人机交互课程设计实验报告

目录

一 课设概述

实验数据由16名视力正常或矫正至正常的健康受试者在一项持续注意力驾驶实验中使用32通道神经扫描系统（30通道EEG加上2通道耳垂）以500Hz的采样率记录参与者的头皮EEG信号，每隔5-10秒随机出现一次车道偏移事件，记录对实验者对扰动的反应时间，然后转换为嗜睡指数(即给出的取值在0与1之间的resTime）。
课设要求：1.处理原始EEG数据并划分trial。2.构建模型学习train_data文件夹中的数据与驾驶疲劳度，以估计test_data文件夹中数据对应的驾驶疲劳度

二 总体方案设计

1.trail的划分

在处理的过程中依据idsNaN文件中0的索引产生相应数据的时间窗口的索引，并对其进行处理。

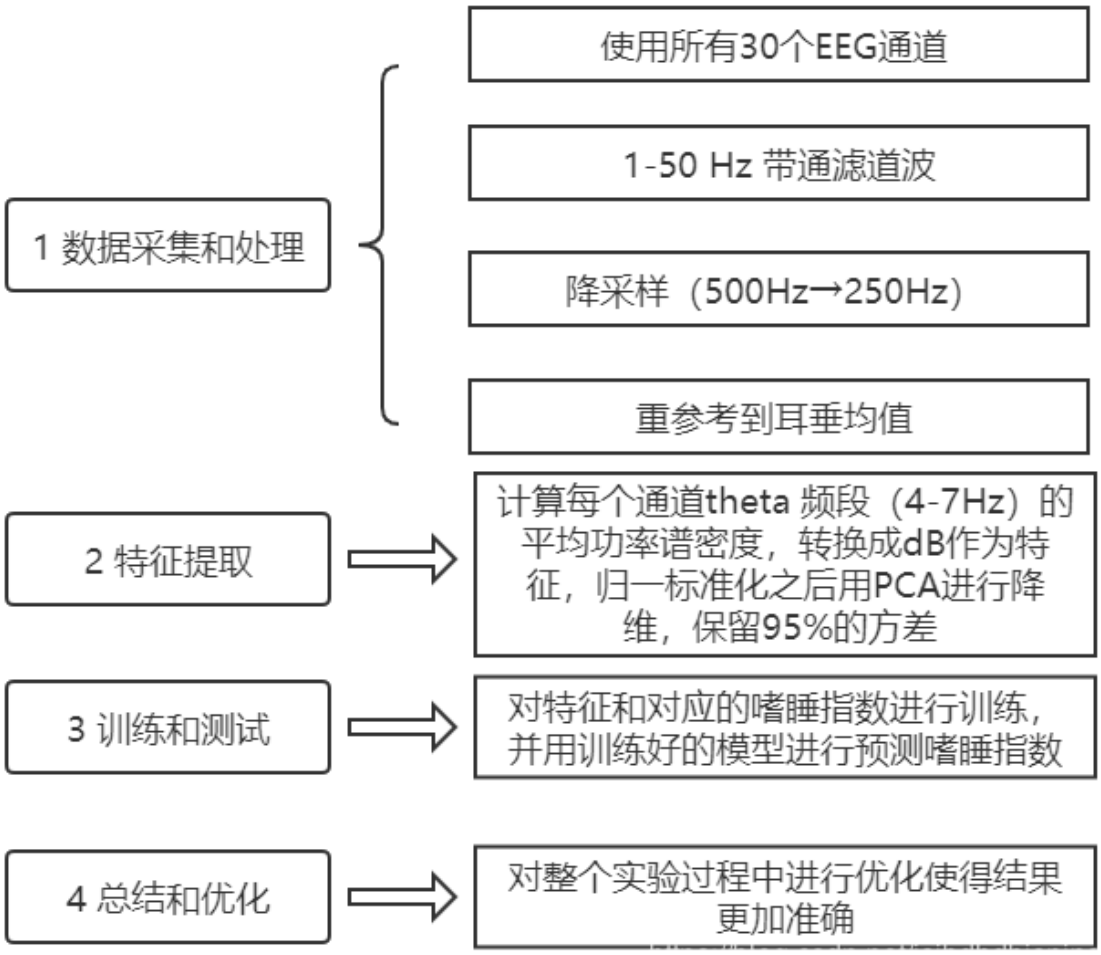
2.实验工具及环境

使用python语言实现，借助pycharm及以下相关库

```
import math
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy.io as scio
from scipy import signal
from scipy.signal import butter, lfilter
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso,LassoCV,LassoLarsCV
```

3.实验方案设计

对所有30个通道的采样数据进行1-50Hz带通滤波，原始采样频率降采样至250Hz，重参考到耳垂通道，提取每次扰动之前30秒EEG数据，计算每个通道theta 频段（4-7Hz）的平均功率谱密度，转换成dB作为特征，对特征和对应的嗜睡指数进行训练，并用训练好的模型进行预测嗜睡指数，最后对实验进行改进。



> 此实验中原始信号已经重参考到耳垂均值，并且已经降采样至250Hz

三 关键技术

1.带通滤波

由于在脑电信号的采集过程中，会受到周围噪声的干扰以及工频噪声(50Hz)的影响，因此，需要首先对原始信号进行噪声的滤除，通过利用带通滤波器，将1Hz - 50Hz频率之间的信号取出。本实验使用使用巴特沃斯带通滤波器进行信号的滤波。

2.平均功率谱密度（PSD）

功率谱密度是指用密度的概念表示信号功率在各频率点的分布情况，是对随机变量均方值的量度，本实验中计算每个通道theta 频段（4-7Hz）的平均功率谱密度，转换成dB作为特征，即一个输入变量有30维特征来描述，对应于一个resTime值

3.主成分分析（PCA)

PCA是一种分析、简化数据集的技术。主成分分析经常用于减少数据集的维数，同时保持数据集中的对方差贡献最大的特征，本实验中先对数据进行归一标准化使得均值为0方差为1，再读取30个特征对方差的贡献，我们发现前6个特征的方差已经达到95%，所以选择降维之后的维数为6.

4.LASSO回归

LASSO回归的目标函数在一般的线性回归的基础上加入了正则项，在保证最佳拟合误差的同时，使得参数尽可能的“简单”，使得模型的泛化能力强。正则项一般采用一范数，使得模型更具有泛化性，同时可以解决线性回归中不可逆情况。本实验一开始选用多元线性回归，发现实验结果中绝大多数值都近似相等，最终选择泛化能力更强的LASSO回归之后效果明显更佳。

四 源程序设计

本实验共包含三个函数，分别为计算功率谱密度，计算用于训练的输入输出参数，计算用于测试的输入参数。

1.计算功率谱密度

输入参数：待处理的数据（7500*30的二维矩阵，即30s时间窗口内30个通道的采样数据）
输出参数：各个通道theta频段平均功率谱密度（含30个值的列表）

```
def compute_power_spectral_density(windowed_signal):
    ret,db_list = [],[]
    windowed_signal = list(map(list, zip(*
windowed_signal)))
    PSD_FREQ = np.array([[4, 8]])
    # Welch parameters
    sliding_window = 7500
    SAMPLING_FREQUENCY = 250
    overlap = 0.667 #每个30s的窗口有20s的重叠 20/30=0.667
    n_overlap = int(sliding_window * overlap)
    # compute psd using Welch method
    freqs, power = signal.welch(windowed_signal,
fs=SAMPLING_FREQUENCY,
                                nperseg=sliding_window,
noverlap=n_overlap)
    for psd_freq in PSD_FREQ:
        tmp = (freqs >= psd_freq[0]) & (freqs <
psd_freq[1])
        a_ndarray=power[:, tmp].mean(1)
        h_list=a_ndarray.tolist()
        for i in h_list:
            db_list.append(10*math.log(i,10))
        ret.append(h_list)
    return ret
```

2.计算用于训练的输入输出参数

输入参数：数据文件的路径和idsNaN文件的路径
输出参数：用于训练的train_x和train_y（train_x为二维矩阵，列数为6，即PCA降维之后的维数，train_y为一个列矩阵）

```
def get_trainxy(datapath,idspath):
    data = scio.loadmat(datapath)
    idsdata = scio.loadmat(idspath)
    train_x,train_y,filtedData,delete_channels=[],[],[],[]
    #带通滤波
    for i in range(30):
```

```
        b, a = signal.butter(8, [0.008, 0.4], 'bandpass')
        onechannel = signal.filtfilt(b, a,
data['eeg_data'][i]) #1-50Hz滤波
        filtedData.append(onechannel)
    filtedData_t = list(map(list, zip(* filtedData)))
 #100000*30
    for j in range(len(idsdata['idsNaN'])):
        x = filtedData_t[j * 2500:j * 2500 + 7500]
        res = compute_power_spectral_density(x)
        if idsdata['idsNaN'][j]==0 :    #依此来划分trial
            train_x.append(res[0])
    for i in range(len(data['resTime'])):
            train_y.append([data['resTime'][i][0]])
    #PCA
    x = StandardScaler().fit_transform(train_x)
    pca = PCA(n_components=6)
    principalComponents = pca.fit_transform(x)
    train_x = principalComponents.tolist()
    return train_x, train_y
```

3.计算用于测试的输入参数

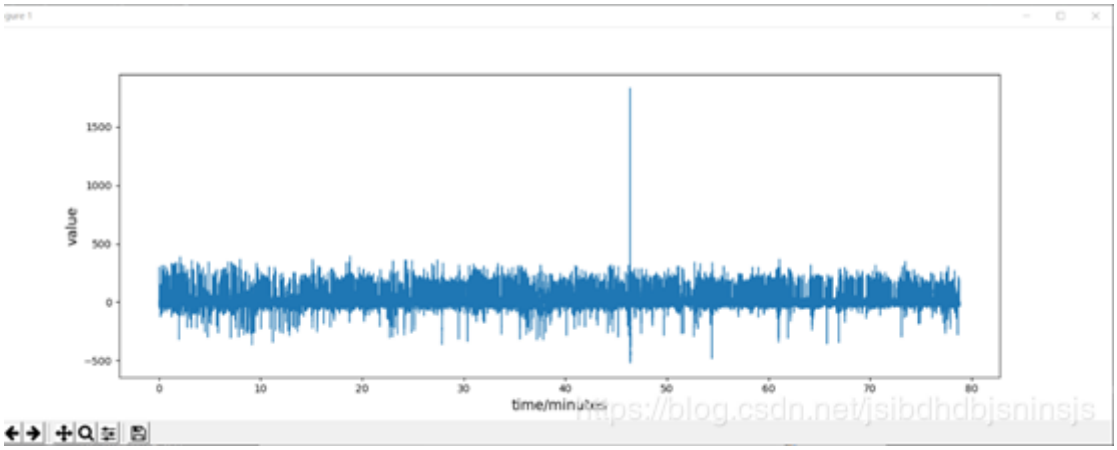
此函数与函数2实现过程相同，区别在于不输出train_y

```
def get_testx(datapath,idspath):
    data = scio.loadmat(datapath)
    idsdata = scio.loadmat(idspath)
    train_x,train_y,filtedData,delete_channels=[],[],[],[]
    for i in range(30):
        b, a = signal.butter(8, [0.008, 0.4], 'bandpass')
        onechannel = signal.filtfilt(b, a,
data['eeg_data'][i]) #1-50Hz滤波
        filtedData.append(onechannel)
    filtedData_t = list(map(list, zip(* filtedData)))
 #100000*30
    for j in range(len(idsdata['idsNaN'])):
        x = filtedData_t[j * 2500:j * 2500 + 7500]
        res = compute_power_spectral_density(x)
        if idsdata['idsNaN'][j]==0 :  #依此来划分trial
            train_x.append(res[0])
    x = StandardScaler().fit_transform(train_x)
```

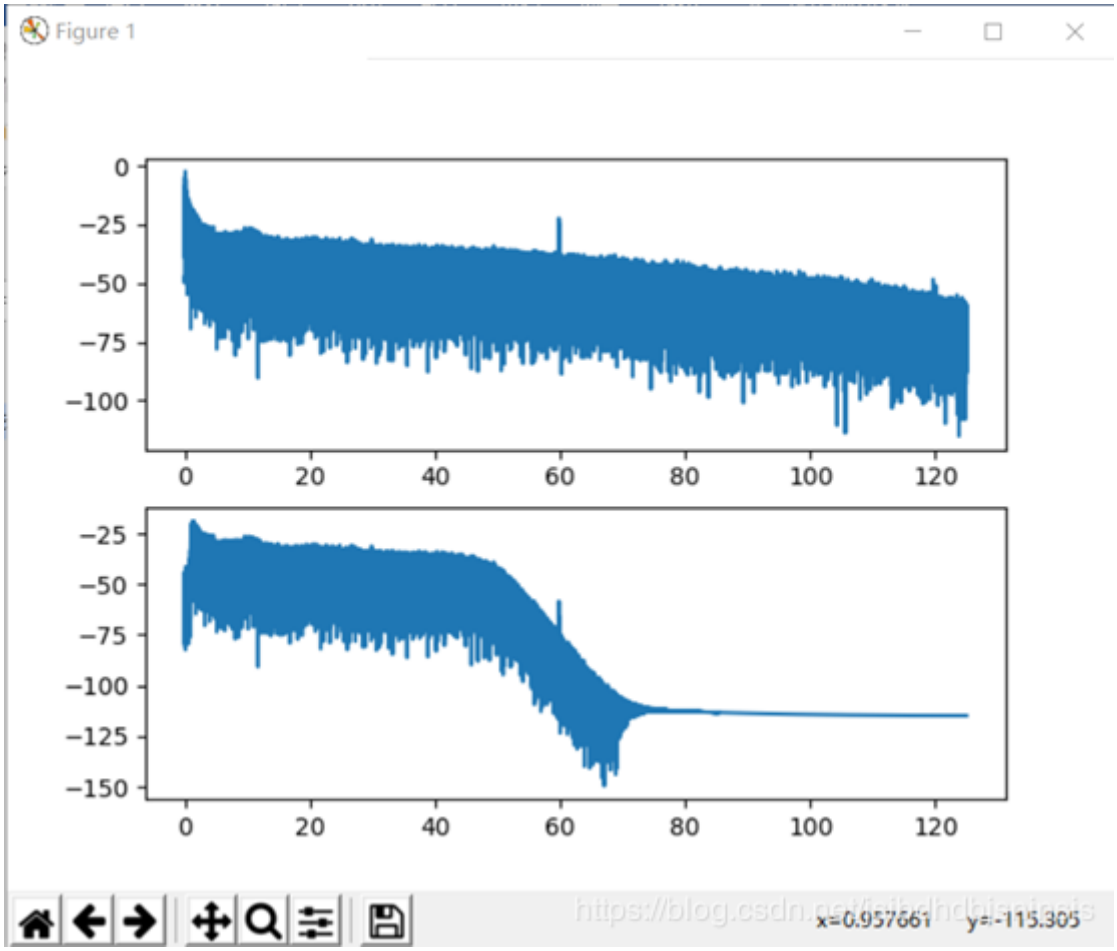
```
    pca = PCA(n_components=6)
    principalComponents = pca.fit_transform(x)
    train_x = principalComponents.tolist()
    return train_x
```

五 实验结果及评价

- 原始数据可视化



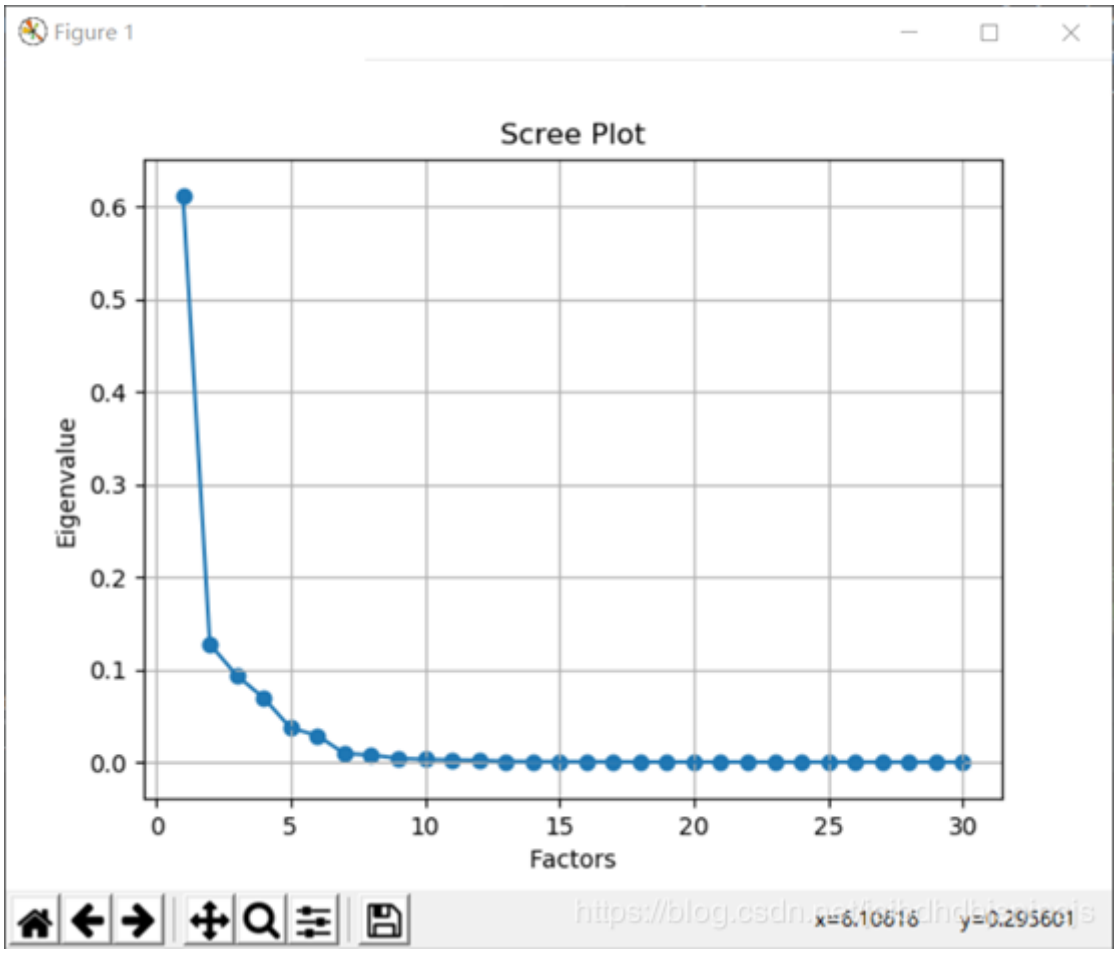
- 带通滤波前后数据频域比较



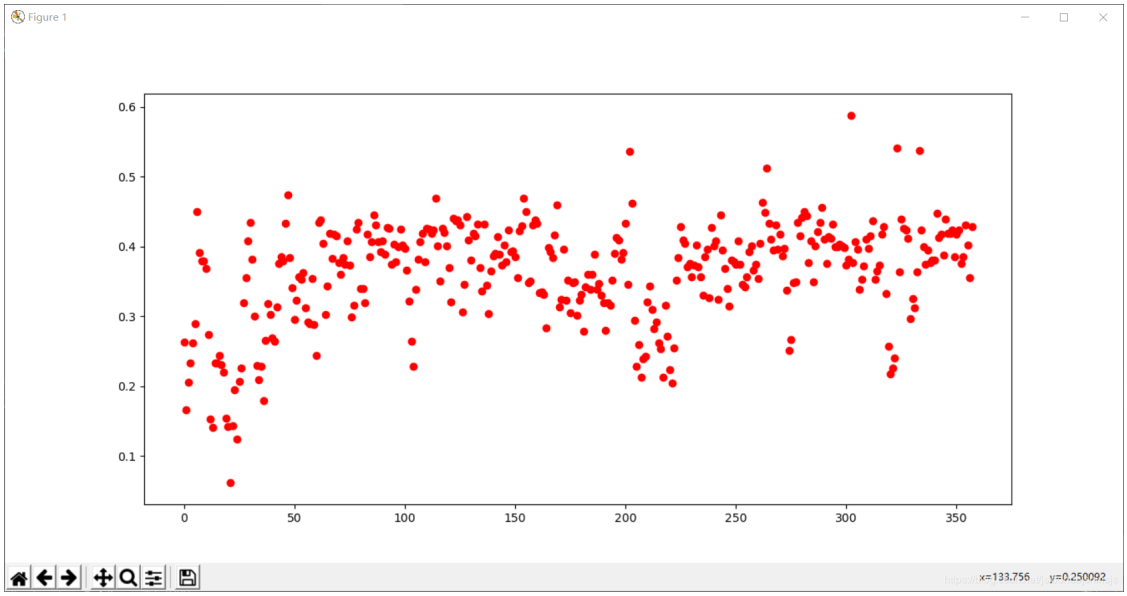
- PCA选择降维后的维数，观察降到多少维度比较合适，下面是30个变量方差的占比，（以070207.mat文件数据为例）

```
[6.12147275e-01 1.27455588e-01 9.35988026e-02 6.93650437e-02
 3.69438872e-02 2.86664876e-02 9.39269842e-03 7.56115944e-03
 4.13183513e-03 3.00002160e-03 2.42237989e-03 1.97620604e-03
 8.03998120e-04 6.22534334e-04 4.87300295e-04 4.11573605e-04
 3.79631782e-04 1.97863260e-04 1.17693824e-04 1.02684093e-04
 7.70789066e-05 6.07784666e-05 2.81409437e-05 2.39349680e-05
 1.37906108e-05 6.97300211e-06 1.79814370e-06 1.62246200e-06
 1.21546755e-06 3.11794521e-09]
```

- 更加直观的可视化，可以看出前6个变量所占比已经超过95%



- 最终预测结果的可视化



六 总结与体会

先说一下实验中存在的问题，通过参考老师的论文，论文中提到对于最大值大于20dB的通道应该去除掉，我一开始担心可能某一个resTime对应的特征变为29个，有的对应30 维，维度不一致，PCA之后可能会破坏它的信息，最终发现这种顾虑是多余的，但是在实现的过程中，发现有很多通道都需要去除，最终也是没能找到问题的根源，我又想到用纵向删除的方法来做，即舍弃该窗口的数据和对应的resTime，即在训练集中减少部分，但是发现这样无法保证对训练集和测试集的操作保持一致，因为有的训练集数据会被去除二根本无法得到对应的resTime值，最终没有使用这一策略，所以本次的实验还是有一定的改进空间。

之前的课程设计都是图像，视频相关的数据，更加侧重的是预处理这一块，最终的追踪识别也和机器学习的回归分类这些算法大不相同，或者是简单的坐标点集，又缺少数据的预处理，而本次课设系统的将数据预处理和回归预测模型结合在一起，使我受益匪浅，而且通过实验过程中不断的查阅资料和文献，我对脑电信号处理相关的知识有了更透彻的了解。

实验过程中能够将课件上或者论文中学习到的知识复现是件很开心的事情，专注于算法速度的提升也是件很纯粹并且很有乐趣的事情。总的说来实验的过程很有挑战性但也极其有趣味性。并且通过此过程我们更加熟悉了python的使用，对脑电信号有了更深刻的认识，在实现课程上学习到的算法时也加深了我们对算法的理解。

七 参考文献

1.D.Wu,V.Lawhern,S.Gordon,B.Lance and C-T Lin,"Driver Drowsiness Estimation from EEG Signals Using Online Weighted Adaptation Regularization for Regression(OwARR)," IEEE Transactions on Fuzzy Systems,25(6),pp.1522-1535,2017.