Jul 07, 2025

Filter Bank CSP with Riemannian Weighting for Disability-Centric Motor Imagery Brain Computer Interface

Brain Informatics

DOI
https://dx.doi.org/10.17504/protocols.io.kxygx4mekl8j/v1

Souissi Jihen[1]

[1]FSM Monastir

Souissi Jihen
FSM Monastir

OPEN ACCESS

DOI: https://dx.doi.org/10.17504/protocols.io.kxygx4mekl8j/v1

External link: https://doi.org/10.1186/s40708-026-00295-0

Protocol Citation: Souissi Jihen 2025. Filter Bank CSP with Riemannian Weighting for Disability-Centric Motor Imagery Brain Computer Interface. **protocols.io** https://dx.doi.org/10.17504/protocols.io.kxygx4mekl8j/v1

Manuscript citation:
Jihen S, Karmani S, Belwafi K, Jemmali M, Djemal R (2026) Filter bank CSP with Riemannian weighting for disability-centric motor imagery brain computer interface. Brain Informatics 13(1). doi: 10.1186/s40708-026-00295-0

Protocol status: Working
We use this protocol and it's working

Created: July 06, 2025

Last Modified: July 07, 2025

Protocol Integer ID: 221808

Keywords: centric motor imagery brain computer interface, centric motor imagery brain computer interface this protocol, filter bank csp with riemannian weighting, bci, particular the feature extraction phase, motor imagery, feature extraction phase, filter bank csp, eeg, csp method, modified csp method, disability, processing signal, riemannian weighting

Abstract

This protocol is devoted to describing the steps involved in processing signal-based (EEG) motor imagery (MI) for brain-computer interfaces (BCI), from pre-processing to classification. Emphasis is placed on the various methods used, in particular the feature extraction phase carried out over several frequency bands using a modified CSP method.

Intoduction

1 Brain-computer interfaces (BCI) based on motor imagery (MI) enable users to interact with external
systems using only their brain activity, without muscle movement.
Electroencephalography (EEG) is widely used in BCI applications due to itsnon-invasive nature .
This protocol describes a comprehensive methodology for EEG signal processing using the Common Spatial Patterns (CSP) algorithm, combined with the OVR approach for multi-class context adaptation, in a non-Euclidean space. This is a Riemannian space with a variable metric for EEG signals. The approach includes multiband filtering of EEG signals, spatial feature extraction using modified CSP, named RWFBCSP, to improve class separability by incorporating Riemannian similarity weighting. We apply classification techniques such as Linear Discriminant Analysis (LDA), Random Forest Classifier (RFC) and Multilayer Perceptron (MLP) to distinguish motor imagery tasks. This methodology is validated using BCI Competition IV data (dataset 2a).

Tools used

2
- **dataset**: BCI Competition IV set 2a
- **Langage :** Python 3.6
- **libraries :** MNE, NumPy, scikit-learn and SciPy

EEG signal pre-processing

3
- The development process in Python consists in loading the EEG data with the dataset extension (.gdf).

- Segmentation of EEG data into epochs synchronized to the stimuli (tmin/tmax setting).as well as the choice of motor tasks (event_id) corresponding to each class.

```python
epochs = mne.Epochs(raw_data, events, event_id=stims, tmin=-0.5, tmax=1.5, baseline=(-0.2, 0), preload=True)
```

- Band-pass filtering using a FIR butterworth filter

EEG feature extraction(CSP Model)

4 This step consists in developing a customized and optimized function, based on the classic CSP model, for the extraction of discriminating features related to different motor

tasks.
The function named RW-FBCSP (Riemannian-Weighting Filter Bank Common Spatial Pattern), due to the use of the CSP model with the OVR approach in a Riemannian space with Similarity Weighting.
The filtered EEG signals are passed through this function to obtain a new, more compact and informative representation of the signals (reduced feature vector), suitable for the classification stage.

Classification step

5 The feature vectors obtained for each subject and the associated labels are used to train the three classifiers: LinearDiscriminantAnalysis , RandomForestClassifier and MLPClassifier.
Each classifier is trained and tested with each person's data, and for each subject the Accuracy, Precision, Recal and F1-score values are calculated and stored in global lists.
These values are averaged to obtain the average test results for the nine subjects.
For better performance, a majority voting method is used. This technique consists in creating a pipeline to exploit the results of the 3 classifiers to give a final decision.
Finally, a portion of the code displays a preview of the results of merging the predictions from the three classifiers.

- exemple of voting classifier

```
>>> Voting classifier - Subject 1

Individual classifier decisions (first 10 samples):
Index | LDA | MLP | RF  | Final Decision |  Majority vote
-------------------------------------------------------------
    0 |   2 |   0 |   2 |              2 | 2 (2 votes)
    1 |   1 |   2 |   1 |              1 | 1 (2 votes)
    2 |   1 |   1 |   2 |              1 | 1 (2 votes)
    3 |   0 |   0 |   0 |              0 | 0 (3 votes)
    4 |   1 |   1 |   1 |              1 | 1 (3 votes)
    5 |   0 |   0 |   0 |              0 | 0 (3 votes)
    6 |   1 |   1 |   1 |              1 | 1 (3 votes)
    7 |   2 |   2 |   2 |              2 | 2 (3 votes)
    8 |   1 |   1 |   1 |              1 | 1 (3 votes)
    9 |   3 |   3 |   2 |              3 | 3 (2 votes)
```

Evaluation step

6 - Metric used: Accuracy, Precision, Recal and F1-score

- Method: Cross-validation

```python
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

- Average scores for the 9 subjects

Comments

7

- The processing steps are applied independently to the training and test sets for each subject.
- The extracted training and test data are used to train the classifiers and evaluate their performance, respectively. A validation step is used to optimize the hyperparameters, so the data are divided as follows: all test data are for evaluation. The training set is divided into a subset used for classifier training (80%) and another dedicated to hyperparameter validation (20%).
  for KFold(n_splits=k) : **Validation set =1/k (1 of k "folds") = 1/5 = 20% (if k=5)**
  Training set =(k-1)/k = 4/5 = 20% (if k=5)
- After each classification step (performed by one of the three classifiers), a piece of code is used to present the classification model's performance in a structured way. Pandas.DataFrame is used to present summary tables.
- Example of a summary table: test results obtained by RFC

```
Test Metrics per Subject:

  Subject  Accuracy (%)  Precision (%)  Recall (%)  F1-score (%)
 Person 1         81.25          83.27       81.25         81.47
 Person 2         68.06          68.81       68.06         68.07
 Person 3         90.62          90.68       90.62         90.58
 Person 4         80.21          80.59       80.21         80.14
 Person 5         72.22          73.80       72.22         72.46
 Person 6         69.44          69.56       69.44         69.38
 Person 7         89.24          89.45       89.24         89.24
 Person 8         86.46          86.56       86.46         86.49
 Person 9         90.62          90.98       90.62         90.64

Average Test Metrics Across All Subjects:
Accuracy:  80.90%
Precision: 81.52%
Recall:    80.90%
F1-score:  80.94%
```

Summary of RFC results