基于脑电信号的情绪识别方法

18数据科学 高剑雄

201800620686

摘　要：针对脑电图(EEG)特征提取单一、分类准确率低等问题，为获得可靠区分性特征，提高分类准确率，采用一种分解脑电信号的特征提取方法。该方法根据脑波频率将脑电信号分解为δ、θ、α、β、γ五大类脑波的时域、频域信号，继而分别对时频两域信号进行特征提取，再利用线性判别分析(LDA)进行二次特征提取。使用三类不同情绪状态的 EEG数据集进行实验，分别在LR、KNN、DNN、SVM、RF分类器上进行测试，五种分类器的平均准确率均已超过85%。后尝试对特征数据集进行特征裁剪后，各分类器的准确率有所提升。其中，在SVM分类器下的平均准确率达到了90.32%。特征裁剪后，提高了识别的准确率，同时大大地节省了运算时间，具备了一定的实用价值，可运用于实际工程领域。

受制于硬件条件，本文未能尝试一维CNN、RNN方法对频域信号进行特征提取，从而与$CNN+LSTM$等深度学习方法进行对比。

关键词：情绪识别、脑电信号、特征提取、LDA降维、分类算法

引　言：情绪识别是让计算机感知人类情绪状态，进而进行人机情绪交互的关键技术，已经成为数字信号处理与人工智能领域的研究热点，在医学和工程领域具有非常重要的意义。

但由于人类情绪是多样与复杂的，通过计算机对其进行识别仍存在不少问题，其主要表现为如下几点：

i. 目前的研究成果中，情绪识别大多依赖于已测得的数据集，且特征提取难。

ii. 大多模型算法复杂度仍较高，无法做到实时识别，限制了情绪识别的应用。

iii. 现有识别之法正确率虽然较高，但仍需大量的数据集验证才能投入实际运用。

而脑电图($electroencephalogram, EEG$)以高时间分辨率及低伪装性在情绪识别领域中表现优异。研究 [1]表明，EEG高频成分可以反映人的情绪和认知状态，其中β波、γ波的表现较之δ、θ、α要好。张克军 [2]等人通过微分熵与LDA结合的特征提取之法提高了识别的运算速率与准确率。目前，脑电信号的处理方式 [3]已成初步成体系，但是针对具体问题，仍有细节可以改进。

1　设计方案与设计思路

1.1　数据研究对象

本文使用名为$SEED$的公开情绪EEG数据集进行试验，该数据集为$BCMI$实验室提供的EEG数据集的集合，该实验由路宝良教授领导。该数据集使用中国影片作为情绪诱导材料，包含三类情绪：积极情绪、中性情绪和消极情绪。在每组实验中，志愿者观看诱导不同情绪状态的电影片段。每个电影剪辑播放约4分钟，在实验中播放3种类型的电影剪辑，每种类型的影片剪辑包含5部电影，共15部电影。每次播放影片前有5s的提示，播放后有4s的反馈时间，志愿者影片播放完毕后有15s的休息时间。共15名视觉、听觉和情绪状态正常的志愿者参加了实验(7名男性8名女性，平均年龄为平均年龄为23.27岁，标准差为2.37)。当志愿者观看电影时，EEG信号通过电极帽记录，采样频率为1000hz。该实验使用国际$10-20$系统和62通道电极帽，具体位置如图1所示。每位志愿者相隔约1周参加一次实验，参共3次。因此，该实验共形成675个数据样本。最后，对测得数据进行200hz的下采样和$0.5 \sim 70hz$范围的滤波以获得预处理的EEG数据集。

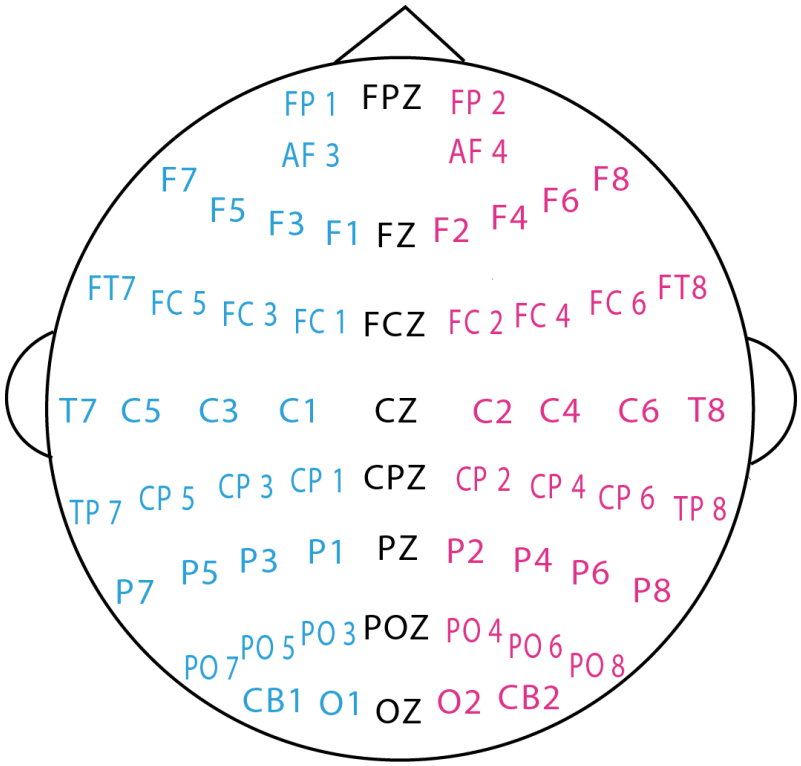


图 1: 脑电信号通道及其位置

1.2 项目方案

图 2: 项目流程图

1.3 项目设计思路

本节以其中一位中文姓名首字母缩写为djc的实验者的第一次实验数据为例，叙述项目设计思路(其他实验者的实验数据以及该实验者的其他实验数据均按照此思路进行处理)。

首先，引入其EEG数据集，该数据包含该实验者观看15个影片剪辑片段的15个62通道实验数据。

1.3.1 时域滤波

已知，该EEG信号已被200hz下采样，及0.5 ~ 70hz范围的带通滤波，符合$Nyquist$采样定理，后续处理不会产生混叠现象。因此，可直接对其进行下一步滤波处理。

EEG信号中可能混有50hz的工频干扰信号，故本文利用 $matlab$ 设计了一个4阶，归一化截止频率为0.5的巴特沃斯低通滤波器，将该滤波器参数代入 $filtfilt$ 函数中，实现零相位滤除干扰信号，低通滤波器参数如下：

b	0.0939809	0.3759234	0.5638851	0.3759234	0.0939809
a	1	7.910e-16	0.4860288	2.628e-16	0.0176648

充分查阅EEG信号相关资料及文献，得知其可根据频谱划分为五大基本类型，分别为：δ波、θ波、α波、β波、γ波。其中α波与β波又都可以划分为三小类。将上述脑波种类及其频率范围整理如下表：

脑波种类	频段
δ	$0.1 \sim 4Hz$
θ	$4 \sim 9Hz$
α_S	$8 \sim 9hz$
α_M	$9 \sim 12hz$
α_F	$12 \sim 14hz$
β_{LR}	$12 \sim 16Hz$
β_{MR}	$16 \sim 20Hz$
β_{HR}	$20 \sim 30Hz$
γ	$> 30hz$

表 1: 脑波种类及其频段

为了尽可能多的提取信号中的特征信息，本文利用 $matlab$ 设计了$9+2=11$个不同的巴特沃斯带通滤波器，其中前9个巴特沃斯带通滤波器参照上表设计，后2个滤波器所通过的频段为$8 \sim 12hz$与$14 \sim 30Hz$，从而提取出完整的α波与β波。将上述11个滤波器的参数代入$filtfilt$ 函数中，分别对数据集中的62通道EEG信号进行零相位滤波，从而得到相应频段的脑波，将原始EEG信号拓展为$62 \times 12 = 744$通道的时域信号。

1.3.2 快速傅里叶变换

通过**时域滤波**步骤，已得到744个通道的时域信号，分别对其进行快速傅里叶变换(FFT)，从而得到744通道的频域信号，对其进行归一化提取其单侧幅值频谱。因而，原始EEG信号被拓展为$744 \times 2 = 1488$通道的信号。

1.3.3 特征提取

本次实验中，实验者djc共观看了15部影片剪辑。因此，分别对15个1488通道的EEG信号进行时域频域特征提取。其中，从时频两域提取的特征内容如下所示：

时域特征 均值、标准差、变异系数、极大极小值、极差、四分位数、功率、绝对中值误差、信息熵、绝对平均值、峰度、偏度、波形因子、峰值因子、脉冲因子、裕度因子

频域特征 均值、标准差、变异系数、极大值、四分位数、功率、绝对中值误差、信息熵、绝对平均值、峰度、偏度、波形因子、峰值因子、脉冲因子、裕度因子

因此，从实验者djc观看每一部影片时的EEG信号中提取了$1488 \times 32 = 23808$个特征。

1.3.4 特征降维

对所有的EEG信号进行特征提取后，得到尺寸为$(675, 23809)$的特征数据集。面对如此庞大的特征，直接将其放入分类器中进行训练并分类是不合适的。于是，本文利用线性判别分析 (LDA) 算法对其进行降维，以实现二次特征提取并降低分类算法的时间复杂度。

LDA 算法的基本目标是将数据集样本的高维特征投影到低维空间中，以达到提取信息和降维的效果。降维后，特征具有最大类间距以及最小类内距，它是一种有效的降维之法。

LDA 算法降维后，特征数据集尺寸缩小为$(675, 3)$，适合投入绝大多数的分类器中。

1.3.5 模式识别

本文对特征集采用5种模式识别（分类）方法$(LR、KNN、MLP、SVM、RF)$进行测试比较研究。每种测试实验中将675个样本随机（实验中采用多个随个随机种子的方法）分配到互斥的训练集(70%)和测试集(30%)，以保证结果的正确性。

注：疫情居家受硬件条件制约，没有足够的算力进行深度学习训练，在此简要说明思路：

1、对12通道的时域EEG信号提取出的特征放入DNN训练。

2、保留12通道的频域EEG信号，直接投入一维卷积神经网络中提取特征

3、将DNN与CNN合并，再次经过一个的DNN神经网络，从而融合两域特征，输出分类结果。

1.3.6 特征裁剪

我们知道，在实际工作中，特别是在需要实时反馈的工作设备上，信号处理直至模式识别所占用的时间越少越好，从而提高设备工作效率。而在本实验中，23808个特征的数据集在分类器中表现较好，但信号处理过程中需要花费太多时间。因此尝试对其特征进行裁剪，以提高效率。

利用**1.3.2 快速傅里叶变换**的结果，发现当信号的频率大于$12hz$时，幅频响应下降很快，因此尝试只采用频率大于$12hz$的特征。实际操作时，将特征减半，只使用原始数据、δ波、θ波、α波、α_S波、α_M波提取出的特征。此时，特征被减少一半，剩下11909个。

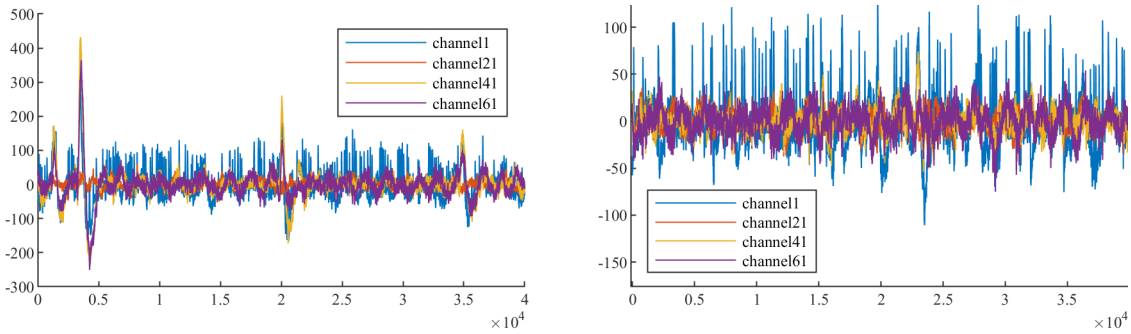
同时，不删除另一半特征，从中取出β波、γ波的特征，共9920个。

分别对两份子数据集进行降维，将降维后的数据投入分类器中，具体操作与**1.3.5 模式识别**中相同。

2　设计结果

2.1　读取数据结果

利用 *matlab* 的 *load* 函数依次加载 EEG 数据集。因其数据量庞大，这里同样以其中中文姓名首字母缩写为 *djc* 的实验者的第一次实验数据为例，截取其观看影片剪辑一、二、三、九时脑电信号的$FP1$、$FC4$、$TP8$、$O2$(对应channel1、21、41、61)信号前40000个采样点信号进行可视化，如下图所示（已注明影片剪辑序号及其情绪状态）：



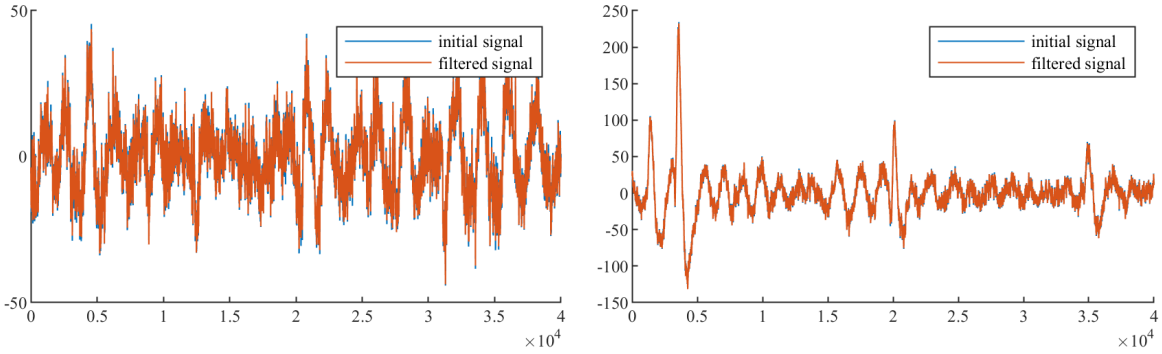
(a) 影片剪辑一[积极情绪]　(b) 影片剪辑二[中性情绪]

(c) 影片剪辑三[消极情绪]　(d) 影片剪辑九[积极情绪]

图 3：读取数据

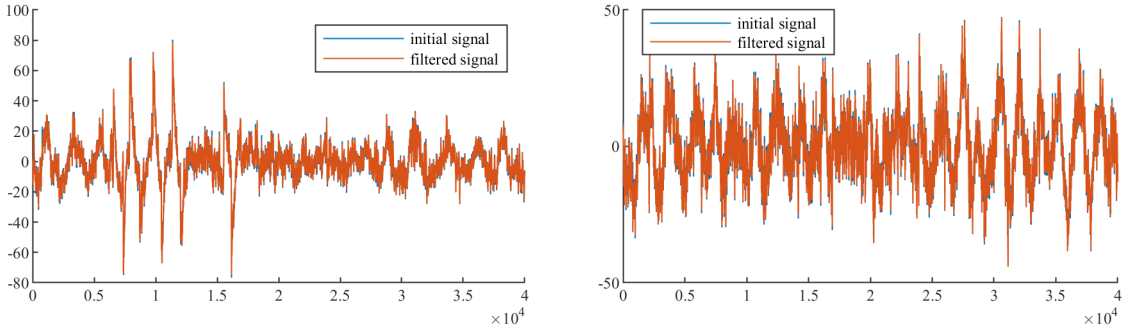
2.2　滤波结果

利用 *matlab* 对所有的 62 通道的EEG信号进行$50hz$的低通滤波以滤除工频干扰信号。选取实验者*djc*观看影片剪辑一、二、三时EEG信号的$FC1$、PZ(对应*channel*18、46)信号，作下图展示滤波结果：



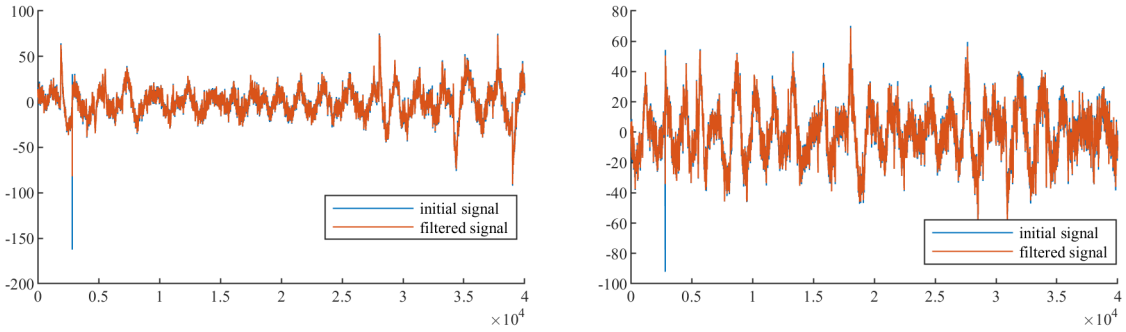
(a) $FC1[channel18]$ (b) $PZ[channel46]$

图 4: 影片剪辑一



(a) $FC1[channel18]$ (b) $PZ[channel46]$

图 5: 影片剪辑二



(a) $FC1[channel18]$ (b) $PZ[channel46]$

图 6: 影片剪辑三

2.3 脑波提取及快速傅里叶变换结果

据表一中不同脑波对应的频率，分别对所有的滤波后的EEG信号进行带通滤波，从而提取出δ波、θ波、α波、α_S波、α_M波、α_F波、β波、β_{LR}波、β_{MR}波、β_{HR}波、γ波等脑波信号。

下将分别展示：

i. α波、α_S波、α_M波、α_F波时频两域图像

ii. β波、β_{LR}波、β_{MR}波、β_{HR}波时频两域图像

iii. 滤波后的完整EEG信号、δ波、θ波、α波、β波、γ波时频两域图像

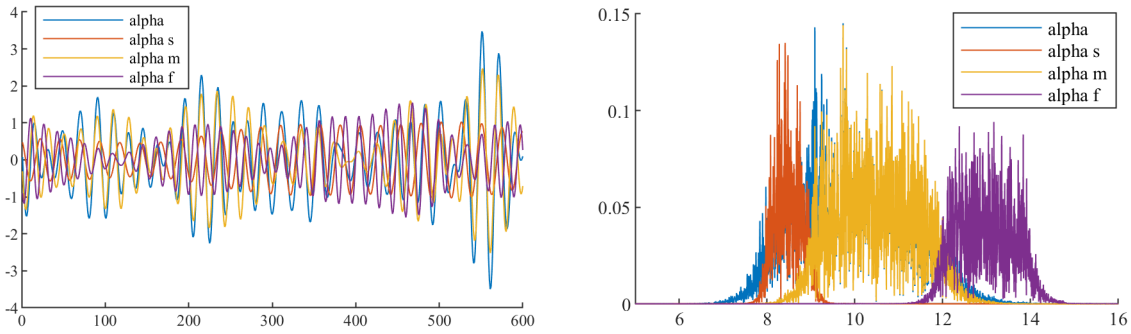
2.3.1 α类、β类脑波 时频两域图像

以影片剪辑一的$FC1(channel18)$信号为例。为了保证图像的清晰性，从该信号中截取第$1501-2100$共600个采样点，对应$3s$信号。分别从该信号中提α波、α_S波、α_M波、α_F波、β波、β_{LR}波、β_{MR}波、β_{HR}波的时域信号。

i. 将α波、α_S波、α_M波、α_F波时域信号组合，并对其进行快速傅里叶变换。

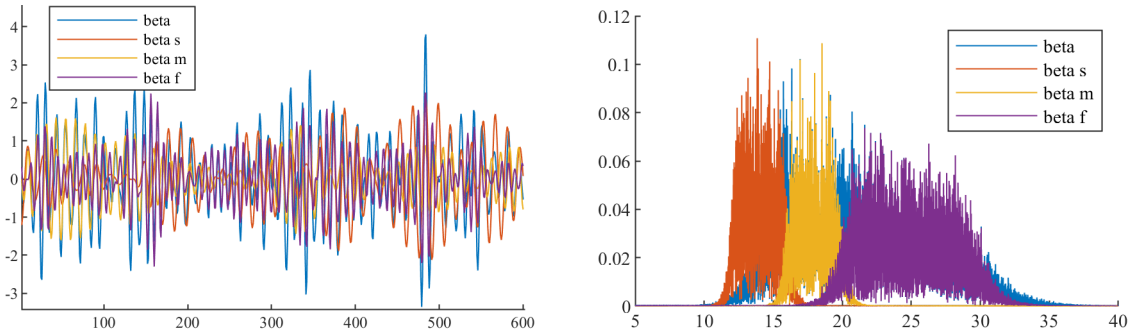
ii. 将β波、β_{LR}波、β_{MR}波、β_{HR}波时域信号组合，并对其进行快速傅里叶变换。

其结果如下图所示：



(a) α类脑波时域信号　　(b) α类脑波频域信号

图 7: α类脑波 时频两域图像



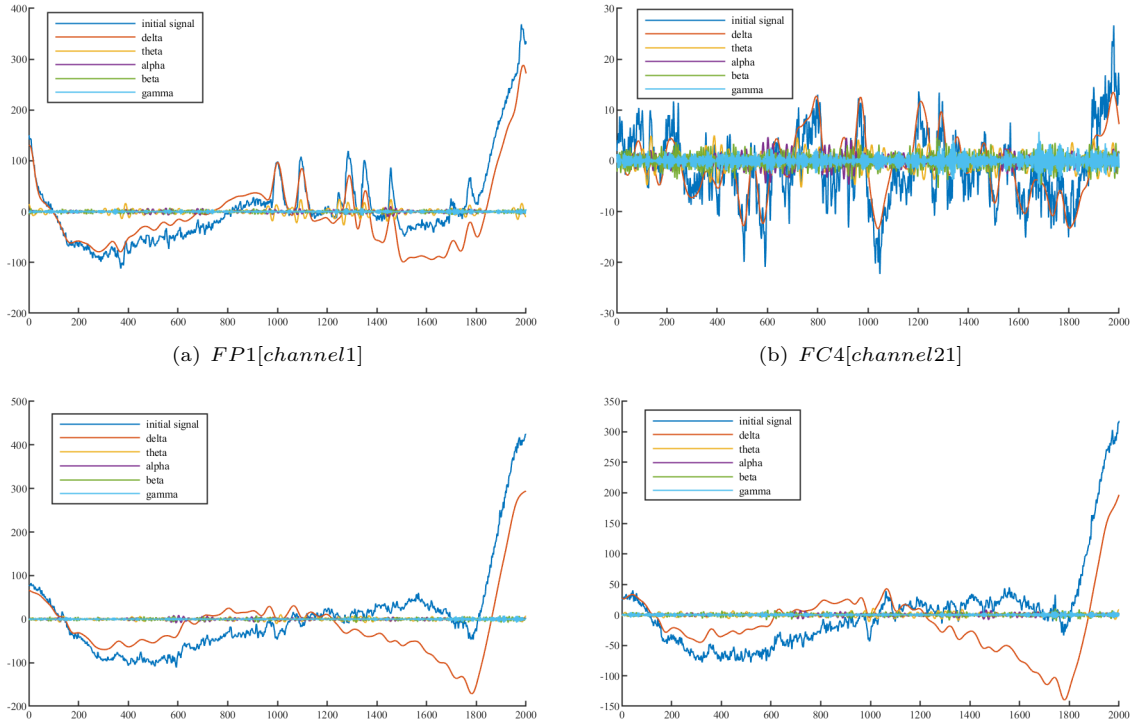
(a) β类脑波时域信号　　(b) β类脑波频域信号

图 8: β类脑波 时频两域图像

通过时频两域图像，我们验证了滤波操作的正确性。图像清晰地展示了α波、β波及其子波的特征各异，从而证明我们充分提取了α波、β波频段的特征。

2.3.2 δ波、θ波、α波、β波、γ波、滤波后完整EEG信号时频两域图像

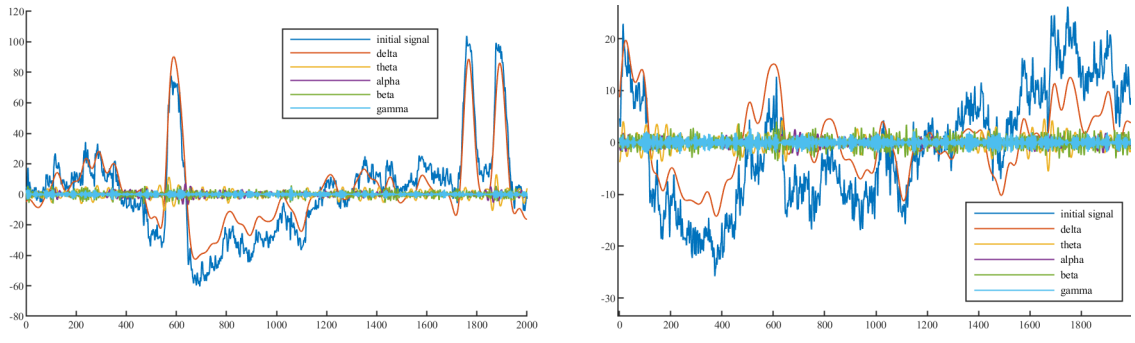
将实验者djc观看影片剪辑一、二、三时的EEG信号的$FP1$、$FC4$、$TP8$、$O2$(对应$channel1$、21、41、61)信号分解为δ波、θ波、α波、β波、γ波5类脑波，结合滤波后完整EEG信号，其结果如下图：



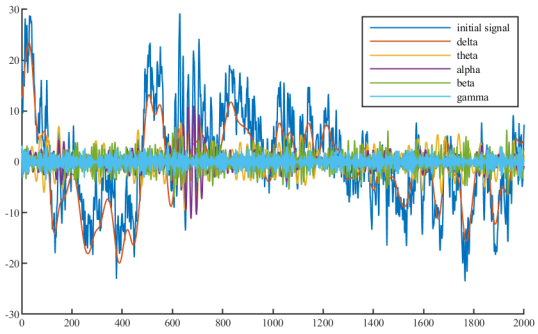
(a) $FP1[channel1]$ (b) $FC4[channel21]$

(c) $TP8[channel41]$ (d) $O2[channel61]$

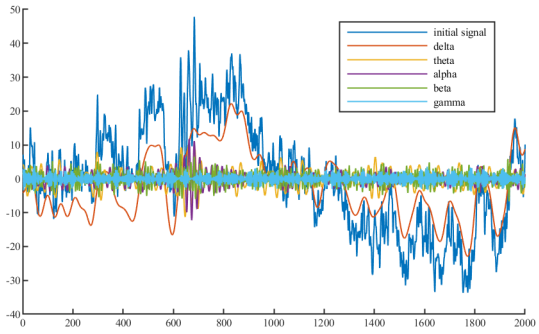
图 9: 影片剪辑一[积极情绪]



(a) $FP1[channel1]$ (b) $FC4[channel21]$

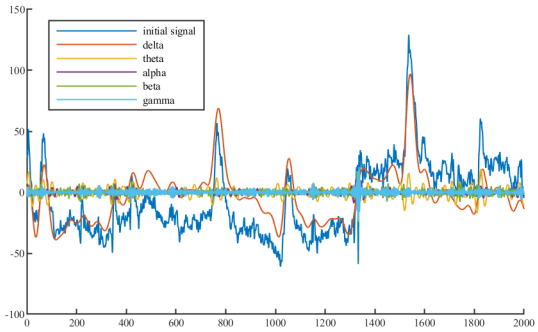


(c) $TP8[channel41]$

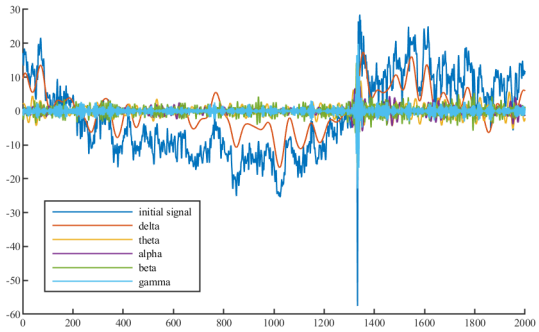


(d) $O2[channel61]$

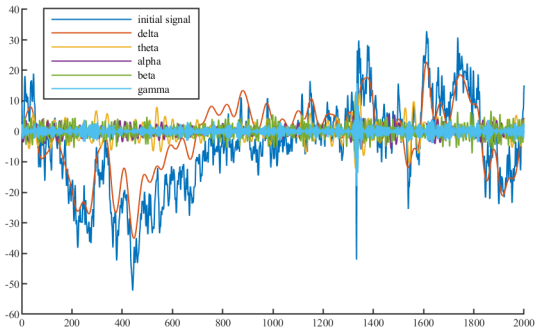
图 9: 影片剪辑二[中性情绪]



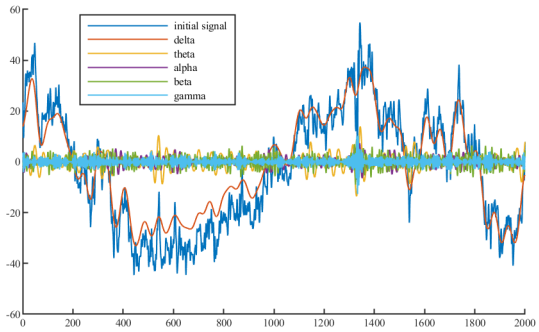
(a) $FP1[channel1]$



(b) $FC4[channel21]$



(c) $TP8[channel41]$



(d) $O2[channel61]$

图 10: 影片剪辑三[消极情绪]

下分别对滤波后的完整EEG信号、δ波、θ波、α波、β波、γ波6个脑波信号进行快速傅里叶变换，以影片剪辑一的$FP1(channel1)$为例，做出其频谱图像，如下图所示：

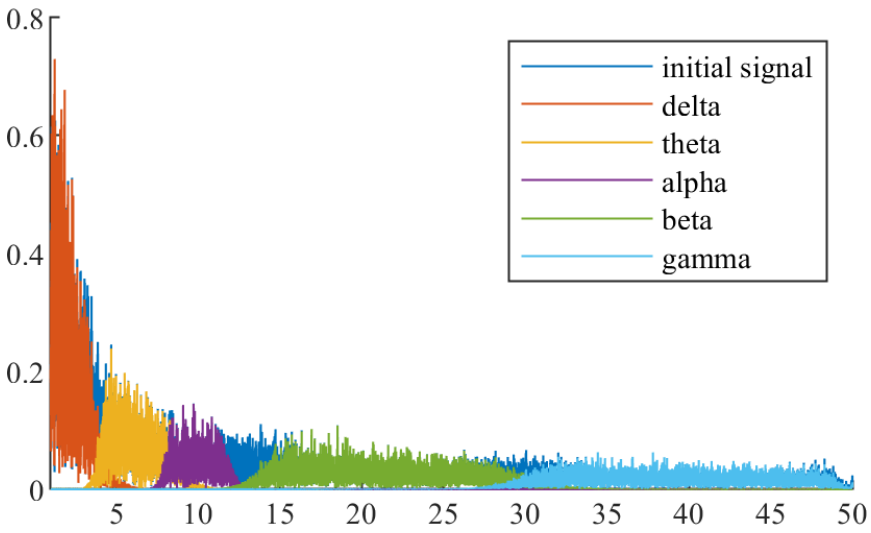


图 11: 5类脑波频谱

通过频谱图像，验证了本文操作的正确性。同时，我们从中可以看出：

i. 不同情绪对应的脑电信号是有显著区别的。

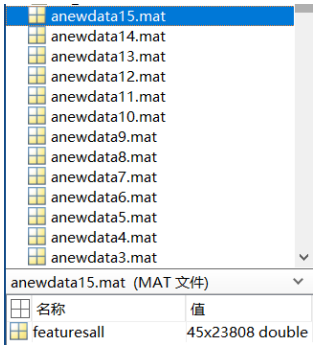
ii.情绪相关脑电信号能量集中在$0 \sim 12hz$，即δ波、θ波、α波，这对我们后续进行特征裁剪是有帮助的。

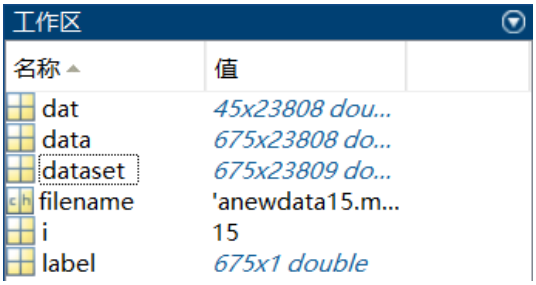
至此，已从原始EEG数据集中提取出12通道的脑电时域信号与12通道的脑电频域信号，且已经验证其操作的正确性。

2.4 特征提取及合并

对675个数据的12通道时频两域EEG信号进行特征提取。**2.3脑电提取及快速傅里叶变换结果**与此步骤的特征提取，耗时非常长，且占用电脑内存大（实际操作时尽量分批进行），该过程进行了**约5个小时**，最终结果如下左图所示。

将所有实验者的特征数据合并，同时加上每一个数据的$label$，得到一个(675×23809)的特征数据集，如下右图所示。





2.5 数据降维与特征裁剪

2.5.1 数据降维

引入*python*著名机器学习开源库*scikit-learn*，分别调用PCA、LDA函数，对数据集降维。LDA降维后的数据在在实际测试中的表现远优于PCA，在此只展示LDA的过程与结果，其具体操作如图12所示：

```
if __name__ == '__main__':
    np.random.seed(4)
    # dataset = pd.read_csv('dataset.csv').values
    # dataset = pd.read_csv('dataset_more_features.csv').values
    dataset = pd.read_csv('dataset_all_features.csv').values
    np.random.shuffle(dataset)

    data, label = dataset[:, :-1], dataset[:, -1]
    data = get_lda_data(data, label, 2)
    print(data.shape)
    # # data = get_pca_data(data, 10)

(675, 2)
[Finished in 13.1s]
```

2.5.2 特征裁剪

2.3.2 δ波、θ波、α波、β波、γ波、滤波后完整EEG信号时频两域图像 的结论指出，信号的能量集中在δ波、θ波、α波上，因此，我们进行特征裁剪，删去其他种类脑波所提取出的特征，如此得到一个(675×11905)的特征数据集。

与此同时，以往的研究 [1]表明，EEG高频成分，即β波和γ波比δ波、θ波、α能更好地区分开不同情绪。因此，本文保留β波和γ波种提取的特征，同样投入分类算法进行实验，于是我们又得到一个(675×9921)的特征数据集。具体操作如下图所示：

```
dataset = pd.read_csv('dataset_all_features.csv').values
print(dataset[0].shape)
newdataset = [[[]for i in range(9920)]for i in range(675)]
for k in range(675):
    for j in range(62):
        for i in range(160):
            newdataset[k][i + j * 160] = dataset[k][224 + j * 384 + i]
    newdataset[k] = np.array(newdataset[k])
print(newdataset[i])
dfdata = pd.DataFrame(newdataset)
datapath1 = 'new_features.csv'
dfdata.to_csv(datapath1, index=False)
```

对于上述两个子数据集，本文同样对其进行LDA降维后投入分类器。

2.6 算法分类结果

在分类过程中，为了保证所得结论的一般性，本文选择了21个伪随机数，在投入算法前将数据集打乱。不同的特征提取之法，对应的其结果如下表：

EEG特征提取方法	分类方法	平均准确率%	**F1/%**	$Kappa$系数
$\delta+\theta+\alpha+LDA$	LR	90.27	90.21	0.85
	DNN	86.49	86.39	0.80
	SVM	90.32	90.24	0.86
	KNN	89.12	89.07	0.84
	RF	88.86	88.76	0.83
$\beta+\gamma+LDA$	LR	89.12	89.07	0.84
	DNN	85.22	85.16	0.78
	SVM	89.16	89.12	0.84
	KNN	87.94	87.95	0.82
	RF	87.12	87.10	0.81
$\delta+\theta+\alpha+\beta+\gamma+LDA$	LR	88.74	88.71	0.83
	DNN	85.74	85.68	0.79
	SVM	88.53	88.51	0.83
	KNN	87.68	87.72	0.81
	RF	87.87	87.82	0.82

通过上表，我们可以看出，将全部特征都投入分类器中的效果并不是最理想的。且EEG信号低频部分在情绪识别方面比其高频部分表现更好。其中，LR与SVM分类器的效果较好，由于SVM的平均准确率在各种特征提取方法之下都是最高的，可视化SVM的平均混淆矩阵：

(c) $\delta+\theta+\alpha+LDA$ (d) $\beta+\gamma+LDA$

图 12: 平均值混淆矩阵

3 总结

本文提出了一种基于分解 EEG 信号结合 LDA 的情绪识别之法，极大地提高了 EEG 信号情绪识别的准确率，且该特征提取之法在较多的分类器上表现较好。

该方法对特征进行裁剪，减少了半数特征，旨在减少特征提取中花费的时间，同时保证了一定的准确率。在实际运用中，不管是医学领域，还是工业领域，都能在一定程度上提高工作效率。

基于本项目，后续若有时间将会测试更多数据集来验证本文之法。如果有硬件资源的支持，会尝试深度学习之法，如$LSTM+CNN$，来进一步提高情绪识别的效率。

最后，对情绪识别进行的简答展望。我觉得在该领域，仍有很多的地方是可以优化改进的，直至该技术可以投入工业生产。我期待在不久的将来，情绪识别技术会真正落地，在实际产品中得到非常好的应用。

附录

4 源代码介绍

4.1 使用软件

matlab、*python*

4.2 环境配置

matlab : *matlab R*2020*a*
python: *python*3.6 *Idle*: *sublime*3

4.3 项目程序运行顺序

1、http://bcmi.sjtu.edu.cn/~seed/downloads.html#seed-access-anchor申请数据集，并下载。
2、将数据集与所有的.m文件放在同一文件夹下。
3、依次运行**read_signal.m**，**cat_features.m**，得到完整特征数据集。
4、运行**deal.py**特征裁剪，代码提取高频脑波特征，获取低频需另改。
5、运行**algorithm.py**进行算法测试。

4.4 代码部分

4.4.1 matlab部分

read_signal.m（用于信号处理与特征提取）

```
%% 加载数据集
clc
clear
close all;
dat1_1 = load('predataset/1_1.mat');
dat1_2 = load('predataset/1_2.mat');
dat1_3 = load('predataset/1_3.mat');
dat2_1 = load('predataset/2_1.mat');
dat2_2 = load('predataset/2_2.mat');
dat2_3 = load('predataset/2_3.mat');
dat3_1 = load('predataset/3_1.mat');
dat3_2 = load('predataset/3_2.mat');
dat3_3 = load('predataset/3_3.mat');
dat4_1 = load('predataset/4_1.mat');
dat4_2 = load('predataset/4_2.mat');
dat4_3 = load('predataset/4_3.mat');
dat5_1 = load('predataset/5_1.mat');
dat5_2 = load('predataset/5_2.mat');
dat5_3 = load('predataset/5_3.mat');
```

```
% dat6_1 = load('predataset/6_1.mat');
% dat6_2 = load('predataset/6_2.mat');
% dat6_3 = load('predataset/6_3.mat');
% dat7_1 = load('predataset/7_1.mat');
% dat7_2 = load('predataset/7_2.mat');
% dat7_3 = load('predataset/7_3.mat');
% dat8_1 = load('predataset/8_1.mat');
% dat8_2 = load('predataset/8_2.mat');
% dat8_3 = load('predataset/8_3.mat');
% dat9_1 = load('predataset/9_1.mat');
% dat9_2 = load('predataset/9_2.mat');
% dat9_3 = load('predataset/9_3.mat');
% dat10_1 = load('predataset/10_1.mat');
% dat10_2 = load('predataset/10_2.mat');
% dat10_3 = load('predataset/10_3.mat');
% dat11_1 = load('predataset/11_1.mat');
% dat11_2 = load('predataset/11_2.mat');
% dat11_3 = load('predataset/11_3.mat');
% dat12_1 = load('predataset/12_1.mat');
% dat12_2 = load('predataset/12_2.mat');
% dat12_3 = load('predataset/12_3.mat');
% dat13_1 = load('predataset/13_1.mat');
% dat13_2 = load('predataset/13_2.mat');
% dat13_3 = load('predataset/13_3.mat');
% dat14_1 = load('predataset/14_1.mat');
% dat14_2 = load('predataset/14_2.mat');
% dat14_3 = load('predataset/14_3.mat');
% dat15_1 = load('predataset/15_1.mat');
% dat15_2 = load('predataset/15_2.mat');
% dat15_3 = load('predataset/15_3.mat');

%% 合并数据集，方便操作
initial1 =[dat1_1,dat1_2,dat1_3];
initial2 =[dat2_1,dat2_2,dat2_3];
initial3 =[dat3_1,dat3_2,dat3_3];
initial4 =[dat4_1,dat4_2,dat4_3];
initial5 =[dat5_1,dat5_2,dat5_3];
% initial6 =[dat6_1,dat6_2,dat6_3];
% initial7 =[dat7_1,dat7_2,dat7_3];
% initial8 =[dat8_1,dat8_2,dat8_3];
% initial9 =[dat9_1,dat9_2,dat9_3];
% initial10 =[dat10_1,dat10_2,dat10_3];
% initial11 =[dat11_1,dat11_2,dat11_3];
% initial12 =[dat12_1,dat12_2,dat12_3];
% initial13 =[dat13_1,dat13_2,dat13_3];
% initial14 =[dat14_1,dat14_2,dat14_3];
% initial15 =[dat15_1,dat15_2,dat15_3];

%% 滤波并特征提取
get_data(initial1,1);
get_data(initial2,2);
get_data(initial3,3);
get_data(initial4,4);
get_data(initial5,5);
% get_data(initial6,6);
```

```
% get_data(initial7,7);
% get_data(initial8,8);
% get_data(initial9,9);
% get_data(initial10,10);
% get_data(initial11,11);
% get_data(initial12,12);
% get_data(initial13,13);
% get_data(initial14,14);
% get_data(initial15,15);
```

get_data.m（信号处理与特征提取步骤的自定义函数）

```
function get_data(initial,num)
    filename = strcat('anewdata',num2str(num),'.mat')
    featuresall=[];
    for x = 1:3
        % 保存的文件名
        initial_dataset = initial(x);
        fileds = fieldnames(initial_dataset);
        features = [];
        for index=1:length(fileds)
            k = fileds(index);
            key = k{1};
            data = initial_dataset.(key);
            fs = 200;
            squence_data = [];
            squence_fft = [];
            disp('开始数字滤波')
            disp(index)
            for i=1:62
                channel = data(i,:);
                channel = low_pass_filter(50*2/fs,4,channel);
                N = length(channel);
                % 整体的快速傅里叶变换
                channel_fft = fft(channel);
                channel_fft = abs(channel_fft(1:floor(N/2)) * 2/N);% 滤波，快速傅里叶变换
                % 滤得 delta 波
                [Delta_data,Delta_fft] = filter_and_fft([0.1*2 4*2]/fs,4,channel);
                % 滤得theta波
                [theta_data,theta_fft] = filter_and_fft([4*2 9*2]/fs,4,channel);
                % 滤得alpha波
                [alpha_data,alpha_fft] = filter_and_fft([8*2 12*2]/fs,4,channel);
                [alpha_s_data,alpha_s_fft] = filter_and_fft([8*2 9*2]/fs,4,channel);
                [alpha_m_data,alpha_m_fft] = filter_and_fft([9*2 12*2]/fs,4,channel);
                [alpha_f_data,alpha_f_fft] = filter_and_fft([12*2 14*2]/fs,4,channel);
                % 滤得beta波
                [beta_data,beta_fft] = filter_and_fft([14*2 30*2]/fs,4,channel);
                [beta_SMR_data,beta_SMR_fft] = filter_and_fft([12*2 16*2]/fs,4,channel);
                [beta_sr_data,beta_sr_fft]   = filter_and_fft([16*2 20*2]/fs,4,channel);
                [beta_hr_data,beta_hr_fft]   = filter_and_fft([20*2 30*2]/fs,4,channel);
                % 滤得gamma波
                [gamma_data,gamma_fft]   = filter_and_fft([30*2 50*2]/fs,4,channel);
```

```
41                  % 时域
42                  subdataset = [  channel
43                                  Delta_data
44                                  theta_data
45                                  alpha_data
46                                  alpha_s_data
47                                  alpha_m_data
48                                  alpha_f_data
49                                  beta_data
50                                  beta_SMR_data
51                                  beta_sr_data
52                                  beta_hr_data
53                                  gamma_data   ];
54                  % 频域
55                  sub_squence_fft = [ channel_fft
56                                      Delta_fft
57                                      theta_fft
58                                      alpha_fft
59                                      alpha_s_fft
60                                      alpha_m_fft
61                                      alpha_f_fft
62                                      beta_fft
63                                      beta_SMR_fft
64                                      beta_sr_fft
65                                      beta_hr_fft
66                                      gamma_fft   ];
67                  squence_data = [squence_data
68                                  subdataset];
69                  squence_fft = [squence_fft
70                                  sub_squence_fft];
71              end
72              % 特征提取
73              feature_single = [];
74              for i=1:744
75                  A = squence_data(i,:);
76                  % 时域特征提取部分
77                  B = squence_fft(i,:);
78                  % 频域
79                  % 时域
80                  feature_single = [feature_single,mean(A)];
81                  feature_single = [feature_single,std(A)];
82                  feature_single = [feature_single,std(A)/mean(A)];
83                  feature_single = [feature_single,min(A)];
84                  feature_single = [feature_single,max(A)];
85                  feature_single = [feature_single,max(A)-min(A)];
86                  feature_single = [feature_single,iqr(A)];
87                  feature_single = [feature_single,mean(abs(A).^2)];
88                  feature_single = [feature_single,mad(A)];
89                  % 绝对中值误差
90                  feature_single = [feature_single,entropy(A)];
91                  % 信息熵
92                  feature_single = [feature_single,mean(abs(A))];
93                  % 绝对值的平均值(整流平均值)
94                  feature_single = [feature_single,kurtosis(A)];
95                  % 峰度
```

```
96                  feature_single = [feature_single,skewness(A)];
97                  % 偏度
98                  feature_single = [feature_single,rms(A)/mean(abs(A))];
99                  % 波形因子
100                 feature_single = [feature_single,(max(A)-min(A))/rms(A)];
101                 % 峰值因子
102                 feature_single = [feature_single,(max(A)-min(A))/mean(abs(A))];
103                 % 脉冲因子
104                 feature_single = [feature_single,(max(A)-min(A))/mean(sqrt(abs(A)))^2];
105                 % 裕度因子
106             % 频域
107                 feature_single = [feature_single,mean(B)];
108                 feature_single = [feature_single,std(B)];
109                 feature_single = [feature_single,std(B)/mean(B)];
110                 feature_single = [feature_single,max(B)];
111                 feature_single = [feature_single,iqr(B)];
112                 feature_single = [feature_single,mean(abs(B).^2)];
113                 feature_single = [feature_single,mad(B)];
114                 % 绝对中值误差
115                 feature_single = [feature_single,entropy(B)];
116                 % 信息熵
117                 feature_single = [feature_single,mean(abs(B))];
118                 % 绝对值的平均值(整流平均值)
119                 feature_single = [feature_single,kurtosis(B)];
120                 % 峰度
121                 feature_single = [feature_single,skewness(B)];
122                 % 偏度
123                 feature_single = [feature_single,rms(B)/mean(abs(B))];
124                 % 波形因子
125                 feature_single = [feature_single,(max(B)-min(B))/rms(B)];
126                 % 峰值因子
127                 feature_single = [feature_single,(max(B)-min(B))/mean(abs(B))];
128                 % 脉冲因子
129                 feature_single = [feature_single,(max(B)-min(B))/mean(sqrt(abs(B)))^2];
130                 % 裕度因子
131             end
132             features = [features
133                         feature_single];
134         end
135         featuresall = [featuresall
136                        features ];
137     end
138     save(filename, 'featuresall');
139 end
```

filter_and_fft.m（带通滤波与快速傅里叶变换函数）

```
1  function [newdata,fftdata] = filter_and_fft(Wn,n,signal)
2      % Wn:  归一化截止频率
3      % n:  阶数
4      % signal:  需要滤波的信号
5      % 获取巴特沃斯滤波器参数
```

```
    [b,a] = butter(n,Wn,'bandpass');
    N = length(signal);
    % 滤波
    data = filtfilt(b,a,signal);
    % 快速傅里叶变换
    fftdata = fft(data);
    newdata = ifft(fftdata);
    fftdata = abs(fftdata(1:floor(N/2))*2/N);
end
```

low_pass_filter.m（低通滤波函数）

```
function data = low_pass_filter(Wn,n,signal)
    % Wn:  归一化截止频率
    % n:  阶数
    % signal:  需要滤波的信号
    % 获取巴特沃斯滤波器参数
    [b,a]=butter(n, Wn,'low');
    data = filtfilt(b,a,signal);
end
```

cat_features.m（合并特征）

```
% 将特征与标签整合为一个变量，方便后续操作
clc
clear
data = [];

label = load('predataset/label.mat').label;
%% 拼接特征，形成数据集
% for j=1:15
    % for i=1:3
        % filename = strcat(num2str(j),'_',num2str(i),'.mat');
        % dat = load(filename).features;
        % data = cat(1,data,dat);
    % end
% end
%
% label = repmat(label,1,45)';
% dataset = cat(2,data,label);
for i =1:15
    filename = strcat('anewdata',num2str(i),'.mat');
    dat = load(filename).featuresall;
    data = cat(1,data,dat);
end
label = repmat(label,1,45)';
dataset = cat(2,data,label);
```

vision.m（信号可视化）

```
%% 可视化部分
%
clc
clear
close all;
figure,
hold on
fs = 200;

dat1_1 = load('predataset/1_1.mat');
data = dat1_1.djc_eeg1;

% 通道1
% channel = data(1,1:37000);
channel = data(18,:);
% channel1 = data(1,:);
% channel21 = data(21,:);
% channel41 = data(41,:);
% channel61 = data(61,:);
% plot(channel1(1:40000));
% plot(channel21(1:40000));
% plot(channel41(1:40000));
% plot(channel61(1:40000));
% legend('channel1','channel21','channel41','channel61')
% x_data = 1:600;
% plot(channel(1:40000))
% channel = low_pass_filter(50*2/fs,4,channel);
% channel = resample(channel,fs,200);
% plot(channel(1:40000))
% legend('initial signal','filtered signal')
N = length(channel);
n = 0:N-1;
f = n*fs/N;

% 整体的快速傅里叶变换
channel_fft = fft(channel);
channel_fft = abs(channel_fft(1:floor(N/2)) * 2/N);
% 滤波，快速傅里叶变换
% 滤得delta波
[Δ_data,Δ_fft] = filter_and_fft([0.1*2 4*2]/fs,4,channel);
% 滤得theta波
[theta_data,theta_fft] = filter_and_fft([4*2 9*2]/fs,4,channel);
% 滤得alpha波
[alpha_data,alpha_fft] = filter_and_fft([8*2 12*2]/fs,4,channel);
[alpha_s_data,alpha_s_fft] = filter_and_fft([8*2 9*2]/fs,4,channel);
[alpha_m_data,alpha_m_fft] = filter_and_fft([9*2 12*2]/fs,4,channel);
[alpha_f_data,alpha_f_fft] = filter_and_fft([12*2 14*2]/fs,4,channel);
% 滤得beta波
[beta_data,beta_fft] = filter_and_fft([14*2 30*2]/fs,4,channel);
[beta_SMR_data,beta_SMR_fft] = filter_and_fft([12*2 16*2]/fs,4,channel);
[beta_sr_data,beta_sr_fft]   = filter_and_fft([16*2 20*2]/fs,4,channel);
[beta_hr_data,beta_hr_fft]   = filter_and_fft([20*2 30*2]/fs,4,channel);
```

```
% 滤得gamma波
[gamma_data,gamma_fft]   = filter_and_fft([30*2 50*2]/fs,4,channel);

% 时域
% plot(x_data,alpha_data(1501:2100))
% plot(x_data,alpha_s_data(1501:2100))
% plot(x_data,alpha_m_data(1501:2100))
% plot(x_data,alpha_f_data(1501:2100))
% plot(x_data,beta_data(1501:2100))
% plot(x_data,beta_SMR_data(1501:2100))
% plot(x_data,beta_sr_data(1501:2100))
% plot(x_data,beta_hr_data(1501:2100))
% 频域
% hold on
% legend('Initial FFT')
% xlim([5,40])
% plot(f(1:N/2),channel_fft)
% plot(f(1:N/2),delta_fft)
% plot(f(1:N/2),theta_fft)
plot(f(1:N/2),alpha_fft)
plot(f(1:N/2),alpha_s_fft)
plot(f(1:N/2),alpha_m_fft)
plot(f(1:N/2),alpha_f_fft)
% plot(f(1:N/2),beta_fft)
% plot(f(1:N/2),beta_SMR_fft)
% plot(f(1:N/2),beta_sr_fft)
% plot(f(1:N/2),beta_hr_fft)
% plot(f(1:N/2),gamma_fft)

% legend('initial signal','delta','theta','alpha','beta','gamma')
legend('alpha','alpha s','alpha m','alpha f')
% legend('beta','beta s','beta m','beta f')
```

entropy.m（计算信息熵函数）

```
% 计算信息熵
function signalentropy = entropy(channel)
    p = hist(channel(:),length(channel));% 根据信号长度得到直方图
    p = p/sum(p);
    i = find(p);
    signalentropy = -sum(p(i).*log2(p(i)));% 计算信源熵
end
```

4.4.2 python部分

algorithm.py（情绪识别核心算法）

```
import numpy as np
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn import linear_model
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier


def get_accuracy(label, pred):
  length = len(label)
  num = len(label)
  for i in range(length):
    if label[i] != pred[i]:
      num -= 1
  return num / length * 100


def get_pca_data(data, dimension):
  pca = PCA(n_components=dimension)  # reduce to dimension
  pca.fit(data)
  newdata = pca.fit_transform(data)  # reduced data
  return newdata


def get_lda_data(data, label, dimension):
  lda = LDA(n_components=dimension)
  lda_feature = lda.fit(data, label).transform(data)
  return lda_feature


def confmartix(trueLabel, predictLabel):
  confusion_array = confusion_matrix(trueLabel, predictLabel)
  return confusion_array


def kappa(matrix):
  n = np.sum(matrix)
  sum_po = 0
  sum_pe = 0
  for i in range(len(matrix[0])):
    sum_po += matrix[i][i]
    row = np.sum(matrix[i, :])
    col = np.sum(matrix[:, i])
    sum_pe += row * col
  po = sum_po / n
  pe = sum_pe / (n * n)
  # print(po, pe)
  return (po - pe) / (1 - pe)


```

```
def rf(train_data, train_label, test_data):
  clf = RandomForestClassifier(n_estimators=100)
  # train the model
  clf.fit(train_data, train_label)
  pred_label = clf.predict(test_data)
  return pred_label


def svm_algorithm(train_data, train_label, test_data):
  model = svm.SVC(kernel='linear', C=0.1)
  model.fit(train_data, train_label)
  pred_label = model.predict(test_data)
  return pred_label


def lr(train_data, train_label, test_data):
  lr = linear_model.LogisticRegression(solver='lbfgs')
  lr.fit(train_data, train_label)
  pred_label = lr.predict(test_data)
  return pred_label


def knn(train_data, train_label, test_data):
  knn = KNeighborsClassifier(n_neighbors=5)
  knn.fit(train_data, train_label)
  pred_label = knn.predict(test_data)
  return pred_label


def dnn(train_data, train_label, test_data):
  clf = MLPClassifier(solver='lbfgs', alpha=1e-2,
  hidden_layer_sizes=(64, 32), random_state=1)
  clf.fit(train_data, train_label)
  pred_label = clf.predict(test_data)
  return pred_label


def vision_confuse_matrix(matrix):
    sns.set()
  f, ax = plt.subplots()
  matrix = pd.DataFrame(matrix)
  matrix.columns = ['negative', 'neutral', 'positive']
  sns.heatmap(matrix, annot=True, ax=ax, cmap="RdPu")

  ax.set_title('confusion matrix')  # title
  ax.set_xlabel('Predict_Label')  # x-axis
  ax.set_ylabel('True_Label')  # y-axis
  plt.show()


if __name__ == '__main__':
  seeds = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50,
        55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
  acc_array = []
  fone_array = []
```

```
    kappa_array = []
    for seed in seeds:
        np.random.seed(seed)
        dataset = pd.read_csv('dataset_more_features.csv').values  # low
        # dataset = pd.read_csv('dataset_all_features.csv').values # all
        # dataset = pd.read_csv('new_features.csv').values # high
        np.random.shuffle(dataset)
        data, label = dataset[:, :-1], dataset[:, -1]
        data = get_lda_data(data, label, 2)
        # print(data.shape)
        # data = get_pca_data(data, 1000)
        train_data, test_data = data[0:int(675 * 0.7)], data[int(675 * 0.7):]
        train_label, test_label = label[0:int(675 * 0.7)], label[int(675 * 0.7):]

        # pred_label = rf(train_data, train_label, test_data) # rf
        # pred_label = dnn(train_data, train_label, test_data) # dnn
        # pred_label = knn(train_data, train_label, test_data) # knn
        pred_label = svm_algorithm(train_data, train_label, test_data)  # svm
        # pred_label = lr(train_data, train_label, test_data) # lr
        # calculate acc
        acc = get_accuracy(test_label, pred_label)
        acc_array.append(acc)
        # calculate confusion_matrix
        confarrary = confmartix(test_label, pred_label)
        # calculate kappa
        kappa_array.append(kappa(confarrary))
        ar = classification_report(test_label, pred_label, output_dict=True, target_names=[
        'negative', 'neutral', 'positive'])
        # calculate F1
        print((ar['negative']['f1-score'] + ar['positive']
        ['f1-score'] + ar['neutral']['f1-score']) / 3)
        fone_array.append((ar['negative']['f1-score'] + ar['positive']
        ['f1-score'] + ar['neutral']['f1-score']) / 3)
        print('accuracy: {0:.5f}\%'.format(acc))
        print('kappa: {0:.5f}\%'.format(kappa(confarrary)))
    fone_array = np.array(fone_array)
    acc_array = np.array(acc_array)
    kappa_array = np.array(kappa_array)
    confusion_arraysss = np.round(confusion_arraysss / 21)
    print('-------------------------------------------')
    print('mean accuracy: {0:.5f}\%'.format(np.mean(acc_array)))
    print('mean kappa: {0:.5f}\%'.format(np.mean(kappa_array)))
    print('mean f1: {0:.5f}\%'.format(np.mean(fone_array)))
    print(confusion_arraysss)
    vision_confuse_matrix(confusion_arraysss)
```

deal.py（特征裁剪）

```
import numpy as np
import pandas as pd


```

```
dataset = pd.read_csv('dataset_all_features.csv').values
print(dataset[0].shape)
newdataset = [[[]for i in range(9920)]for i in range(675)]
for k in range(675):
for j in range(62):
for i in range(160):
newdataset[k][i + j * 160] = dataset[k][224 + j * 384 + i]
# newdataset[k][i + j * 160] = 1
newdataset[k] = np.array(newdataset[k])
print(newdataset[i])
dfdata = pd.DataFrame(newdataset)
datapath1 = 'new_features.csv'
dfdata.to_csv(datapath1, index=False)
```

mat_to_csv.py（转.mat文件为.csv）

```
import pandas as pd
import scipy
from scipy import io
features_struct = scipy.io.loadmat('all_features.mat')
features = features_struct['dataset']
dfdata = pd.DataFrame(features)
datapath1 = 'dataset_all_features.csv'
dfdata.to_csv(datapath1, index=False)
```

参考文献

[1] Wei-Long Zheng, Hao-Tian Guo, and Bao-Liang Lu. Revealing critical channels and frequency bands for emotion recognition from eeg with deep belief network. In *2015 7th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 154–157. IEEE, 2015.

[2] 张克军, 韩娜, 陈欣然, and 缪睿. 基于脑电图的新型情绪识别特征提取方法. 华南师范大学学报 *(*自然科学版*)*, 51(5):6–11, 2019.

[3] 陈泽龙 and 谢康宁. 脑电 eeg 信号的分析分类方法. 中国医学装备大会暨 *2019* 医学装备展览会论文汇编, 2019.