The Eelbrain Python Toolkit
Christian Brodbeck

christianbrodbeck.net / christianbrodbeck@me.com
Overview

Agenda

‣ What you can do with Eelbrain
‣ How easy it is
‣ How to get started

Eelbrain in the Python eco-system

Tutorial manuscript

Components of Eelbrain

‣ Time-series data
‣ Deconvolution
‣ Visualization
‣ Mass-univariate statistics
Eelbrain: toolkit for deconvolution analysis
Eelbrain in the Python eco-system

- **MNE**
  - EEG/MEG data I/O
  - Preprocessing
  - Many analysis tools

- **Eelbrain**
  - N-dimensional time series data
  - Deconvolution
  - High-level visualization
  - Mass-univariate statistics

- **NumPy**
  - N-dimensional array class
  - Optimized array operations

- **SciPy**
  - Extends NumPy for scientific applications (linear algebra, signal processing, image processing, …)

- **Matplotlib**
  - Low-level plotting functions

- **Jupyter**
  - Notebook environment

- **Pingouin**
  - Univariate statistics
  - Machine learning (e.g. classification)

- **R**
  - Export to other platforms
Tutorial for using Eelbrain with the Alice EEG dataset

- Introduces deconvolution analysis and demonstrates several applications
- Analysis of the Alice audiobook listening EEG dataset
- Source code from raw data to figures: https://github.com/Eelbrain/Alice

Get the manuscript!

- Tell us how to improve it
- Ask questions on GitHub
  - On the Alice data/analysis: https://github.com/Eelbrain/Alice/discussions
  - On Eelbrain: https://github.com/christianbrodbeck/Eelbrain/discussions

---

Eelbrain: A Python toolkit for time-continuous analysis with temporal response functions

Christian Brodbeck†, Proloy Das†, Joshua P. Kulasingham†, Shohini Bhattasali†, Phoebe Gaston†, Philip Resnik‡ & Jonathan Z. Simon‡

1) University of Connecticut
2) Massachusetts General Hospital
3) University of Maryland, College Park

* christianbrodbeck@me.com

Abstract

Even though human experience unfolds continuously in time, it entails cascading processes building hierarchical cognitive structures. From perception, humans transform a continuously varying array of phonemes into words, and meaning, and these levels all have different, interdependent time courses. Deconvolution analysis has recently emerged as a promising approach to learn how electrophysiological brain responses related to such complex mental processes change over time. We introduce Eelbrain, a Python toolkit, which makes this kind of analysis more accessible. Here we demonstrate its use, using continuous speech as a sample paradigm. Using the Alice audiobook dataset of audiobook listening, a companion GitHub repository provides working code for the analysis, from raw data to group level statistics. A case study uses a hypothesis-driven approach in which the experimenter specifies different representations of the data and uses these as predictor variables for the electrophysiological signal. This allows one to investigate the relationship between brain activity and speech perception. In particular, we decompose the brain signal into distinct responses associated with e.g., the syllable duration or the presence of a particular vowel, by estimating a multivariate temporal response function (mTRF), quantifying the influence of each predictor on brain responses as a function of time (lags). This allows asking detailed questions about brain function, such as the role of the brainstem in continuous speech perception. The approach is implemented using Eelbrain, a Python toolkit that is freely available online.
Tutorial with code: Auditory TRFs

A) Predictors

![Predictors Graph]

B) Envelope

![Envelope Graph]

C) Envelope + onsets

![Envelope + onsets Graph]
Design

**Easy to install**

- Using Anaconda distribution (download the GUI installer)
- Single command to install all libraries you need with an environment file:
  
  ```bash
  $ conda env create --file=environment.yml
  ```

**Well documented**

- Command reference
- Code examples

**Open source**

- GitHub: [https://github.com/christianbrodbeck/Eelbrain](https://github.com/christianbrodbeck/Eelbrain)
**eelbrain.boosting**

```
eelbrain.boosting(y, x, tstart, tstop, scale_data=True, delta=0.005, mindelta=None, error='l2', basis=0, basis_window='hamming', partitions=None, model=None, validate=1, test=0, ds=None, selective_stopping=0, partition_results=False, debug=False)
```

Estimate a linear filter with coordinate descent

**Parameters**

- `y` (**NDVar**) – Signal to predict.
- `x` (**NDVar** or **sequence of NDVar**) – Signal to use to predict `y`. Can be sequence of NDVars to include multiple predictors. Time dimension must correspond to `y`.
- `tstart` (**scalar** or **sequence of scalar**) – Start of the TRF in seconds. A list can be used to specify different values for each item in `x`.
- `tstop` (**scalar** or **sequence of scalar**) – Stop of the TRF in seconds. Format must match `tstart`.
- `scale_data` (**bool** or **inplace**) – Scale `y` and `x` before boosting: subtract the mean and divide by the standard deviation (when `error='l2'`) or the mean absolute value (when `error='l1'`). Use `inplace` to save memory by scaling the original objects specified as `y` and `x` instead of making a copy.
- `delta` (**float**) – Step for changes in the kernel.
- `mindelta` (**Optional [ float ]**) – If the error for the training data can't be reduced, divide `delta` in half until `delta < mindelta`. The default is `mindelta = delta`, i.e. `delta` is constant.
- `error` (**literal [ 'l1', 'l2' ]**)
partition_results (bool) - Keep results (TPR and model evaluation) for each test-partition. This is disabled by default to reduce file size when saving results.

date (bool) - Add additional attributes to the returned result.

See also

plot.preview_partitions

preview data partitions for cross-validation

Notes

The boosting algorithm is described in 1.

In order to predict data, use the `convolve()` function:

```python
>>> ds = datasets.get_uts()
>>> ds['a1'] = epoch_impulse_predictor('uts', 'A=a1', ds=ds)
>>> ds['a0'] = epoch_impulse_predictor('uts', 'A=a0', ds=ds)
>>> res = boosting('uts', ['a0', 'a1'], 0, 0.5, partitions=10, model='A', ds=ds)
>>> y_pred = convolve(res.h_scaled, ['a0', 'a1'], ds=ds)
```

References

1

Data partitions for boosting

The boosting algorithm can use two different forms of cross-validation: cross-validation as stopping criterion (always on), and cross-validation for model evaluation (optional). This requires partitioning the data into different segments. The `eelbrain.plot.preview_partitions()` function is for exploring the effect of different parameters on the way the data is split.

Validation

During boosting, every training step consists in modifying one element of the kernel/TRF. After every such step, the new TRF is evaluated against the validation data. For continuous data (without `Case` dimension), the default is to split the data into 10 equal-length segments, and perform 10 model fits, each using one of the segments as validation set. In the plots below, each "Split" shown on the y-axis corresponds to a separate run of the boosting algorithm. The results returned by the `boosting()` function would be based on the average TRF of those 10 runs.

```python
# sphinx_gallery_thumbnail_number = 6
from eelbrain import *

p = plot.preview_partitions()
```

The number of partitions can be controlled with the `partitions` parameter:
Design principles

- Analysis is script-based: completely reproducible
- Concise, high-level commands
- Focus on outcome, not implementation

Examples

- Automatic handling of sensor positions for head-maps
- Meaningful indexing
Code Examples
Time-series representations

Represent data with different dimensions

- Time
- Sensors (EEG, MEG)
- Frequency (e.g., of a spectrogram)
- …

Keeps track of meta-information for you

- EEG sensor positions
- Sampling rate
- …
Boosting algorithm

- Boosting algorithm for estimating sparse mTRFs
  - Coordinate descent
  - Early stopping based on cross-validation to prevent over-fitting
- Can handle
  - Large numbers of predictors
  - Correlated predictors

Built-in $k$-fold cross-validation

- Cross-validated predictive power with one command
- Data partitioning based on trials or continuous time
- Retrieve results from different folds as independent estimates
Mass-univariate statistics

- Permutation tests for multiple comparison correction
  - Max-statistic (Nichols & Holmes, 2002)
  - Cluster-mass based correction (Maris & Oostenveld, 2007)
  - Threshold-free cluster enhancement (Smith & Nichols, 2009)

- Works with arbitrary dimensions

- Currently available tests:
  - One-sample, independent and related $t$-tests
  - Pearson correlation
  - ANOVA (fixed, repeated measures, mixed, nested)
  - Two-stage tests for arbitrary linear models

Basic univariate statistics

- ANOVA, $t$-tests, …

- Export to R, …
Extends to source localization

With MNE and boosting

- Example: Brodbeck et al., 2021

Joint source localization and mTRF estimation

- With Neuro-Current Response Functions (Das et al., 2020; https://github.com/proloyd/neuro-currentRF)
Thank you!

If you’re interested:

- Download the Eelbrain tutorial preprint: https://github.com/Eelbrain/Alice/discussions/2
- Download the Alice dataset and analysis code: https://github.com/Eelbrain/Alice
  - Reproduce our results
  - Find new, better predictors for the EEG responses
- Ask questions on GitHub Discussions: https://github.com/Eelbrain/Alice/discussions
- Help us make the tutorial better!