

cnspp_tutorial_eelbrain_answers

July 20, 2022

1 Introduction to Encoding/Decoding Analysis with Eelbrain

Tutorial for CNSP workshop 18-20 July 2022

<https://cnsppworkshop.net/>

Please see README.pdf for installation instructions

Author: Joshua P. Kulasingham ***

This tutorial will go through several common steps when analyzing M/EEG signals using eelbrain.

Two datasets are used in this tutorial.

- 1) LalorNatSpeech: This dataset will be used to demonstrate the following steps
 - Loading CND files
 - Preprocessing EEG data
 - Encoders/Decoders

- 2) Alice TRFs: This dataset consists of precomputed TRFs from the Alice Dataset. Statistical tests on TRFs across subjects will be demonstrated using these precomputed TRFs.

For more details on the dataset, see: <https://aclanthology.org/2020.lrec-1.15>

For more details on the estimated TRFs, see: <https://www.biorxiv.org/content/10.1101/2021.08.01.454687v1>

2 0 Setup

```
[1]: # basic imports
%matplotlib inline
import numpy as np
import scipy
import time

# M/EEG packages
import mne
import eelbrain as eel
import trftools
mne.set_log_level('ERROR')
```

```
# filepaths accounting for OS differences
from pathlib import Path
cwd_path = Path('.')
data_path = cwd_path / 'data'
results_path = cwd_path / 'results'
```

3 1 Basic TRF

This section gives a feel for the core steps involved in computing TRFs
Extensive details of each step will be provided in later sections 2-5

3.1 1.1 Load preprocessed data

```
[2]: eegs = eel.load.unpickle(data_path / 'Section1' / '01_basic_trf_eegs.pickle')
      preds = eel.load.unpickle(data_path / 'Section1' / '01_basic_trf_preds.pickle')

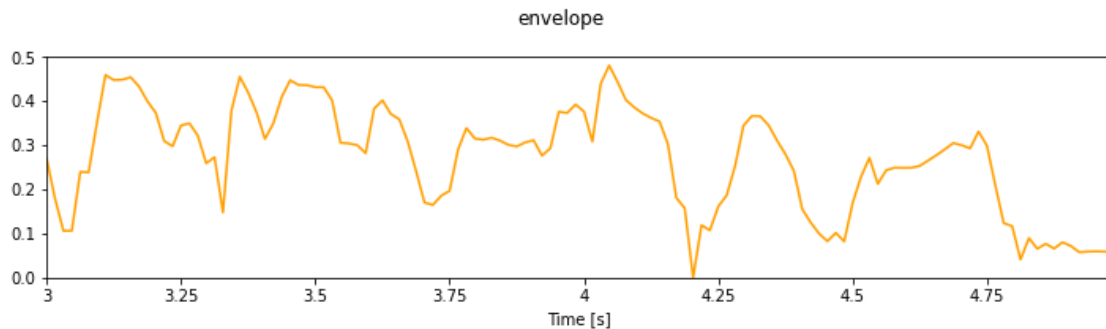
      print('eegs', eegs)
      print('preds', preds)
```

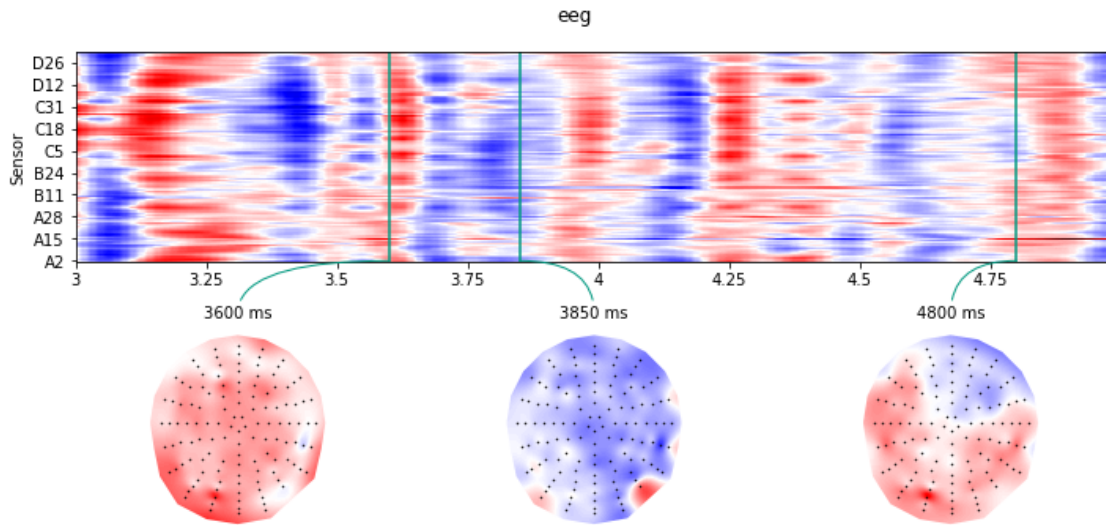
```
eegs <NDVar: 4 case, 124 sensor, 9920 time>
preds <NDVar: 4 case, 9920 time>
```

3.2 1.2 Visualize data

```
[3]: # plot predictor (speech envelope)
      p = eel.plot.UTS(preds[1].sub(time=(3, 5)), w=10, h=3, title='envelope')

      # plot eeg
      # Sensor x Time plot, with sensor topographies of interesting time points
      ↪displayed
      time_points = [3.6, 3.85, 4.8]
      p = eel.plot.TopoArray(eegs[1].sub(time=(3, 5)), t=time_points, h=5, w=10,
      ↪title='eeg')
```





3.3 1.3 Compute trf

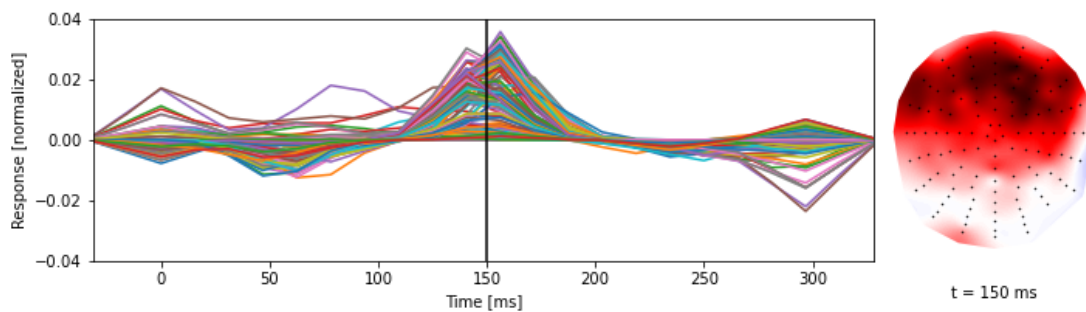
```
[ ]: # this may take a minute or so
res = eel.boosting(eegs, preds, 0, 0.3, basis=0.08, partitions=2)
```

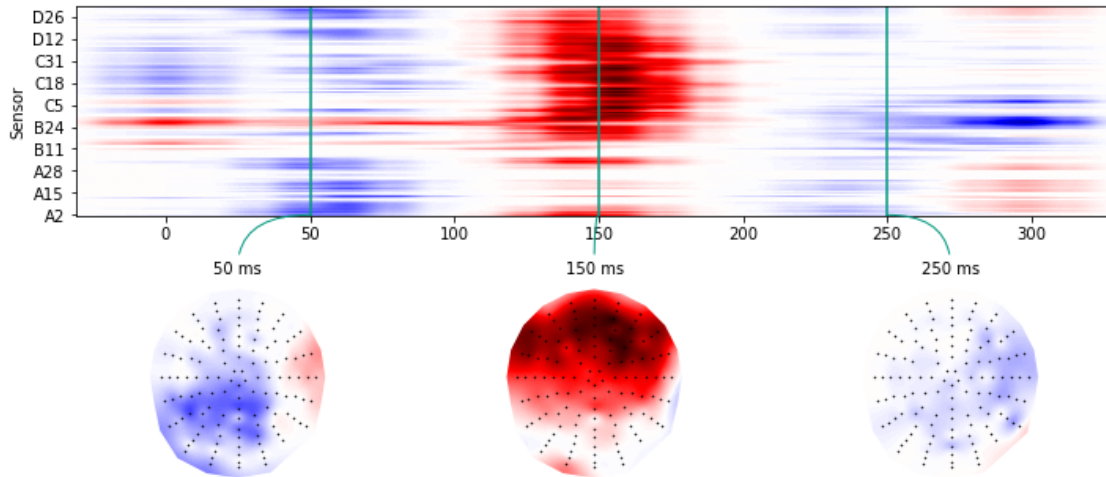
```
[4]: # to load a precomputed TRF, uncomment line below
res = eel.load.unpickle(data_path / 'results' / 'Section1_trf.pickle')
```

3.4 1.4 Plot trf

```
[5]: print('correlation accuracy = ', res.r.mean())
p = eel.plot.TopoButterfly(res.h)
p.set_time(0.15)
p = eel.plot.TopoArray(res.h, t=(0.05, 0.15, 0.25), h=5, w=10)
```

correlation accuracy = 0.05423599448228214





4 2 Data Types: NDVars

For more details see <https://eelbrain.readthedocs.io/en/stable/generated/eelbrain.NDVar.html>

4.1 2.1 Basic operations on NDVars

```
[6]: print(eegs)
eeg = eegs[0] # first trial
print(eeg)
# Access data as a numpy array using NDVar.x
eeg_data = eeg.x
print('data shape', eeg_data.shape)
# list dimensions using NDVar.dimnames
print('dimensions', eeg.dimnames)
```

```
<NDVar: 4 case, 124 sensor, 9920 time>
<NDVar: 124 sensor, 9920 time>
data shape (124, 9920)
dimensions ('sensor', 'time')
```

4.1.1 2.1.1 UTS dimension

For more details, see <https://eelbrain.readthedocs.io/en/stable/generated/eelbrain.UTS.html>

```
[7]: # UTS dimension (Uniform Time Series)
print(eeg.time, 'UTS(tstart, tstep, nsamples)')
print(f'fs = {1/eeg.time.tstep}, {eeg.time.tmin = }, {eeg.time.tmax = }')
```

```
UTS(0.0, 0.015625, 9920) UTS(tstart, tstep, nsamples)
fs = 64.0, eeg.time.tmin = 0.0, eeg.time.tmax = 154.984375
```

```
[8]: # creating a UTS dimension
tstart = 0
tstep = 1/100
nsamples = 100
newUTS = eel.UTS(tstart, tstep, nsamples)
print(f'{newUTS = }')
print(f'{newUTS.times}')

```

```
newUTS = UTS(0.0, 0.01, 100)
[0.  0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1  0.11 0.12 0.13
 0.14 0.15 0.16 0.17 0.18 0.19 0.2  0.21 0.22 0.23 0.24 0.25 0.26 0.27
 0.28 0.29 0.3  0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.4  0.41
 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.5  0.51 0.52 0.53 0.54 0.55
 0.56 0.57 0.58 0.59 0.6  0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69
 0.7  0.71 0.72 0.73 0.74 0.75 0.76 0.77 0.78 0.79 0.8  0.81 0.82 0.83
 0.84 0.85 0.86 0.87 0.88 0.89 0.9  0.91 0.92 0.93 0.94 0.95 0.96 0.97
 0.98 0.99]
```

```
[9]: # subsetting a dimension
eeg1 = eeg.sub(time=(1, 4)) # 1 to 4 seconds
print(f'eeg has UTS dimension = {eeg.time}')
print(f'eeg1 has UTS dimension = {eeg1.time}')

```

```
eeg has UTS dimension = UTS(0.0, 0.015625, 9920)
eeg1 has UTS dimension = UTS(1.0, 0.015625, 192)
```

4.1.2 2.1.2 Sensor dimension

For more details see <https://eelbrain.readthedocs.io/en/stable/generated/eelbrain.Sensor.html#eelbrain.Sensor>

```
[10]: # sensor dimension
print(f'{eeg.sensor = }')
print(f'first 3 eeg sensor names are {eeg.sensor.names[:3]}')

# subsetting a sensor dimension
eeg1 = eeg.sub(sensor=['C2', 'C3'])
print(f'eeg1 has sensor names {eeg1.sensor.names}')

```

```
eeg.sensor = <Sensor n=124, name='BioSemi'>
first 3 eeg sensor names are ['A2', 'A3', 'A4']
eeg1 has sensor names ['C2', 'C3']
```

4.1.3 2.1.3 Creating an NDVar

```
[11]: # creating an NDVar
ntrials = 10
sensordim = eeg.sensor
nsamples = 1000
```

```

tstart = 0
fs = 100
tstep = 1/fs

# create a random numpy array
data_array = np.random.randn(ntrials, len(sensordim), nsamples)

# create the time dimension
timedim = eel.UTS(0, tstep, nsamples)

# create the NDVar
# eel.Case is a dimension that is used for subjects/trials
newNDVar = eel.NDVar(data_array,
                     dims=(eel.Case, sensordim, timedim),
                     name='newNDVar')
print(newNDVar)

```

<NDVar 'newNDVar': 10 case, 124 sensor, 1000 time>

4.1.4 2.1.4 Operations on NDVars

```

[12]: eeg_abs = eeg1.abs() # absolute value
eeg_max = eeg1.max() # maximum over all dimensions
eeg_max_time = eeg1.max('time') # maximum over time dimensions
eeg_min = eeg1.min() # minimum over all dimensions
eeg_mean = eeg1.mean('sensor') # mean over sensor dimension
eeg_std = eeg1.std('sensor') # standard deviation over sensor dimension
eeg_rms = eeg1.rms('sensor') # r.m.s. over sensor dimension
eeg_norm = eeg1.norm('sensor') # l2 norm over sensor dimension

# chaining operations
eeg_rms_mean_max = eeg1.rms('sensor').mean('time').max()

```

For even more operations, see <https://eelbrain.readthedocs.io/en/stable/reference.html#ndvar-operations>

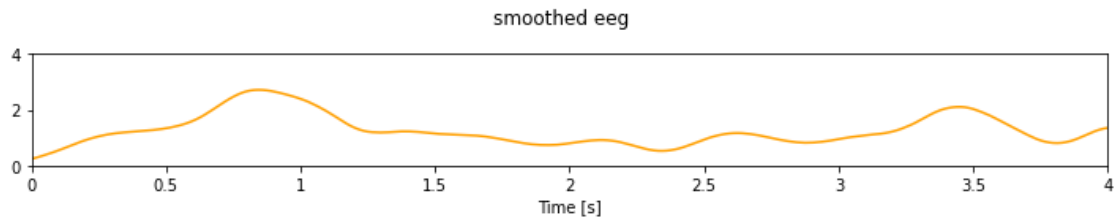
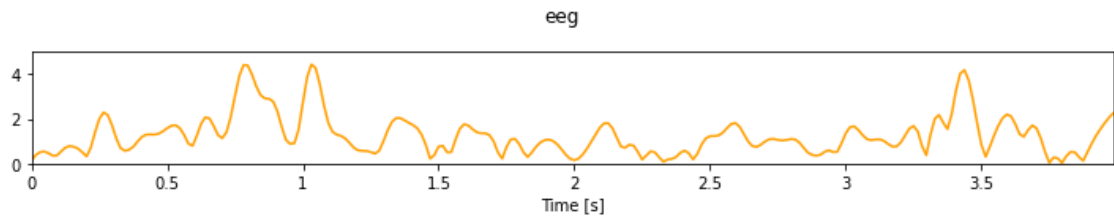
4.2 2.2 Plotting NDVars

```

[13]: # plotting time series
plot_data = eeg1.rms('sensor').sub(time=(0,4))
p = eel.plot.UTS(plot_data,
                 h=2, w=10, title='eeg')

# smoothing
plot_data = eeg1.rms('sensor').smooth('time', 0.5)
p = eel.plot.UTS(plot_data, xlim=(0, 4),
                 h=2, w=10, title='smoothed eeg')

```

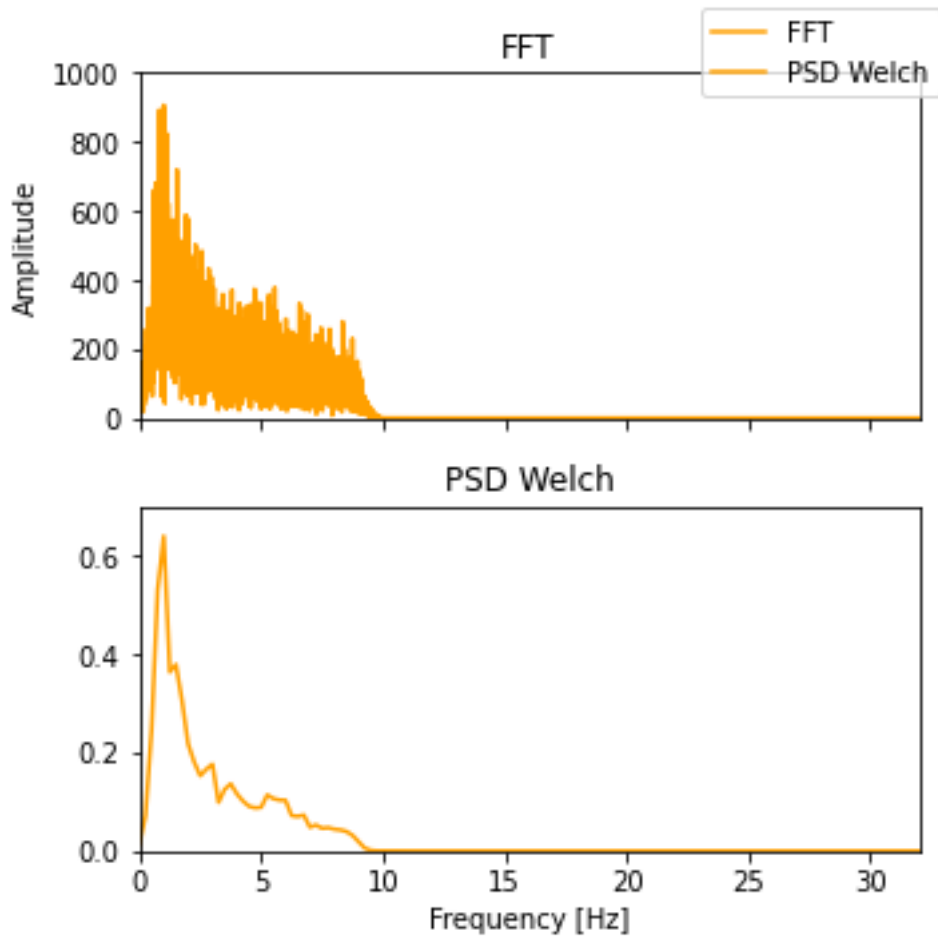


```
[14]: # frequency domain
eegfft = eeg1.fft().mean('sensor')
eegfft.name = 'FFT'
eegpsd = eel.psd_welch(eeg1).mean('sensor')
eegpsd.name = 'PSD Welch'

print(eegfft)
print(eegpsd)
p = eel.plot.UTS([eegfft, eegpsd], h=5, w=5)
```

```
<NDVar 'FFT': 4961 frequency>
```

```
<NDVar 'PSD Welch': 129 frequency>
```

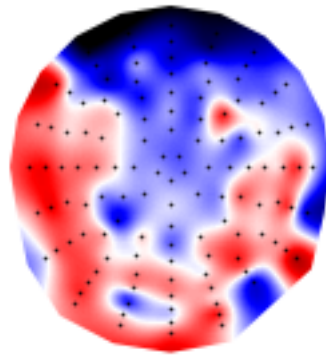


```
[15]: # Topmaps
p = eel.plot.Topomap(eegs[0].sub(time=1.2), title='Topomap')

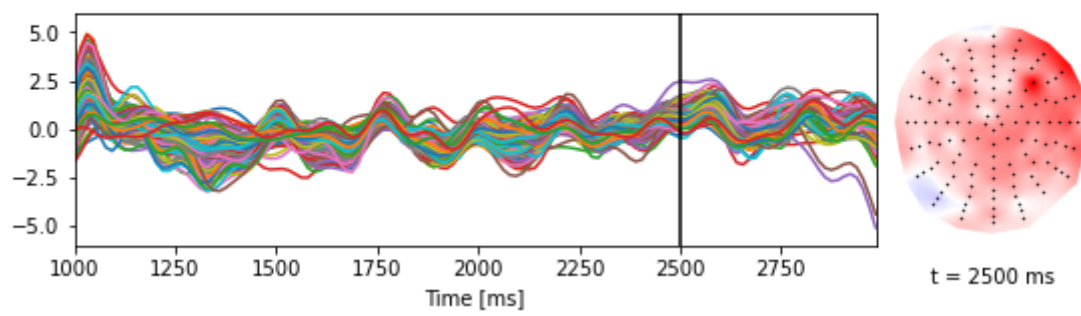
plot_data = eegs[0].sub(time=(1,3))
p = eel.plot.TopoButterfly(plot_data, title='TopoButterfly', w=8)
p.set_time(2.5) # set time point of topography to be shown on the right

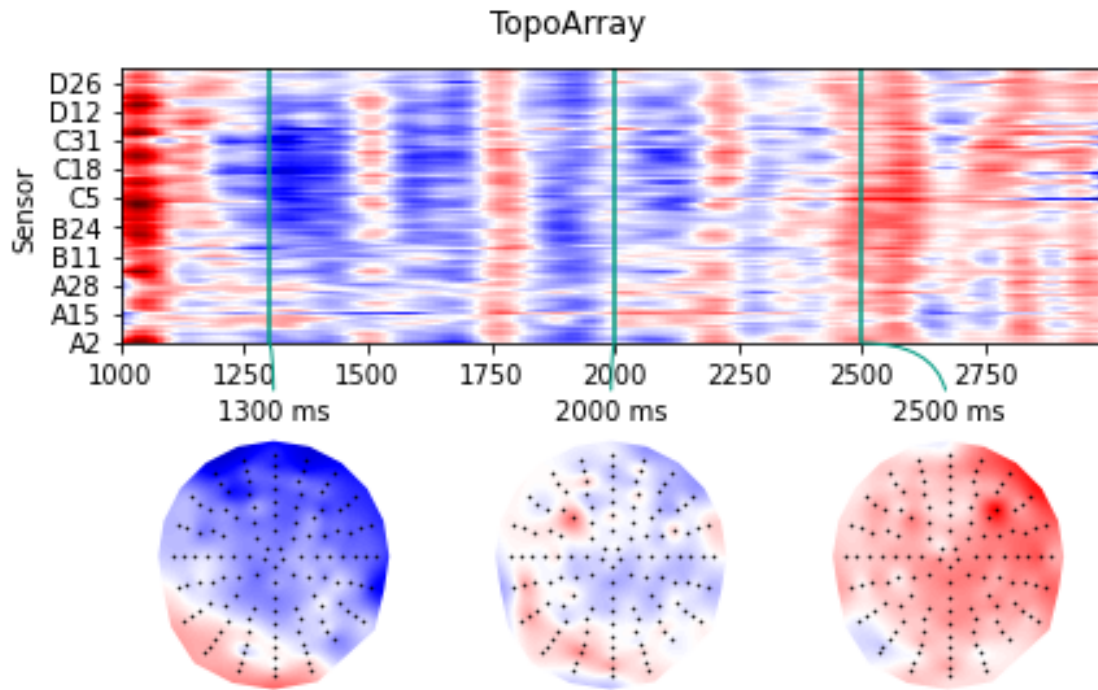
p = eel.plot.TopoArray(plot_data, t=(1.3, 2, 2.5), title='TopoArray', h=4, w=6)
```


Topomap



TopoButterfly





5 3 Preprocessing

5.1 3.1 Speech feature extraction

```
[16]: # loading wav files
wav = eel.load.wav(data_path / 'Section3' / 'audio1.wav')
wav = wav.sub(channel=1) # select only one channel
print(wav)

# extracting envelopes
env = wav.envelope()
print(env)
```

```
<NDVar 'audio1.wav' int16: 7830528 time>
<NDVar 'audio1.wav': 7830528 time>
```

```
[17]: env = eel.resample(env, 1000) # resample with anti-aliasing
env = eel.filter_data(env, None, 8) # lowpass at 8 Hz
env = eel.resample(env, 100) # resample to 64 kHz
env = env.clip(min=0) # clip below zero
env.name += ' envelope' # change name of NDVar

# normalize
```

```
env.x = env.x / env.max()
wav.x = wav.x / wav.max()

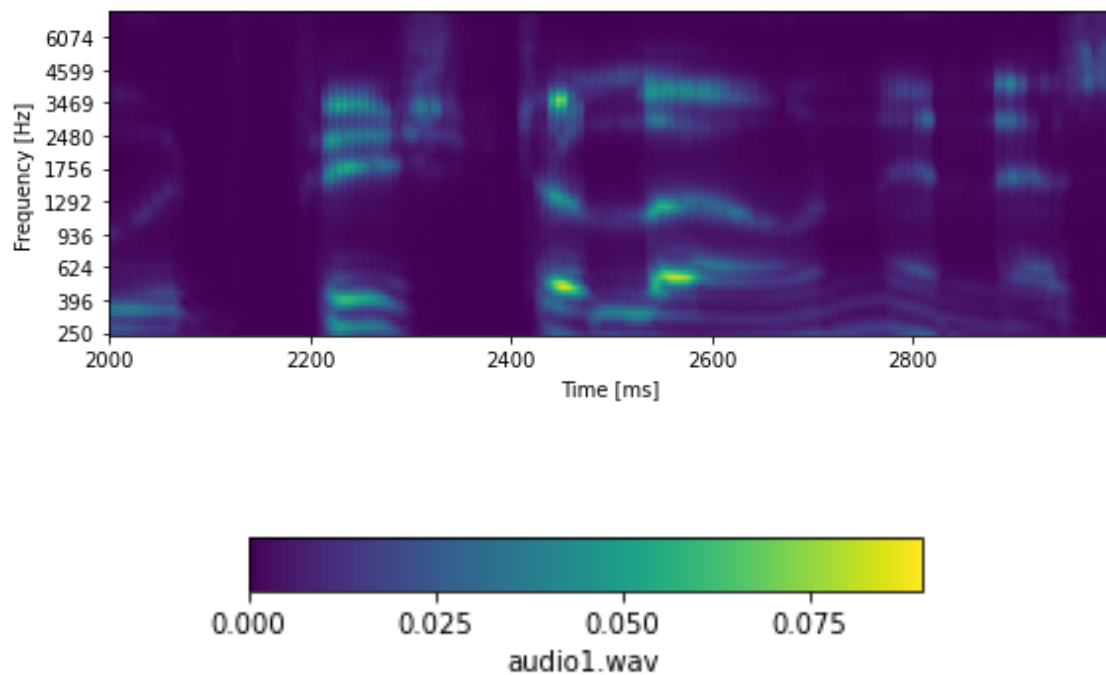
print(env)
```

<NDVar 'audio1.wav envelope': 17756 time>

```
[ ]: # extracting spectrograms
# this may take a minute
# this block can be skipped
gt = trftools.gammatone_bank(wav, f_min=250, f_max=8000, n=64, tstep=0.001)
eel.save.pickle(gt, results_path / 'Section3_gt.pickle')
```

```
[18]: # to load pre-computed gt uncomment line below
# gt = eel.load.unpickle(data_path / 'results' / 'Section3_gt.pickle')
print(gt)
p = eel.plot.Array(gt.sub(time=(2,3)), cmap='viridis', w=8, h=3)
cbar = p.plot_colorbar()
```

<NDVar 'audio1.wav': 64 frequency, 177562 time>

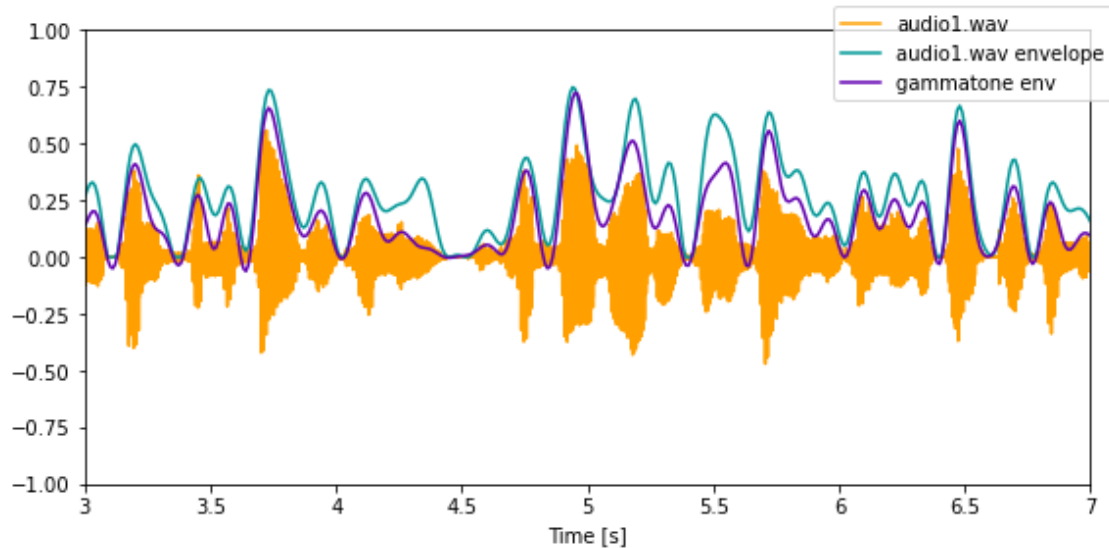


```
[19]: # average across frequency to extract envelope
gt_env = gt.mean('frequency')
gt_env = eel.resample(eel.filter_data(gt_env, None, 8), 100) # lowpass and
↳downsample
```

```
gt_env.x = gt_env.x / gt_env.max()
gt_env.name = 'gammatone env'
print(gt_env)
```

<NDVar 'gammatone env': 17756 time>

```
[20]: # plot results
plot_data = [[wav, env, gt_env]]
p = eel.plot.UTS(plot_data, xlim=(3, 7), legend=True)
```



5.2 3.2 Loading EEG files

EEG files can be loaded in several ways

1. Load EEG raw files using mne load functions https://mne.tools/stable/reading_raw_data.html
For example, `mne.io.read_raw_bdf()`
Then load mne .fif files directly into eelbrain
Automatically generates NDVars with appropriate sensor dimensions
See Alice dataset scripts <https://github.com/Eelbrain/Alice>
2. Load files using pickle format as shown above
3. Load MATLAB files using scipy, pymatreader etc.
4. Load CND files into eelbrain

5.2.1 3.2.1 Loading CND files

This section requires that the LalorNatSpeech dataset is downloaded and is in the correct location.
Download the dataset from the CNSP workshop website.
Place it in the following folder, relative to this folder

```
tutorials
    eelbrain_tutorial
        cnspr_tutorial_eelbrain.ipynb (this file)
    ...
datasets
    LalorNatSpeech
    other datasets
```

```
[21]: # loading CND files
fs = 64
fs_orig = 128
cnd_path = cwd_path.absolute().parents[1] / 'datasets' / 'LalorNatSpeech' / 'dataCND'
data = eel.load.cnd(cnd_path / 'dataSub10.mat') # data is an eelbrain Dataset object
print(data.head(), '\n') # view a summary of the Dataset
print(data['eeg'][0])
print(data['extChan'][0])
```

eeg	origTrialPosition	extChan
<NDVar 'eeg'...	0	<NDVar 'Mast...
<NDVar 'eeg'...	1	<NDVar 'Mast...
<NDVar 'eeg'...	2	<NDVar 'Mast...
<NDVar 'eeg'...	3	<NDVar 'Mast...
<NDVar 'eeg'...	4	<NDVar 'Mast...
<NDVar 'eeg'...	5	<NDVar 'Mast...
<NDVar 'eeg'...	6	<NDVar 'Mast...
<NDVar 'eeg'...	7	<NDVar 'Mast...
<NDVar 'eeg'...	8	<NDVar 'Mast...
<NDVar 'eeg'...	9	<NDVar 'Mast...

<NDVar 'eeg': 23186 time, 128 sensor>
 <NDVar 'Mastoids': 23186 time, 2 channel>

```
[22]: # reshape eegs with first dimension as sensor
eegs = []
for eeg in data['eeg']:
    eeg1 = eel.NDVar(eeg.get_data(('sensor', 'time')), # re-order dimensions
                    eeg.get_dims(('sensor', 'time')))
    eeg1 = eeg1.sub(time=(0, 150)) # crop the first 150 seconds to match the stimulus
    eegs.append(eeg1)
eegs = eel.combine(eegs)
print(eegs)
```

<NDVar: 20 case, 128 sensor, 19200 time>

```
[23]: # do the same for the reference channels
exts = []
for ext in data['extChan']:
    ext1 = eel.NDVar(ext.get_data(('channel', 'time')), # re-order dimensions
                    ext.get_dims(('channel', 'time')))
    ext1 = ext1.sub(time=(0, 150)) # crop the first 150 seconds to match the
    ↪ stimulus
    exts.append(ext1)
exts = eel.combine(exts)
print(exts)
```

<NDVar: 20 case, 2 channel, 19200 time>

5.2.2 3.2.1 Loading MATLAB files

```
[24]: # loading .mat stimulus files
stim_mat = scipy.io.loadmat(cnd_path / 'dataStim.mat')
stim = stim_mat['stim'][0][0][-1] # select stimulus field
print('stim', stim.shape, '(features, trials)')
print('stim[0,0]', stim[0,0].shape, 'numpy array')
```

stim (2, 20) (features, trials)
stim[0,0] (22729, 1) numpy array

```
[25]: # dataStim.mat is a CND file, so it can also be loaded as shown below
data_stim = eel.load.cnd(cnd_path / 'dataStim.mat')
data_stim.head()
```

```
[25]: stimIdxs    Speech_Envelope_Vectors    Word_Onset_Vectors    condIdxs    condition
-----
```

0	<NDVar 'Spee...	<NDVar 'Word...	0	L
1	<NDVar 'Spee...	<NDVar 'Word...	0	L
2	<NDVar 'Spee...	<NDVar 'Word...	0	L
3	<NDVar 'Spee...	<NDVar 'Word...	0	L
4	<NDVar 'Spee...	<NDVar 'Word...	0	L
5	<NDVar 'Spee...	<NDVar 'Word...	0	L
6	<NDVar 'Spee...	<NDVar 'Word...	0	L
7	<NDVar 'Spee...	<NDVar 'Word...	0	L
8	<NDVar 'Spee...	<NDVar 'Word...	0	L
9	<NDVar 'Spee...	<NDVar 'Word...	0	L

```
[26]: # store envelopes as a list of NDVars
envs = []
for st in stim[0]:
    env = eel.NDVar(st[:,0], eel.UTS(0, 1/fs_orig, len(st))) # create NDVar
    env = env.sub(time=(0, 150)) # crop to 150s
    envs.append(env)
```

```
envs = eel.combine(envs)
print(envs)
```

<NDVar: 20 case, 19200 time>

5.3 3.3 Filtering and downsampling

Simple FIR filters are used here

For more flexibility take a look at the following - filtering with mne https://mne.tools/stable/generated/mne.filter.filter_data.html - filtering with scipy

```
[27]: lf = 1 # lower passband edge frequency
      hf = 8 # higher passband edge frequency

      envs_f = eel.filter_data(envs, None, hf)
      envs_fd = eel.resample(envs_f, fs).clip(min=0)

      # operations on a single line
      eegs_fd = eel.resample(eel.filter_data(eegs, lf, hf), fs)
      exts_fd = eel.resample(eel.filter_data(exts, lf, hf), fs)

      print(envs_fd, eegs_fd, exts_fd)
```

<NDVar: 20 case, 9600 time> <NDVar: 20 case, 128 sensor, 9600 time> <NDVar: 20 case, 2 channel, 9600 time>

5.4 3.4 Reject bad channels, re-reference

```
[28]: # determine bad channels using correlation with neighboring sensors
      eeg_concat = eel.concatenate(eegs_fd) # concatenate eegs across trials
      nc = eel.neighbor_correlation(eeg_concat)
      print('bad channels are', nc.sensor.names[nc < 0.3])
      print('indices', [i for i in range(128) if nc[i] < 0.3])
```

bad channels are ['A1', 'B8', 'B9', 'D13']
indices [0, 39, 40, 108]

```
[29]: # reject bad channels, re-reference and normalize
      goods = nc.sensor.names[nc > 0.3] # good channels

      eeg_preprocessed = eegs_fd.sub(sensor=goods) # select only good channels
      eeg_preprocessed -= exts_fd.mean('channel') # re-reference
      eeg_preprocessed.x /= eeg_preprocessed.std() # normalize

      print(eeg_preprocessed)
```

<NDVar: 20 case, 124 sensor, 9600 time>

5.5 3.5 Artifact rejection using ICA

```
[30]: # ica using mne
data_for_raw = eel.concatenate(eeg_preprocessed) # concatenate data

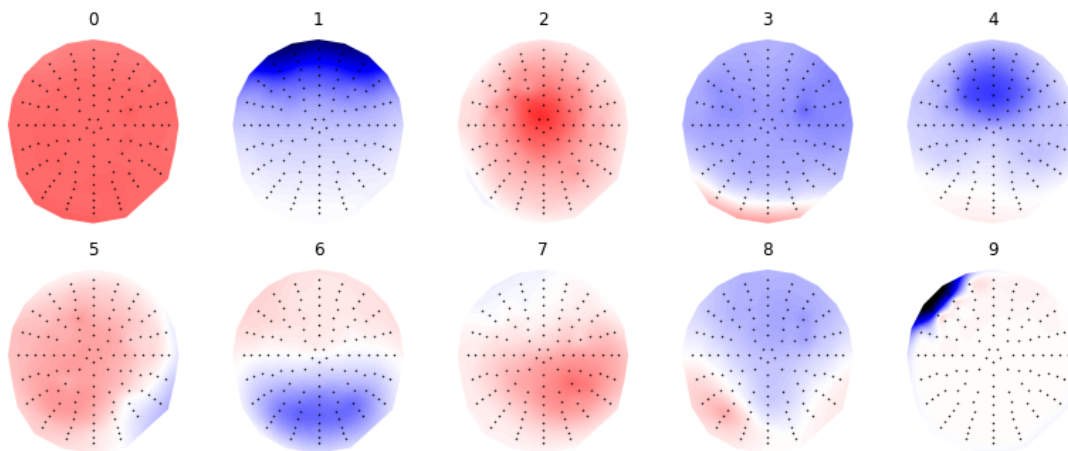
# to apply ica, we need to convert our data to mne raw objects
# for more details on these steps, see https://mne.tools/stable/generated/mne.io.RawArray.html
info = mne.create_info(ch_names=data_for_raw.sensor.names, sfreq=fs,
    ch_types='eeg')
info.set_montage('biosemi128')
raw = mne.io.RawArray(data_for_raw.x, info=info)
```

```
[ ]: # run ICA (takes about a minute)
# this block can be skipped
ica = mne.preprocessing.ICA(n_components=0.99)
ica.fit(raw, picks=data_for_raw.sensor.names)

#ica.save(results_path / 'Section3_dataSub10-ica.fif', overwrite=True)
```

```
[32]: # to load pre-computed ica file uncomment line below
ica = mne.preprocessing.read_ica(data_path / 'results' /
    'Section3_dataSub10-ica.fif')

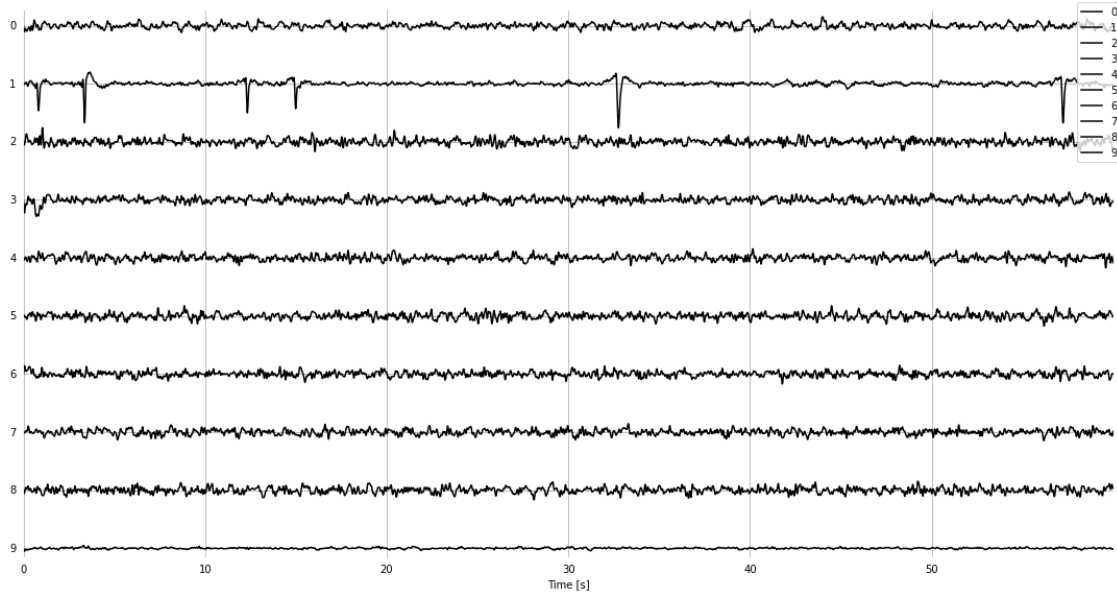
# plot the ICA components
ica_comps = eel.NDVar(ica.get_components().T, (eel.Case, eeg_preprocessed.
    sensor)) # get components as NDVar
p = eel.plot.Topomap(ica_comps[:10], '.case', ncol=5) # plot components
```



```
[33]: # plot component time_courses
ica_sources = ica.get_sources(raw).get_data() # get time courses
```



```
ica_sources = eel.NDVar(ica_sources, (eel.Case, data_for_raw.time)) # convert
↳ to NDVar
p = eel.plot.LineStack(ica_sources[:10].sub(time=(0,60)), h=8, w=15) # plot
↳ time courses
```

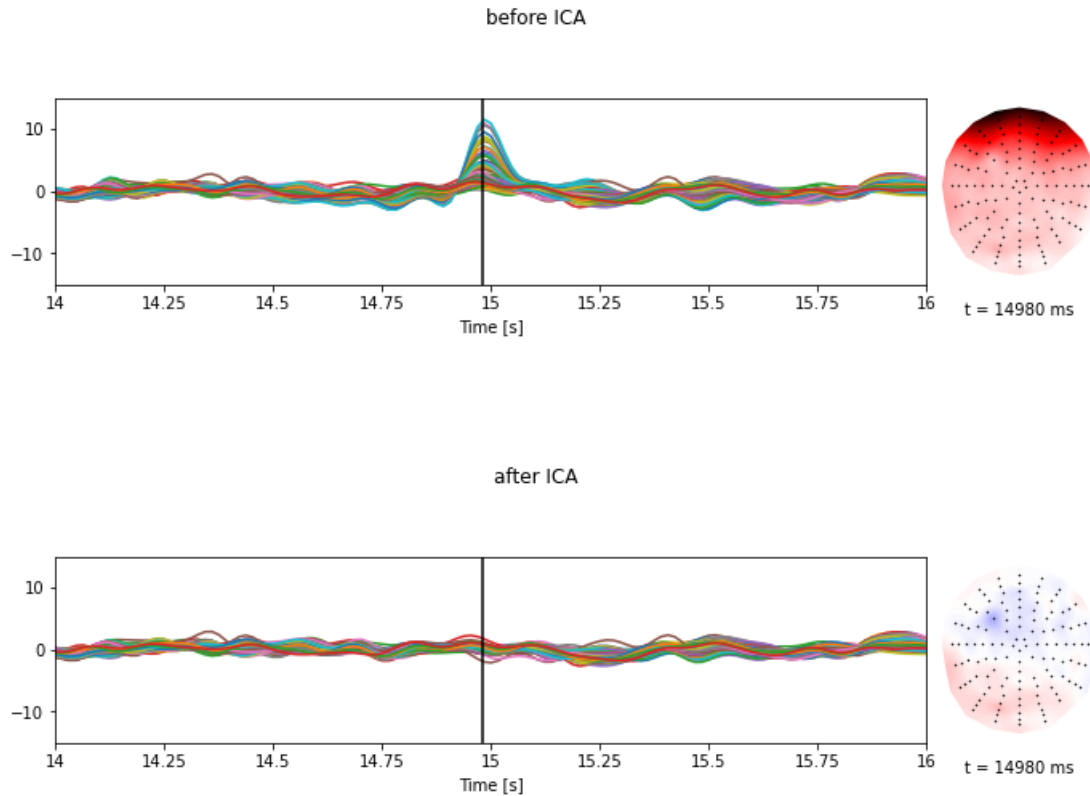


```
[34]: ica.exclude = [1, 9] # reject 1st component (eyeblinks)
ica.exclude
```

```
[34]: [1, 9]
```

```
[35]: # apply ICA to data
# we need to again create mne objects to apply the ICA
ep = mne.EpochsArray(eeg_preprocessed.x, info) # create mne Epochs object
ep_ica = ica.apply(ep) # apply ICA
eeg_ica = eel.NDVar(ep_ica.get_data(), eeg_preprocessed.dims) # convert to NDVar
```

```
[36]: # plot eeg before and after ICA
p = eel.plot.TopoButterfly(eeg_preprocessed[0],
                           xlim=(14, 16), vmax=15, vmin=-15, title='before ICA')
p.set_time(14.98) # time point with an eyeblink
p = eel.plot.TopoButterfly(eeg_ica[0],
                           xlim=(14, 16), vmax=15, vmin=-15, title='after ICA')
p.set_time(14.98)
```



6 4.0 Encoding and Decoding Models (TRFs)

6.1 4.1 TRFs with boosting

See the documentation for more details on boosting <https://eelbrain.readthedocs.io/en/stable/generated/eelbrain.b>

```
[ ]: # this may take a few minutes
# this block can be skipped and the pre-computed TRF loaded below

# boosting arguments
tstart = -0.05 # start lag of TRF in seconds
tstop = 0.35 # stop lag of TRF in seconds
basis = 0.03 # basis function width for smoother TRFs (hamming window of 30 ms ↵
↵width)
test = True # perform cross-validation with held-out test data for model fit ↵
↵calculation
partitions = 4 # number of partitions for cross_validation
eeg_ica.name = 'eeg'
envs_fd.name = 'env'
res1 = eel.boosting(eeg_ica, envs_fd, tstart, tstop, basis=basis, test=test, ↵
↵partitions=partitions, partition_results=True)
```

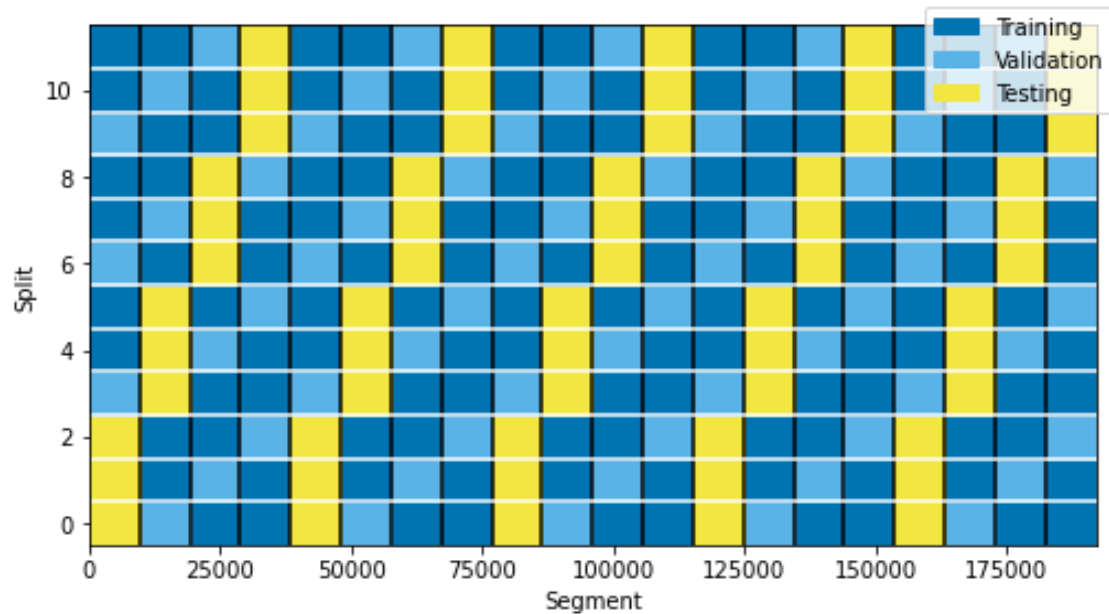
```
eel.save.pickle(res1, results_path / 'Section4_TRF_env.pickle')
```

```
[37]: # to load precomputed TRFs, uncomment line below
# res1 = eel.load.unpickle(data_path / 'results' / 'Section4_TRF_env.pickle')

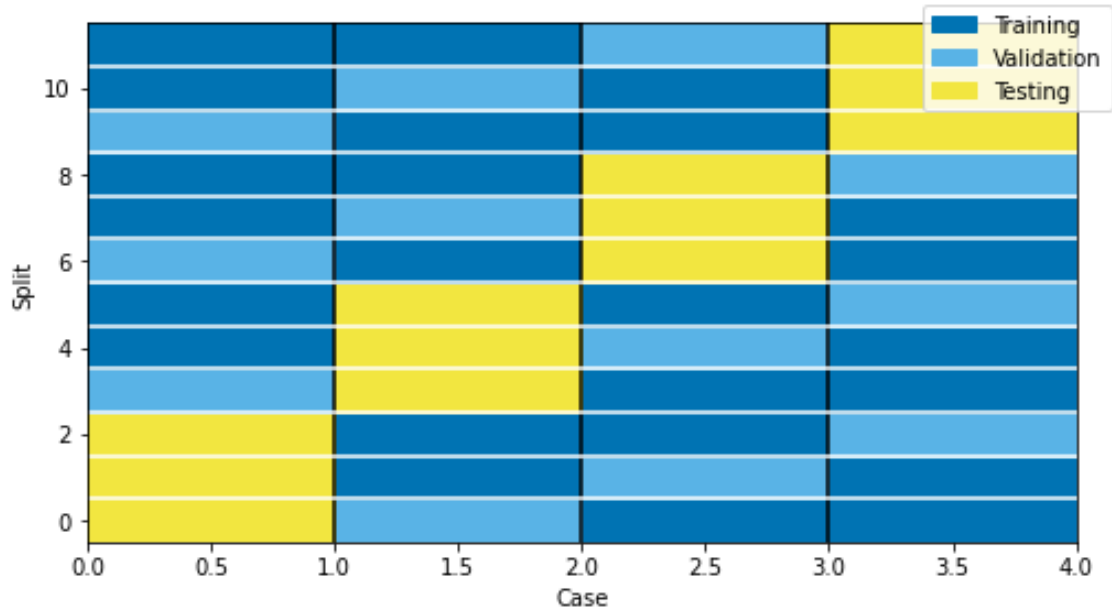
print(res1)
```

```
<boosting eeg ~ env, -0.05 - 0.35, basis=0.03, partitions=4, test=True>
```

```
[38]: # crossvalidation folds
_ = res1.splits.plot(h=4)
```



```
[39]: _ = eel.plot.preview_partitions(4, 4, test=True, validate=True, h=4)
```



6.2 4.2 Exercise - Visualize TRFs

The BoostingResult object `res1` has several properties of interest. - `res1.r` = The model fit (pearson correlation between actual and predicted signals) - `res1.h` = The estimated TRF

Plot these two properties to view the estimated TRFs.

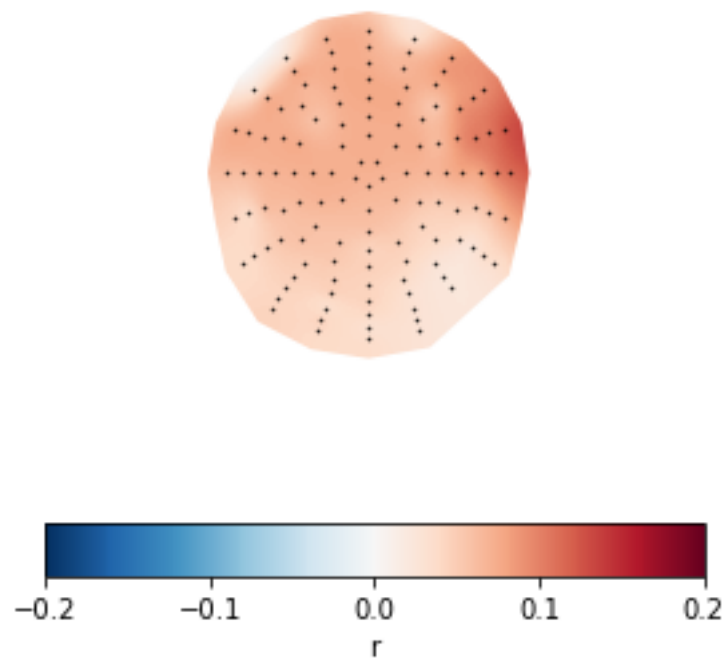
You can use some of eelbrain's plotting functions.

To learn more about all the properties of the BoostingResult object, see <https://eelbrain.readthedocs.io/en/stable/generated/eelbrain.BoostingResult.html>

```
[40]: # plot model fit correlation

# write code here
p = eel.plot.Topomap(res1.r)
p.plot_colorbar();
print(f'rmean = {res1.r.mean():.5g}')
print(f'rmax = {res1.r.max():.5g} at sensor {res1.r.argmax()}')
```

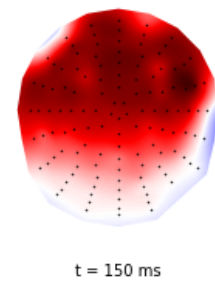
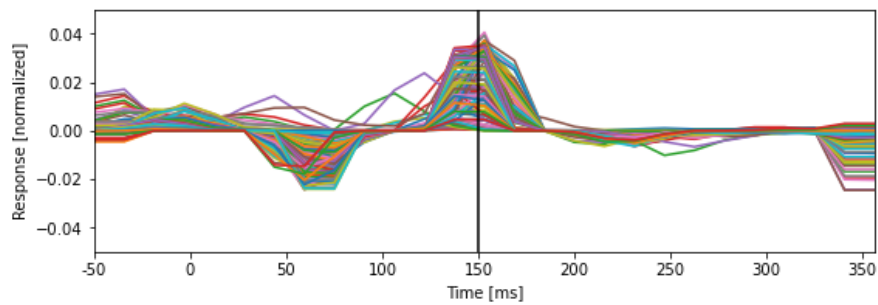
```
rmean = 0.060194
rmax = 0.12122 at sensor B27
```

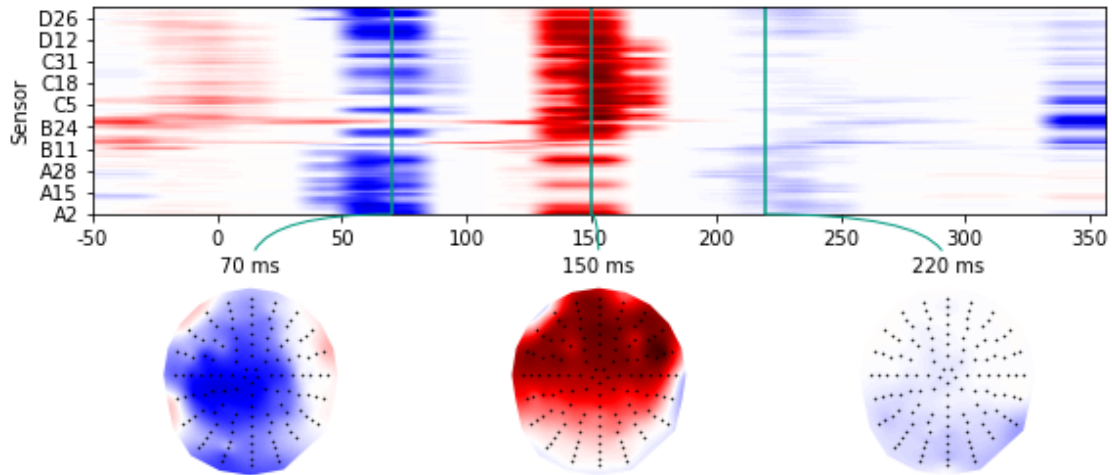


[41]: *# plot TRFs*

write code here

```
eel.plot.TopoButterfly(res1.h, t=0.15, h=3, w=10);
eel.plot.TopoArray(res1.h, t=(0.07, 0.15, 0.22), h=4, w=8);
```





6.3 4.3 Exercise - TRFs to multiple predictors

TRFs to multiple predictors can be jointly estimated by passing the predictors as a list to the boosting function.

e.g., `res = boosting(eeg, [pred1, pred2], -0.1, 0.5)`

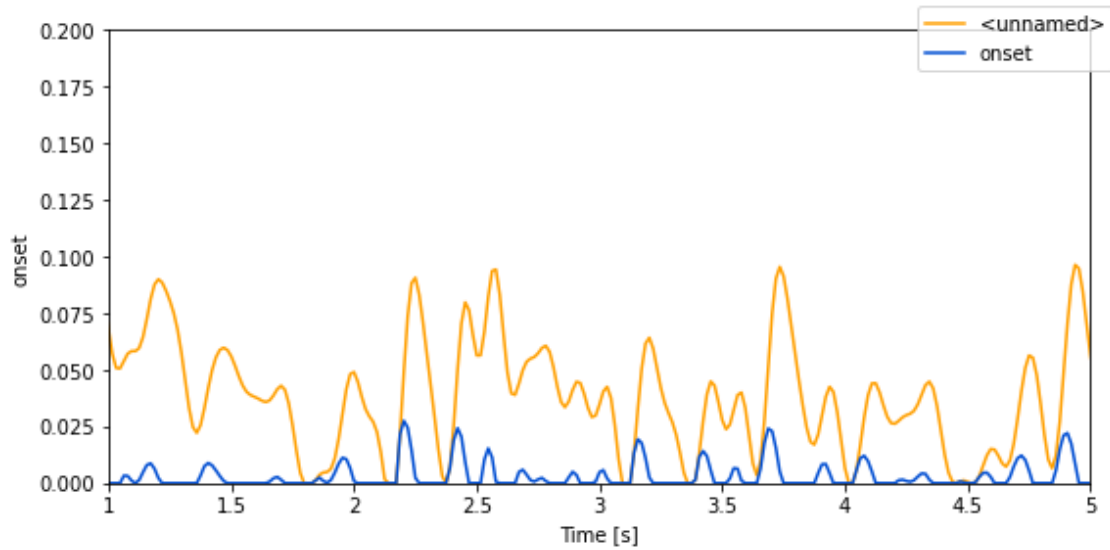
1. Extract the onsets from the speech envelope and plot them
2. Jointly estimate TRFs to both envelopes and onsets and plot the TRFs

```
[42]: # extract onsets

# write code here
onsets = envs_fd.diff().clip(min=0, name='onset')
```

```
[43]: # plot onsets and envelopes

# write code here
eel.plot.UTS([[envs_fd[0], onsets[0]]], xlim=(1,5), legend=True);
```



```
[ ]: # estimate TRFs to envelopes and onsets

# write code here
res2 = eel.boosting(eeg_ica, [envs_fd, onsets], -0.05, 0.35, basis=0.03,
    ↪ partitions=4, test=True)
```

```
[44]: # loading precomputed TRF
res2 = eel.load.unpickle(results_path / 'Section4_trf_env+onset.pickle')
```

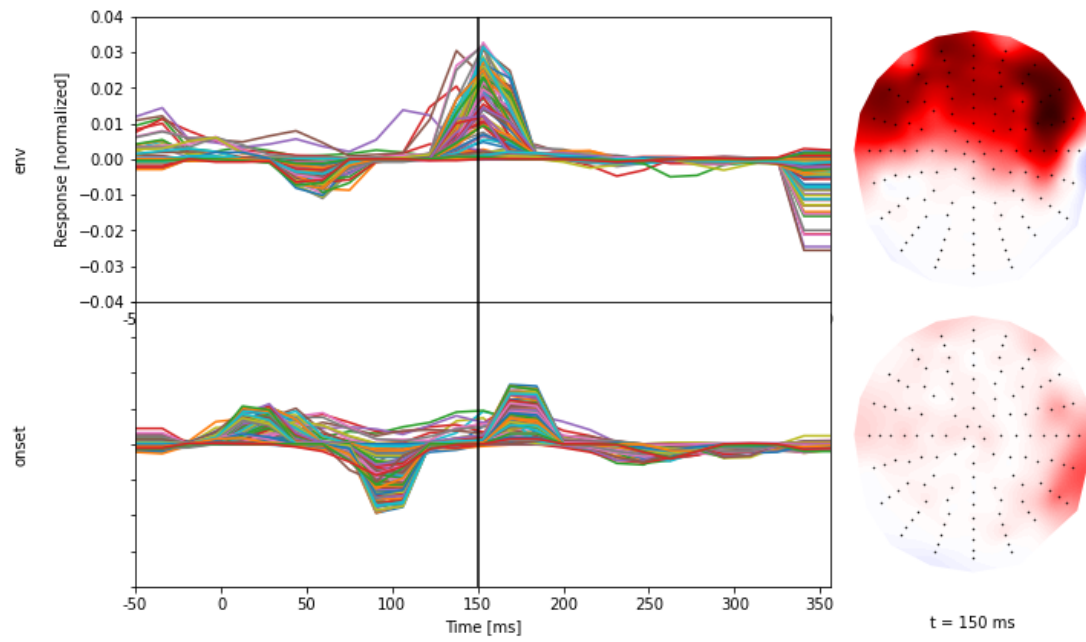
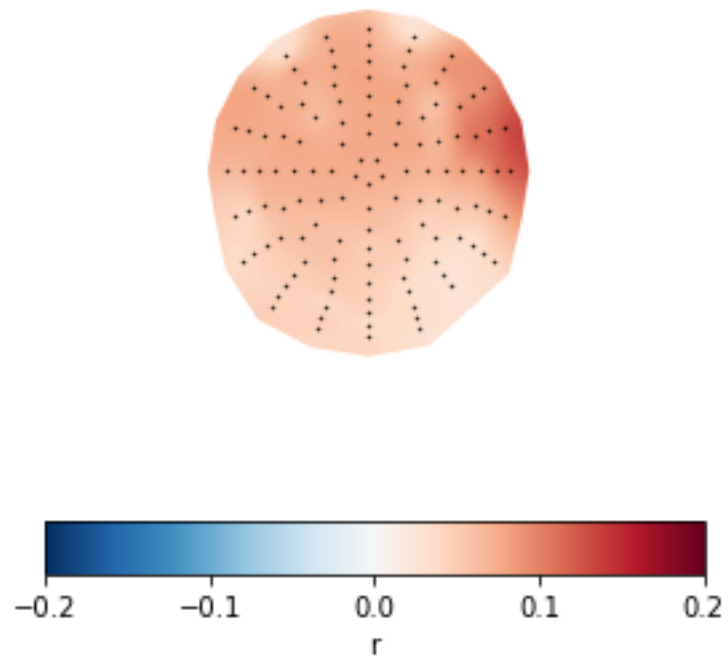
```
[45]: # plot TRFs to envelopes and onsets

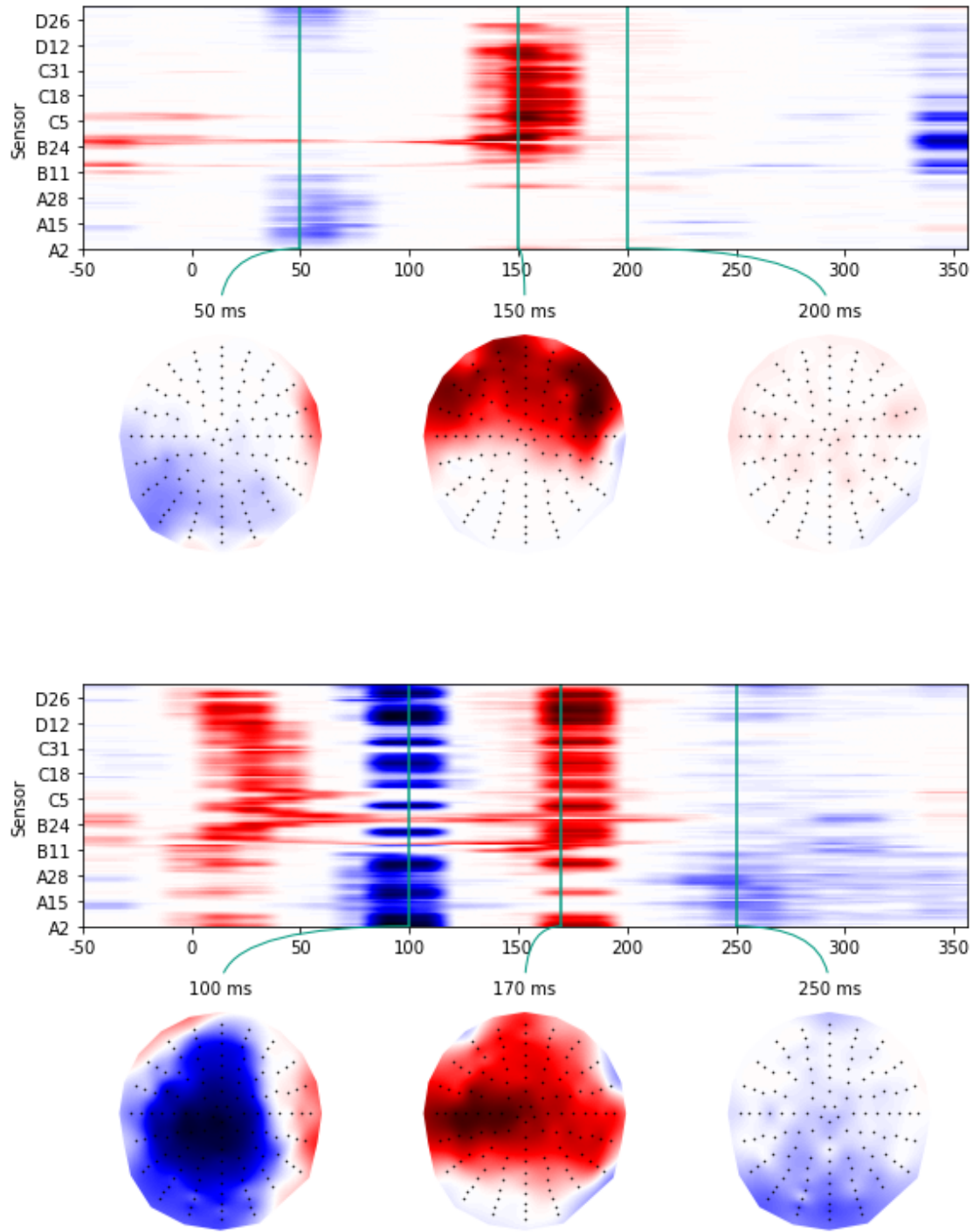
# write code here
p = eel.plot.Topomap(res2.r)
p.plot_colorbar();
print(f'rmean = {res2.r.mean():.5g}')
print(f'rmax = {res2.r.max():.5g} at sensor {res2.r.argmax()}')

eel.plot.TopoButterfly([res2.h[0], res2.h[1]], t=0.15);

eel.plot.TopoArray(res2.h[0], t=(0.05, 0.15, 0.2), w=8);
eel.plot.TopoArray(res2.h[1], t=(0.1, 0.17, 0.25), w=8);
```

```
rmean = 0.063537
rmax = 0.12424 at sensor B27
```





6.4 4.4 Excercise - Decoding models

Decoders can be estimated by switching the response and predictor variables in the boosting function.

The time lags should now also be specified in reversed order (i.e., negative time)

Reconstruct the speech envelope from the eeg using the decoding approach

What is the model fit correlation?

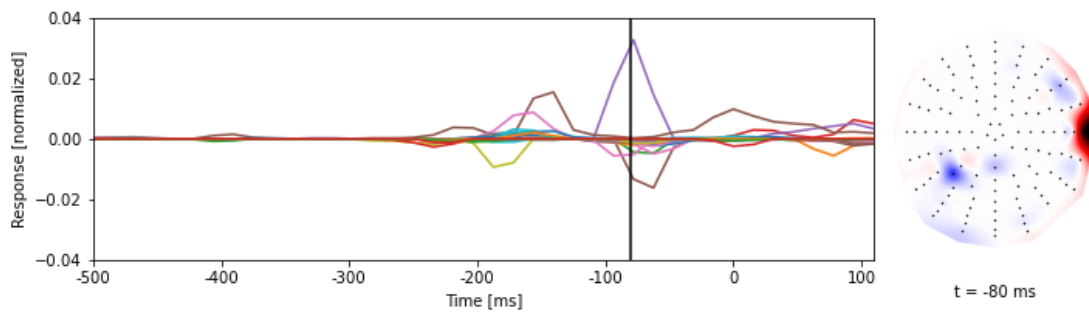
```
[ ]: # Estimate decoders

# write code here
resd = eel.boosting(envs_fd, eeg_ica, -0.5, 0.1, basis=0.03, test=True,
↳ partitions=4)

[46]: # loading precomputed decoder
resd = eel.load.unpickle(results_path / 'Section4_decoder.pickle')

[47]: print(f'decoder average r = {resd.r.mean():.5g}')
eel.plot.TopoButterfly(resd.h, t=-0.08);
```

decoder average r = 0.18497



7 5.0 Statistical Tests

7.1 5.1 Load Alice dataset TRFs

Precomputed TRFs are loaded for running statistical tests across multiple subjects

Following TRF models are loaded - envelope : model with only the envelope TRF - envelope+onset : model with both envelope and onset TRFs - envelope+onset_shuff0 : model with both envelope and onset TRFs. But the onset predictor has been shuffled (shifted by 30s) to serve as a null model.

```
[48]: # get subject names
subject_paths = [f for f in data_path.glob('Alice_TRFs/S*')]
subjects = [f.stem for f in subject_paths]
print(subjects)
```

```
['S01', 'S03', 'S04', 'S05', 'S06', 'S08', 'S10', 'S11', 'S12', 'S13', 'S14',
'S15', 'S16', 'S17', 'S18', 'S19', 'S20', 'S21', 'S22', 'S25', 'S26', 'S34',
'S35', 'S36', 'S37', 'S38', 'S39', 'S40', 'S41', 'S42', 'S44', 'S45', 'S48']
```

```
[49]: # define models and trfs in dictionaries
models = ['envelope', 'envelope+onset', 'envelope+onset_shuff0']
trf_names = ['envelope', 'onset']
model_fits = {}
model_trfs = {}

# load models
for model in models:
    # initialize lists to store TRFs
    model_fits[model] = []
    for trf_name in trf_names:
        if trf_name in model: # add onset TRFs only for onset models
            model_trfs[model+'_'+trf_name] = []

    for subject in subjects:
        temp = eel.load.unpickle(data_path / 'Alice_TRFs' / subject /
    ↪f'{subject}_{model}.pickle')
        model_fits[model].append(temp.r) # store model fit correlation

        for i, trf_name in enumerate(trf_names):
            if trf_name in model:
                trf = temp.h[i].sub(time=(0,0.3)) # extract trf from 0-300 ms
                model_trfs[model+'_'+trf_name].append(trf)

# convert lists to NDVars for statistical testing
for k in model_fits.keys():
    model_fits[k] = eel.combine(model_fits[k]) # combine from lists into NDVars
    ↪with Case dimension

for k in model_trfs.keys():
    model_trfs[k] = eel.combine(model_trfs[k]) # combine from lists into NDVars
    ↪with Case dimension
```

```
[50]: model_fits
```

```
[50]: {'envelope': <NDVar 'Correlation': 33 case, 61 sensor>,
      'envelope+onset': <NDVar 'Correlation': 33 case, 61 sensor>,
      'envelope+onset_shuff0': <NDVar 'Correlation': 33 case, 61 sensor>}
```

```
[51]: model_trfs
```

```
[51]: {'envelope_envelope': <NDVar 'envelope': 33 case, 61 sensor, 30 time>,
      'envelope+onset_envelope': <NDVar 'envelope': 33 case, 61 sensor, 30 time>,
      'envelope+onset_onset': <NDVar 'onset': 33 case, 61 sensor, 30 time>,
      'envelope+onset_shuff0_envelope': <NDVar 'envelope': 33 case, 61 sensor, 30
time>,
```

```
'envelope+onset_shuff0_onset': <NDVar: 33 case, 61 sensor, 30 time>}
```

7.2 5.2 Cluster based statistical tests

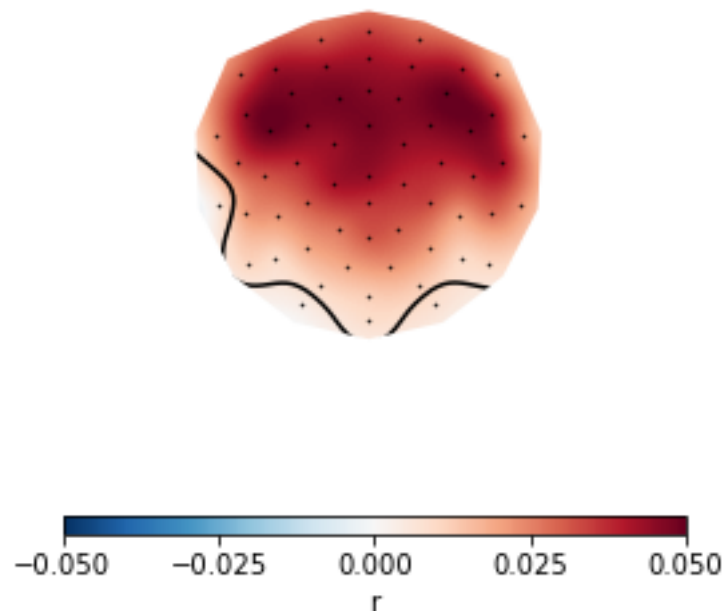
```
[53]: # running tests may take a while
# run this block to load precomputed test results
# test blocks can be skipped and pre-computed tests can be loaded
# tests = eel.load.unpickle(data_path / 'results' / 'Section5_tests.pickle')

[ ]: # do not run this if you are loading precomputed tests
tests = {}

[ ]: # prediction accuracy of a single model
# test if significantly > 0
# permutation tests based on TFCE
tests['r_envelope'] = eel.testnd.TTestOneSample(model_fits['envelope'], tail=1,
↪tfce=True, samples=1000)

[54]: # plot results
print(tests['r_envelope'])
p = eel.plot.Topomap(tests['r_envelope'])
_ = p.plot_colorbar(width=0.1)
```

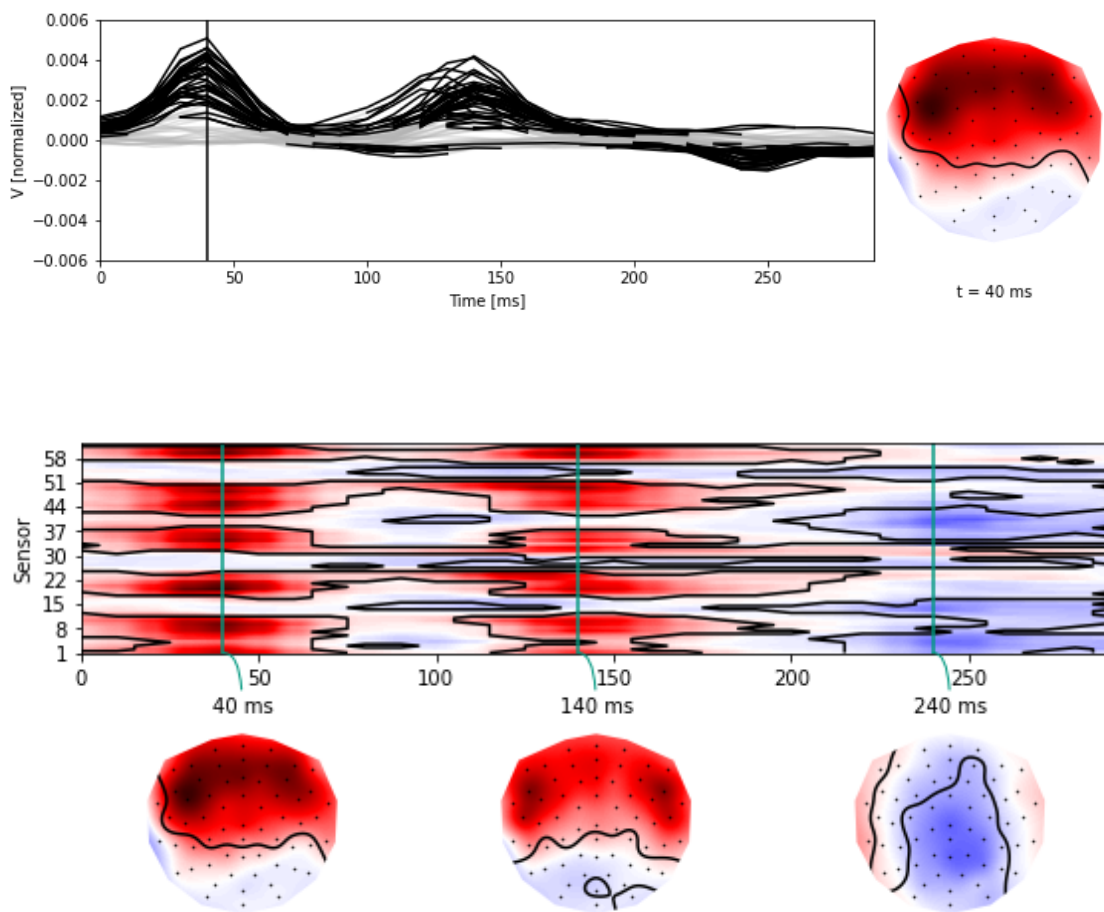
```
<TTestOneSample 'Correlation', tail=1, samples=10000, tfce=True, p < .001>
```



```
[ ]: # test envelope TRFs have consistent non-zero activity across subjects
tests['trf_envelope_only'] = eel.testnd.
    ↪TTestOneSample(model_trfs['envelope_envelope'],
                    tail=0, tfce=True,
    ↪samples=1000)
```

```
[55]: print(tests['trf_envelope_only'])
p = eel.plot.TopoButterfly(tests['trf_envelope_only'], color='black')
p.set_time(0.04)
p = eel.plot.TopoArray(tests['trf_envelope_only'], w=8, h=4, t=(0.04, 0.14, 0.
    ↪24))
```

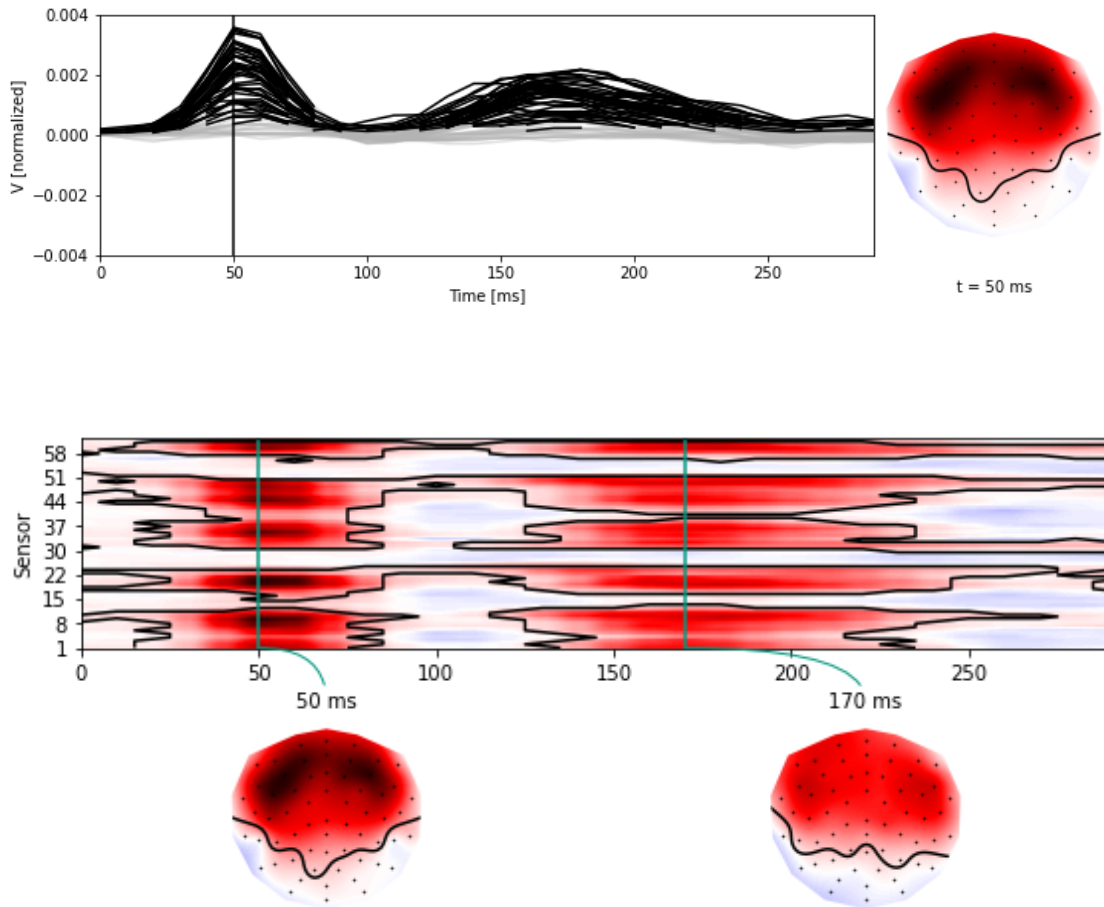
<TTestOneSample 'envelope', samples=10000, tfce=True, p < .001>



```
[ ]: # test onset TRFs
tests['trf_onset'] = eel.testnd.
    ↪TTestOneSample(model_trfs['envelope+onset_onset'],
                    tail=0, tfce=True, samples=10000)
```

```
[56]: print(tests['trf_onset'])
p = eel.plot.TopoButterfly(tests['trf_onset'], color='k')
p.set_time(0.05)
p = eel.plot.TopoArray(tests['trf_onset'], w=8, h=4, t=(0.05, 0.17))
```

```
<TTestOneSample 'onset', samples=10000, tfce=True, p < .001>
```



7.3 5.3 Exercise - Compare two models

Use a cluster based permutation test to compare the prediction accuracy of the model fits of the envelope vs. the envelope + onset models

You can use the following function `eel.testnd.TTestRelated(ndvar1, ndvar2, ...)`

Note, if you want to view the keys in a dictionary, use `model_fits.keys()`

Plot the test result. Does adding the onsets significantly improve the model fits?

```
[ ]: # test if model fit increases when adding onset predictor
```

```

# write code here
tests['r_compare_env_to_env+onset'] = eel.testnd.
↳ TTestRelated(model_fits['envelope+onset'],

↳ model_fits['envelope'],

↳ samples=1000, tail=1)
tfc=True,

```

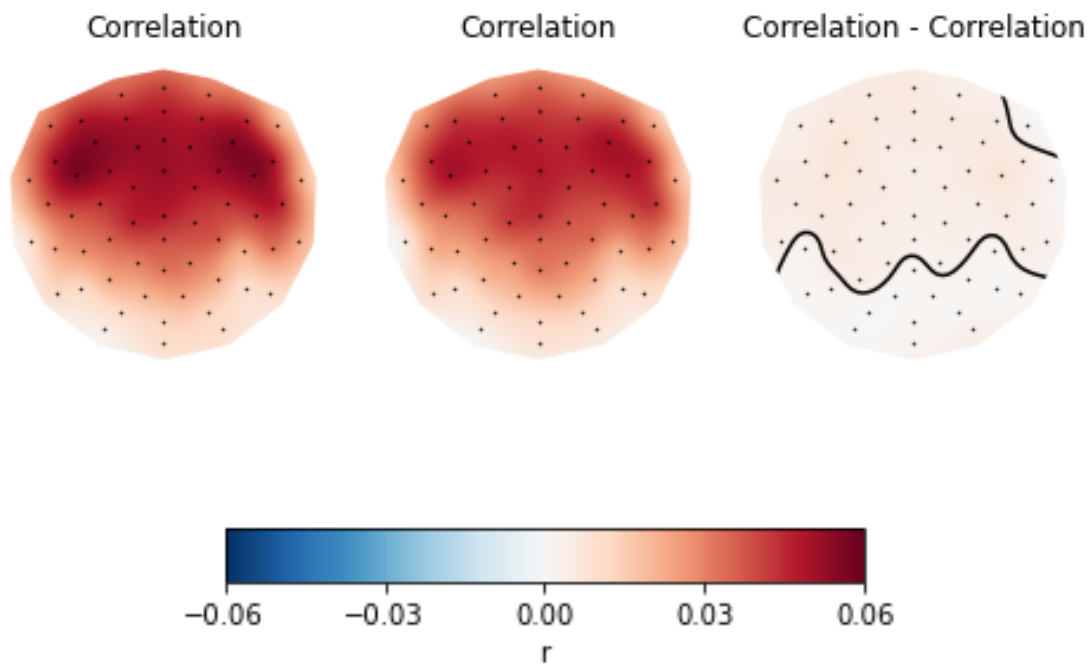
```

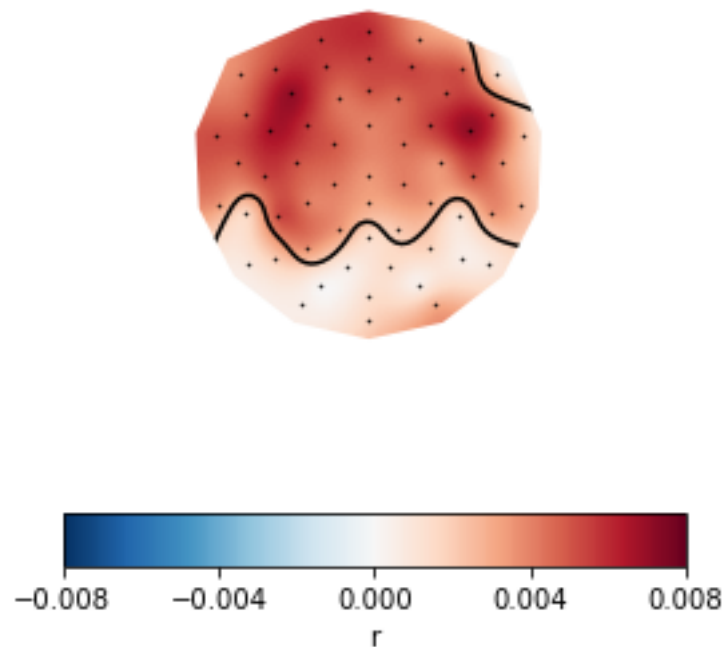
[59]: # plot the test result

# write code here
p = eel.plot.Topomap(tests['r_compare_env_to_env+onset'], ncol=3)
p.plot_colorbar();

masked_diff = tests['r_compare_env_to_env+onset'].masked_difference()
p = eel.plot.Topomap(masked_diff)
p.plot_colorbar();

```





Testing a model with only 1 predictor vs. a model with 2 predictors may not be the best method. An improved method would be to test models with the same number of predictors, but with one predictor shuffled in time.

1. Does the shuffled onset predictor model have significant TRFs?
Test the onset TRFs for the shuffled onset model.
Plot the test result. Is there any significant activity?
2. Repeat the above exercise comparing the model fits for the envelope+onset vs. envelope+onset_shuff0 models.
Plot the test result. Does onset predictor significantly improve the model fits?

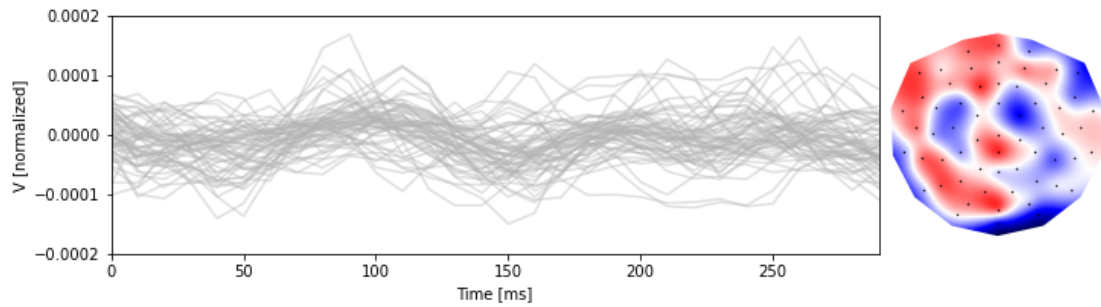
```
[ ]: # test if the shuffled onset TRF has significant activity

# write code here
tests['trf_shuffled_onset'] = eel.testnd.
    ↳TTestOneSample(model_trfs['envelope+onset_shuff0_onset'],
                    tfce=True, tail=0,
    ↳samples=1000)
```

```
[62]: # plot results

# write code here
print(f'p-value = {tests["trf_shuffled_onset"].p.min()}')
p = eel.plot.TopoButterfly(tests['trf_shuffled_onset'], color='black')
```


p-value = 0.8146



```
[ ]: # test if model fit increases compared to a null model (shifted predictor)

# write code here
tests['r_compare_onset_to_shuff_onset'] = eel.testnd.
    ↳TTestRelated(model_fits['envelope+onset'],

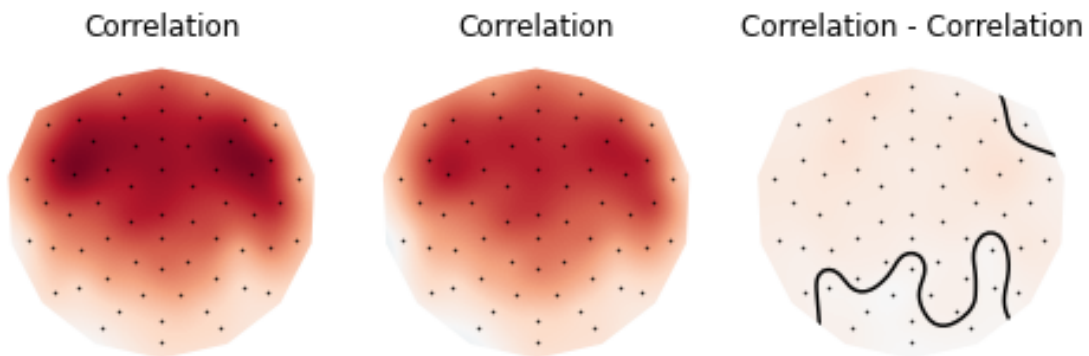
    ↳model_fits['envelope+onset_shuff0'],

    ↳samples=1000, tail=1)

                                     ↳
                                     tfce=True, ↳
```

```
[63]: # plot the test result

# write code here
eel.plot.Topomap(tests['r_compare_onset_to_shuff_onset'], ncol=3);
```



8 6 Additional Topics - see Eelbrain documentation

- Source localization
Eelbrain allows easy computation and visualization of source localized responses based on

mne functions

- ANOVA, ANCOVA

More complex tests are available directly in Eelbrain

- MneExperiment pipeline

Pipeline for preprocessing and TRF estimation that can be specified using high-level functions

[]: