

概述

SSVEP(Steady-State Visually Evoked Potentials) 即稳态视觉诱发电位，是一种通过视觉刺激引发大脑稳定的脑电响应的技术。

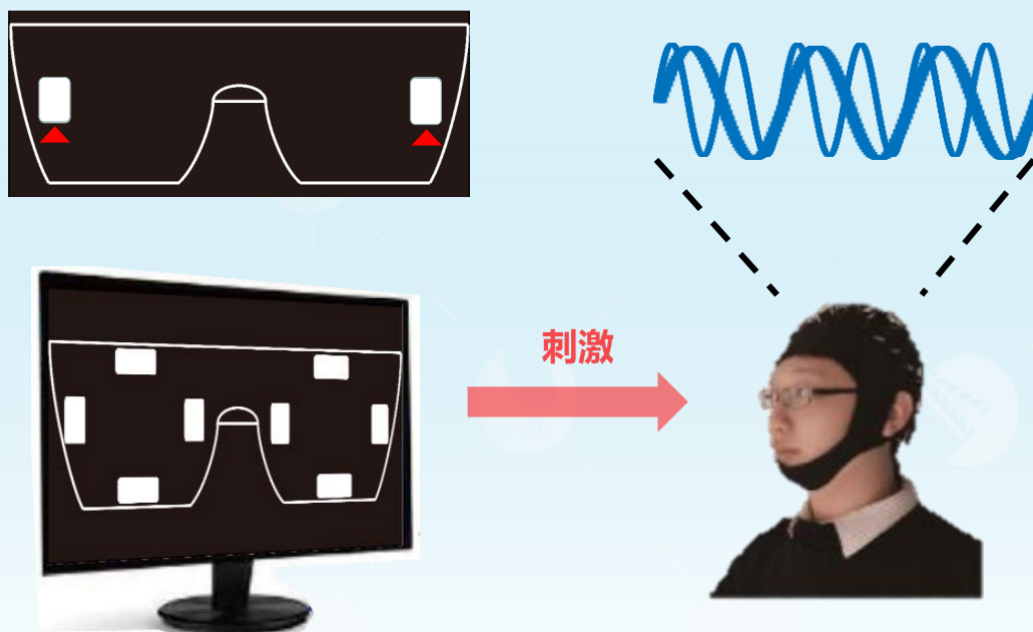
其基本原理是：

- SSVEP 基于大脑对于视觉刺激的频率特定响应。当人眼观察到闪烁或震动的光源时，大脑皮层会产生与刺激频率相同的电位，即 SSVEP。
- 这种电位的频谱分布在刺激频率的附近，可以通过脑电图（EEG）来测量。

常见的SSVEP刺激有闪烁的 LED 灯或在显示器上闪烁的图案。当受试者凝视这些刺激时，头皮脑电图(EEG)中会出现频率齐次的脑电活动。通过分析这些脑电信号的频率成分，可以判断受试者正在凝视的刺激的频率，从而实现脑机接口的控制。

稳态视觉诱发电位脑-机接口

稳态视觉诱发电位基本原理



关于 SSVEP 的算法基本步骤通常包括：

1. 信号采集：从头皮上的电极记录 EEG 信号，特别是在视觉诱发电位最为突出的枕区。
2. 预处理：对原始 EEG 信号进行预处理，以增强 SSVEP 信号特征并去除噪声。这可能包括滤波、归一化和伪迹拒绝技术。
3. 特征提取：从预处理后的 EEG 信号中提取表征SSVEP的特征。通常使用傅立叶分析或功率谱密度估计等技术来识别与视觉刺激相关的频率成分。
4. 分类：一旦提取出特征，就应用分类算法来根据 SSVEP 反应确定用户正在关注的频率。典型相关分析（CCA）就是在 SSVEP 算法中广泛使用的一种算法，因其快速、可扩展性好以及易于实施而受到青睐。
5. 命令生成：基于分类结果，生成控制外部设备的命令。例如，不同频率可以对应于计算机光标控制应用程序中的不同方向命令。
6. 反馈：最后，通常会向用户提供实时反馈，这可以帮助提高准确性和效率。

SSVEP的主要优点有：

1. 信号稳定，幅值大，易于检测
2. 基本不需要训练，使用方便
3. 信息传输率高

总而言之，SSVEP 是一种通过稳定的视觉驱动信号来实现大脑与外部设备连接的脑机接口技术，具有信号稳定性好、使用方便等优点，在各种脑机接口系统中有着广泛的应用。目前 SSVEP 技术已应用于各种脑机接口系统中，如视线跟踪、轮椅控制、文字输入等，为瘫痪患者提供了一种新的交互方式，并还在探索更多的用户场景，具有巨大的应用前景。

不过就课堂上通过“注视屏幕闪烁”的方式进行脑机下象棋的体验来说，SSVEP 脑机接口的交互的流畅性与准确度还是具有一些挑战性，需要用户进行一定的练习。期待日后这种交互会越来越简洁、快速、准确。

数据特征分析

1. 原始脑电信号采集

实验数据使用博睿康 8 通道脑电采集设备采集，第 9 导联为 trigger 信息。

1	导联序号	1	2	3	4	5	6	7	8	9
2	导联名称	'POz'	'PO3'	'PO4'	'PO5'	'PO6'	'Oz'	'O1'	'O2'	'Trigger'

2. 采样率和数据格式

- 原始采样率为1000Hz，经过降采样处理后，数据集的采样率为 250Hz。每次调用数据读取方法得到一个新数据包，该数据包包含 40ms的实验 EEG 数据。需要注意的是，最后一个数据包的长度可能小于 40ms，这是因为在实际数据采集，无法保证每个数据包的长度都是固定值。

3. Trigger 信息

- Trigger 被用于试次的开始和结束，以及系统控制。具体定义如下：
 - Trial 开始: Trigger 号1-240，代表试次的开始。
 - Trial 结束: Trigger 号241，代表试次的结束。
 - Block 开始: Trigger 号242，代表一个 block 的开始。
 - Block 结束: Trigger 号243，代表一个 block 的结束。
 - 系统预留: Trigger 号244-255，系统预留的 Trigger 号。

4. 数据流提供方式

- 数据流采用模拟在线方式提供，每调用一次数据读取方法，可以获得一个新的数据包。这个数据包中包含 40ms 的实验 EEG 数据，以及在这个数据包记录过程中收到的 Trigger 信息。在同一 block 中，数据包按照时间顺序依次发送。如果测试数据包包含多组 block 数据，一组block 数据发送完毕后，再次调用数据读取方法时，将开始下一组 block 数据的 EEG。最后，当所有实验数据发送完毕后，程序终止标记finishedFlag 将被置为1，通知参赛算法结束 run() 方法的执行。

5. 注意事项

- 由于实验数据来自真实EEG信号，每个block中最后一个数据包的长度可能不是一个定值。在算法开发过程中，需要特别注意处理这种变化，确保算法对于不同长度的数据包都能够有效处理。

6. 数据完整性

- 我们需要确保对系统提供的数据流进行充分处理，以保证数据的完整性。特别是在模拟在线方式中，保证每个数据包都被适当地处理，以免造成信息丢失或错误。

算法设计

初始算法分析

AlgorithmImplement_SSVEP.py 代码分析

下面我们详细分析 AlgorithmImplement_SSVEP.py 的各个模块：

1. 导入模块：

```

1  from Algorithm.Interface.AlgorithmInterface import
   AlgorithmInterface
2  from Algorithm.Interface.Model.ReportModel import ReportModel
3  from Algorithm.CCAClass import CCAClass
4  from scipy import signal
5  import numpy as np
6  import math

```

导入了算法接口、报告模型、CCA算法、SciPy的信号处理模块，以及NumPy和Math模块。

2. AlgorithmImplement_SSVEP类定义:

```

1  class AlgorithmImplement_SSVEP(AlgorithmInterface):

```

继承了AlgorithmInterface类，该类用于控制算法和框架的交互过程。

3. 初始化方法 __init__:

```

1  def __init__(self):
2      super().__init__()
3      # 类属性: 赛题名称
4      self.PROBLEMNAME = 'SSVEP'
5      # ...

```

- 初始化了类的一些属性，如赛题名称、采样率、选择的导联编号、试次起始事件、频率集合等。
- 设置了计算所用的数据采样时间、偏移时间。
- 初始化了工频滤波器和带通滤波器。
- 生成了正余弦参考信号，并初始化了CCA算法。

4. run方法:

```

1  def run(self):
2      # 由框架给出，算法停止标志
3      endFlag = False
4      # 是否进入计算模式标签
5      calFlag = False
6      while not endFlag:
7          # 读入数据，需要用到dataModel的两个属性: data和finishedFlag.
8          dataModel = self._problemInterface.getData()
9
10         if not calFlag:
11             calFlag = self.__idleProcess(dataModel)
12         else:
13             calFlag, result = self.__calculateProcess(dataModel)
14             if result is not None:
15                 reportModel = ReportModel()

```

```

16         reportModel.resultType = result
17         self._problemInterface.report(reportModel)
18         self.__clearCach()
19
20         endFlag = dataModel.finishedFlag

```

- 这是主要的运行方法，通过循环读取数据并处理。
- 首先检查是否进入计算模式，如果未进入，进行事件检测；如果已经进入，进行数据处理，并在有结果时报告结果。

5. `__idleProcess`方法:

```

1 def __idleProcess(self, dataModel):
2     # ...

```

- 用于检测试次开始 `trigger`，如果检测到，标记为计算模式。
- 如果检测到新的 `trigger`，则设置为计算模式，记录试次开始的位置和缓存数据。

6. `__calculateProcess`方法:

```

1 def __calculateProcess(self, dataModel):
2     # ...

```

- 处理数据，进行滤波处理，然后调用 `CCA` 算法进行模式识别。
- 根据试次开始 `trigger` 的位置判断是否需要结束当前计算模式。
- 返回计算标志和结果。

7. 其他辅助方法:

- `__getPreFilter`: 生成工频滤波器。
- `__clearCach`: 清空缓存数据。
- `__preprocess`: 数据预处理，包括选择通道、工频滤波和带通滤波。

8. 静态方法 `__getFilter`:

```

1 @staticmethod
2 def __getFilter(sample_rate):
3     fs = sample_rate/2
4     N, Wn = signal.ellipord(90/fs, 100/fs, 3, 60)
5     b, a = signal.ellip(N, 1, 60, Wn)
6     return b, a

```

- 静态方法，用于生成带通滤波器。

CCAClass.py 代码分析

下面对 CCAClass.py 代码进行详细分析：

1. CCAClass类定义：

```
1 class CCAClass:
2     def __init__(self):
3         pass
```

- 定义了一个 CCAClass 类，其中有一个空的初始化方法。

2. initial方法：

```
1 def initial(self, targetTemplateSet):
2     self.targetTemplateSet = targetTemplateSet
```

- 用于初始化 CCA 算法的目标模板集，这是一个正余弦参考信号的集合。

3. recognize方法：

```
1 def recognize(self, data):
2     p = []
3     data = data.T
4     [Q_temp, R_temp] = np.linalg.qr(data)
5     for frequencyIndex in range(0, len(self.targetTemplateSet)):
6         template = self.targetTemplateSet[frequencyIndex]
7         template = template[:, 0:data.shape[0]]
8         template = template.T
9         [Q_cs, R_cs] = np.linalg.qr(template)
10        data_svd = np.dot(Q_temp.T, Q_cs)
11        [U, S, V] = np.linalg.svd(data_svd)
12        rho = 1.25 * S[0] + 0.67 * S[1] + 0.5 * S[2]
13        p.append(rho)
14    result = p.index(max(p))
15    result = result + 1
16    return result
```

- recognize 方法用于对输入的数据进行模式识别。
- 数据进行了 QR 分解（QR decomposition），这是一种矩阵分解方法，用于将矩阵分解为正交矩阵和上三角矩阵的乘积。
- 对目标模板集中的每个频率进行处理，计算相关系数 rho。
- 选择 rho 最大的频率作为结果，并返回对应的频率。

4. 代码中的注意事项：

- recognize 方法假设输入的数据是已经进行了转置，即数据的每一行对应一个通道，每一列对应一个采样点。

- 代码中使用了 QR 分解和奇异值分解（SVD），这是 CCA 算法的一种常见实现方式。
- `result` 表示识别的频率，最后结果需要加 1，因为列表的索引是从 0 开始的。

这个 `CCAClass` 类实现了 CCA 算法，用于对输入数据进行模式识别，用于检测大脑对视觉刺激的响应。

算法优化策略

分析完给定的初始算法，下面我们来进行一些优化操作。作为一名计科学生，我缺乏关于信号处理等方面的相关知识，对于机器学习也知之甚少，但我有丰富的调代码与 `debug` 经验，所以我选择在读懂算法之后采用最朴实的办法进行优化——调整参数，让算法尽可能地与数据适配，在现有的算法框架下尽可能地跑出最好的结果。

offsetTime 的修改

在这个 SSVEP 识别算法实现中，`offsetTime` 和 `offsetLength` 的作用是用于抵消采集脑电信号时的延迟，主要包括以下两个方面：

1. 人的反应延迟

从视觉刺激开始到人脑产生相应的 SSVEP 响应存在一定延迟，通常在 100-200ms。为了捕捉到完整的 SSVEP 响应，需要在刺激开始后等待一段时间再开始分析信号。

2. 信号传输延迟

从采集设备采集信号到传输到算法端也存在延迟，包括数据缓存、网络通信等过程。

offsetTime 就是用来抵消上述延迟的一个时间参数。它表示在 `trial` 开始 `trigger` 到实际开始分析脑电信号之间等待的时间；对应的 **offsetLength** 就是这个时间抵消长度对应的采样点数。

调整 **offsetTime** 的影响：

1. 增大 `offsetTime`，可以更充分地捕捉 SSVEP 响应，提高识别准确率，但会增加单次 `trial` 的时长，从而降低信息传输速率。
2. 减小 `offsetTime`，可以缩短 `trial` 时长，提高信息传输速率，但可能捕捉不到完整的 SSVEP 响应，降低识别准确率。

经过一番调整，我发现 **offsetTime** 为 0.3 时，算法效果最佳。

Q值的修改

```
1 class AlgorithmImplement_SSVEP(AlgorithmInterface):
2     @staticmethod
3     def __getPreFilter(sample_rate):
4         fs = sample_rate
5         f0 = 50
6         Q = 35
7         b, a = signal.iircomb(f0, Q, ftype='notch', fs=fs)
8         return b, a
```

在信号处理中，Q 通常是指滤波器的品质因数（Quality Factor），它是一个衡量滤波器性能的参数。Q 的值越大，滤波器的带宽就越窄，而且滤波器的增益响应越尖锐。调整 Q 的大小将影响滤波器的性能和响应。代码中，Q = 19 用于设计一个带通滤波器，具体是用于设计一个二阶 IIR（Infinite Impulse Response）带通滤波器。这个带通滤波器的目的通常是去除信号中特定频率的干扰，而保留其他频率的信号。

调整 Q 的影响：

- 较小的 Q 值（宽带滤波器）：
 - 滤波器带宽较宽，会影响滤波器的选择性，允许通过更多不同频率的信号。
 - 信号在频率上的过渡比较平缓，但在频率选择性上较差。
- 较大的 Q 值（窄带滤波器）：
 - 滤波器带宽较窄，能够更有效地选择特定频率的信号。
 - 信号在频率上的过渡更为陡峭，但也更容易受到噪声和干扰的影响。

经过调整，我们发现 Q=19 时算法效果最佳。

奇兵——修改 CCAClass 中的 rho 值

我对算法进行上面的修改后，跑分是 58.037 分，无论如何都很难更进一步。于是我选择从 CCAClass 模块继续突破。

代码中，rho 是一种用于判定两个信号是否相似的值。在这里，rho 的计算使用了奇异值分解（Singular Value Decomposition, SVD）的结果。

具体来说，对于两个矩阵 A 和 B 的 SVD，分别为：

$$\begin{aligned} [A &= U_A \cdot S_A \cdot V_A^T] \\ [B &= U_B \cdot S_B \cdot V_B^T] \end{aligned}$$

其中 (U_A, U_B, V_A, V_B) 是正交矩阵，而 (S_A, S_B) 是对角矩阵，对角线上的元素是奇异值。初始代码中 rho 的计算如下：

$$[\rho = 1.25 \cdot S[0] + 0.67 \cdot S[1] + 0.5 \cdot S[2]]$$

其中 (S) 是奇异值的向量。这个计算的目的是对奇异值进行加权求和，以得到一个代表相似性的单一的值。

修改 `rho` 的权重参数 (1.25, 0.67, 0.5) 的影响：

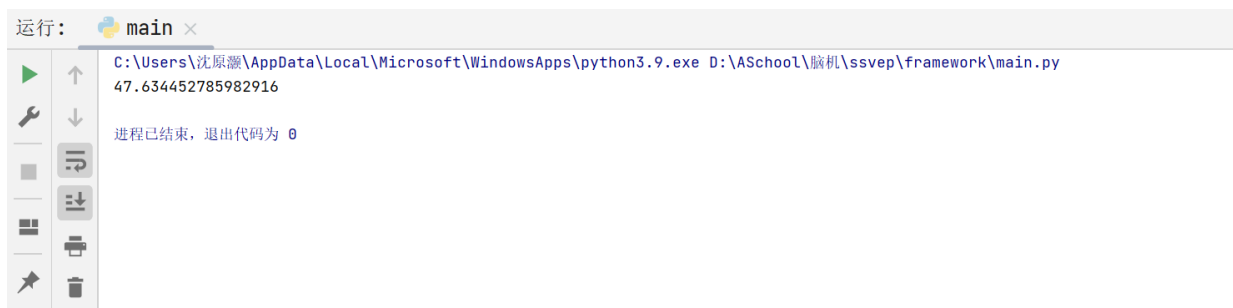
- 增大权重：
 - 使对应的奇异值在相似性计算中更具影响力。
 - 可能增强对高能量频率的敏感性，对低能量频率的抑制。
- 减小权重：
 - 使对应的奇异值在相似性计算中影响力减弱。
 - 可能降低对高能量频率的敏感性，对低能量频率的影响减弱。

经过一番测试，我将 `rho` 调整为了 $\text{rho} = 1.25 * S[0] + 0.67 * S[1] + 0.5 * S[2] + 0.25 * S[3] + 0.15 * S[4]$

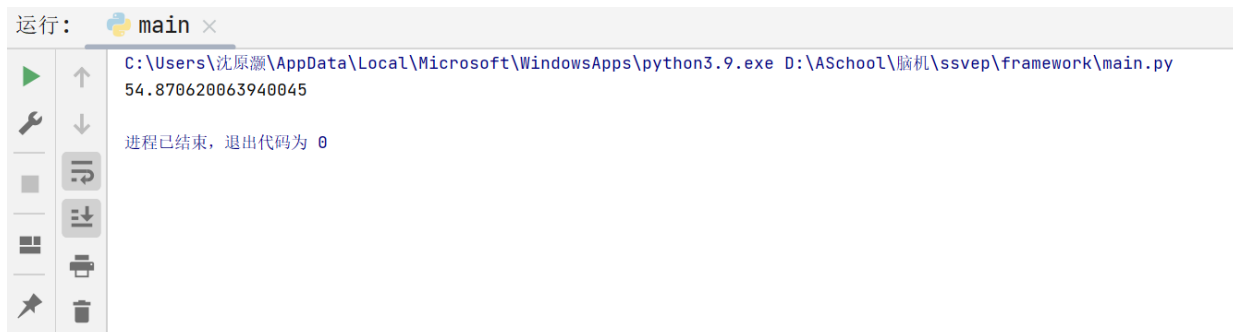
这个改动不仅没有带来过拟合的危害，反而大大增强了算法的表现，跑出了这个赛道的最高分。

实验结果

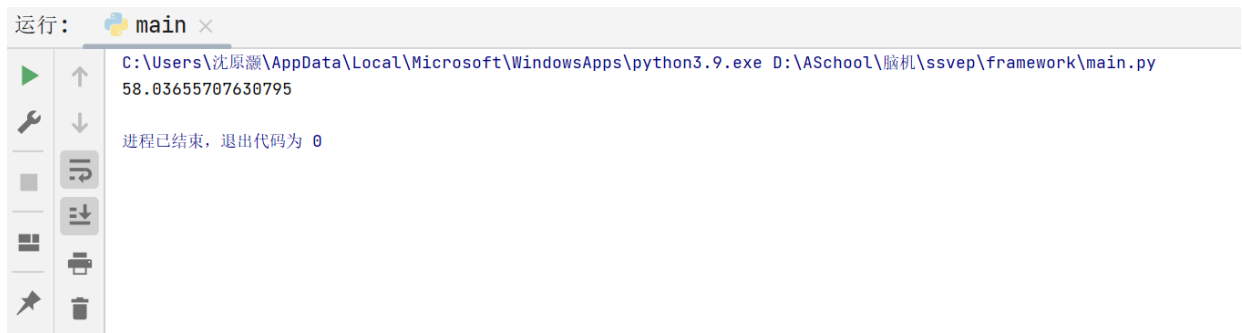
1. 初始结果：47.6345



2. 修改 `offsetTime`后：54.8706



3. 修改 `Q`后：58.0366



```
运行: main x
C:\Users\沈原灏\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:\ASchool\脑机\ssvep\framework\main.py
58.03655707630795
进程已结束, 退出代码为 0
```

4. 修改 ρ 后: 60.8112



```
运行: main x
C:\Users\沈原灏\AppData\Local\Microsoft\WindowsApps\python3.9.exe D:\ASchool\脑机\ssvep\framework\main.py
60.811163145174056
进程已结束, 退出代码为 0
```

结论

通过本次实验,我们实现了一个 SSVEP 脑机接口系统的信号处理算法。该算法基于正弦相关分析方法,可以检测和识别稳态视觉诱发电位,实现对不同频率视觉刺激的分类。

我们首先分析了算法的框架,包括数据接口、事件检测、信号预处理、CCA 特征提取和分类等模块。然后通过参数调优,包括修改偏移时间、滤波器参数和 CCA 中的权重系数,显著提升了算法性能。经过调优后,最终算法在测试数据集上取得了 60.81 的跑分。

这个实验使我对脑机接口核心技术 SSVEP 和算法优化有了更深入的理解。同时,我也通过查阅论文了解到一些 SSVEP 目前的挑战与趋势:

目前, SSVEP 甚至已经应用到脑控机械臂、无人机、小车寻路等领域,但普遍存在一些问题:

- 在范式设计方面,缺乏指令数更多、舒适度更高的实验范式
- 在信号处理方面,缺乏分类精度更高、泛化能力更强的信号处理技术
- 在控制方式方面,缺乏控制效率更高、稳定性和鲁棒性更强的控制方式

同时,大家也正在致力于使用机器学习、多模态、改进算法思想与框架等多种方式来进一步发展这个领域,非常超乎我的想象力。期待脑机接口有朝一日可以“飞入寻常百姓家”。