Published in A Beginner's Guide to Brain-Computer Interfaces

Tim de Boer Follow

Sep 7, 2022 · 5 min read · ✦ · Listen

Effective Machine Learning for Brain Computer Interfaces With Code Examples

Explaining Common Spatial Patterns and Riemannian Geometry — A Beginner's Guide to Brain-Computer Interfaces (part 9)

After we applied frequency band filtering, we are almost ready to let a ing model learn

and predict motor imagery (MI) states!

In this blog post we first explain another kind of feature engineering, called spatial filtering, after which we can finally talk about applying machine learning (ML) to get our predictions. Both steps are easily combined in Python using scikit-learn, and in the last section we will show the code (spoiler: it can be done in only **3 lines of code!**).

Let's dive in!

Figure 1.

Spatial filtering

EEG measures your brain activity from the scalp. Because of the distance between brain and electrodes, i.e., the scalp, relevant signals we retrieve from EEG data can be smeared around multiple EEG electrodes. In combination with the fact that EEG only has a limited amount of electrodes, we can say that EEG has poor spatial resolution. In order to extract or recover the relevant original brain signals, information of several channels can be combined using weighted linear combinations. This process is called spatial filtering [Lotte, 2014].

Two commonly used spatial filtering methods are the Common Spatial Patterns filter (CSP) and Riemannian Geometry (RG). Both have reached state-of-the-art performance on various BCI tasks. In the following sections, both methods will be explained.

> *This blog post serves as an intuitive explanation to both methods., and for mathematical details the interested reader is referred this paper for CSP, and this paper for RG.*

Common Spatial Pattern

CSP transforms the EEG data by minimizing the variance of one class, and simultaneously maximizing the variance of the other class.

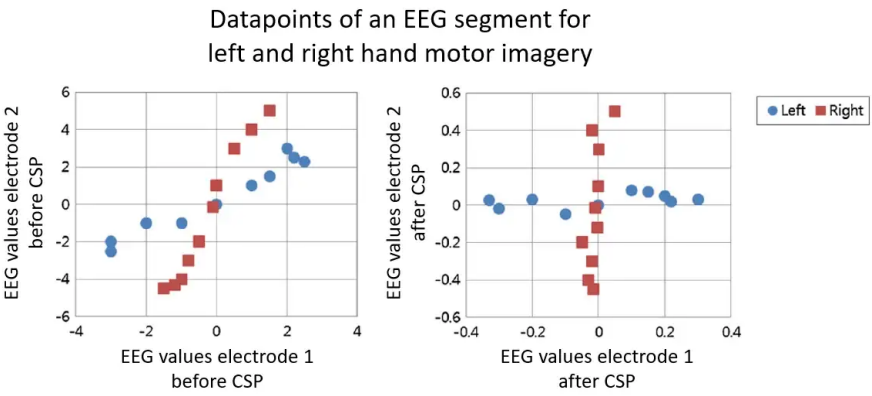

Figure 2: Data transformation by CSP. Image adapted from this study.

An example of the data transformation by CSP for two-channel EEG data is presented in Figure 2. The axes in the left graph represent electrical potential values measured by two EEG channels, with each point representing a datapoint of one segment of EEG data (for example, a segment of 1 second). In the right graph, the datapoints are presented in the new space after applying the CSP filters.

CSP can also act as a dimensionality reduction algorithm, as data from multiple electrode channels can be transformed to any chosen number of axes, which are ordered from high to low by the difference in variance between classes. One can choose the number of returned axes as the *components* parameter in CSP, and this is often chosen as a lower amount than

the amount of electrode channels. Lastly, CSP has been developed for binary classification, but an extension of CSP has been developed for multiclass classification.

Riemmanian Geometry

Just as CSP, RG has reached state-of-the-art performance for multiple BCI tasks. An additional advantage of RG is that it does not have any parameters to tune.

RG maps data on a geometrical curved space, called a manifold. From EEG data (Figure 3a), a covariance matrix is calculated (Figure 3b), which is used to map the data to the RG manifold for each segment (Figure 3c).

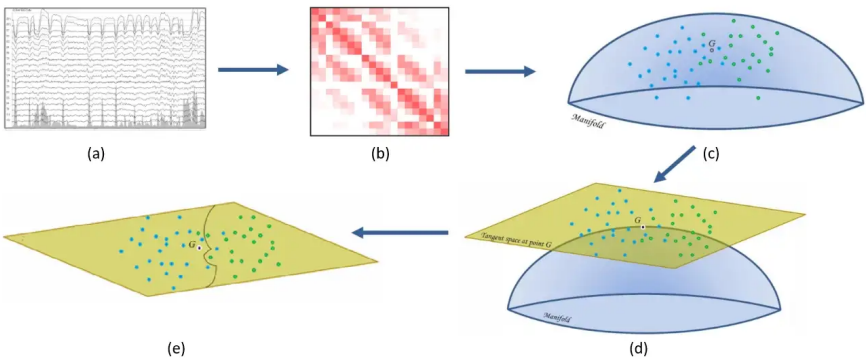

Figure 3: Riemannian Geometry explained. From EEG data (a) to covariance matrices (b) to the RG manifold (c) to creation of the

tangent space (d) which can then be used by ML to create a decision line (e). Adapted from this presentation.

Using labeled training data, a mean covariance matrix can be constructed for each class, visually presented in the manifold in Figure 3c as *G*. A property of this manifold is that it can be locally approximated by a hyperplane, which is named a tangent space (TG) (Figure 3d). Intuitively, an image of earth from space can be seen as a manifold, and a 2D satellite image of a part of the earth can be seen as a TG. The TG is useful, as traditional ML algorithms assume that data is presented in a hyperplane. The TG can serve as a hyperplane (Figure 3e), while the RG manifold is not a hyperplane.

So, which ML models can we apply?

Machine Learning for motor imagery prediction

After spatial filtering, the next step in the MI-BCI pipeline is classification of the MI state using ML.

As data is already heavily processed after applying CSP or RG, often traditional linear ML algorithms are applied, like linear discriminant analysis (LDA),

Support Vector Machine (SVM), an a somewhat more complex algorithm Random Forest (RF) too.

Let's start coding!

Right know we know the ingredients of our pipeline for motor imagery prediction. Let's put it into code!

First of all, let's talk about the data format. Our data is essentially a 3D array, with multiple segments of a (channels, timepoints) format, and scikit-learn likes to work with Numpy arrays. For this purpose, the numpy function *stack* can come in handy.

> *np.stack: Join a sequence of arrays along a new axis.*

So, by putting our segments (each an np.array of (channels, timepoints) into a list or array, we can use np.stack in order to get our 3D array.

To apply CSP, we need to install the MNE library. Then, we import CSP from mne.decoding.

> *MNE-Python: Open-source Python package for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, sEEG, ECoG, NIRS, and more.*

Regarding Riemannian Geometry, a library called PyRiemann has been developed.

> *PyRiemann: a Python machine learning package based on scikit-learn API. It provides a high-level interface for processing and classification of multivariate time series through the Riemannian geometry of symmetric positive definite (SPD) matrices.*

Then, we can use the scikit-learn function *Pipeline* together with the ML models provided by scikit-learn to stitch everything neatly together as follows.

```
1  from sklearn.discriminant_analysis import
   LinearDiscriminantAnalysis as LDA
2  from sklearn.ensemble import RandomForestClassifier
   as RFC
3  from sklearn.pipeline import Pipeline
4  from mne.decoding import CSP
5  from pyriemann.estimation import Covariances
6  from pyriemann.tangentspace import TangentSpace
7
8  csp = Pipeline(steps=[('csp', CSP()),('lda',
   LDA())])
```

As you can see, the code essentially boils down to only 3 lines: initialization of the pipeline, fit, and predict. That's quite easy!

When I started out with BCIs, I did not know anything about these algorithms, and applied traditional feature engineering and ML. The results were pretty bad! By reading this blog post, I hope now you have a grasp of how to effectively use ML for EEG data using specialized algorithms like CSP and RG.

Please check out the publication page, where more practical BCI tutorials like this one will be posted in the future, and give me a follow to be notified for further posts!

Machine Learning Brain Computer Interface Eeg

Neurotechnology Bci

Get an email whenever Tim de Boer publishes.

Your email

Subscribe

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.