AI Capstone: Project 1 Report

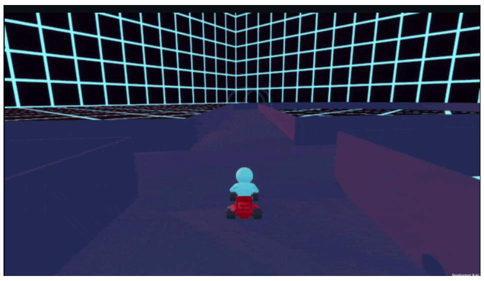
110550090 王昱力

Research

In this project, I have delved into a new field of computer science known as BCI (Brain-Computer Interface). In this field, our primary focus is on brain signal data. There are various types of brain signals, and the type I utilized in this project is called EEG (Electroencephalography). So, what can we do with brain signals? In recent applications, treating stroke patients by applying brain signals corresponding to the affected body parts that patients cannot control, as well as mind control, are the two main reasons that sparked my interest.

Dataset

1. Data type : electrical activity with time series (Figure 2)
2. Amount and Composition : EEG signals were collected using motor imagery, and I have extracted 544 events from the raw EEG data.
    a. motor imagery : a type of EEG commonly used in experiments. It involves recording and analyzing brain activity when the subject imagines performing a certain movement.
3. Process of data collection : designing a Unity game wherein participants can imagine themselves as characters in the game, while simultaneously recording EEG data.
    a. github link : https://github.com/xEvheMary/MI-BCI-UnityKart?tab=readme-ov-file


game

SMARTINGPRO

    b. Hardware: **SMARTING PRO** (worn on the head to collect brain signals with 32 channels).

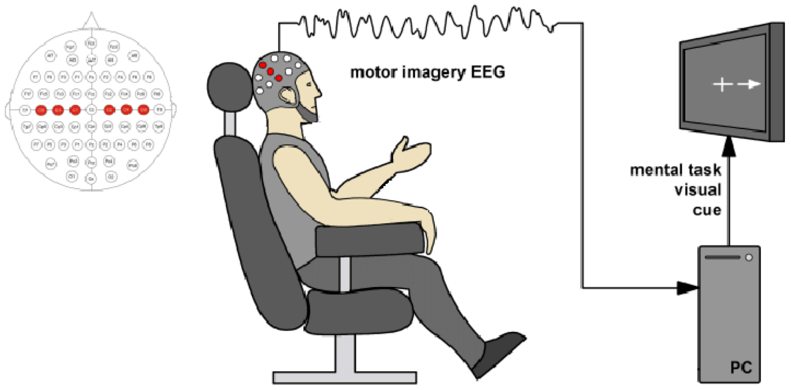
    c. Software: Python, OpenVibe (used to transfer the data collected from Cygnus to Python for processing) , Cygnus (GUI for the SMARTING PRO).
4. Amount of the dataset : 544 events, collected from the raw_data.

Methods

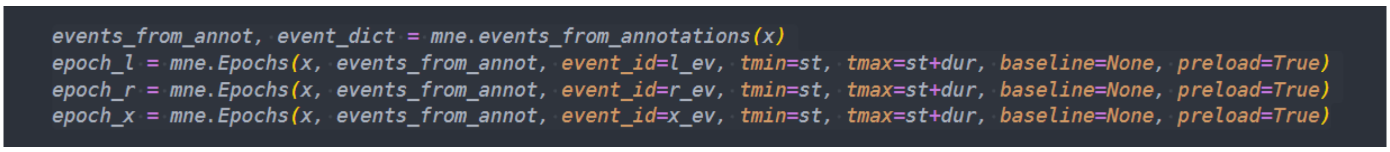
Public libraries :

numpy, matplotlib, math, sklearn, seaborn, mne(designed for data of EEG/MEG), pytorch

Experiment Flow Diagram:

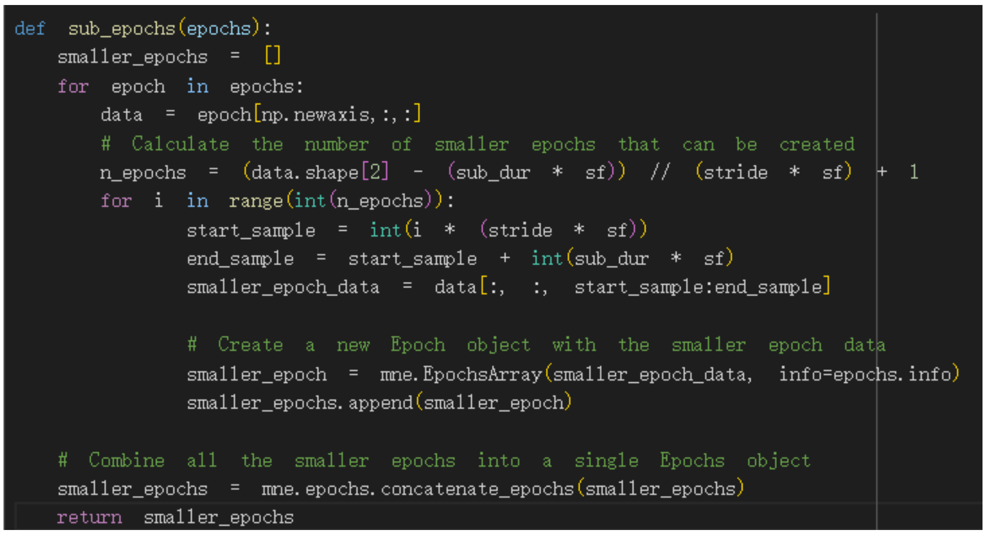


In the beginning, I mentioned that I am using a 32-channel EEG device to collect brain signals. In Figure 1, we can observe the standard positions of each channel relative to the head. The 32 channels utilized in this project are listed as follows:
ch_list = ['Fp1', 'Fp2', 'AF3', 'AF4', 'F7', 'F3', 'Fz', 'F4', 'F8', 'FT7', 'FC3', 'FCz','FC4', 'FT8', 'T7', 'C3', 'Cz', 'C4', 'T8', 'TP7', 'CP3', 'CPz', 'CP4', 'TP8', 'P7', 'P3', 'Pz', 'P4', 'P8', 'O1', 'Oz', 'O2']
To detect signals related to motor imagery cases, we will focus more on ['C3','Cz','C4'] channels. It has been proven that these channels are highly related to capturing signals associated with "imagining hand movements". In the experimental design, participants were instructed to imagine moving their left or right hand to turn the kart left or right, respectively.
In Figure 2, we can observe the intensity of each channel. Since we have designed the game, we know exactly during which time section we should turn left or right, or even just move forward. Therefore, the method to determine which channels will be evoked or undergo significant changes while turning the kart left is to analyze the combination of each channel during the epochs in that specific time. Thus, in this experiment, we are going to observe frequencies between 1Hz and 40Hz. For turning right and moving forward, the process is the same.
In Figure 3, I have depicted a simple graph representing the experiment. For instance, during the time interval from 0 to 6 seconds, we know that the kart is moving forward. Therefore, we will concentrate on the EEG signals to understand the characteristics of the EEG curves under such conditions. Our final goal is to predict how given brain signals correspond to reactions of the kart.
To delve deeper into my experiment structure, the mne library provides a function, `mne.events_from_annotations(x)`, where x represents the raw data. This function returns the number of different types of events that occur during the entire recording. Additionally, by utilizing the `mne.Epochs` function, we can precisely obtain the timing of the events (left, right, forward) in the raw data. Here's what it will look like:



```
events_from_annot, event_dict = mne.events_from_annotations(x)
epoch_l = mne.Epochs(x, events_from_annot, event_id=l_ev, tmin=st, tmax=st+dur, baseline=None, preload=True)
epoch_r = mne.Epochs(x, events_from_annot, event_id=r_ev, tmin=st, tmax=st+dur, baseline=None, preload=True)
epoch_x = mne.Epochs(x, events_from_annot, event_id=x_ev, tmin=st, tmax=st+dur, baseline=None, preload=True)
```

To collect more data, I've divided each epoch into multiple sub-epochs. In Figure 3, the blue box (sub_dur) represents the duration of each sub-epoch I've chosen, and 0.25 (s) indicates the interval by which the box will move with each stride. Using this method, we can gather more features and expand our dataset for analysis. Here's what it will look like :



```python
def  sub_epochs(epochs):
    smaller_epochs = []
    for epoch in epochs:
        data = epoch[np.newaxis, :, :]
        # Calculate the number of smaller epochs that can be created
        n_epochs = (data.shape[2] - (sub_dur * sf)) // (stride * sf) + 1
        for i in range(int(n_epochs)):
                start_sample = int(i * (stride * sf))
                end_sample = start_sample + int(sub_dur * sf)
                smaller_epoch_data = data[:, :, start_sample:end_sample]

                # Create a new Epoch object with the smaller epoch data
                smaller_epoch = mne.EpochsArray(smaller_epoch_data, info=epochs.info)
                smaller_epochs.append(smaller_epoch)

    # Combine all the smaller epochs into a single Epochs object
    smaller_epochs = mne.epochs.concatenate_epochs(smaller_epochs)
    return smaller_epochs
```

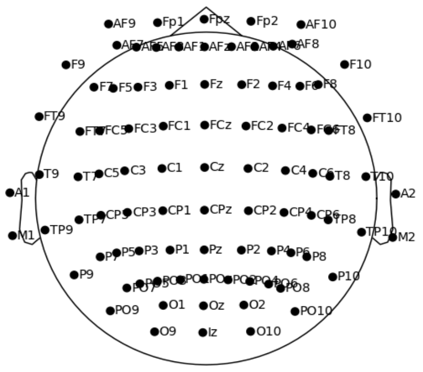


Figure 1

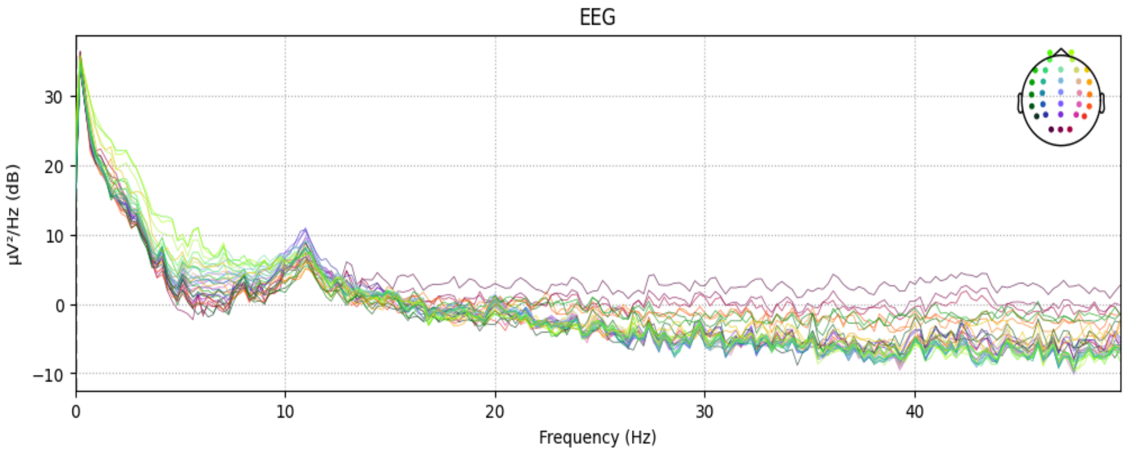


Figure 2

Explanation of Figure 2:

Figure 2 displays the signal power of each channel in different frequency ranges, typically measured in dB/Hz (decibels per Hertz). A noticeable sharp decline in the 0-10 Hz range may be attributed to external factors such as eye movements, noise in the experimental environment, or physiological conditions of the subjects.

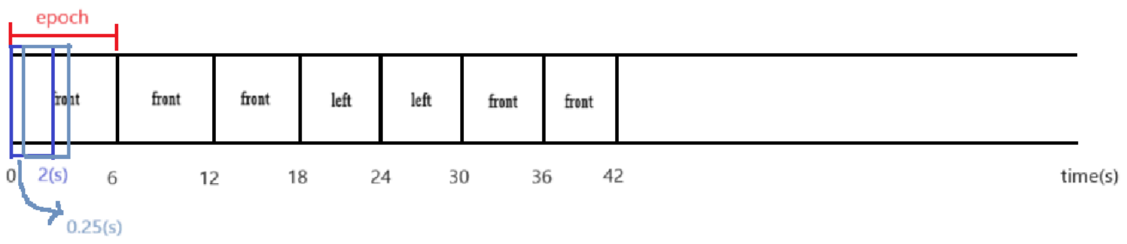


Figure 3

Supervised methods :

Experiments(ML):

Introduction to CSP:
CSP (Common Spatial Patterns) is a feature extraction method (dimensionality reduction) employed in brain signal analysis. In essence, CSP optimizes linear combinations of EEG channels to enhance the discriminative capacity of spatial patterns. It achieves this by maximizing the variance of one class while simultaneously minimizing the variance of another.

	without cross-validation(CSP + LDA)	**5-fold-cross-validation(CSP + LDA)**
Training data	**Accuracy**	**Accuracy**
20% (108)	**0.65**	**0.53**
40% (217)	**0.7003**	**0.63**
60% (326)	**0.7018**	**0.65**
80% (435)	**0.67**	**0.72**

Summarization/Discussions :

From the results, we can draw a conclusion and infer that without cross-validation, the amount of data seems to have little influence on the accuracy for EEG datasets. However, when cross-validation is applied, the accuracy tends to increase with the size of the dataset.

With/Without dimensionality reduction

	CSP + LDA	CSP + SVM	LDA	SVM
Training data	Accuracy	Accuracy	Accuracy	Accuracy
20% (108)	0.53	0.66	0.41	0.49

40% (217)	**0.63**	0.7264	0.422	0.504
60% (326)	**0.65**	0.7448	0.34	0.467
80% (435)	**0.72**	0.7494	0.39	0.477

Summarization/Discussions :

From the results, we can draw a conclusion and infer that by employing feature extraction methods, the accuracy significantly improves in EEG datasets. Additionally, we observe that in the classification of EEG datasets, SVM outperforms LDA.

Augmentation(CSP + SVM)

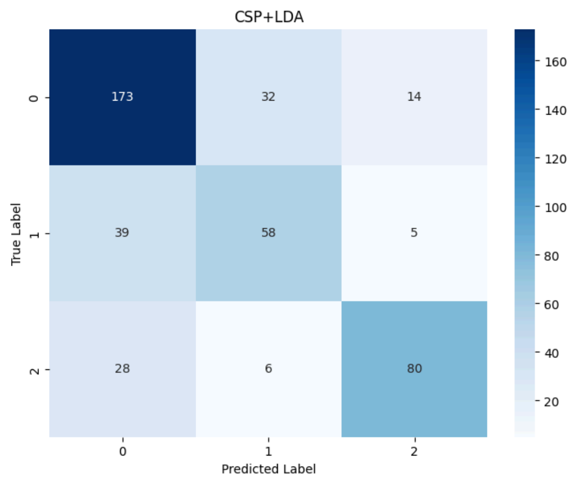
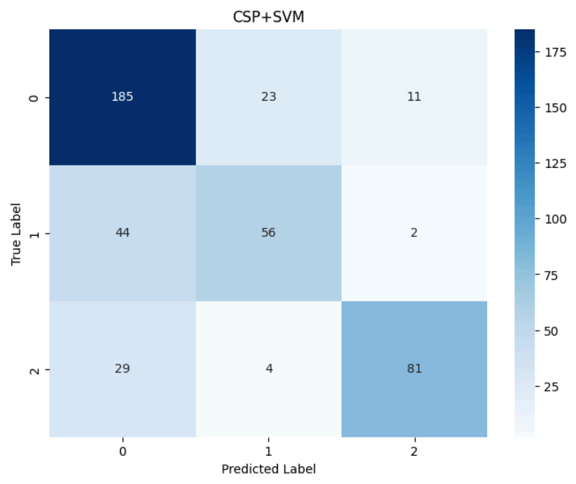
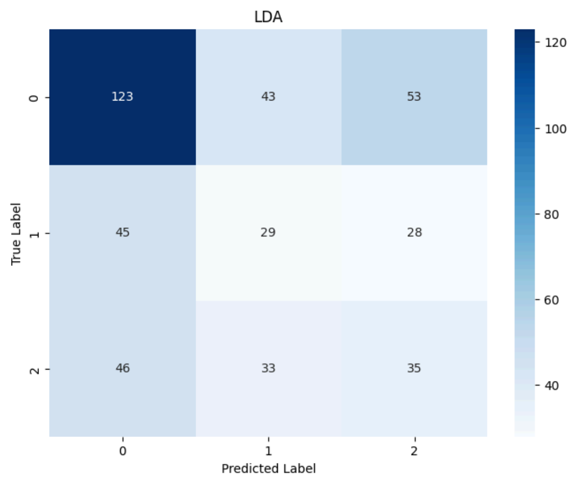
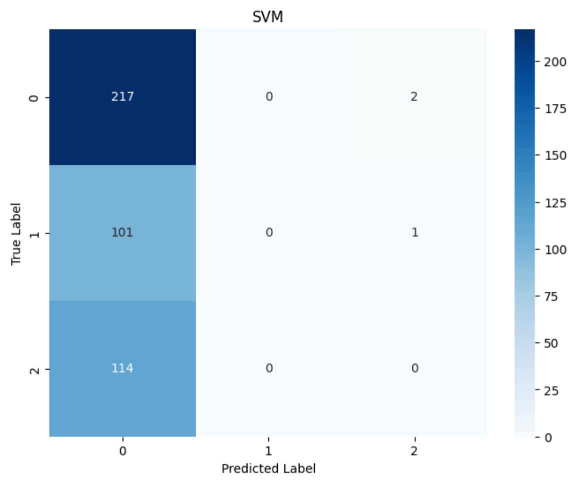
	1~40(Hz)	8~12(Hz)	13~30(Hz)
Training data	Accuracy	Accuracy	Accuracy
20% (108)	0.499	0.66	0.593
40% (217)	0.534	0.72	0.612
60% (326)	0.526	0.7448	0.639
80% (435)	0.533	0.7494	0.666

Summarization/Discussions :

From the results, we can observe that the frequency range between 8~12Hz exhibits significantly better performance, while the range from 13~30 Hz is secondary. Conversely, the range from 1~40 Hz performs worse. This suggests that there is a meaningful difference in performance across different frequency bands. In BCI, the frequency range between 8~12 Hz is referred to as the Alpha (α) band, while the range from 13~30 Hz is known as the Beta (β) band. In a relaxed state, higher alpha band activity is typically detected, whereas during the preparation or execution of movement, alpha band activity decreases. Conversely, for the beta band, activity increases during the preparation and execution of movement. Therefore, we can capture changes in band power to determine the state the participants are in. Compared to the raw data (1~40Hz), the observed results seem reasonable.

Confusion matrix

CSP + LDA	**CSP + SVM**

	
LDA	**SVM**
	

Summarization/Discussions :

From the results, we can observe that implementing the dimension-reduction method (CSP) leads to an increase in the number of accurately predicted labels. However, for SVM, although the accuracy is high, upon plotting the confusion matrix, I discovered that the results are not entirely ideal. This is because most of the predicted labels are concentrated in label 0. The higher accuracy is attributed to the dataset containing more label 0 events.

Experiments(Deep Learning):

Reference of EEG model structure :
https://github.com/High-East/BCI-ToolBox/blob/master/models/EEGNet/EEGNet.py

Different method of cross-validation

	StratifiedGroupKFold(K = 5)	**StratifiedShuffleSplit(K = 5)**

learning rate	Accuracy	Accuracy
0.0005	0.396	0.75
0.001	0.428	0.862
0.005	0.36	0.53
0.01	0.38	0.61

Summarization/Discussions :

From the results, we can conclude that, for EEG datasets, ShuffleSplit tends to yield better accuracy than GroupKFold. This observation suggests that weaker group correlation within each group or event might contribute to this outcome. As for the learning rate, we find that the optimal value lies at 0.001.

SMOTE

	K-fold(unbalance)	K-fold(balance)	Shuffle(unbalance)	Shuffle(balance)
learning rate	Accuracy	Accuracy	Accuracy	Accuracy
0.0005	0.396	0.43	0.75	0.574
0.001	0.428	0.323	0.862	0.78
0.005	0.36	0.38	0.53	0.61
0.01	0.38	0.37	0.61	0.61

Summarization/Discussions :

From the results, we observe that without resampling the data, the accuracy is better on average. Additionally, except for the K-fold (balanced) cases, a learning rate of 0.001 yields the best accuracy in the other three cases. I believe the reason for unbalanced data outperforming balanced data is that, for EEG data, signals from every channel affect the results. Therefore, instead of resampling the dataset, we need to collect more data for those classes with fewer samples to enable the model to learn from diverse cases.

Unsupervised methods :

PCA-KMeans

	1~40(Hz)		8~12(Hz)		13~30(Hz)	
	PCA(10)	PCA(5)	PCA(10)	PCA(5)	PCA(10)	PCA(5)
ARI	0.0006	0.0035	-0.002	-0.002	-0.002	-0.0019

MI	**0.002**	**0.002**	**0.001**	**0.0015**	**0.0006**	**0.0005**
FMI	**0.355**	**0.365**	**0.353**	**0.351**	**0.354**	**0.352**

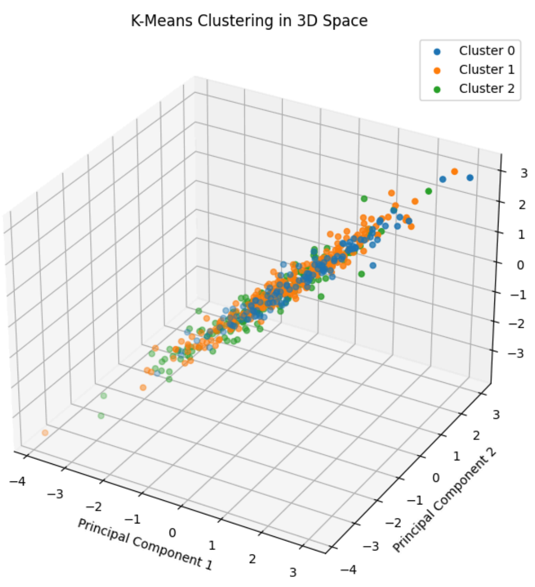
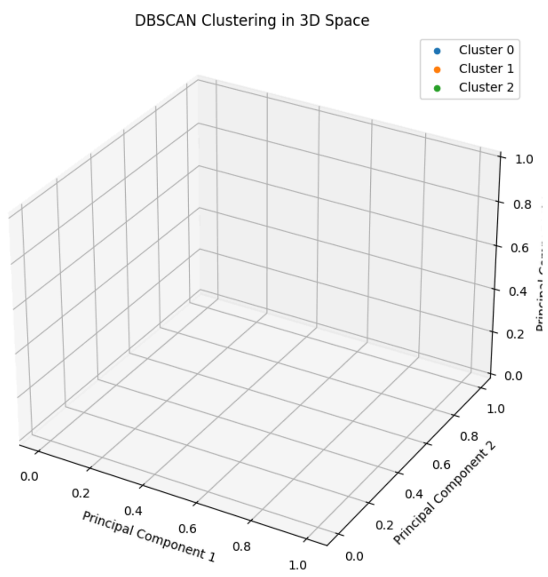
Summarization/Discussions :

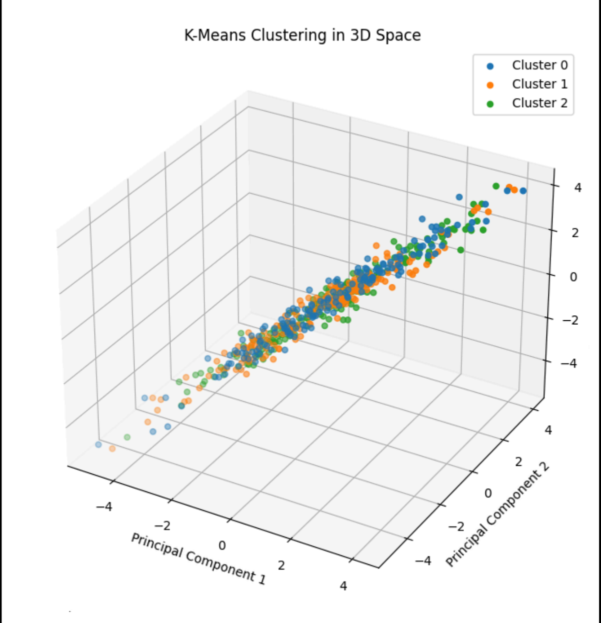
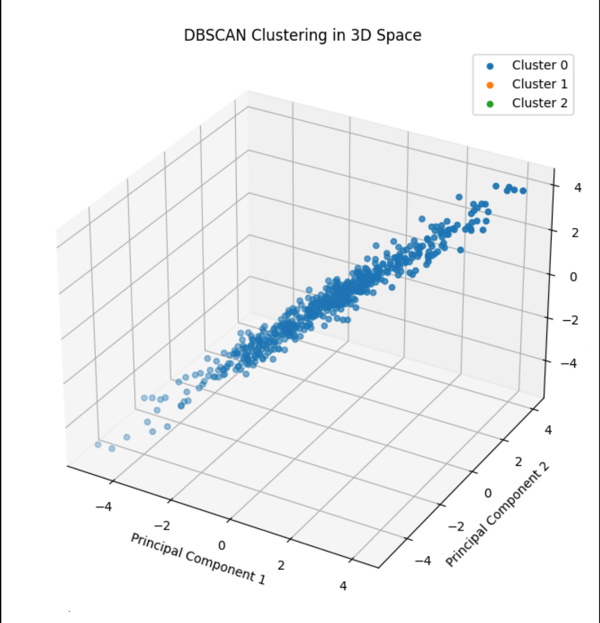
From the results, we observe that in three bandpass cases (1~40, 8~12, 13~30), the Adjusted Rand Index (ARI) is nearly close to 0, indicating a tendency towards random allocation. Similarly, for Mutual Information (MI), the value is also close to 0, suggesting a weak relationship between the real label and predicted label. As for the Fowlkes-Mallows Index (FMI), although the results seem better than the other two indices, they are still quite low. However, it indicates that when predicted as the same cluster, the EEG data might share some common information. I believe that when conducting unsupervised learning on EEG data, using FMI could be a valuable tool.

Additional Experiments

	ICA(5)-KMeans	**ICA(5)-DBSCAN**	**PCA(5)-KMeans**	**PCA(5)-DBSCAN**
ARI	**-0.0016**	**0**	**-0.002**	**0**
MI	**0.0004**	**0**	**0.0015**	**0**
FMI	**0.366**	**0.611**	**0.351**	**0.611**

Confusion matrix

ICA(5)-KMeans	**ICA(5)-DBSCAN**
	

PCA(5)-KMeans	**PCA(5)-DBSCAN**
	

Summarization/Discussions :

After conducting the analysis based on PCA and K-Means, I found the results to be of little value. Therefore, I attempted to use another dimension reduction method (ICA) and an unsupervised method (DBSCAN) to gain further insights into the EEG data. However, the results were similarly unsatisfactory. I experimented with various parameters in both DBSCAN and ICA to find the optimal clustering approach, but the outcomes remained poor. Additionally, even when combining ICA with DBSCAN, the model failed to separate any clusters despite trying numerous parameter combinations. In summary, I believe the poor results stem from the inherent difficulty in identifying variance between different labels in brain signals, as they may be influenced by subtle factors.

Discussion

1. **Based on your experiments, are the results and observed behaviors what you expect?**
   I think the result made from augmentation parts makes me excited, since it truly matches what I expected and also makes me understand what factors will affect alpha bands. However, speaking of the unsupervised-learning part, I think the result frustrated me.
2. **Discuss factors that affect the performance, including dataset characteristics.**
   For EEG datasets, I believe they are influenced by individual differences and the level of concentration during recording sessions. Since recording typically lasts 1 to 2 hours, it can be challenging to maintain focus during the whole period. Additionally, motor imagery tasks involve imagining movements without physically performing them, which can be abstract. This ambiguity raises uncertainty about whether the

collected data aligns with our expectations. Moreover, preprocessing, dimension reduction, and selecting the right channels for analysis also affect performance.

3. **Describe experiments that you would do if there were more time available.**
   If more time were available, I would like to try real-time experiences in the Unity game. However, it remains difficult to simultaneously record data and perform instant reactions in the game due to the preprocessing of raw data. Additionally, BCI (Brain-Computer Interface) is a very new field in computer science, so it's challenging to find information and helpful papers for research. In addition to motor imagery, which I currently use to record data, there are many other techniques used to collect EEG data, such as SSVEP (Steady-State Visually Evoked Potential) and ERP (Event-Related Potential). If there were more time, I would be willing to delve into these methods and learn more about EEG.
4. **Indicate what you have learned from the experiments and remaining questions.**
   Before starting this project, I had only a vague concept of EEG. However, as I progressed with the project, I gradually gained a deeper understanding of this field and learned how to code specifically for the data we collected. It also made me realize the considerable effectiveness of deep learning compared to traditional machine learning methods. Additionally, I had never attempted dimension reduction in a dataset before, and this project provided me with the opportunity to learn about PCA and how to implement it. However, despite reading some papers suggesting that using SMOTE could improve EEG performance, in my case, it didn't seem to have that effect. I believe this could be a challenging question for me to solve in the future.

Reference

https://github.com/High-East/BCI-ToolBox/blob/master/models/EEGNet/EEGNet.py
https://github.com/xEvheMary/MI-BCI-UnityKart?tab=readme-ov-file
https://mne.tools/stable/api/python_reference.html
https://www.mdpi.com/2075-4418/13/6/1122

```
# ------------------------------------------------------------------
deep
learning-----------------------------------------------------------
------
!pip install mne &> /dev/null
!pip install skorch -U &> /dev/null
!pip install mne-icalabel &> /dev/null
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
import os
import numpy as np
import mne
import matplotlib.pyplot as plt
import math
from scipy.io import loadmat
from mne.preprocessing import ICA
from mne_icalabel import label_components
from sklearn.base import BaseEstimator, TransformerMixin
import pickle
from skorch import NeuralNetClassifier
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score, cross_val_predict
from mne.decoding import CSP
from mne.filter import filter_data
from sklearn.preprocessing import StandardScaler
```

```
Mounted at /content/drive
```

```
mne.set_log_level(False)

sbj_n = 4 #@param {type:"integer"}
sbj_path = 'Subject ' + str(sbj_n)

base_path = '/content/drive/MyDrive/EEG data'
subj_folder = os.path.join(base_path, sbj_path)
l = [file for file in os.listdir(subj_folder) if file.endswith('gdf')]

type_1 = [file for file in os.listdir(subj_folder) if
file.endswith('1.gdf')]
type_2 = [file for file in os.listdir(subj_folder) if
file.endswith('2.gdf')]

smaller = min(len(type_1), len(type_2))
type_1 = type_1[:smaller]
type_2 = type_2[:smaller]

# Constant
ch_list = ['Fp1', 'Fp2', 'AF3', 'AF4', 'F7', 'F3', 'Fz', 'F4', 'F8',
'FT7', 'FC3', 'FCz','FC4', 'FT8', 'T7', 'C3', 'Cz', 'C4', 'T8', 'TP7',
'CP3', 'CPz', 'CP4', 'TP8', 'P7', 'P3', 'Pz', 'P4', 'P8', 'O1', 'Oz',
'O2']
# For epoching
st = 0
# For sub-epochs
sub_dur = 2
stride = 0.25
# For cross-validation dataset
n_splits = 4

def load_ds(pt, f):       # For BCIC Dataset
  fp = os.path.join(pt, f)
  print("File path: {}".format(fp))
  if fp.endswith('gdf'):
    raw_data = mne.io.read_raw_gdf(fp, preload=True)
  else:
    raw_data = None
  return raw_data

def select_event(events_from_annot, event_dict, sfreq):
  # specify needed event
  l_ev = event_dict['769']
  r_ev = event_dict['770']
  x_ev = event_dict['33024']
  end_trial = event_dict['800']
  needed_event = [l_ev, r_ev, x_ev, end_trial]
  # Remove unecessary last part
  re = events_from_annot[::-1, 2].tolist()  # reverse
  last_id = re.index(end_trial)             # Find index of last end
signal
  if last_id > 0:                           # Filter out last part
    events_annot = events_from_annot[:-last_id]
  else:
    events_annot = events_from_annot[:]
  # Filter put other event except the one used
  mask = np.isin(events_annot[:, 2], needed_event)
  filtered_events = events_annot[mask]
  # Get durations
  a = np.diff(filtered_events[:, 0])/sfreq
  dur = round(np.mean(a[::2]),2)
  print('Average duration of this session trials : ',dur)
  return needed_event, dur

def sub_epochs(epochs):
  smaller_epochs = []
  sf = epochs.info['sfreq']
  for epoch in epochs:
    data = epoch[np.newaxis,:,:]
    # Calculate the number of smaller epochs that can be created
    n_epochs = (data.shape[2] - (sub_dur * sf)) // (stride * sf) + 1
    for i in range(int(n_epochs)):
        start_sample = int(i * (stride * sf))
        end_sample = start_sample + int(sub_dur * sf)
        smaller_epoch_data = data[:, :, start_sample:end_sample]

        # Create a new Epoch object with the smaller epoch data
        smaller_epoch = mne.EpochsArray(smaller_epoch_data,
info=epochs.info)
        smaller_epochs.append(smaller_epoch)
  # Combine all the smaller epochs into a single Epochs object
  smaller_epochs = mne.epochs.concatenate_epochs(smaller_epochs)
  return smaller_epochs

def create_label(size, lbl):
  return np.full(size, lbl)

def epoch_array(epoch_l, epoch_r, epoch_x):
  mini_ep_l = sub_epochs(epoch_l)   # Create sub epochs
  mini_ep_r = sub_epochs(epoch_r)
  mini_ep_x = sub_epochs(epoch_x)
  ec_l = len(mini_ep_l)/8       # Calculate sub-epochs per trial
  ec_r = len(mini_ep_r)/8
  ec_x = len(mini_ep_x)/8
  ep_l = mini_ep_l.get_data(copy=True)   # Turn to array
  ep_r = mini_ep_r.get_data(copy=True)
  ep_x = mini_ep_x.get_data(copy=True)
  lbl_l = create_label(ep_l.shape[0], 1)    # Create Labels
  lbl_r = create_label(ep_r.shape[0], 2)
  lbl_x = create_label(ep_x.shape[0], 0)
  # Combine arrays
  epoch_data = np.concatenate((ep_l, ep_r, ep_x),axis=0)
  label_data = np.concatenate((lbl_l, lbl_r, lbl_x),axis=0)
  return epoch_data, label_data

def balance_dataset(X, y):
  unique, counts = np.unique(y, return_counts=True)
  groups = np.hstack((np.repeat(np.arange(8), ([int(counts[1]/8)]*8)),
np.repeat(np.arange(8), ([int(counts[2]/8)]*8)),
np.repeat(np.arange(8), ([int(counts[0]/8)]*8))))
  # Randomly sample from class 0, get indices
  balanced_indices = []
  for group_val in range(8):
      class_2_indices = np.where((groups == group_val) & (y == 0))[0]
      selected_indices = np.random.choice(class_2_indices,
size=int(counts[1]/8), replace=False)
      balanced_indices.extend(selected_indices)
  balanced_indices.sort()
  balanced_X = np.concatenate([X[y != 0], X[balanced_indices]])
  balanced_y = np.concatenate([y[y != 0], y[balanced_indices]])
  unique2, counts2 = np.unique(balanced_y, return_counts=True)
  new_groups = np.hstack((np.repeat(np.arange(8),
([int(counts2[1]/8)]*8)), np.repeat(np.arange(8),
([int(counts2[2]/8)]*8)), np.repeat(np.arange(8),
([int(counts2[0]/8)]*8))))
  return balanced_X, balanced_y, new_groups

def save_epoch(X, y, file_name):
  # Names
  epoch_folder = subj_folder + '/Epoch'
  rec_name = file_name[:-4] + '_data.txt'
  lbl_name = file_name[:-4] + '_label.txt'
  # Path
  ep_file = os.path.join(epoch_folder, rec_name)
  lbl_file = os.path.join(epoch_folder, lbl_name)
  arr_reshaped = X.reshape(X.shape[0], -1)
  np.savetxt(ep_file, arr_reshaped)
  np.savetxt(lbl_file, y)

def load_dataset(file_list: list):
  combined_set = []
  for data in file_list:
    x = load_ds(subj_folder, data)    # Load data
    combined_set.append(x)
  raw_cat = mne.concatenate_raws(combined_set)
  channel_mapping = {old_name: new_name for old_name, new_name in
zip(raw_cat.ch_names, ch_list)}   # Remap channels
  raw_cat.rename_channels(channel_mapping)
  montage = mne.channels.make_standard_montage('standard_1020')
  _ = raw_cat.set_montage(montage)
  events_from_annot, event_dict = mne.events_from_annotations(raw_cat)
  needed_event, dur = select_event(events_from_annot, event_dict,
raw_cat.info['sfreq'])
  return raw_cat, needed_event, dur, events_from_annot

for data in type_1:
  x = load_ds(subj_folder, data)    # Load data
  events_from_annot, event_dict = mne.events_from_annotations(x)
  needed_event, dur = select_event(events_from_annot, event_dict,
x.info['sfreq'])
  print(needed_event)

File path: /content/drive/MyDrive/EEG data/Subject 4/record-
[2024.03.10]_S9_1.gdf
Average duration of this session trials :  6.21
[3, 4, 2, 5]
File path: /content/drive/MyDrive/EEG data/Subject 4/record-
[2024.03.10]_S7_1.gdf
Average duration of this session trials :  6.21
[3, 4, 2, 5]
```

```python
def create_dataset(file_list: list):
  combined_set = []
  combined_labels = []
  combined_groups = []
  for data in file_list:
    x = load_ds(subj_folder, data)    # Load data
    events_from_annot, event_dict = mne.events_from_annotations(x)
    # Event select
    needed_event, dur = select_event(events_from_annot, event_dict,
x.info['sfreq'])
    x = x.set_eeg_reference("average")
    print(needed_event)
    # Load data into epochs
    #epoch_base = mne.Epochs(x.copy().crop(tmin=10-dur, tmax=10),
events_from_annot)

    #--------------------origin-------------------------
    epoch_l = mne.Epochs(x, events_from_annot,
event_id=needed_event[0], tmin=st, tmax=st+dur, baseline=None,
preload=True)
    epoch_r = mne.Epochs(x, events_from_annot,
event_id=needed_event[1], tmin=st, tmax=st+dur, baseline=None,
preload=True)
    epoch_x = mne.Epochs(x, events_from_annot,
event_id=needed_event[2], tmin=st, tmax=st+dur, baseline=None,
preload=True)
    #--------------------origin-------------------------

    # epoch_l = mne.Epochs(x, events_from_annot,
event_id=needed_event[1], tmin=st, tmax=st+dur, baseline=None,
preload=True)
    # epoch_r = mne.Epochs(x, events_from_annot,
event_id=needed_event[3], tmin=st, tmax=st+dur, baseline=None,
preload=True)
    # epoch_x = mne.Epochs(x, events_from_annot,
event_id=needed_event[2], tmin=st, tmax=st+dur, baseline=None,
preload=True)

    if len(epoch_x) > (len(epoch_l)+len(epoch_r)):
      epoch_x.drop([-1])    # Remove last epoch since it's
    # Bandpass filter
    #epoch_base.filter(l_freq=1, h_freq=40)

    # retain alpha power band ?
    epoch_l.filter(l_freq=6, h_freq=14)
    epoch_r.filter(l_freq=6, h_freq=14)
    epoch_x.filter(l_freq=6, h_freq=14)
    X, y = epoch_array(epoch_l, epoch_r, epoch_x)

    new_X, new_y, current_group = balance_dataset(X, y)
    save_epoch(new_X, new_y, data)
    combined_set.append(new_X)
    combined_labels.append(new_y)
    combined_groups.append(current_group)

    # without balance data
    # ------------------------------------------------------
    # unique, counts = np.unique(y, return_counts=True)
    # groups = np.hstack((np.repeat(np.arange(8),
([int(counts[1]/8)]*8)), np.repeat(np.arange(8),
([int(counts[2]/8)]*8)), np.repeat(np.arange(8),
([int(counts[0]/8)]*8))))
    # save_epoch(X, y, data)
    # combined_set.append(X)
    # combined_labels.append(y)
    # combined_groups.append(groups)
    # ------------------------------------------------------

  if len(combined_set) > 1:
    result_X = np.vstack(combined_set)
    result_y = np.concatenate(combined_labels)
    result_group = np.concatenate(combined_groups)
  else:
    result_X = combined_set[0]
    result_y = combined_labels[0]
    result_group = combined_groups[0]
  return result_X, result_y, result_group

from sklearn.model_selection import (
    GroupKFold,
    KFold,
    StratifiedGroupKFold,
    StratifiedKFold,
    StratifiedShuffleSplit
)

cmap_data = plt.cm.Paired
cmap_cv = plt.cm.coolwarm
```

#Type1

```
ds, lbl, grouping = create_dataset(type_1)

print(len(ds))
print(len(lbl))
print(len(grouping))
```

```
1088
1088
1088
```

```
cv = StratifiedGroupKFold(n_splits)

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from scipy import signal
from skorch.callbacks import Checkpoint, EpochScoring, EarlyStopping

t = 2
sf = 128
cls_n = 3

# https://github.com/High-East/BCI-ToolBox/blob/master/models/EEGNet/
EEGNet.py
class EEGNet(nn.Module):
    def __init__(self, in_chn, n_cls, input_ts, f1=8, f2=16, d=2,
drop_prob=0.5):
        super(EEGNet, self).__init__()

        self.F1 = f1   # High Frequency pattern
        self.F2 = f2   # Lower Frequency patter
        self.D = d    # Dilation (?), spatial?
        #
        self.kernel_l = math.ceil(sf/2)
        self.chn = in_chn
        self.cls = n_cls
        self.drop_prob = drop_prob
        self.tp = input_ts

        # Spectral
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, self.F1, (1, self.kernel_l), padding=(0,
math.ceil(self.kernel_l//2)), bias=False),
            nn.BatchNorm2d(self.F1)
        )

        # Spectral-specific Spatial
        self.conv2 = nn.Sequential(
            nn.Conv2d(self.F1, self.D*self.F1, (self.chn, 1),
groups=self.F1, bias=False),
            nn.BatchNorm2d(self.D*self.F1),
            nn.ELU(),
            nn.AvgPool2d((1, 4)),
            nn.Dropout(self.drop_prob)
        )

        # Temporal
        self.conv3 = nn.Sequential(
            nn.Conv2d(self.D*self.F1, self.D*self.F1, (1,
math.ceil(self.kernel_l//4)), padding=(0, 8), groups=self.D*self.F1,
bias=False),
            nn.Conv2d(self.D*self.F1, self.F2, (1, 1), bias=False),
            nn.BatchNorm2d(self.F2),
            nn.ELU(),
            nn.AvgPool2d((1, 8)),
            nn.Dropout(self.drop_prob)
        )

        #self.classifier = nn.Linear(math.ceil(self.kernel_l/4)*
math.ceil(self.tp//32), self.cls, bias=True)
        self.classifier = nn.Linear(self.F2 * math.ceil(self.tp//32),
self.cls, bias=True)
        #self.softmax = nn.Softmax()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(-1, self.F2*math.ceil(self.tp//32))
        x = self.classifier(x)
        #x = self.softmax(x)
        return x

new_sf = t * sf

n_epoch = 200
# learn_r = 0.001
learn_r = 0.0005
n_batch = 32

x_resample = signal.resample(ds, new_sf, axis=2)
y_resample = lbl.astype(np.int64)
x_resample = np.expand_dims(x_resample, axis=1).astype(np.float32)

net = NeuralNetClassifier(
    EEGNet,
    module__in_chn=x_resample.shape[-2],
    module__n_cls=cls_n,
    module__input_ts=x_resample.shape[-1],
    criterion = torch.nn.CrossEntropyLoss(),
    optimizer = torch.optim.AdamW,
    iterator_train__shuffle=True,
    batch_size = n_batch,
    callbacks=[
        EpochScoring(scoring='accuracy', name='train_acc',
on_train=True),
        Checkpoint(monitor='valid_loss_best'),  # save based on
validation loss
        #EarlyStopping(patience=50, monitor='valid_loss')
    ],
    max_epochs=n_epoch,
    lr=learn_r,
    device='cuda' if torch.cuda.is_available() else 'cpu'
)

scores = cross_val_score(net, x_resample, y_resample, groups=grouping,
cv=cv, n_jobs=None)

print("Cross-validated accuracy scores:", scores)
print("Mean accuracy:", np.mean(scores))

Cross-validated accuracy scores: [0.40073529 0.25       0.25
0.36029412]
Mean accuracy: 0.31525735294117646
```

###Shuffle

```
cv2 = StratifiedShuffleSplit(n_splits)

scores2 = cross_val_score(net, x_resample, y_resample,
groups=grouping, cv=cv2, n_jobs=None)

print("Cross-validated accuracy scores:", scores2)
print("Mean accuracy:", np.mean(scores2))

Cross-validated accuracy scores: [0.69724771 0.26605505 0.25688073
0.6146789 ]
Mean accuracy: 0.45871559633027525
```

#Type2

```
ds2, lbl2, grouping2 = create_dataset(type_2)

File path: /content/drive/MyDrive/EEG data/Subject 4/record-
[2024.03.10]_S8_2.gdf
Average duration of this session trials :  6.27
[4, 5, 2, 6]
File path: /content/drive/MyDrive/EEG data/Subject 4/record-
[2024.03.10]_S10_2.gdf
Average duration of this session trials :  6.28
[4, 5, 2, 6]

cv = StratifiedGroupKFold(n_splits)

learn_r = 0.01

x_resample = signal.resample(ds2, new_sf, axis=2)
y_resample = lbl2.astype(np.int64)
x_resample = np.expand_dims(x_resample, axis=1).astype(np.float32)
```

```
net = NeuralNetClassifier(
    EEGNet,
    module__in_chn=x_resample.shape[-2],
    module__n_cls=cls_n,
    module__input_ts=x_resample.shape[-1],
    criterion = torch.nn.CrossEntropyLoss(),
    optimizer = torch.optim.AdamW,
    iterator_train__shuffle=True,
    batch_size = n_batch,
    callbacks=[
        EpochScoring(scoring='accuracy', name='train_acc',
on_train=True),
        Checkpoint(monitor='valid_loss_best'),  # save based on
validation loss
        #EarlyStopping(patience=50, monitor='valid_loss')
    ],
    max_epochs=n_epoch,
    lr=learn_r,
    device='cuda' if torch.cuda.is_available() else 'cpu'
)

scores = cross_val_score(net, x_resample, y_resample,
groups=grouping2, cv=cv, n_jobs=None)

print("Cross-validated accuracy scores:", scores)
print("Mean accuracy:", np.mean(scores))

Cross-validated accuracy scores: [0.375      0.41666667 0.3287037
0.36111111]
Mean accuracy: 0.37037037037037037
```

###Shuffle

```
cv2 = StratifiedShuffleSplit(n_splits)

scores2 = cross_val_score(net, x_resample, y_resample,
groups=grouping2, cv=cv2, n_jobs=None)

print("Cross-validated accuracy scores:", scores2)
print("Mean accuracy:", np.mean(scores2))
# -------------------------------------------------------------------
deep
learning------------------------------------------------------------
------

Cross-validated accuracy scores: [0.4137931  0.8045977  0.42528736
0.8045977 ]
Mean accuracy: 0.6120689655172414
```

```
# ------------------------------------------------------------------
machine
learning------------------------------------------------------------
------
%matplotlib inline
!pip install mne &> /dev/null

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

Mounted at /content/drive

import os
import numpy as np
import mne
import matplotlib.pyplot as plt
import math
from scipy.io import loadmat

mne.set_log_level(False)

def load_ds(pt, f):       # For BCIC Dataset
  fp = os.path.join(pt, f)
  print("File path: {}".format(fp))
  if fp.endswith('gdf'):
    raw_data = mne.io.read_raw_gdf(fp)
  else:
    raw_data = None
  return raw_data

sbj_n = 4 #@param {type:"integer"}
sbj_path = 'Subject ' + str(sbj_n)

base_path = '/content/drive/MyDrive/EEG data'
subj_folder = os.path.join(base_path, sbj_path)
l = [file for file in os.listdir(subj_folder) if file.endswith('gdf')]

print(l)

['record-[2024.03.10]_S9_1.gdf', 'record-[2024.03.10]_S7_1.gdf',
'record-[2024.03.10]_S8_2.gdf', 'record-[2024.03.10]_S10_2.gdf']

picked = l[0]
print(picked)

record-[2024.03.10]_S9_1.gdf

ch_list = ['Fp1', 'Fp2', 'AF3', 'AF4', 'F7', 'F3', 'Fz', 'F4', 'F8',
'FT7', 'FC3', 'FCz','FC4', 'FT8', 'T7', 'C3', 'Cz', 'C4', 'T8', 'TP7',
'CP3', 'CPz', 'CP4', 'TP8', 'P7', 'P3', 'Pz', 'P4', 'P8', 'O1', 'Oz',
'O2']
```

```
x = load_ds(subj_folder, picked)
print(x.annotations)
```

```
File path: /content/drive/MyDrive/EEG data/Subject 4/record-
[2024.03.10]_S9_1.gdf
<Annotations | 67 segments: 1010 (1), 33024 (17), 769 (8), 770 (8),
800 (33)>
```

```
channel_mapping = {old_name: new_name for old_name, new_name in
zip(x.ch_names, ch_list)}
x.rename_channels(channel_mapping)
```

```
<RawGDF | record-[2024.03.10]_S9_1.gdf, 32 x 120544 (241.1 s), ~35 kB,
data not loaded>
```

```
x.compute_psd(fmax=50).plot(picks="data", exclude="bads",
amplitude=False)

# Create a standard montage (e.g., 10-20 system)
montage = mne.channels.make_standard_montage('standard_1020')

# Plot the montage to visualize the electrode positions
fig, ax = plt.subplots(figsize=(6, 6))
_ = montage.plot(show_names=True, axes=ax)
_ = x.set_montage(montage)

events_from_annot, event_dict = mne.events_from_annotations(x)
# print(events_from_annot)
print(event_dict)
```

```
{'1010': 1, '33024': 2, '769': 3, '770': 4, '800': 5}
```

```
l_ev = event_dict['769']
r_ev = event_dict['770']
x_ev = event_dict['33024']
end_trial = event_dict['800']
needed_event = [l_ev, r_ev, x_ev, end_trial]
print(events_from_annot.shape)
```

```
(67, 3)
```

```
re = events_from_annot[::-1, 2].tolist()  # reverse
last_id = re.index(end_trial)
# delete the end_trail event
if last_id > 0:
  events_annot = events_from_annot[:-last_id]
else:
  events_annot = events_from_annot[:]

# Extract rows from events_annot that contain specified event values.
mask = np.isin(events_annot[:, 2], needed_event)
```

```
filtered_events = events_annot[mask]
# print(filtered_events)

# Time difference between
a = np.diff(filtered_events[:, 0])/x.info['sfreq']
# Durations
dur = round(np.mean(a[::2]),2)
print(dur)
```

```
6.21
```

```
unique, counts = np.unique(filtered_events[:,2], return_counts=True)
print(unique)
print(counts)
```

```
[2 3 4 5]
[17  8  8 33]
```

```
st = 0

epoch_l = mne.Epochs(x, events_from_annot, event_id=l_ev, tmin=st,
tmax=st+dur, baseline=None, preload=True)
epoch_r = mne.Epochs(x, events_from_annot, event_id=r_ev, tmin=st,
tmax=st+dur, baseline=None, preload=True)
epoch_x = mne.Epochs(x, events_from_annot, event_id=x_ev, tmin=st,
tmax=st+dur, baseline=None, preload=True)

print(epoch_l)
print(epoch_r)
print(epoch_x)
```

```
<Epochs |  8 events (all good), 0 – 6.21 s, baseline off, ~6.1 MB,
data loaded,
 '3': 8>
<Epochs |  8 events (all good), 0 – 6.21 s, baseline off, ~6.1 MB,
data loaded,
 '4': 8>
<Epochs |  16 events (all good), 0 – 6.21 s, baseline off, ~12.2 MB,
data loaded,
 '2': 16>
```

```
#filter with specific bandpass

# epoch_l.filter(l_freq=1, h_freq=40)
# epoch_r.filter(l_freq=1, h_freq=40)
# epoch_x.filter(l_freq=1, h_freq=40)

# alpha
epoch_l.filter(l_freq=8, h_freq=12)
epoch_r.filter(l_freq=8, h_freq=12)
epoch_x.filter(l_freq=8, h_freq=12)
```

```
# beta
# epoch_l.filter(l_freq=13, h_freq=30)
# epoch_r.filter(l_freq=13, h_freq=30)
# epoch_x.filter(l_freq=13, h_freq=30)
```

```
<Epochs |  16 events (all good), 0 – 6.21 s, baseline off, ~12.2 MB,
data loaded,
 '2': 16>
```

```
ep_n = 0 #@param {type:"integer"}

sub_dur = 2
stride = 0.25
sf = x.info['sfreq']

def sub_epochs(epochs):
  smaller_epochs = []
  for epoch in epochs:
    data = epoch[np.newaxis,:,:]
    # Calculate the number of smaller epochs that can be created
    n_epochs = (data.shape[2] - (sub_dur * sf)) // (stride * sf) + 1
    for i in range(int(n_epochs)):
        start_sample = int(i * (stride * sf))
        end_sample = start_sample + int(sub_dur * sf)
        smaller_epoch_data = data[:, :, start_sample:end_sample]

        # Create a new Epoch object with the smaller epoch data
        smaller_epoch = mne.EpochsArray(smaller_epoch_data,
info=epochs.info)
        smaller_epochs.append(smaller_epoch)

  # Combine all the smaller epochs into a single Epochs object
  smaller_epochs = mne.epochs.concatenate_epochs(smaller_epochs)
  return smaller_epochs
def create_label(size, lbl):
  return np.full(size, lbl)

mini_ep_l = sub_epochs(epoch_l)
mini_ep_r = sub_epochs(epoch_r)
mini_ep_x = sub_epochs(epoch_x)

print(len(mini_ep_l)/8)
print(len(mini_ep_r)/8)
print(len(mini_ep_x)/8)
# 1 Epoch --> 17 sub-epoch --> Use index | 2[Early] ---- 8 [Mid] ----
15[Late] |
```

```
17.0
17.0
34.0
```

```
ep_l = mini_ep_l.get_data()
ep_r = mini_ep_r.get_data()
ep_x = mini_ep_x.get_data()
```

```
<ipython-input-30-744d8960eed5>:1: FutureWarning: The current default
of copy=False will change to copy=True in 1.7. Set the value of copy
explicitly to avoid this warning
  ep_l = mini_ep_l.get_data()
<ipython-input-30-744d8960eed5>:2: FutureWarning: The current default
of copy=False will change to copy=True in 1.7. Set the value of copy
explicitly to avoid this warning
  ep_r = mini_ep_r.get_data()
<ipython-input-30-744d8960eed5>:3: FutureWarning: The current default
of copy=False will change to copy=True in 1.7. Set the value of copy
explicitly to avoid this warning
  ep_x = mini_ep_x.get_data()
```

```
lbl_l = create_label(ep_l.shape[0], 1)
lbl_r = create_label(ep_r.shape[0], 2)
lbl_x = create_label(ep_x.shape[0], 0)
```

```
print(ep_l.shape)
print(lbl_l.shape)
```

```
(136, 32, 1000)
(136,)
```

```
epoch_data = np.concatenate((ep_l, ep_r, ep_x),axis=0)
label_data = np.concatenate((lbl_l, lbl_r, lbl_x),axis=0)
# print(epoch_data)
print(len(label_data))
```

```
544
```

```
print(label_data)
```

###LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(epoch_data,
label_data, test_size=0.2, random_state=42)
X_train_flat = X_train.reshape(X_train.shape[0], -1)


lda = LinearDiscriminantAnalysis()
clf = Pipeline([("LDA", lda)])
clf.fit(X_train_flat, y_train)
```

```
accuracy = clf.score(X_test.reshape(X_test.shape[0], -1), y_test)
print("Classification accuracy: %f" % np.mean(accuracy))

Classification accuracy: 0.394495
```

###SVM

```
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(epoch_data,
label_data, test_size=0.2, random_state=42)
X_train_flat = X_train.reshape(X_train.shape[0], -1)

svc = SVC(kernel='rbf', probability=True)
clf = Pipeline([("SVM", svc)])
clf.fit(X_train_flat, y_train)
accuracy = clf.score(X_test.reshape(X_test.shape[0], -1), y_test)
print("Classification accuracy: %f" % np.mean(accuracy))

Classification accuracy: 0.477064
```

###CSP + LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit, cross_val_score
from sklearn.pipeline import Pipeline
from mne.decoding import CSP
```

####Without Cross-Valid

```
X_train, X_test, y_train, y_test = train_test_split(epoch_data,
label_data, test_size=0.2, random_state=42)
lda = LinearDiscriminantAnalysis()
csp = CSP(n_components=4, reg=None, log=True, norm_trace=False)
clf = Pipeline([("CSP", csp), ("LDA", lda)])
clf.fit(X_train, y_train)
accuracy = clf.score(X_test, y_test)
print("Classification accuracy: %f" % np.mean(accuracy))
```

####Cross-Valid

```
X_train, X_test, y_train, y_test = train_test_split(epoch_data,
label_data, test_size=0.2, random_state=42)
cv = ShuffleSplit(5, test_size=0.2, random_state=42)
cv_split = cv.split(X_train, y_train)
```

```
lda = LinearDiscriminantAnalysis()
csp = CSP(n_components=4, reg=None, log=True, norm_trace=False)

clf = Pipeline([("CSP", csp), ("LDA", lda)])
scores = cross_val_score(clf, X_train, y_train, cv=cv, n_jobs=None)

class_balance = np.mean(y_train == y_train[0])
class_balance = max(class_balance, 1.0 - class_balance)
print(
    "Classification accuracy: %f / Chance level: %f" %
(np.mean(scores), class_balance)
)

Classification accuracy: 0.721839 / Chance level: 0.737931
```

###CSP + SVM

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
cv = ShuffleSplit(5, test_size=0.2, random_state=42)
cv_split = cv.split(X_train, y_train)
svc = SVC(kernel='rbf', probability=True)
csp = CSP(n_components=4, reg=None, log=True, norm_trace=False)
scaler = StandardScaler()
clf = Pipeline([("CSP", csp), ("SVM", svc)])
scores = cross_val_score(clf, X_train, y_train, cv=cv, n_jobs=None)

# Printing the results
class_balance = np.mean(y_train == y_train[0])
class_balance = max(class_balance, 1.0 - class_balance)
print("Classification accuracies: " ,scores)
print("Classification accuracy: %f / Chance level: %f" %
(np.mean(scores), class_balance))

Classification accuracies:  [0.72413793 0.73563218 0.77011494
0.75862069 0.75862069]
Classification accuracy: 0.749425 / Chance level: 0.737931
```

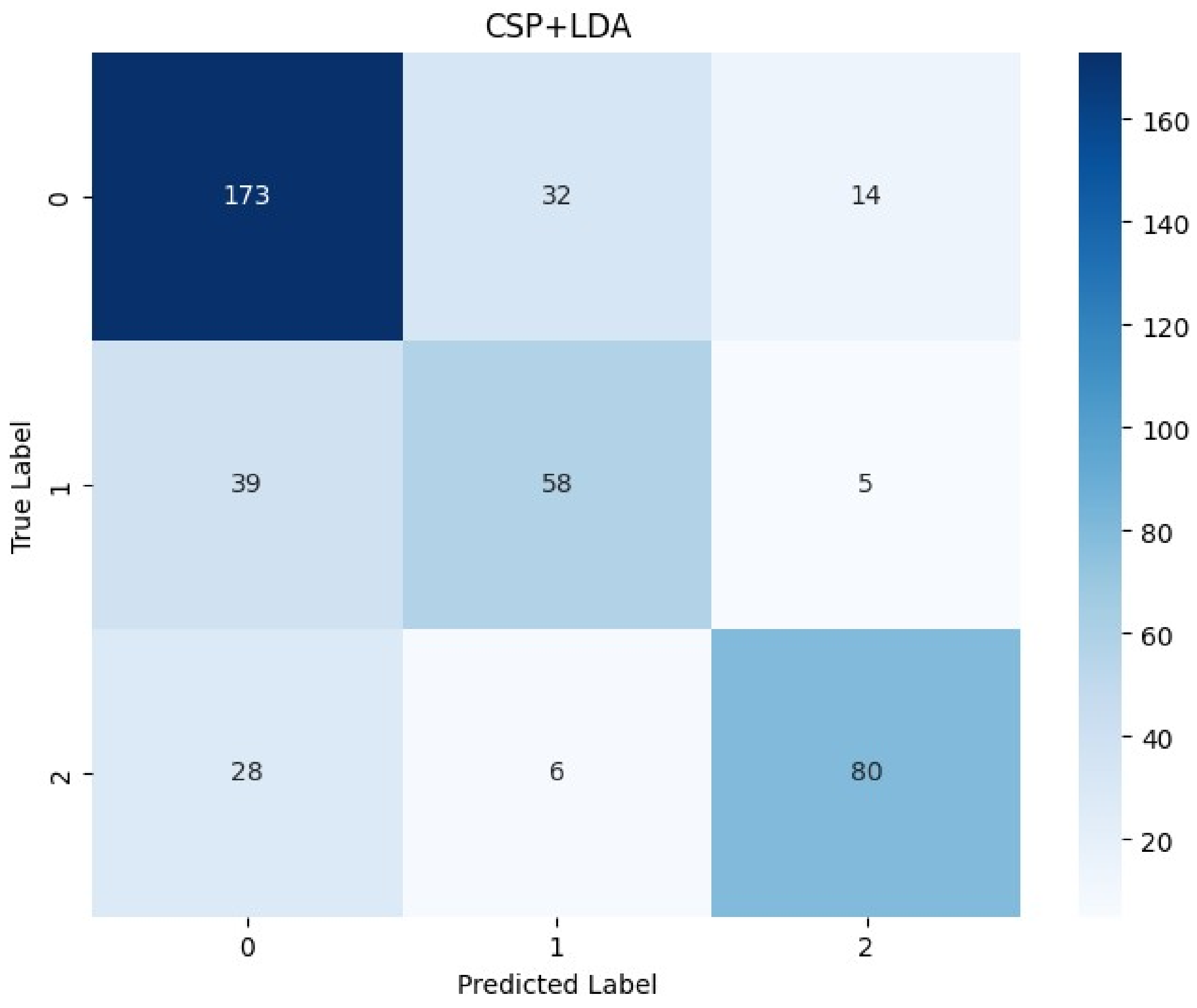
#Confusion matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_predict

cv = 5  # 這裡的5 可以根據需要調整
X_train, X_test, y_train, y_test = train_test_split(epoch_data,
label_data, test_size=0.2, random_state=42)
# X_train_flat = X_train.reshape(X_train.shape[0], -1)
# # 使用 cross_val_predict 進行交叉驗證預測
# clf = Pipeline([("CSP", csp), ("SVM", svc)])
clf = Pipeline([("CSP", csp), ("LDA", lda)])
# clf = Pipeline([("SVM", svc)])
# clf = Pipeline([("LDA", lda)])
y_pred = cross_val_predict(clf, X_train, y_train, cv=cv)

# 計算混淆矩陣
cm = confusion_matrix(y_train, y_pred)

# 可視化混淆矩陣
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['0',
'1', '2'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('CSP+LDA')
plt.show()
```



```
# 訓練模型
clf.fit(X_train, y_train)

# 預測測試集
y_pred = clf.predict(X_test)

# 計算混淆矩陣
cm = confusion_matrix(y_test, y_pred)

# 可視化混淆矩陣
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['foot', 'left_hand', 'right_hand'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

#Unsupervise Test

###PCA

```
from sklearn.decomposition import PCA, FastICA
from mne.decoding import UnsupervisedSpatialFilter
from sklearn.cluster import KMeans
pca = UnsupervisedSpatialFilter(PCA(5), average=False)
pca_data = pca.fit_transform(epoch_data)
print(pca_data.shape)
print(epoch_data.shape)
```

```
(544, 5, 1000)
(544, 32, 1000)
```

####K-Means

```
from sklearn.cluster import KMeans
# 將 PCA 處理後的數據展平為 2D
pca_data_2d = pca_data.reshape(pca_data.shape[0], -1)

# 初始化 K-Means 分群器，假設你希望分成 3 個群集
kmeans = KMeans(n_clusters=3, random_state=42)

# 對資料進行 K-Means 聚類
cluster_labels = kmeans.fit_predict(pca_data_2d)
```

####DBSCAN

```
from sklearn.cluster import DBSCAN
```

```python
dbscan = DBSCAN(eps=0.5, min_samples=10)
# 對資料進行 DBSCAN 聚類
cluster_labels = dbscan.fit_predict(pca_data_2d)
print(cluster_labels)

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Visualize the K-Means clustering results in 3D space
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot points with different colors based on cluster labels
for i in range(3):
    ax.scatter(pca_data_2d[cluster_labels == i, 0],
               pca_data_2d[cluster_labels == i, 1],
               pca_data_2d[cluster_labels == i, 2],
               label=f'Cluster {i}')

# Add labels and title
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('DBSCAN Clustering in 3D Space')

# Add legend
ax.legend()

# Show the plot
plt.show()
```

###ICA

```python
ica = UnsupervisedSpatialFilter(FastICA(5, whiten="unit-variance"),
average=False)
ica_data = ica.fit_transform(epoch_data)
print(ica_data.shape)
```

```
(544, 5, 1000)
```

####K-Means

```python
# 初始化 K-Means 分群器，假設你希望分成 3 個群集
kmeans = KMeans(n_clusters=3, random_state=42)
ica_data_2d = ica_data.reshape(ica_data.shape[0], -1)
cluster_labels = kmeans.fit_predict(ica_data_2d)
# print(cluster_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```

####DBSCAN

```
ica_data_2d = ica_data.reshape(ica_data.shape[0], -1)
dbscan = DBSCAN(eps=0.2, min_samples=8)
# 對資料進行 DBSCAN 聚類
cluster_labels = dbscan.fit_predict(ica_data_2d)
print(cluster_labels)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
for i in range(3):
    ax.scatter(ica_data_2d[cluster_labels == i, 0],
               ica_data_2d[cluster_labels == i, 1],
               ica_data_2d[cluster_labels == i, 2],
               label=f'Cluster {i}')

# 添加標籤和標題
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('DBSCAN Clustering in 3D Space')

# 添加圖例
ax.legend()

# ax.view_init(elev=0, azim=60)

# 顯示圖形
plt.show()

from sklearn.metrics import adjusted_rand_score, mutual_info_score,
fowlkes_mallows_score

# Assuming ground_truth_labels are the ground truth labels
# Assuming cluster_labels are the labels predicted by K-Means

# Adjusted Rand Index (ARI)
ari = adjusted_rand_score(label_data, cluster_labels)
print("Adjusted Rand Index (ARI):", ari)

# Mutual Information (MI)
mi = mutual_info_score(label_data, cluster_labels)
print("Mutual Information (MI):", mi)
```

```
# Fowlkes-Mallows Index (FMI)
fmi = fowlkes_mallows_score(label_data, cluster_labels)
print("Fowlkes-Mallows Index (FMI):", fmi)
```

```
Adjusted Rand Index (ARI): 0.0
Mutual Information (MI): 0.0
Fowlkes-Mallows Index (FMI): 0.6114319153500704
```

```
# ---------------------------------------------------------------------
machine
learning--------------------------------------------------------------
------
```