1, 1, 1, facecolor='white')
    ax1.set_xlabel('turns')
    ax1.set_ylabel('sample_size')
    ax1.set_ylim(0, 1000)
    x_major_locator=MultipleLocator(1)
    ax1.xaxis.set_major_locator(x_major_locator)

    ind = np.arange(1,11)
    p1 = plt.bar(ind, train_num, width=0.35)
    p2 = plt.bar(ind, all_num, width=0.35, bottom=train_num)
    plt.legend((p1[0], p2[0]), ('train_samples',
        'unlabeled_samples'),loc=3)

    ax2 = ax1.twinx()
    ax2.plot(ind, val_acc, '--',color='green', label='val_acc')
    ax2.set_ylabel('val_acc')
    ax2.set_ylim(60, 74)
    ax2.legend(loc=4)
    plt.savefig(pic_dir + 'results1.png', dpi=300)

if __name__ == '__main__':
    drawPics()
    drawPics_Q4_splitTest()
    drawPics_Q4_featureSelection()
    drawAtten()
    drawPics_Q3()
```