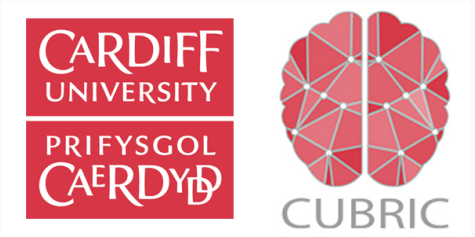


Python in brain research

Dominik Krzemiński, Maciek Szul (@MaciekSzul)

Cardiff University Brain Research Imaging Centre

Table of contents

About CUBRIC

CUBRIC

The £44m Centre, which opened in spring 2016, forms part of the University's Maindy Park site, which is a redevelopment housing a range of buildings promoting the University's ambition to be a world leader in research and innovation.

CUBRIC Research

- **MRI** - four MRI laboratories with a combination of equipment that makes it a unique facility in Europe.
- **MEG** - CTF 275 axial gradiometer design, a system which combines physical first-order axial gradiometers with effective third-order noise cancellation
- **EEG**
- **Brain stimulation labs** - transcranial electrical stimulation and transcranial magnetic stimulation;
- **Cognitive labs**
- **Sleep labs**

Intro

Goal of this talk

To show that Python is a viable environment to conduct cutting-edge research in neuroscience.

Experimental design

Experimental design

Purposes of ”experimental” software in neuroscience:

- Uniform conditions across participants
- Control in dynamic experimental design (detection thresholds, performance)
- Precise timing of events
- Interface with hardware (button boxes, joysticks, motion trackers, photometers)
- Convenient and reliable logging of the output data

Experimental design - issues

- Reproducibility
- Reusability (flexibility)
- Open source code
- Time
- Cost

PsychoPy

- www.psychopy.org
- Free, open-source, Python-based alternative
- Vibrant community of users and developers
- User interface
- Mostly used as a python library
- Cross-platform
- Standalone version
- pavlovia.org - JS platform to run PsychoPy online

PsychoPy - example

```
from psychopy import core, visual, monitors
from psychopy.iohub.client import launchHubServer

width, dist, res = (42.3, 120, ( 1920, 1080))

mon = monitors.Monitor('default')
mon.setWidth(width)
mon.setDistance(dist)
mon.setSizePix(res)

win = visual.Window(size=res,
                    color='#000000',
                    fullscr=True,
                    allowGUI=False,
                    winType='pyglet',
                    units='deg',
                    monitor=mon)

"""JOYSTICK""
kwargs = {'psychopy_monitor_name': 'default', 'xinput.Gamepad': {}}
io = launchHubServer(**kwargs)
gamepad = io.devices.gamepad
```

PsychoPy - example

```
"""EYETRACKER"""
iohub_tracker_class_path = 'eyetracker.hw.sr_research.eyelink.EyeTracker'
eyetracker_config = dict()
eyetracker_config['name'] = 'tracker'
eyetracker_config['enable'] = True
eyetracker_config['stream_events'] = False
eyetracker_config['model_name'] = 'EYELINK 1000 TOWER'
eyetracker_config['default_native_data_file_name'] = output_file_name
eyetracker_config['simulation_mode'] = False
eyetracker_config['runtime_settings'] = dict(sampling_rate=1000,
                                             track_eyes='RIGHT')

io = launchHubServer(**{iohub_tracker_class_path: eyetracker_config})
tracker = io.devices.tracker
calibration = tracker.runSetupProcedure()
```

PsychoPy - example

```
text = visual.TextStim(win, text=' ',
                       color='white',
                       units='deg',
                       height=1,
                       opacity=1.0,
                       pos=(0.0, 0.0))

text.text = 'wait 2 sec'
text.draw()
win.flip()
core.wait(2)

tracker.setRecordingState(True)
tracker.sendMessage("START")

# experiment start
for i in range(1,21):
    text.text = str(i)
    text.pos = (np.random.randn(2)*2)
    text.draw()
    core.wait(2)
    tracker.sendMessage("TRIGGERED")
    win.flip()

# experiment end
tracker.sendMessage("END")
tracker.setRecordingState(False)
tracker.setConnectionState(False)

win.close()
io.quit()
core.quit()
```

Neuroimaging methods

Data collection - EEG

Electroencephalograpy

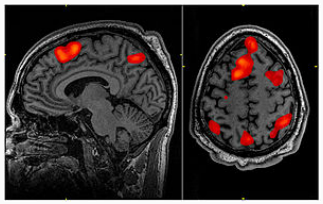
- **Data source:** electrical activity recorded on the scalp
- **Data type:** time series
- **Pros:** cheap, lots of research, good time resolution, direct measurememnt of neuronal activity
- **Cons:** noisy, conductance issues, very poor spatial resolution

Data collection - MEG

Magnetoencephalograpy

- **Data source:** magnetic activity near the sensors
- **Data type:** time series
- **Pros:** excellent time resolution, good to excellent spatial resolution, direct measurememnt of neuronal activity
- **Cons:** expensive, relatively rare,

Data collection - MRI

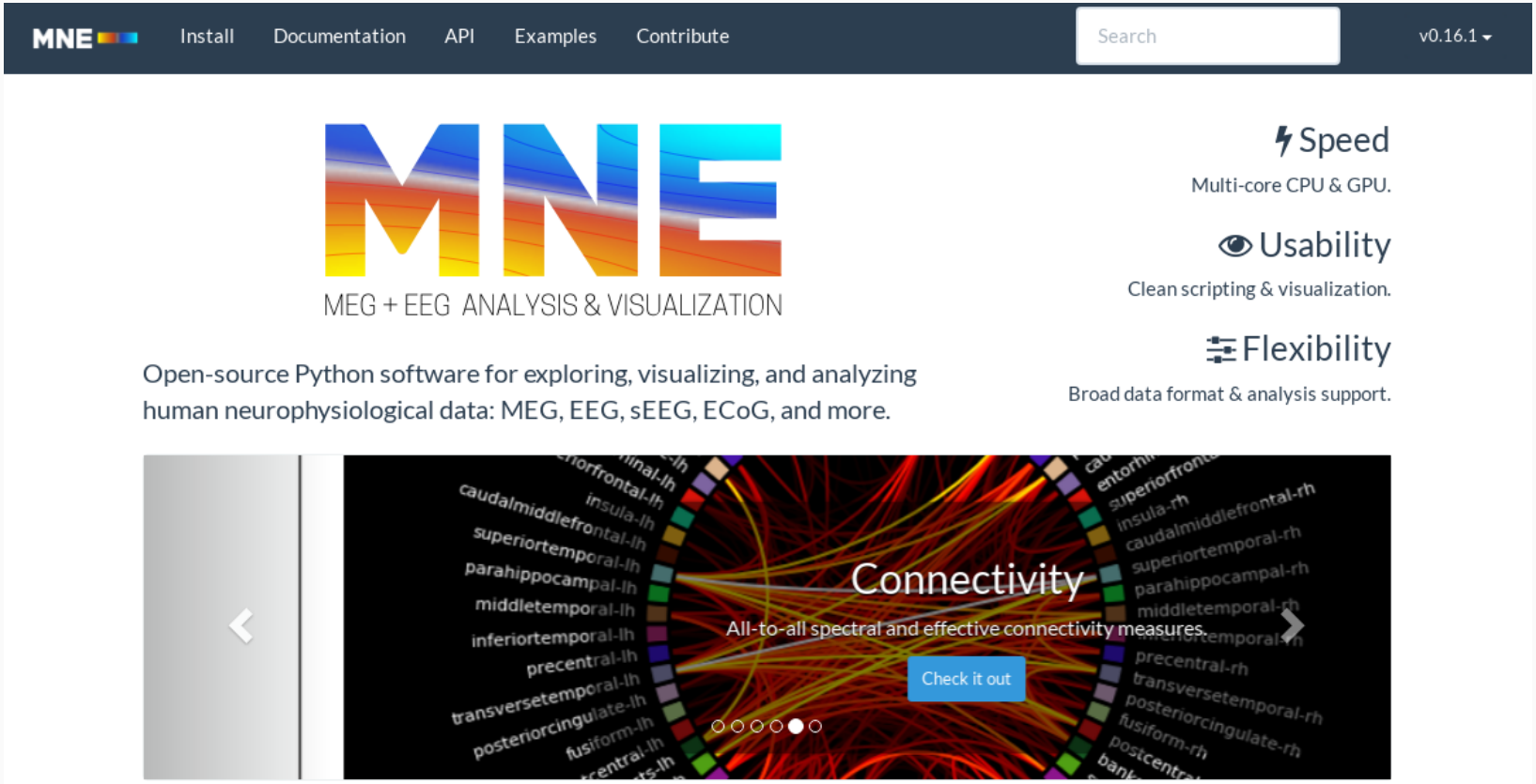


Magnetic Resonance Imaging

- **Data source:** tissue relaxation times, blood oxygenation levels
- **Data type:** images (sometimes time series for functional imaging)
- **Pros:** excellent spatial resolution, huge amount of research, popular, anatomy and function
- **Cons:** limited spatial resolution, low temporal resolution (functional), indirect and complicated measurement of neuronal activity

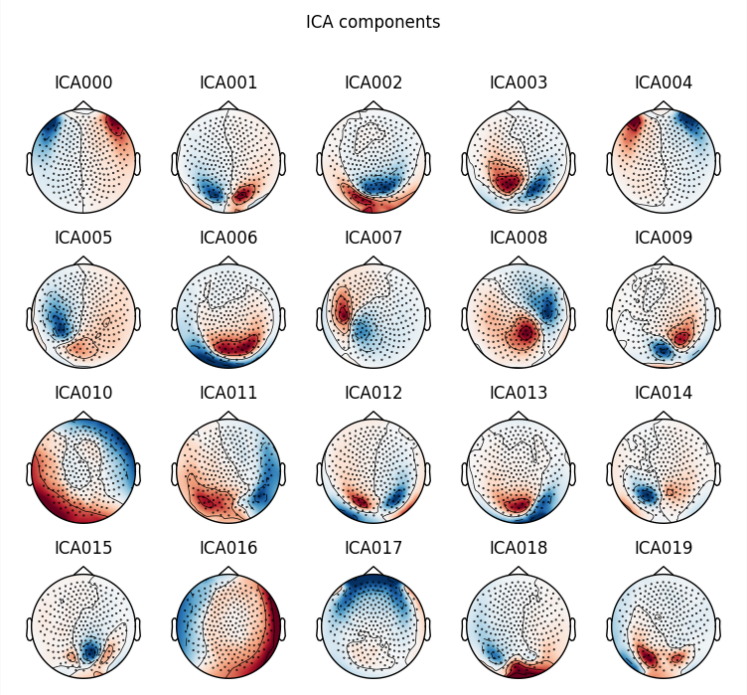
EEG/MEG data processing

MNE


MNE
Install
Documentation
API
Examples
Contribute
Search
v0.16.1
MNE
MEG + EEG ANALYSIS & VISUALIZATION
Open-source Python software for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, sEEG, ECoG, and more.
Speed
Multi-core CPU & GPU.
Usability
Clean scripting & visualization.
Flexibility
Broad data format & analysis support.
Connectivity
All-to-all spectral and effective connectivity measures.
Check it out

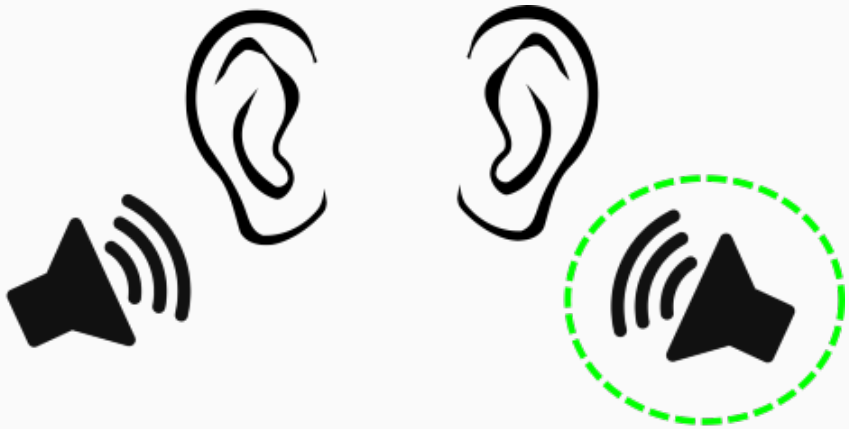
Data preprocessing

1. downsampling;
2. filtering;
3. epoching;
4. artifacts rejection;
    - Bad trials removal;
    - Bad channels removal;
    - Independent Component Analysis.



MNE

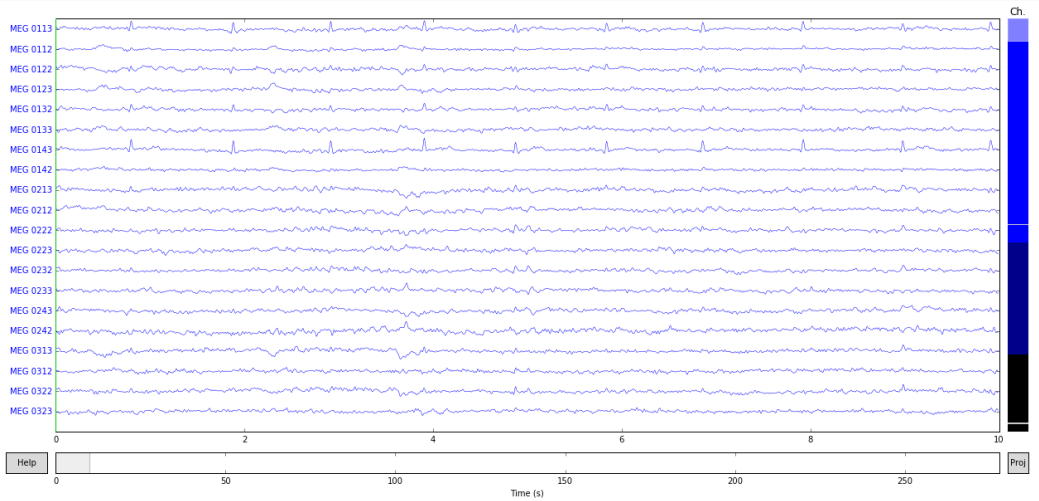
```
import numpy as np
import matplotlib.pyplot as plt

import mne
from mne.datasets import sample

data_path = sample.data_path()
fname_raw = data_path + '/MEG/sample/sample_audvis_filt-0-40_raw.fif'
fname_event = data_path + '/MEG/sample/sample_audvis_filt-0-40_raw-eve.fif'
```



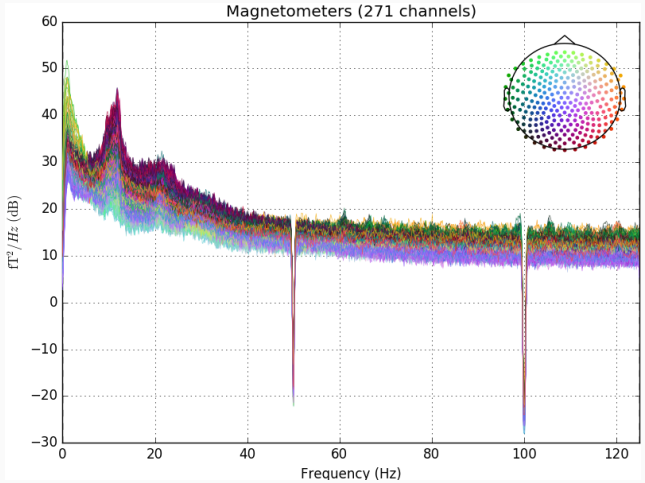
https://martinos.org/mne/stable/auto_examples/connectivity/plot_mixed_source_space_connectivity.html

MNE

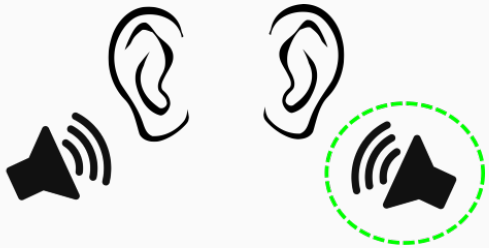
```
raw = mne.io.read_raw_fif(fname_raw)
events = mne.read_events(fname_event)
raw.info['bads'] += ['MEG 2443']
raw.plot()
```




```
raw.plot_psd()
```

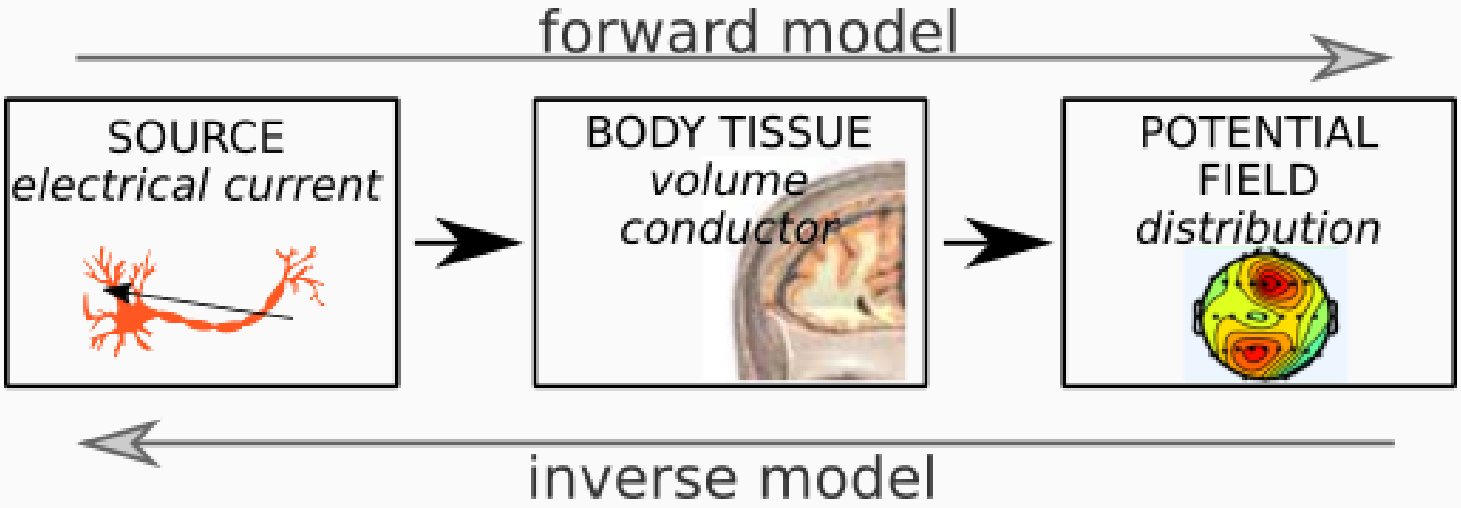


MNE

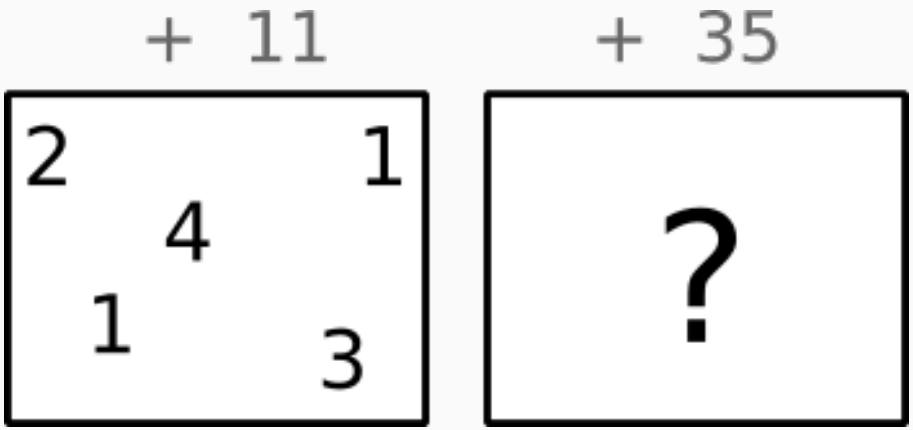


```
# Pick MEG channels
picks = mne.pick_types(raw.info, meg=True, eeg=False, stim=False, eog=True,
                       exclude='bads')

# Define epochs for left-auditory condition
event_id, tmin, tmax = 1, -0.2, 0.5
epochs = mne.Epochs(raw, events, event_id, tmin, tmax, picks=picks,
                    baseline=(None, 0), reject=dict(mag=4e-12, grad=4000e-13,
                                                    eog=150e-6))
```

Forward / Inverse Modeling

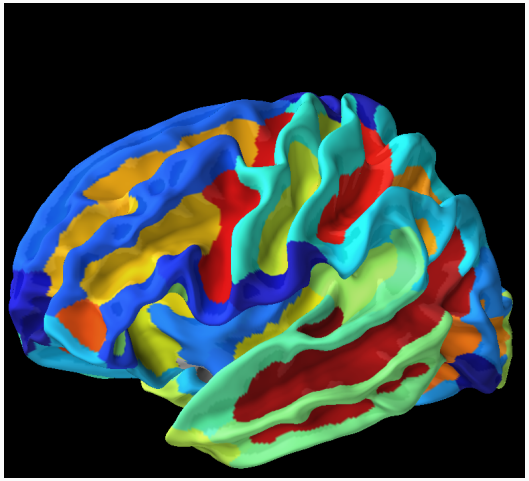


MNE




```
snr = 1.0  # use lower SNR for single epochs
lambda2 = 1.0 / snr ** 2
method = "MNE"  # dSPM, MNE or sLORETA
stcs = apply_inverse_epochs(epochs, inverse_operator, lambda2, method,
                            pick_ori="normal", return_generator=True)
```

MNE

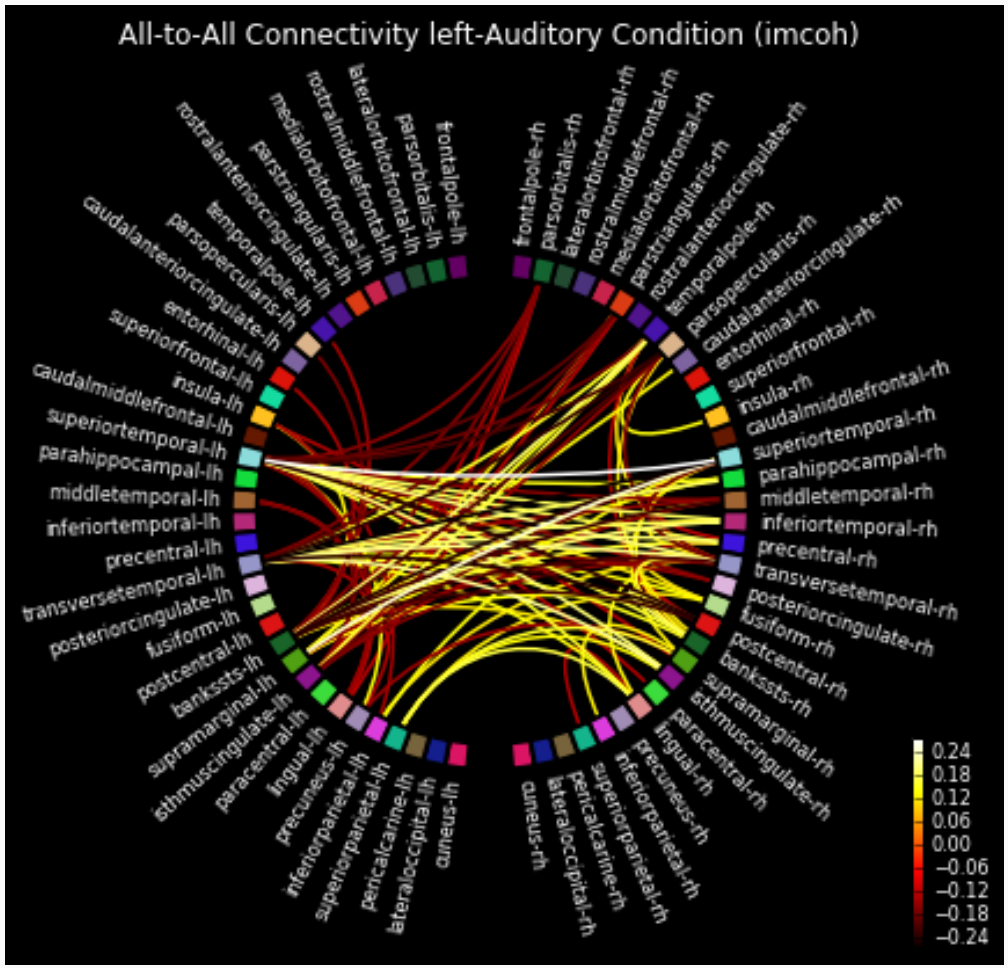


```
labels = mne.read_labels_from_annot('sample', parc='aparc',
                                    subjects_dir=subjects_dir)
label_colors = [label.color for label in labels]

src = inverse_operator['src']
label_ts = mne.extract_label_time_course(stcs, labels, src, mode='mean_flip',
                                         return_generator=True)
```

```
fmin, fmax  = 8., 13.
sfreq = raw.info['sfreq']  # the sampling frequency
con_methods = ['imcoh']
con, freqs, times, n_epochs, n_tapers = spectral_connectivity(
    label_ts, method=con_methods, mode='multitaper', sfreq=sfreq, fmin=fmin,
    fmax=fmax, faverage=True, mt_adaptive=True, n_jobs=1)
```

MNE

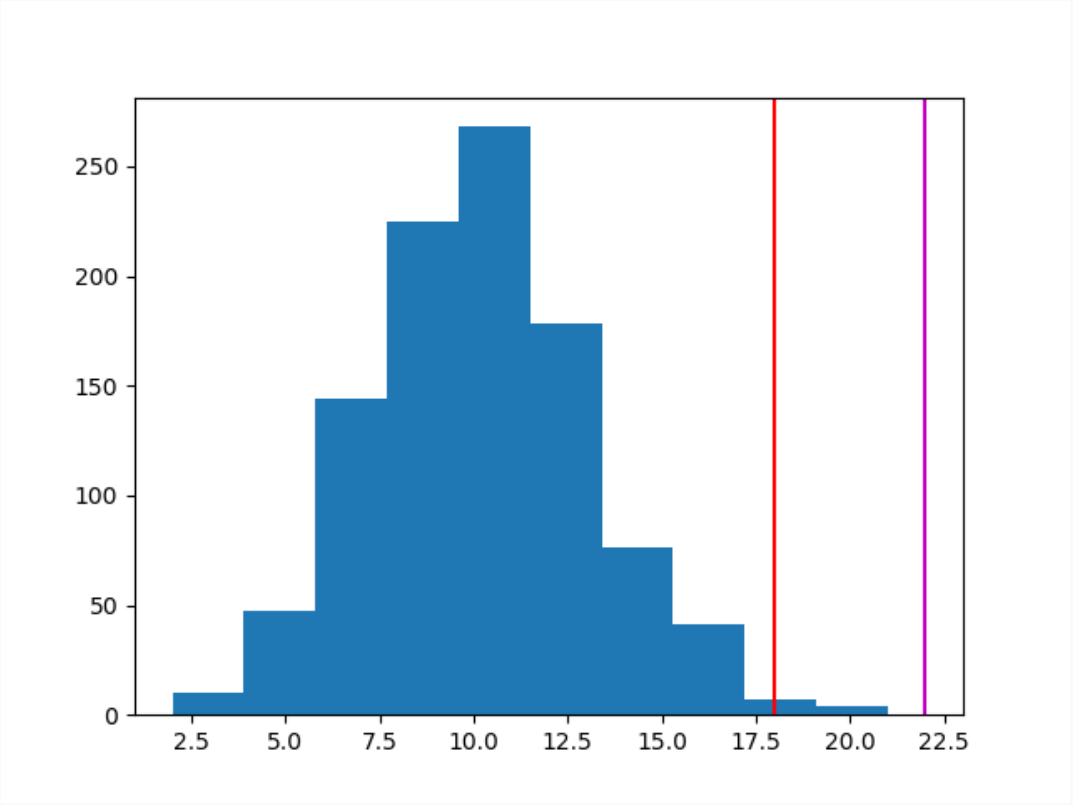
```
# Prettification
# ...
plot_connectivity_circle(con_res['imcoh'], label_names, n_lines=100,
                         node_angles=node_angles, node_colors=label_colors,
                         title='All-to-All Connectivity left-Auditory '
                               'Condition (imcoh)')
```



Statistical analysis

Scipy stats

```
import scipy.stats as ss

data = np.array([...])
plt.hist(data)

data.mean()
# 10.071

alpha = 0.01
xr_ = ss.poisson.ppf(1 — alpha, data.mean())
# 18.0

plt.axvline(x = xr_, color='r')
plt.show()
```



Brain activity decoding

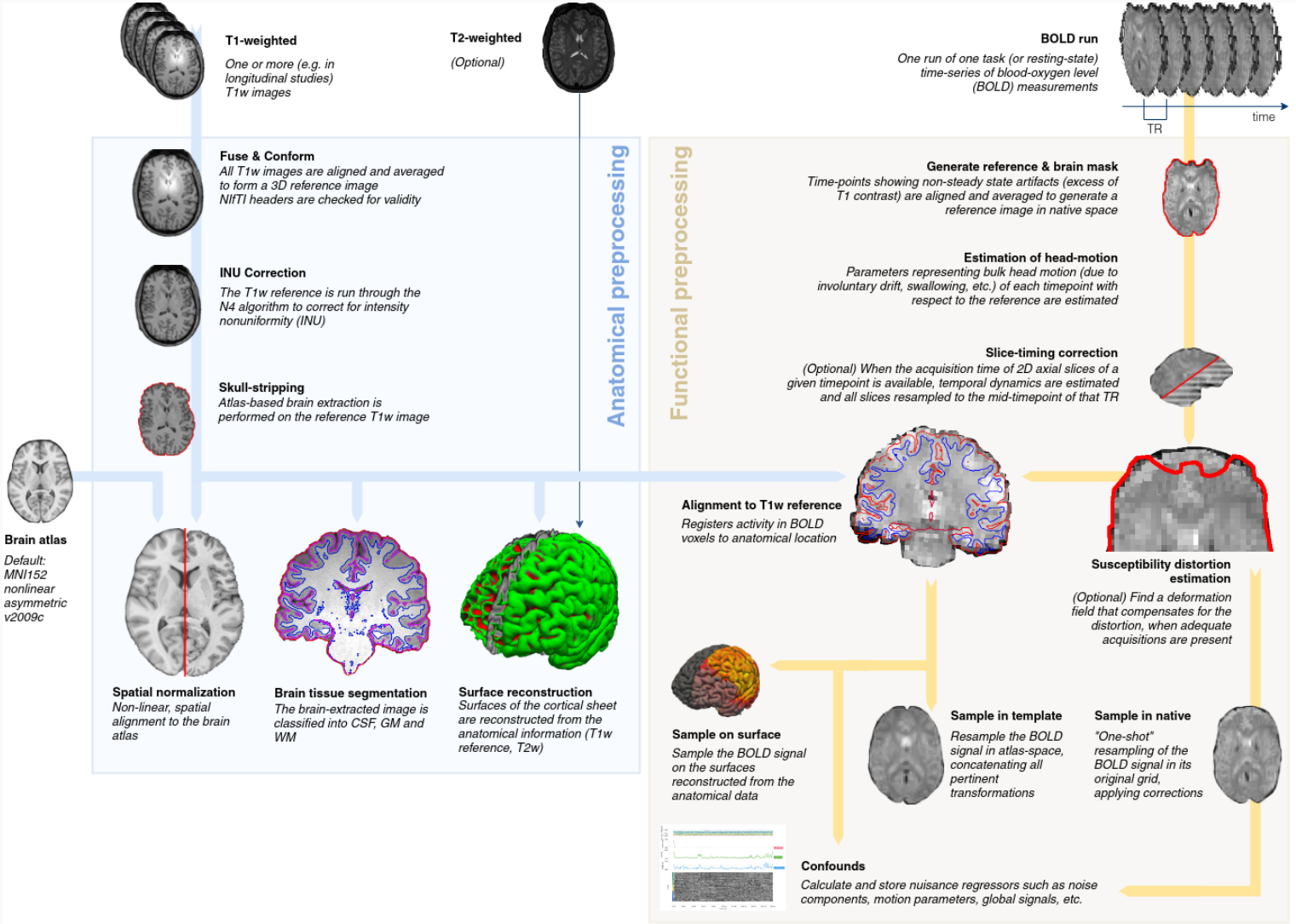
Machine learning in neuroscience

- Historically linked with python (PyMVPA)
- Currently relying heavily on scikit-learn
- Fully open source
- NiPy + scikit-learn = Ni-learn
- MNE

Machine learning in neuroscience - issues

- Preprocessing
- Feature selection
- Biological interpretations (lot of talking)
- Appropriate amount of samples
- Proper labeling
- Reproducibility
- Researchers' degrees of freedom.

Machine learning in neuroscience



fmriprep: A Robust Preprocessing Pipeline for fMRI Data

Basic application of SVM, for simple design (houses vs faces)

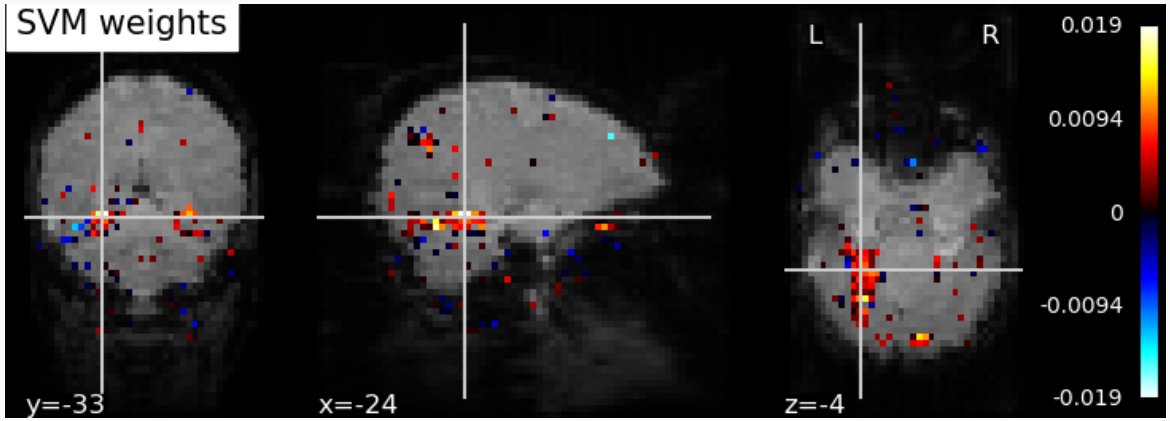

```
import pandas as pd
from nilearn import datasets
from nilearn.input_data import NiftiMasker
from sklearn.svm import SVC
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.pipeline import Pipeline
from sklearn.model_selection import LeaveOneGroupOut, cross_val_score
from nilearn import image
from nilearn.plotting import plot_stat_map, show

haxby_dataset = datasets.fetch_haxby()
behavioral = pd.read_csv(haxby_dataset.session_target[0], sep=" ")
conditions = behavioral['labels']

condition_mask = behavioral['labels'].isin(['face', 'house'])
conditions = conditions[condition_mask]

session = behavioral[condition_mask].to_records(index=False)

# Load the mask
mask_filename = haxby_dataset.mask
# For decoding, standardizing is often very important
# note that we are also smoothing the data
masker = NiftiMasker(mask_img=mask_filename, smoothing_fwhm=4,
                     standardize=True, memory="nilearn_cache", memory_level=1)
func_filename = haxby_dataset.func[0]
X = masker.fit_transform(func_filename)
# Apply our condition_mask
X = X[condition_mask]
```

Basic application of SVM, for simple design (houses vs faces)


```
svc = SVC(kernel='linear')

# Define the dimension reduction to be used.
# Here we use a classical univariate feature selection based on F-test
feature_selection = SelectPercentile(f_classif, percentile=5)

anova_svc = Pipeline([('anova', feature_selection), ('svc', svc)])

anova_svc.fit(X, conditions)
y_pred = anova_svc.predict(X)

cv = LeaveOneGroupOut()

cv_scores = cross_val_score(anova_svc, X, conditions, cv=cv, groups=session)

classification_accuracy = cv_scores.mean()
```

Basic application of SVM, for simple design (houses vs faces)

```
coef = svc.coef_
# reverse feature selection
coef = feature_selection.inverse_transform(coef)
# reverse masking
weight_img = masker.inverse_transform(coef)

# Use the mean image as a background to avoid relying on anatomical data
mean_img = image.mean_img(func_filename)

# Create the figure
plot_stat_map(weight_img, mean_img, title='SVM weights')

# Saving the results as a Nifti file may also be important
weight_img.to_filename('haxby_face_vs_house.nii')

show()
```



http://nilearn.github.io/auto_examples/02_decoding/plot_haxby_anova_svm.html

Open neuroimaging datasets

- `https://github.com/voytekresearch/OpenData` - curated list of datasets in human electrophysiology
- `https://en.wikipedia.org/wiki/List_of_neuroscience_databases` - wiki page with neuroscience databases
- `https://openneuro.org/` - open platform for sharing and analysing neuroimaging data
- `https://github.com/cMadan/openMorph` - curated list of open structural datasets
- `https://neurovault.org/` - repository of unthresholded statistical maps

Bigger support of open scientific practices contributes to growth in available datasets.

Thank you!

Diolch!

Questions?

ARmadillo

Welcome to the ARmadillo web app, which creates Augmented Reality 3D brain images from the **Neurovault** database.

Take it for a spin

Example 1

Open

Example 2

Open

Example 3

Open