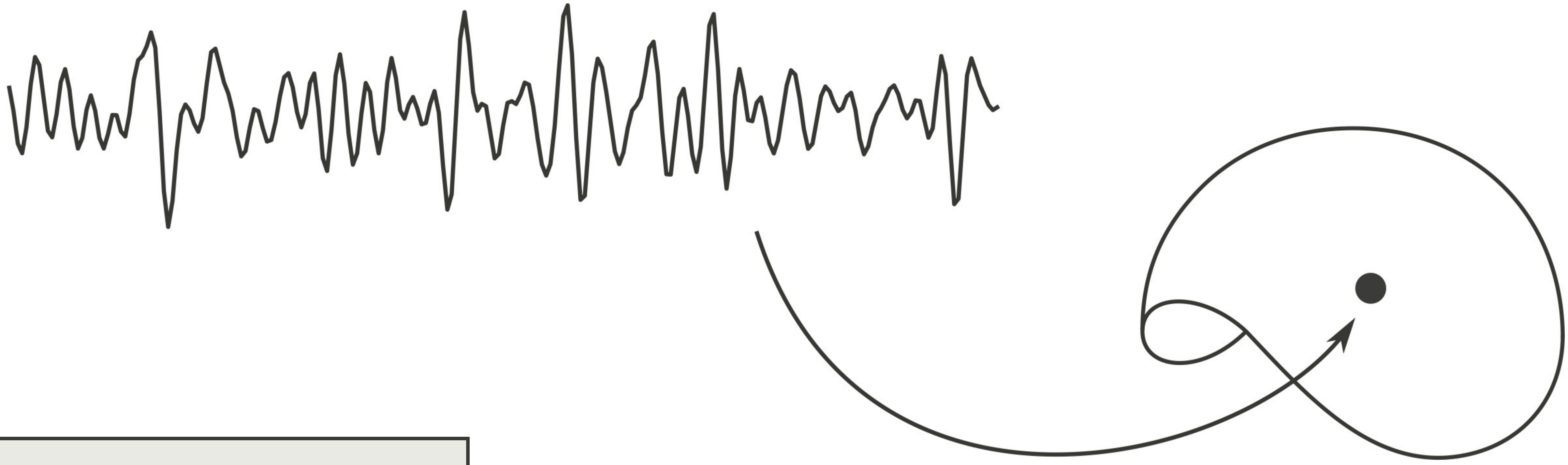
A guided tour on Riemannian methods for BCI with pyRiemann



Pedro L. C. Rodrigues

In collaboration with S. Chevallier, A. Gramfort, and Q. Barthélemy

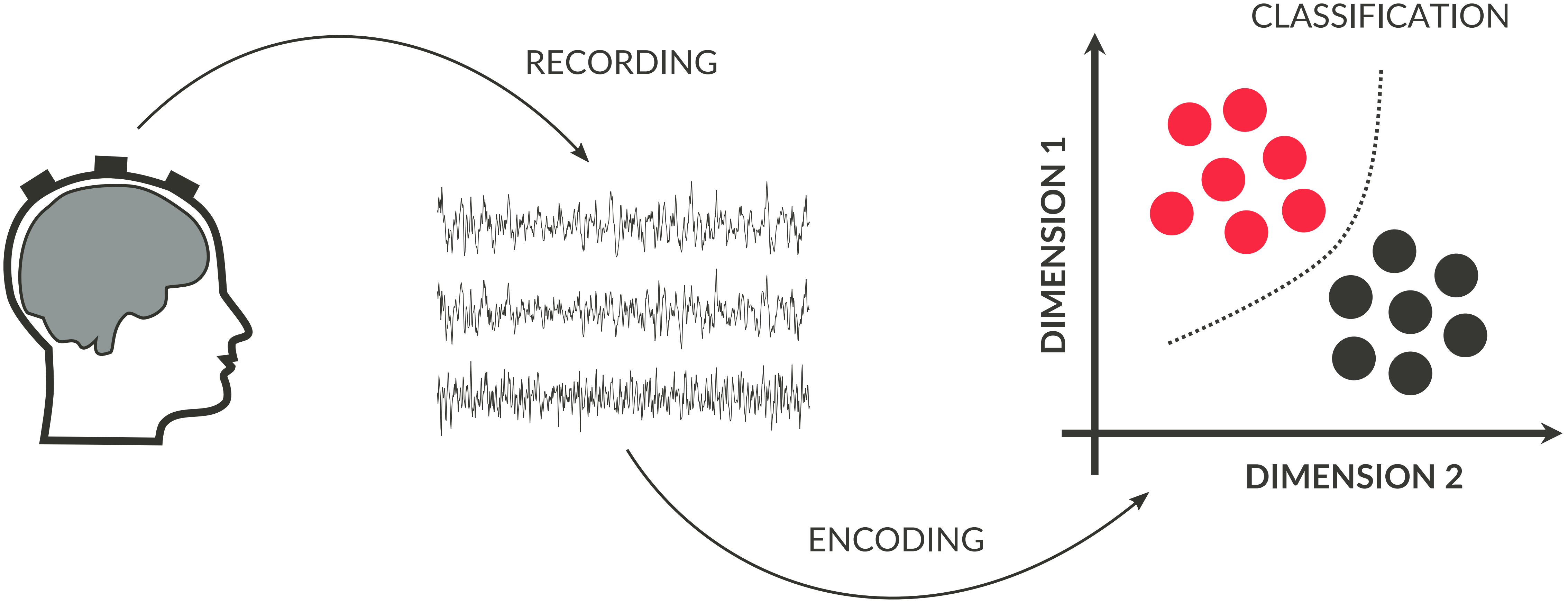
● INTRODUCTION

OVERVIEW

USE CASES

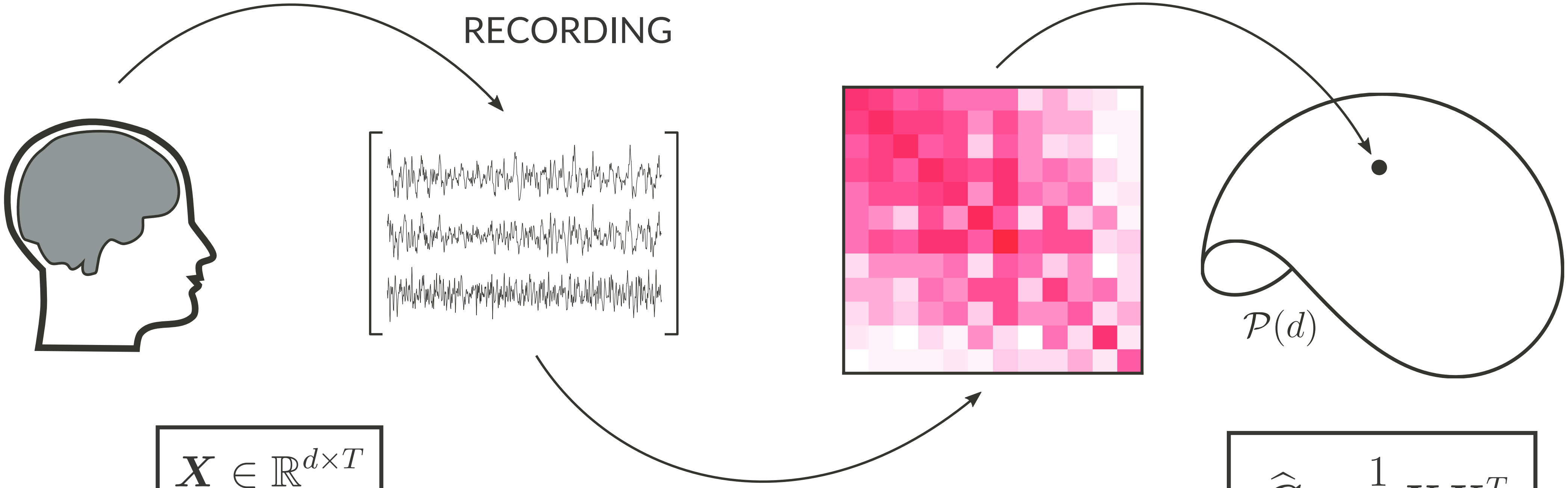
PERSPECTIVES

A typical BCI system is decomposed into **three parts**: recording, encoding, and classification



The RG framework fixes an adequate form of encoding and a way of classifying data

We encode the statistical features of the recording into a spatial covariance matrix



$$\boldsymbol{X} \in \mathbb{R}^{d \times T}$$

(zero-mean time series with *T* samples)

$$\widehat{\boldsymbol{C}} = \frac{1}{T}\boldsymbol{X}\boldsymbol{X}^T$$

(sample covariance matrix)

And then the classification is carried out on a set of matrices

$$\mathcal{D} = \Big\{(\boldsymbol{C}_1, y_1), \ldots, (\boldsymbol{C}_N, y_N)\Big\}$$

Consider the following python script using MOABB and pyRiemann

```python
# choose dataset
dataset = Weibo2014()

# choose paradigm
paradigm = LeftRightImagery()

# extract epochs from the dataset following the paradigm setup
X, labels, meta = paradigm.get_data(dataset=dataset, subjects=[1])

# estimate covariance matrices from the epochs
covs = Covariances(estimator='lwf').fit_transform(X)
```

note the scikit-learn API

How should we train a classifier based on `covs` and `labels`?

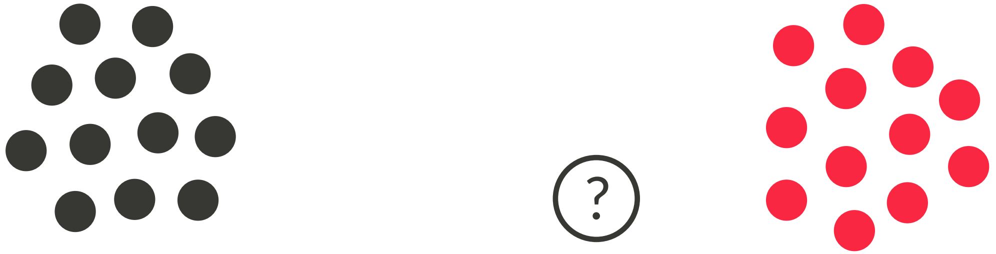
↳ **Naïve idea:** use a nearest centroid classifier based on Euclidean distance of matrices

How should we train a classifier based on `covs` and `labels`?

↳ **Naïve idea:** use a nearest centroid classifier based on Euclidean distance of matrices

$$\mathcal{D} = \left\{ (\boldsymbol{C}_1^{(\ell)}, \texttt{left}), \ldots, (\boldsymbol{C}_N^{(\ell)}, \texttt{left}), (\boldsymbol{C}_1^{(r)}, \texttt{right}), (\boldsymbol{C}_N^{(r)}, \texttt{right}) \right\}$$

Each point represents the covariance matrix of the signal recorded in a BCI epoch

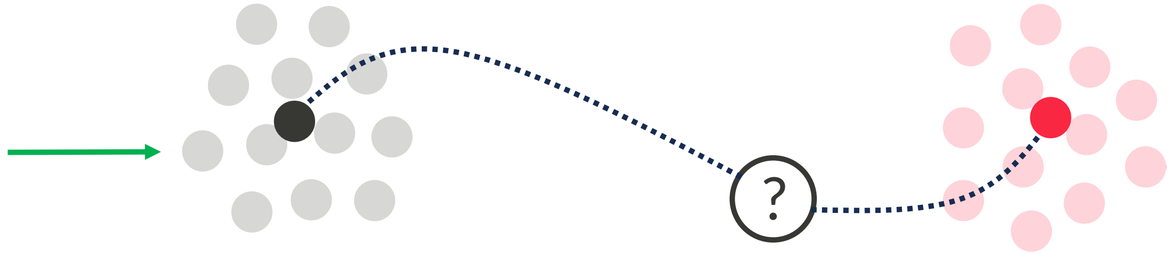


How should we train a classifier based on `covs` and `labels`?

↳ **Naïve idea:** use a nearest centroid classifier based on Euclidean distance of matrices

$$\mathcal{D} = \left\{ (\boldsymbol{C}_1^{(\ell)}, \texttt{left}), \dots, (\boldsymbol{C}_N^{(\ell)}, \texttt{left}), \color{red}{(\boldsymbol{C}_1^{(r)}, \texttt{right}), (\boldsymbol{C}_N^{(r)}, \texttt{right})} \right\}$$

Each point represents the covariance matrix of the signal recorded in a BCI epoch



Compare the Euclidean distance of ? to each centroid and choose the smallest

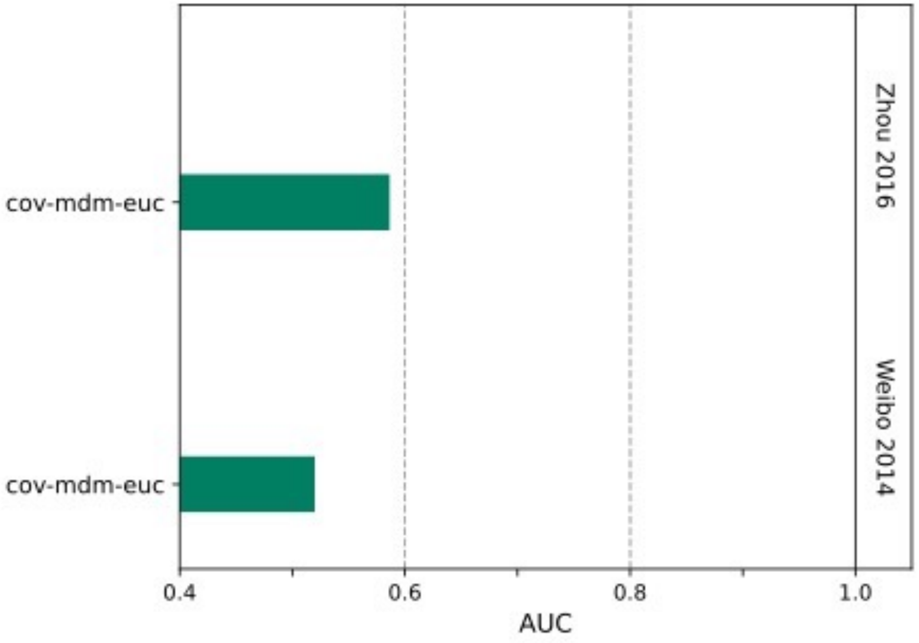
In pyRiemann this is called the Minimum Distance to Mean (MDM) classifier

```
pipeline = make_pipeline(Covariances(estimator='lwf'), MDM(metric='euclid'))
```

note the scikit-learn API

Consider the following python script using MOABB and pyRiemann

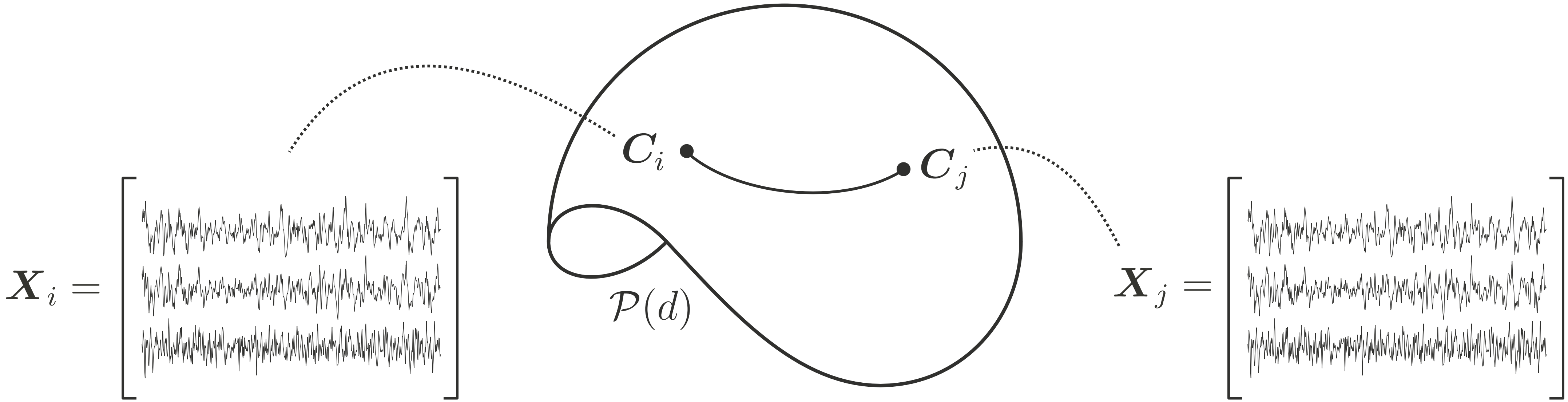
```
# choose which datasets to consider from MOABB
datasets = [Weibo2014(), Zhou2016()]
paradigm = LeftRightImagery()

# instantiate the pipelines
pipelines = {}
pipelines['cov-mdm-euc'] = make_pipeline(
Covariances(estimator='lwf'), MDM(metric='euclid’))

# setup the evaluation procedure
evaluation = WithinSessionEvaluation(
       paradigm=paradigm,
       datasets=datasets,
       overwrite=True)

# run the valuation process
results = evaluation.process(pipelines)
```



The problem with this approach is that we loose the **structure** of the covariance matrices

↳ Covariance matrices are symmetric positive definite matrices and have a lot of structure

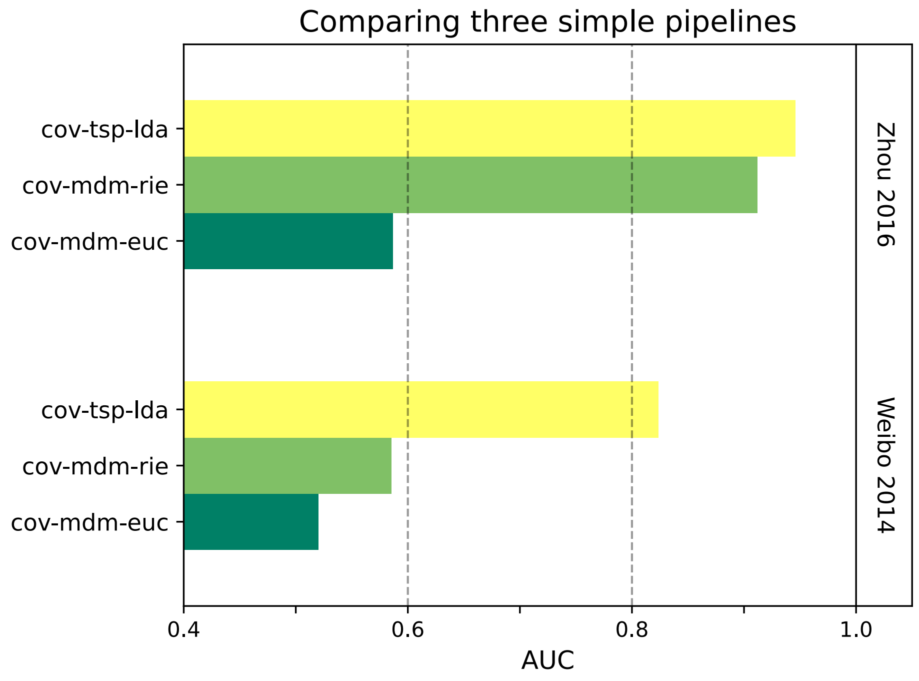
The Riemannian framework handles these matrices respecting their **intrinsic geometry**



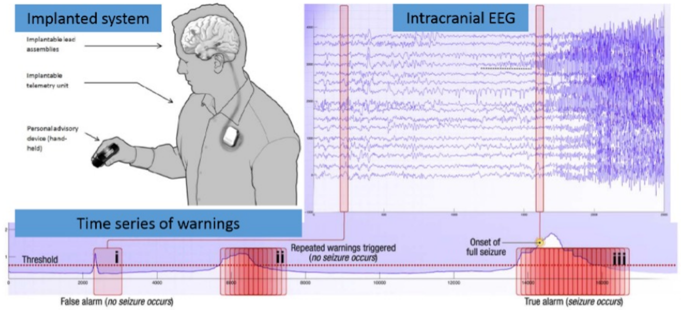
This means we should use MDM with a **Riemannian distance** between matrices !

Consider the following python script using MOABB and pyRiemann

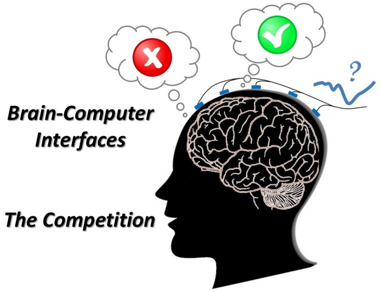
```
# choose which datasets to consider from MOABB
datasets = [Weibo2014(), Zhou2016()]
paradigm = LeftRightImagery()

# instantiate the pipelines
pipelines = {}
pipelines['cov-mdm-euc'] = make_pipeline(
Covariances(estimator='lwf'), MDM(metric='euclid’))

pipelines['cov-mdm-rie'] = make_pipeline(
Covariances(estimator='lwf'), MDM(metric='riemann’))

pipelines['cov-tsp-lda'] = make_pipeline(
Covariances(estimator='lwf’),
TSclassifier(metric='riemann'))
```



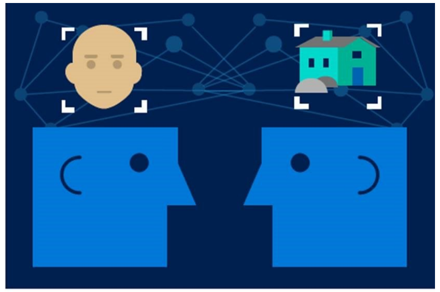
RG is a theoretical framework that has demonstrated quite good results in practice!


Implanted system
Implantable lead assemblies
Implantable telemetry unit
Personal advisory device (hand-held)
Intracranial EEG
Time series of warnings
Threshold
i
ii
iii
Repeated warnings triggered (no seizure occurs)
Onset of full seizure
False alarm (no seizure occurs)
True alarm (seizure occurs)

Seizure prediction competition (2016)



BCI Challenge @ NER (2015)



Microsoft decoding brain signal (2016)

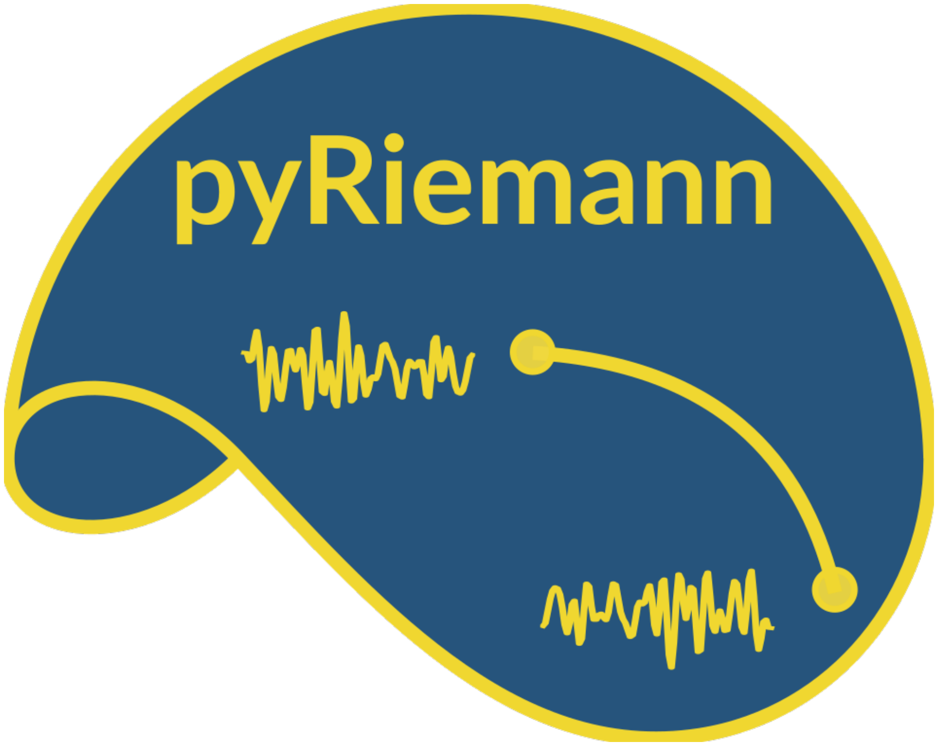
↳ All three competitions (and many others!) were won using tools from RG 😎

The brain behind all this?

Alexandre Barachant
https://alexandre.barachant.org/

↳ Yep, the same guy who started MOABB...

The success of RG methods in BCI pushed Alex to consolidate his codes into a package...



```
pip install pyriemann
```

Documentation: pyriemann.readthedocs.io

GitHub: github.com/pyRiemann/pyRiemann

From its inception in 2015 to now, many people made important contributions to the code

The current core team of developers is…

Sylvain Chevallier
sylvchev

Quentin Barthélemy
qbarthelemy

Alexandre Gramfort
agramfort

Pedro L. C. Rodrigues
plcrodrigues

YOU?

And we have a reference to **cite when using pyRiemann** in your research 😉

↳ Barachant et al. (2023) – https://zenodo.org/record/7642689

INTRODUCTION

● OVERVIEW

USE CASES

PERSPECTIVES

OVERVIEW

pyRiemann is based on the scikit-learn API and has many functionalities such as

Spatial filtering
CSP, Xdawn, SPoC, etc

SPD Matrices Estimation
Covariances, Hankel, etc

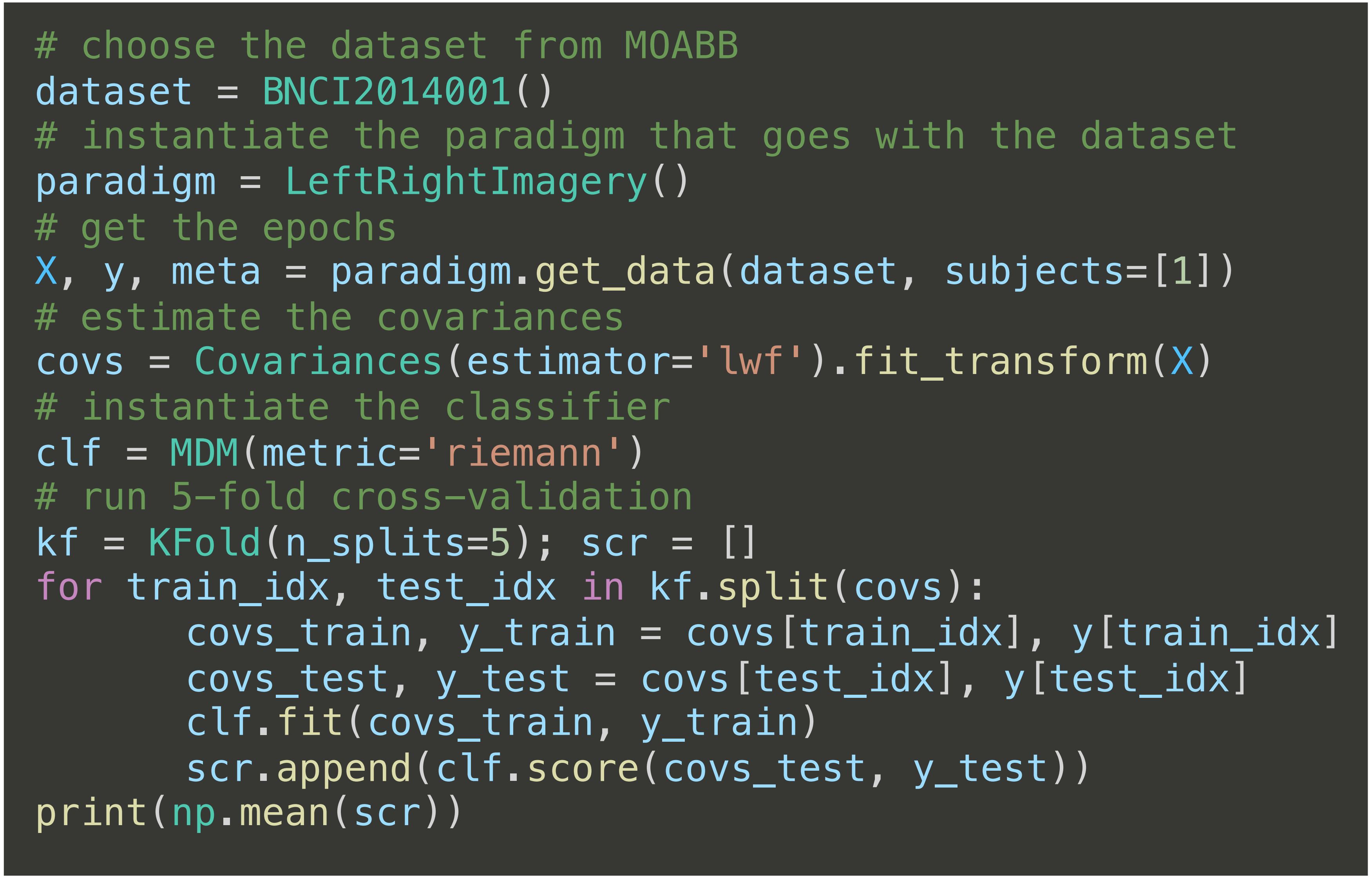
Classification
MDM, kNN, TSClassifier, etc

Transfer Learning
Parallel transport, RPA, MDWM

see pyriemann.readthedocs.io/en/latest/api.html for more

It is easy to use scikit-learn functions (e.g. KFold, GridSearchCV, etc)

Suppose you want to classify MI data using the RG framework



```
# choose the dataset from MOABB
dataset = BNCI2014001()
# instantiate the paradigm that goes with the dataset
paradigm = LeftRightImagery()
# get the epochs
X, y, meta = paradigm.get_data(dataset, subjects=[1])
# estimate the covariances
covs = Covariances(estimator='lwf').fit_transform(X)
# instantiate the classifier
clf = MDM(metric='riemann')
# run 5-fold cross-validation
kf = KFold(n_splits=5); scr = []
for train_idx, test_idx in kf.split(covs):
       covs_train, y_train = covs[train_idx], y[train_idx]
       covs_test, y_test = covs[test_idx], y[test_idx]
       clf.fit(covs_train, y_train)
       scr.append(clf.score(covs_test, y_test))
print(np.mean(scr))
```

Note that for MI we can estimate all of the covariances upfront

Run the usual KFold cross validation using tools from scikit-learn

Suppose you want to classify P300 data using the RG framework

```
# choose the dataset from MOABB
dataset = bi2012(Training=True, Online=False)
# instantiate the paradigm that goes with the dataset
paradigm = P300()
# get the epochs
X, y, meta = paradigm.get_data(dataset, subjects=[1])
# instantiate the classification pipeline
pipeline = make_pipeline(
      XdawnCovariances(nfilter=4, estimator='lwf’),
      TSclassifier(metric='riemann', clf=LDA()))
# run 5-fold cross-validation
kf = KFold(n_splits=5); scr = []
for train_idx, test_idx in kf.split(X):
      X_train, y_train = X[train_idx], y[train_idx]
      X_test, y_test = X[test_idx], y[test_idx]
      pipeline.fit(X_train, y_train)
      scr.append(pipeline.score(X_test, y_test))
print(np.mean(scr))
```

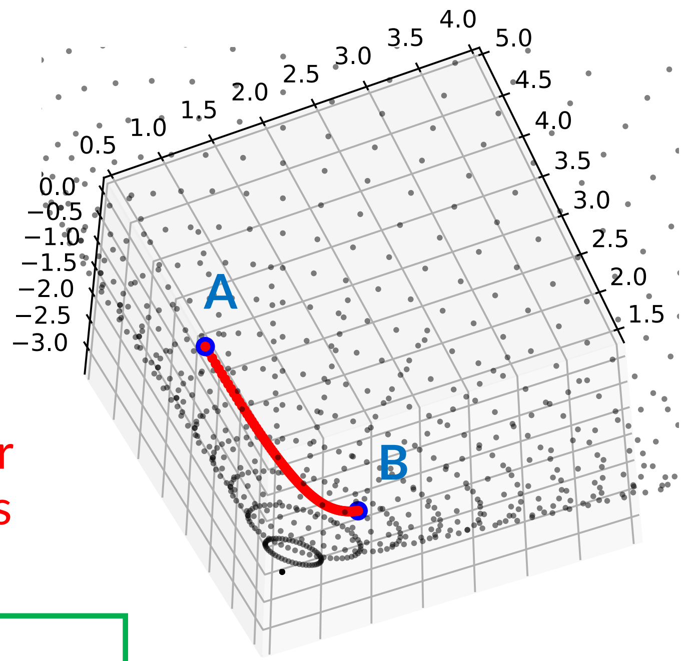
Use the pipeline constructor from scikit-learn

Note that for P300 the covariances are estimated inside the CV folds pyRiemann functions are based on fundamental concepts from RG for SPD matrices such as

geodesic path

$$\boxed{\boldsymbol{C}(\gamma) = \boldsymbol{A}^{1/2}\left(\boldsymbol{A}^{-1/2}\boldsymbol{B}\boldsymbol{A}^{-1/2}\right)^{\gamma}\boldsymbol{A}^{1/2}}$$

```
# walk over the geodesic between A and B
C = []
for gamma in np.linspace(0, 1, 100):
C.append(geodesic_riemann(A, B, gamma))
```



Geodesic path for 2x2 SPD matrices

geodesic distance

$$\boxed{\delta_R^2(\boldsymbol{A}, \boldsymbol{C}(\gamma)) = \|\log(\boldsymbol{A}^{-1/2}\boldsymbol{C}(\gamma)\boldsymbol{A}^{-1/2})\|_F^2}$$

```
# calculate the distances
d = [distance_riemann(A, Ci) for Ci in C]
```

pyRiemann has also other types of distances between SPD matrices

OVERVIEW

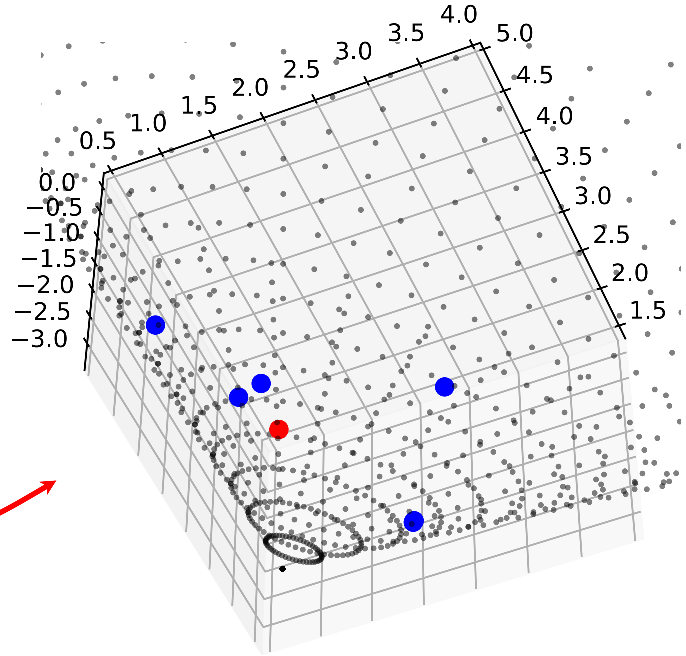
pyRiemann functions are based on fundamental concepts from RG for SPD matrices such as

geometric mean

$$\boldsymbol{M} = \underset{\boldsymbol{X}\in\mathcal{P}(2)}{\operatorname{argmin}} \sum_{i=1}^{N} \delta_R^2(\boldsymbol{C}_i, \boldsymbol{X})$$

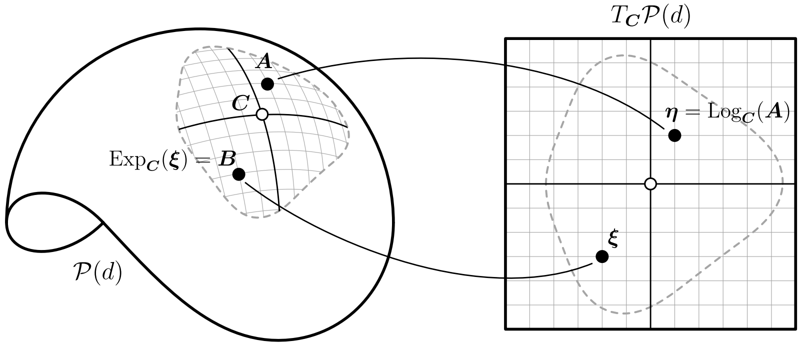
```
# calculate the geometric mean
M = mean_riemann(C)
```

Geometric mean of a set of 2x2 SPD matrices



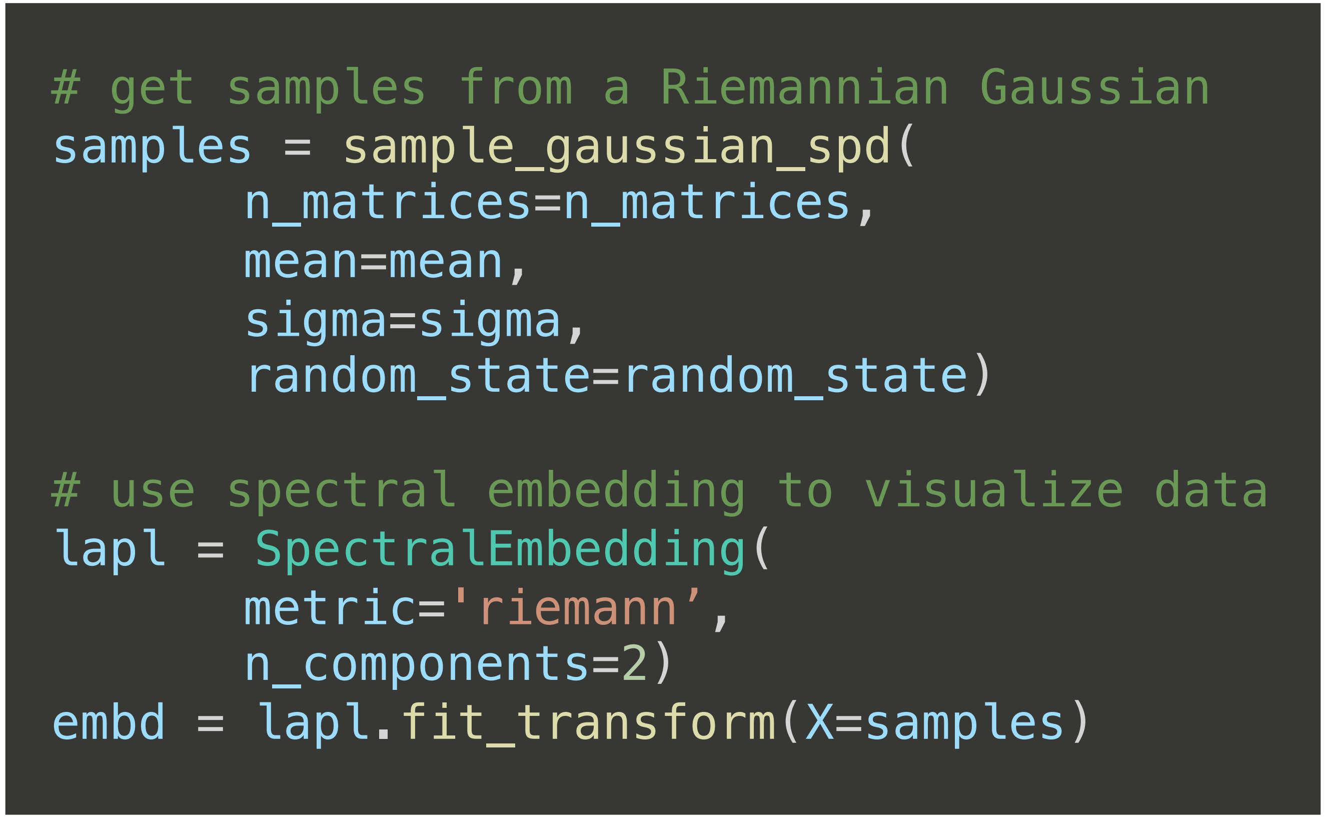
tangent space mapping

```
# project to tangent space
v = tangent_space(C, Cref)
# project back to manifold
C = untangent_space(v, Cref)
```

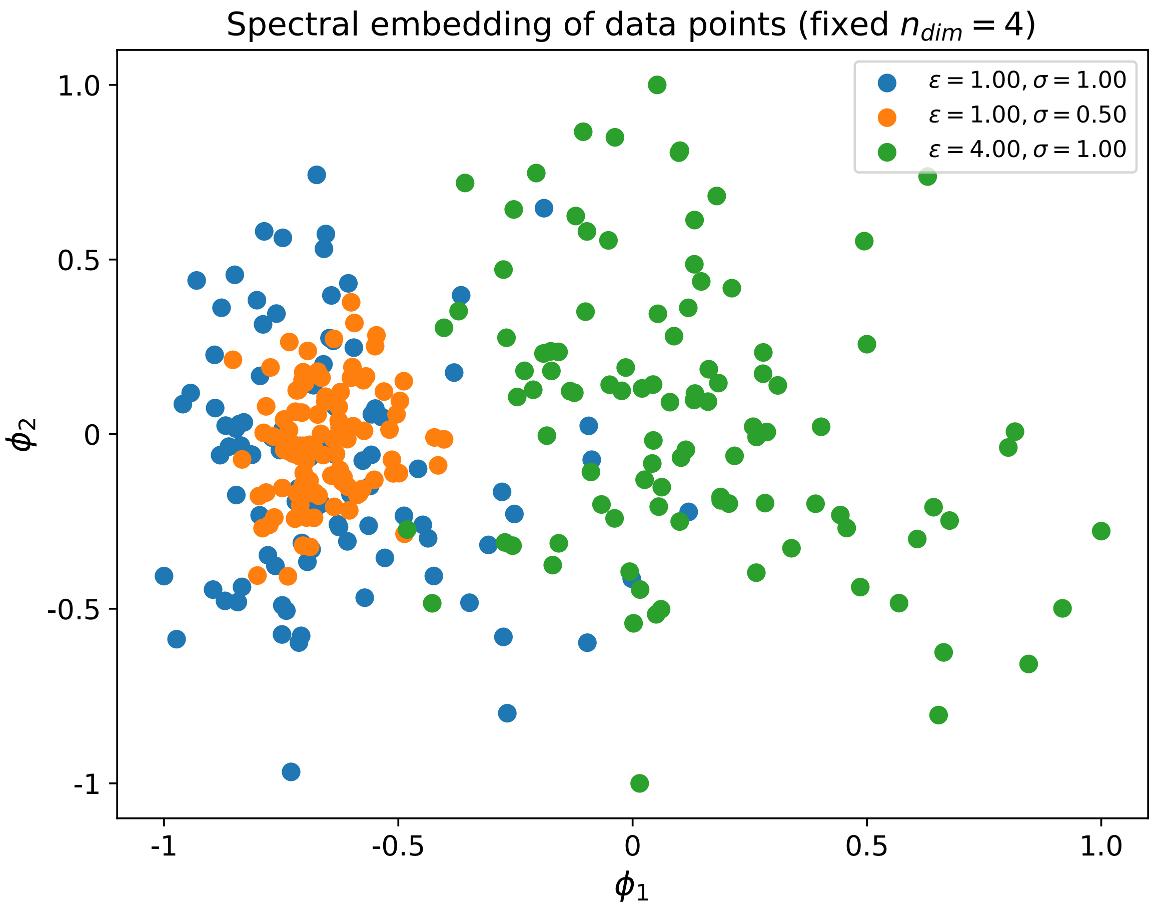

$T_C\mathcal{P}(d)$
$\boldsymbol{A}$
$\boldsymbol{C}$
$\mathrm{Exp}_{\boldsymbol{C}}(\boldsymbol{\xi}) = \boldsymbol{B}$
$\boldsymbol{\eta} = \mathrm{Log}_{\boldsymbol{C}}(\boldsymbol{A})$
$\boldsymbol{\xi}$
$\mathcal{P}(d)$

We can directly use scikit-learn classifiers on the tangent vectors

We can also do some more sophisticated operations on the SPD manifold not related to BCI



```
# get samples from a Riemannian Gaussian
samples = sample_gaussian_spd(
        n_matrices=n_matrices,
        mean=mean,
        sigma=sigma,
        random_state=random_state)

# use spectral embedding to visualize data
lapl = SpectralEmbedding(
        metric='riemann’,
        n_components=2)
embd = lapl.fit_transform(X=samples)
```



The Riemannian Gaussian can be used to generate synthetic data and validate new classifiers
Spectral embedding can be used to study, for example, the presence of outliers in the data

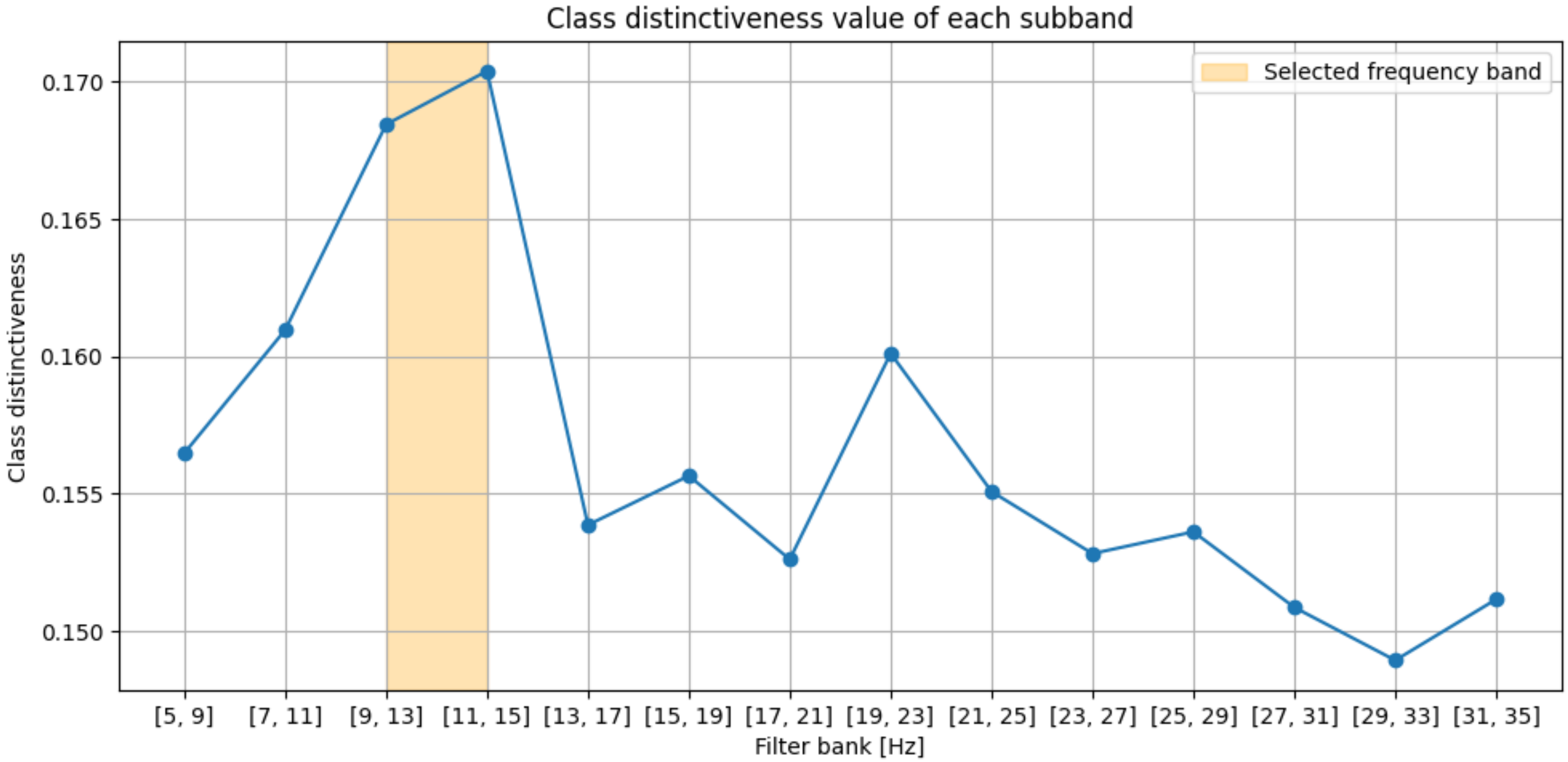
INTRODUCTION

OVERVIEW

● USE CASES

PERSPECTIVES

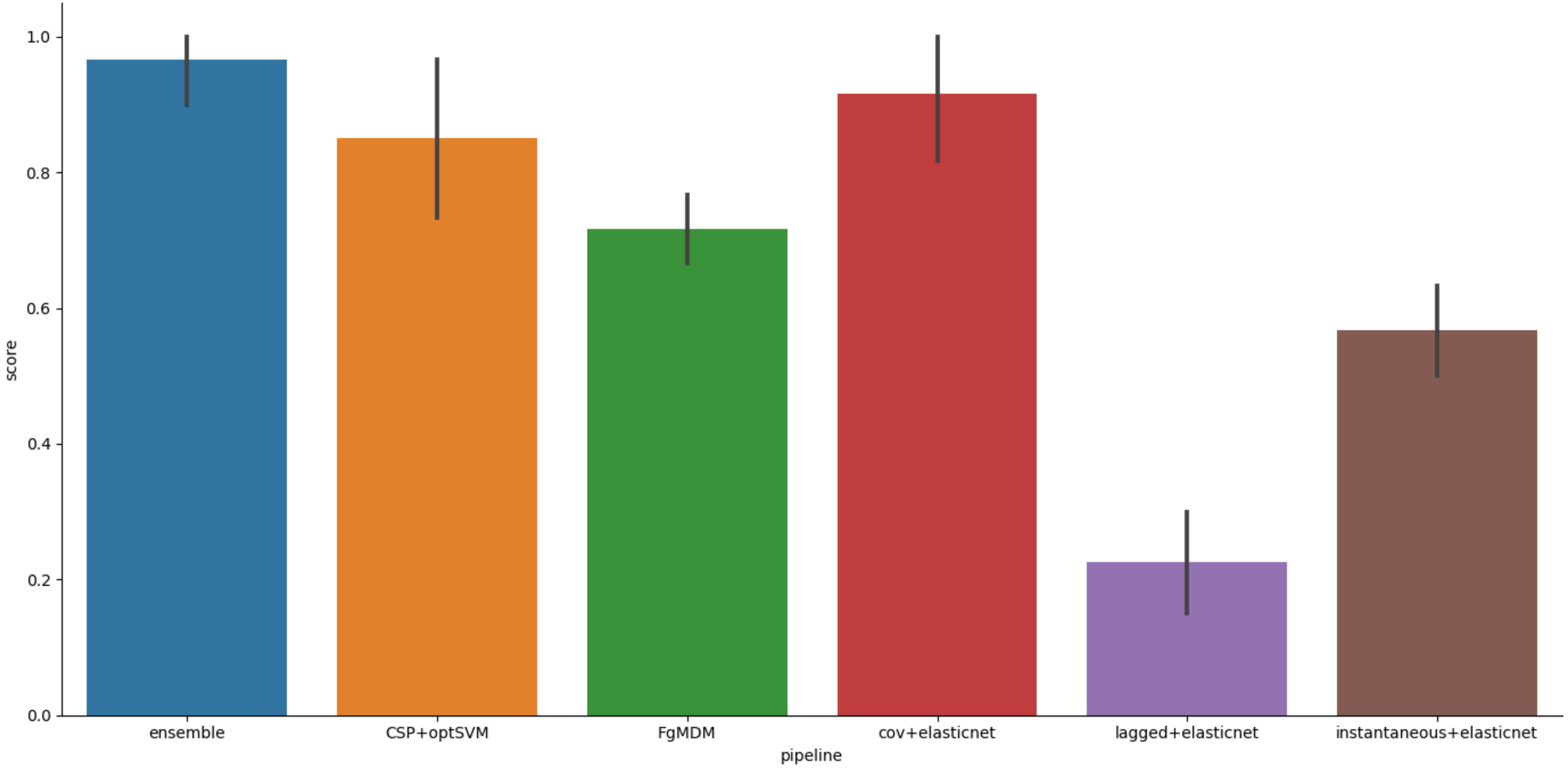
pyRiemann can be used to detect the most discriminative frequency band for motor imagery



Example available at pyRiemann/examples/motor-imagery/plot_frequency_band_selection.py

USE CASE 2: ENSEMBLE LEARNING WITH FUNCTIONAL CONNECTIVITY (FUCONE)

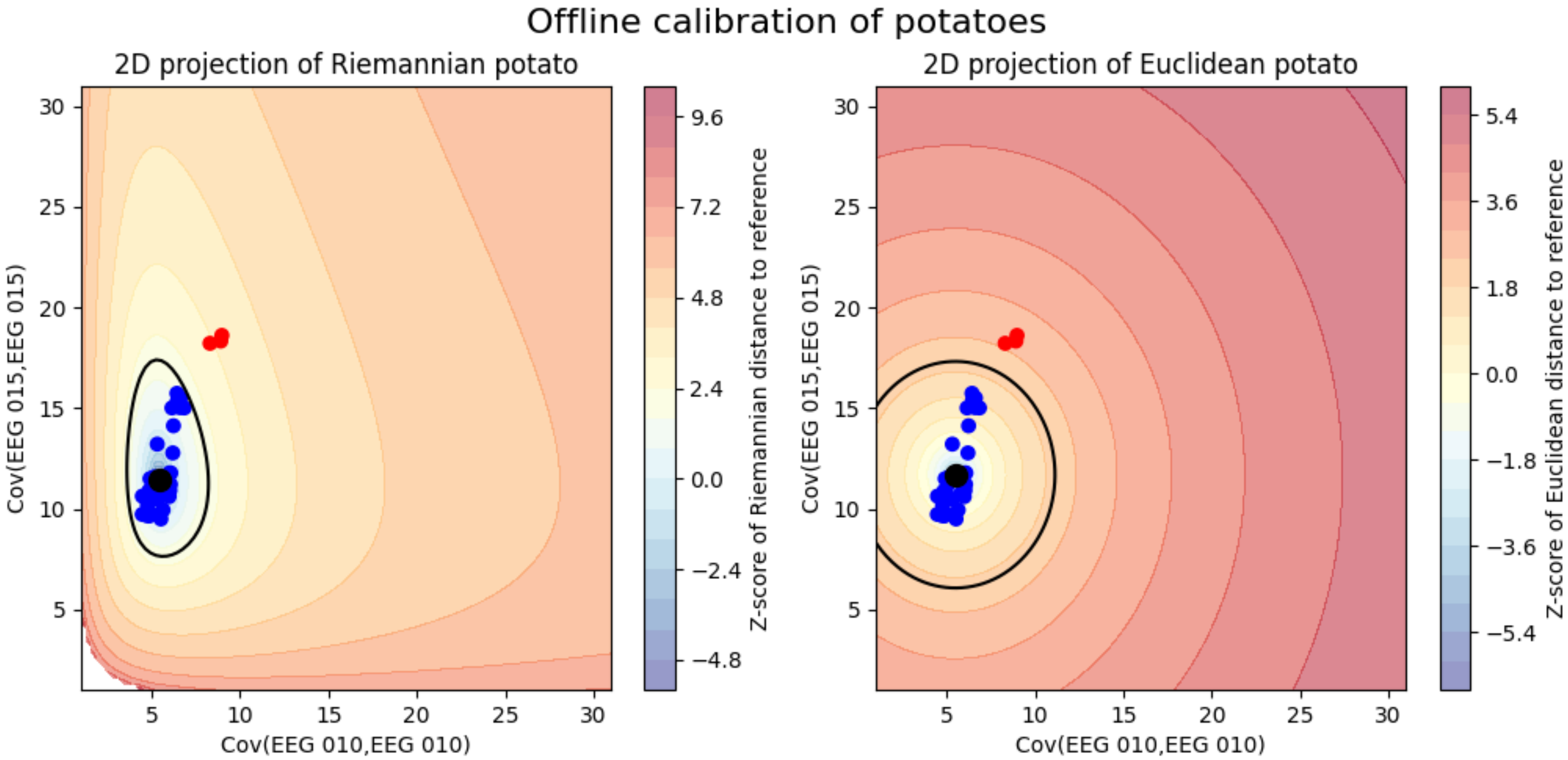
We can easily combine SPD and functional connectivity features for BCI using pyRiemann



Example available at pyRiemann/examples/motor-imagery/plot_ensemble_coherence.py

USE CASE 3: EEG ARTIFACT DETECTION WITH THE RIEMANNIAN POTATO

SPD matrices of EEG artifacts are statistical outliers that can be easily detected in pyRiemann



Example available at pyRiemann/examples/motor-imagery/plot_detect_riemannian_potato_EEG.py

INTRODUCTION

OVERVIEW

USE CASES

● PERSPECTIVES

PERSPECTIVES

pyRiemann has ~25k lines of code but there's still so much more that can be included

- Methods for regression tasks in the SPD manifold (e.g. for brain-age estimation)
- Handling Hermitian positive definite (HPD) matrices as well (e.g. for radar applications)
- Geometry-aware methods for dimensionality reduction of SPD matrices
- Ensemble methods for transfer learning, i.e. multi-source subject calibration

And much more!

Don't hesitate to **FORK** the repo, **CODE** your feature, and **CONTRIBUTE** to pyRiemann

pyRiemann

Thanks for your attention!