Mise en place des outils

`pip`

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py --user
```

Environnement virtuel

Installation de virtualenv

```
pip install --user virtualenv
```

EV pour le TP

Création

```
virtualenv tp-datasim-2020
```

Immersion/Activation

```
source tp-datasim-2020/bin/activate
```

Installation des bibliothèques

```
pip install mne matplotlib sklearn
```

MNE (https://mne.tools) est une bibliothèque didiée à la visualisation et l'analyse de signaux MEG, EEG, sEEG, ECoG.

matplotlib (https://matplotlib.org) est une bibliothèque de visualisation scientifique pour produire des images de qualité (notamment pour les publications scientifiques).

sklearn (https://scikit-learn.org) est une bibliothèque pour l'apprentissage machine.

Le jeu de données

Le jeu de données que nous allons utilisé aujourd'hui contient des données EEG acquisent lors de différentes tâches d'imagerie mentale motrice (IMM). Les 109 sujets ont réalisé 4 types de tâches d'IMM pendant 4 secondes, imaginer :

- bouger la main gauche
- bouger la main droite
- bouger les deux mains simultanément
- bouger les deux pieds simultanément

64 canaux EEG ont été enregistrés à 160Hz.

8-30Hz+CSP+LDA

Charger les données d'un sujet

Pour charger les données d'un sujet, utilisez la fonction suivante :

```
# import the different functionnalities necessary for your work
import numpy as np
import matplotlib.pyplot as plt

from mne import Epochs, pick_types, events_from_annotations
from mne.io import concatenate_raws, read_raw_edf
from mne.datasets import eegbci
from mne.decoding import CSP
from mne.channels import read_layout

from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import ShuffleSplit, cross_val_score

def load_data(subject):
    """
    Load data from a subject designated by his identifier.

    Input:

        subject: int between 1 and 109

    Output:

        raw: mne.Raw, structure containing EEG data

        events: numpy array (n_events, 3)
            first column: date of event in sample
            second column: duration of event
            thrid column: event code
    """
    assert 1 <= subject <= 109

    # dictionnary to specify the label and code of each event of interest
    event_id = dict(left=0, right= 1, hands=2, feet=3)
```

```
    # list of dictionnaries to specify the different tasks of interest
    task = [
            dict(T1=event_id['left'], T2=event_id['right']),
            dict(T1=event_id['hands'], T2=event_id['feet'])
        ]

    # list of dictionnaries to specify the different runs to load for one subject
    runs = [
            dict(id=4,  task=task[0]),
            dict(id=6,  task=task[1]),
            dict(id=8,  task=task[0]),
            dict(id=10, task=task[1]),
            dict(id=12, task=task[0]),
            dict(id=14, task=task[1])
        ]

    # load and concatenate the different files from the specified subject
    # download the files if necessary
    raws = list()
    events_list = list()
    for run in runs:
        # localize the file, download it if necessary
        filename = eegbci.load_data(subject, run['id'])
        # load its contain
        raw = read_raw_edf(filename[0], preload=True)
        events, _ = events_from_annotations(raw, event_id=run['task'])
        # accumulate the data
        raws.append(raw)
        events_list.append(events)
    # concatenate all data in two structures : one for EEG, one for the events
    raw, events = concatenate_raws(raws, events_list=events_list)

    # strip channel names of "." characters
    raw.rename_channels(lambda x: x.strip('.'))

    # delete annotations
    indices = [x for x in range(len(raw.annotations))]
    indices.reverse()
    for i in indices:
        raw.annotations.delete(i)

    return raw, events
```

Visualiser le signal EEG

Observer les signaux des sujets 1 et 2, essayer d'identifier les différents types d'artefacts :

- EMG.
- clignements et mouvements des yeux.
- décollement d'électrodes.
- signaux de mauvaise qualité.

Faites le pour des signaux bruts et filtrés entre 8 et 30Hz. Les artefacts ont-ils disparus ? Quelle est leur influence sur le signal ?

Étape de travail :

- Afficher le signal brut
- Faire vos observations sur le signal brut
- Filtrer les signaux
- Faire vos observations/comparaisons

Utiliser les méthodes Raw.plot, Raw.copy, Raw.filter

Évaluation du modèle

Nous allons essayer de discriminer les classes bouger la main droite, et bouger les pieds. Pour ce faire, vous utiliserez les algorithmes de CSP et LDA.

Évaluer le modèle par validation croisée pour les vingt premiers sujets de la base.

Les différentes étapes du traitement sont :

- Filtrer le signal entre 8 et 30 Hz
- Extraire les fenètres liées aux évènements : bouger la main droite et les pieds.
- Créer le modèle CSP + LDA
- Faire la validation croisée

Identifier un sujet pour qui la validation croisée tend à montrer que cela fonctionne et cela ne fonctionne pas.

Pour chacun de ces deux sujets, afficher les patterns retenu par le CSP. Que constatez vous ?

Utiliser les fonctionnalités suivantes :

- from mne import Epochs, pick_types
- from mne.decoding import CSP
- mne.decoding.CSP.plot_patterns
- from mne.channels import read_layout
- from sklearn.pipeline import Pipeline
- from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
- from sklearn.model_selection import ShuffleSplit, cross_val_score

---

Optimisation des hyper-paramètres

- Identifier les hyper-paramètres de cette méthode.
- Comment les régler ?

Élimination des artéfacts

Utiliser l'ICA pour éliminer les fenètres contaminés par des clignements des yeux:

- En sélectionnant les sources manuellement
- En sélectionnant une source qui est corrélée à la moyenne des électrodes les plus frontales