# Baseline

June 14, 2021

## 1 Hackathon baseline

We provide here a simple pipeline to read the data, train a Tangent Space Classifier and try naive transfer between sessions.

[1]:
```
%matplotlib inline

import os
import mne
import pandas as pd
from mne.externals.pymatreader import read_mat
import numpy as np
import matplotlib.pyplot as plt
import itertools
from glob import glob

import pyriemann

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score
```

**Here set the data_path to corresponding path on your computer**

[8]:
```
data_path = '/home/dcas/l.darmet/data/contest/comeptition_done'
n_subs = 4
n_sessions = 2
diff = ['MATBeasy', 'MATBmed', 'MATBdiff']
```

**Read channels names and position**

[10]:
```
electrodes = pd.read_csv(data_path + '/Electrodes/
 ↪chan_locs_standard',header=None, sep ='\t', names=['ch_names','x','y','z'])
electrodes.head()
```

[10]:
```
    ch_names        x       y       z
0        Fp1 -29.4370  83.917  -6.990
```

```
1         Fz   0.3122  58.512  66.462
2         F3 -50.2440  53.111  42.192
3         F7 -70.2630  42.474 -11.420
4        FT9 -84.0760  14.567 -50.429
```

**Covariance estimation** For robust covariance estimation, we take advantage of shrinkage. Here the Oracle Approximating Shrinkage (OAS) is used. #### Classifier We use a simple Logistic Regression (with a non-optimized L2 penalty) on Tangent Space Features, extracted with Pyriemann toolbox. #### Channel selection A manual and naive EEG channel selection is performed to use 13 electrodes, mostly frontal.

[ ]:

```
lr = LogisticRegression(C=1/10.)
clf = make_pipeline(pyriemann.estimation.Covariances(estimator='oas'),
                    pyriemann.classification.TSclassifier(clf=lr))

ch_slice = ['F7', 'F5', 'F3', 'F1', 'F2', 'F4', 'F6', 'AF3', 'AFz', 'AF4',␣
 ↪'FP1', 'FP2', 'FPz']
```

## 1.1 Single subject epochs classification

[ ]:

```
for sub_n, session_n in itertools.product(range(n_subs), range(n_sessions)):
    epochs_data = []
    labels = []
    for lab_idx, level in enumerate(diff):
        sub = 'P{0:02d}'.format(sub_n+1)
        sess = f'S{session_n+1}'
        path = os.path.join(os.path.join(data_path, sub), sess) + f'/eeg/
 ↪alldata_sbj{str(sub_n+1).zfill(2)}_sess{session_n+1}_{level}.set'
        # Read the epoched data with MNE
        epochs = mne.io.read_epochs_eeglab(path, verbose=False)
        # You could add some pre-processing here with MNE
        # We will just select some channels (mostly frontal ones)
        epochs = epochs.drop_channels(list(set(epochs.ch_names) -␣
 ↪set(ch_slice)))

        # Get the data and concatenante with others MATB levels
        tmp = epochs.get_data()
        epochs_data.extend(tmp)
        labels.extend([lab_idx]*len(tmp))

    epochs_data = np.array(epochs_data)
    labels =  np.array(labels)

    # Compute classification accuracy with 5-folds cross validation
    acc = cross_val_score(clf, X=epochs_data, y=labels, cv=5)
    print(f'Subject {sub} and session {session_n+1}: mean accuracy of {round(np.
↪mean(acc), 2)} with a standard deviation of {round(np.std(acc), 2)}')
```

## 1.2 Transfer from session 1 to session 2 for P01

For subject P01, a model is trained on session 1 and directly used for epochs of session 2

[ ]:

```
sub_n = 0
```

[ ]:

```
session_n = 0

epochs_data = []
labels = []
for lab_idx, level in enumerate(diff):
    sub = 'P{0:02d}'.format(sub_n+1)
    sess = f'S{session_n+1}'
    path = os.path.join(os.path.join(data_path, sub), sess) + f'/eeg/
↪alldata_sbj{str(sub_n+1).zfill(2)}_sess{session_n+1}_{level}.set'
    # Read the epoched data with MNE
    epochs = mne.io.read_epochs_eeglab(path, verbose=False)
    # You could add some pre-processing here with MNE
    # We will just select some channels (mostly frontal ones)
    epochs = epochs.drop_channels(list(set(epochs.ch_names) - set(ch_slice)))

    # Get the data and concatenante with others MATB levels
    tmp = epochs.get_data()
    epochs_data.extend(tmp)
    labels.extend([lab_idx]*len(tmp))

epochs_data = np.array(epochs_data)
labels =  np.array(labels)
# Train the model on all epochs from session 1
clf.fit(epochs_data, labels)
```

[ ]:

```
session_n = 1

epochs_data = []
labels = []
for lab_idx, level in enumerate(diff):
    sub = 'P{0:02d}'.format(sub_n+1)
    sess = f'S{session_n+1}'
    path = os.path.join(os.path.join(data_path, sub), sess) + f'/eeg/
↪alldata_sbj{str(sub_n+1).zfill(2)}_sess{session_n+1}_{level}.set'
    # Read the epoched data with MNE
    tmp = mne.io.read_epochs_eeglab(path, verbose=False)
    # You could add some pre-processing here with MNE
    # We will just select some channels (mostly frontal ones)
    epochs = epochs.drop_channels(list(set(epochs.ch_names) - set(ch_slice)))

    # Get the data and concatenante with others MATB levels
    tmp = epochs.get_data()
    epochs_data.extend(tmp)
    labels.extend([lab_idx]*len(tmp))

epochs_data = np.array(epochs_data)
labels =  np.array(labels)

# Use trained model to predict for all epochs of session 2 and compute accuracy
y_pred = clf.predict(epochs_data)
acc = accuracy_score(labels, y_pred)
print(f'Subject {sub} and transfer from session 1 to 2: mean accuracy of␣
 ↪{round(acc, 2)}.')
```

### Generate a CSV to submit

```
[ ]: submission = pd.DataFrame({'epochID':np.arange(len(y_pred)), 'prediction' :␣
      ↪y_pred})
     submission.head()
```

```
[ ]: submission.to_csv("submission.csv",header=True,index=False)
```