Introduction to BCILAB
A MATLAB Toolbox and EEGLAB Plugin

Christian A. Kothe

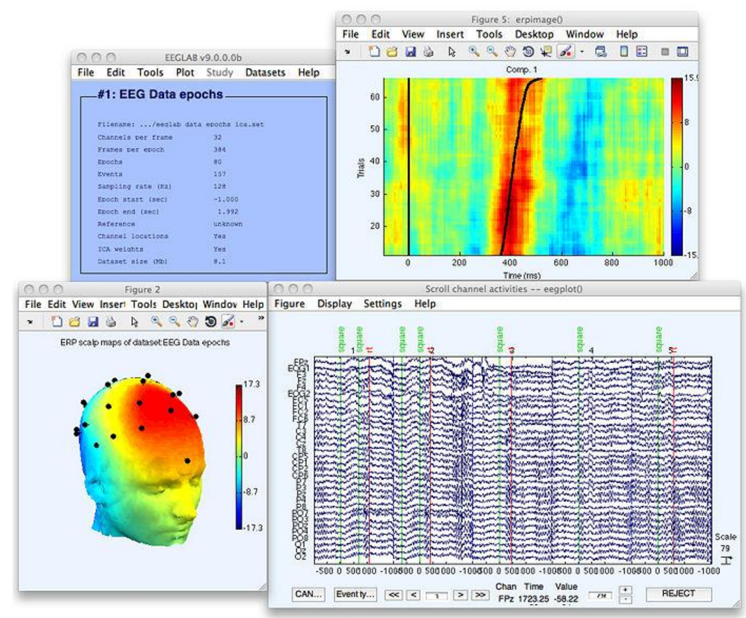
Swartz Center for Computational Neuroscience, UCSD

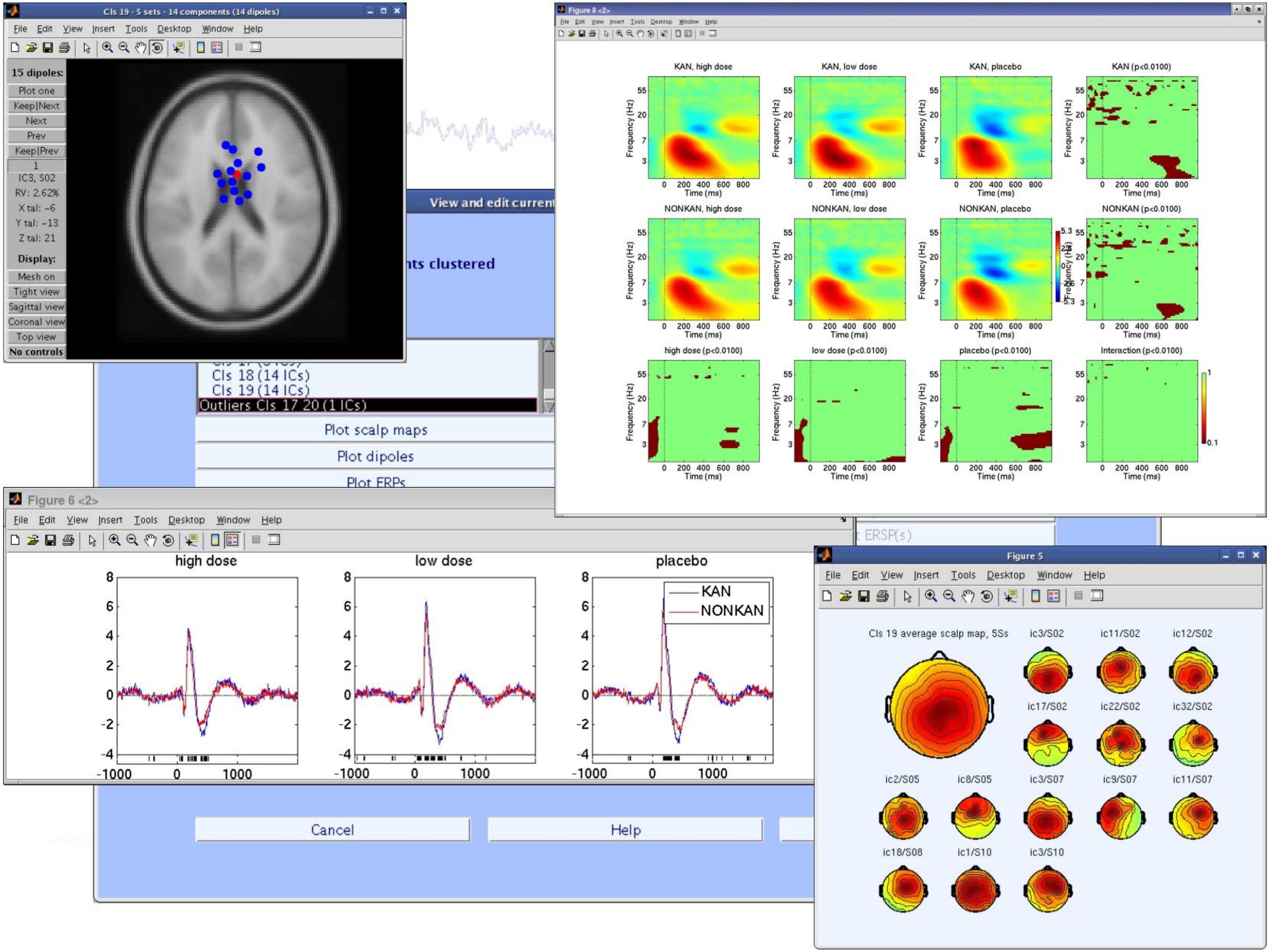
EEGLAB Overview

Swartz Center for Computational Neuroscience

What is EEGLAB?

- Large open-source toolbox for EEG analysis (70k lines, 90k d/loads, 5000-9000 users on discussion list)
- Neuroscience focus & features (ICA, 3d source localization, statistics, multi-subject analysis, graphics)
- Developed by Arno Delorme and Scott Makeig (et al.) under NIH funding
- 20+ plugins (NFT, SIFT, BCILAB, MPT, …)
- Currently being extended for real-time experimentation (MoBILAB, ERICA platform)





Cls 19 - 5 sets - 14 components (14 dipoles)
15 dipoles:
Plot one
Keep|Next
Next
Prev
Keep|Prev
IC3, S02
RV: 2.62%
X tal: -6
Y tal: -13
Z tal: 21
Display:
Mesh on
Tight view
Sagittal view
Coronal view
Top view
No controls
View and edit curren
nts clustered
Cls 18 (14 ICs)
Cls 19 (14 ICs)
Outliers Cls 17 20 (1 ICs)
Plot scalp maps
Plot dipoles
Plot ERPs
Figure 8 <2>
KAN, high dose
KAN, low dose
KAN, placebo
KAN (p<0.0100)
NONKAN, high dose
NONKAN, low dose
NONKAN, placebo
NONKAN (p<0.0100)
high dose (p<0.0100)
low dose (p<0.0100)
placebo (p<0.0100)
Interaction (p<0.0100)
Frequency (Hz)
Time (ms)
Figure 6 <2>
high dose
low dose
placebo
KAN
NONKAN
Cancel
Help
Figure 5
Cls 19 average scalp map, 5Ss
ic3/S02
ic11/S02
ic12/S02
ic17/S02
ic22/S02
ic32/S02
ic2/S05
ic8/S05
ic3/S07
ic9/S07
ic11/S07
ic18/S08
ic1/S10
ic3/S10

EEGLAB articles

Delorme, A., Makeig, S. (2004) EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods*, 134(1), 9-21.

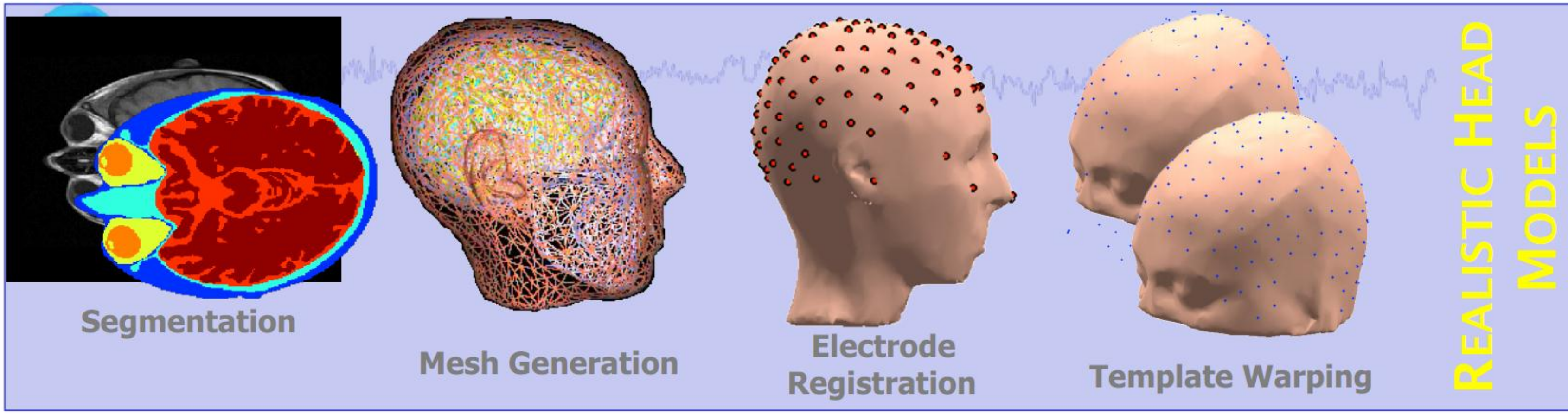
Makeig, S., Debener, S., Onton, J., Delorme, A. (2004) Mining event related dynamics. *Trends in cognitive Neuroscience*, 8(5), 204-210.

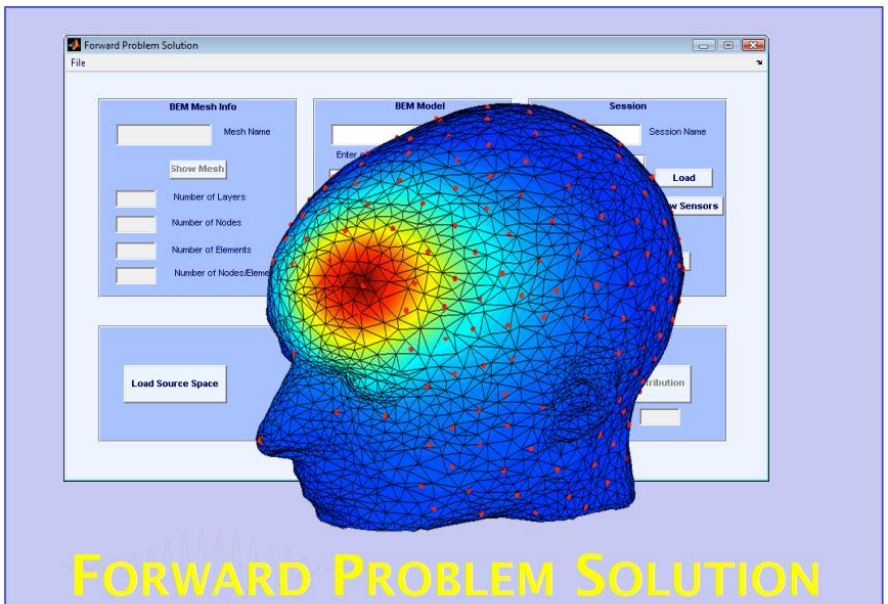
Delorme, A., Kothe, C., Bigdely, N., Vankov, A., Oostenveld, R., Makeig, S. Matlab Tools for BCI Research? In "human-computer interaction and brain-computer interfaces". Editors : Tan, D. and Nijholt, A. To appear in 2010. Springer Publishing.

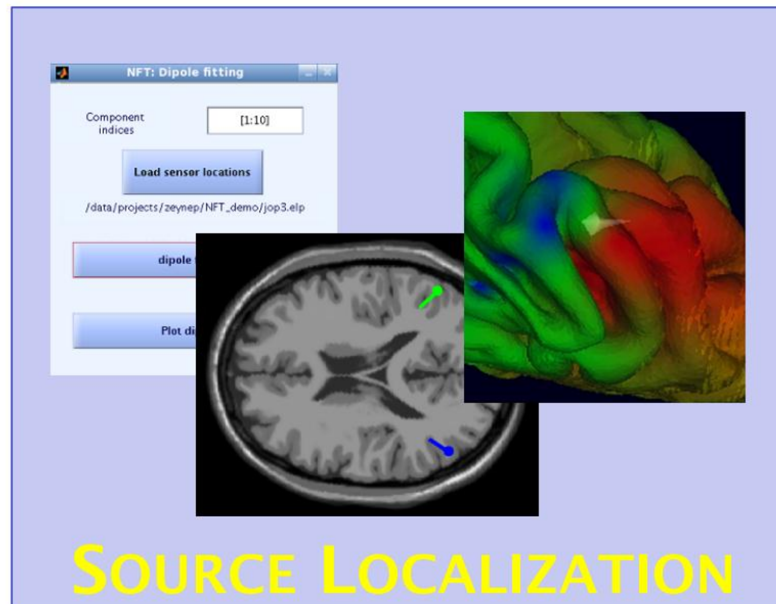
Delorme, A., Mullen, T., Kothe, C., Bigdely-Shamlo, N., Akalin, Z., Vankov, A., Makeig, S. EEGLAB, MPT, NetSIFT, NFT, BCILAB, and ERICA: New tools for advanced EEG/MEG processing. Computational Intelligence, accepted.

Delorme, A., Makeig, S. Open Source Programming for Interpreted Language: Graphic Interface and Macro Bridging Interface. IEEE International Conference on Signal Image Technology and Internet Based Systems. In press.

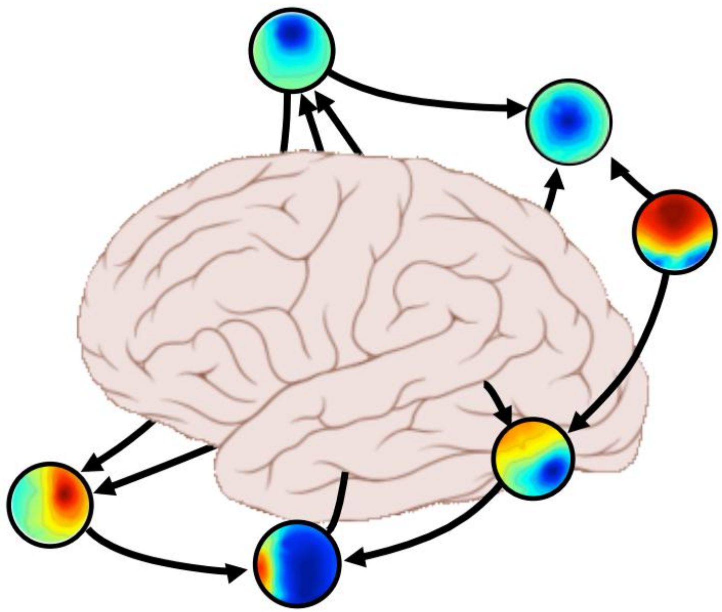
NFT: Neuroelectromagnetic Forward Head Modeling Toolbox







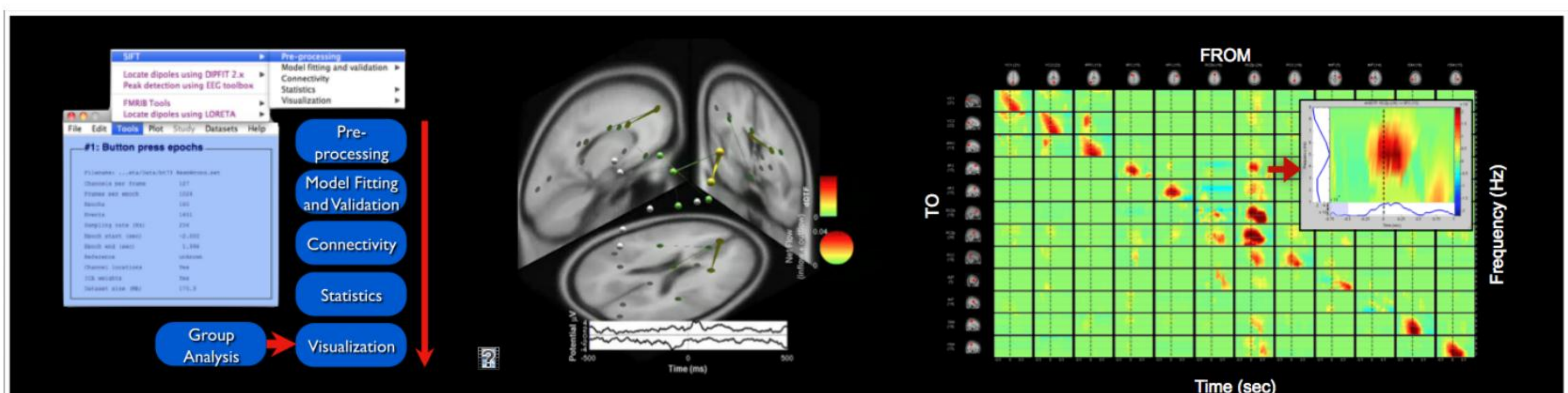
http://sccn.ucsd.edu/nft



SIFT

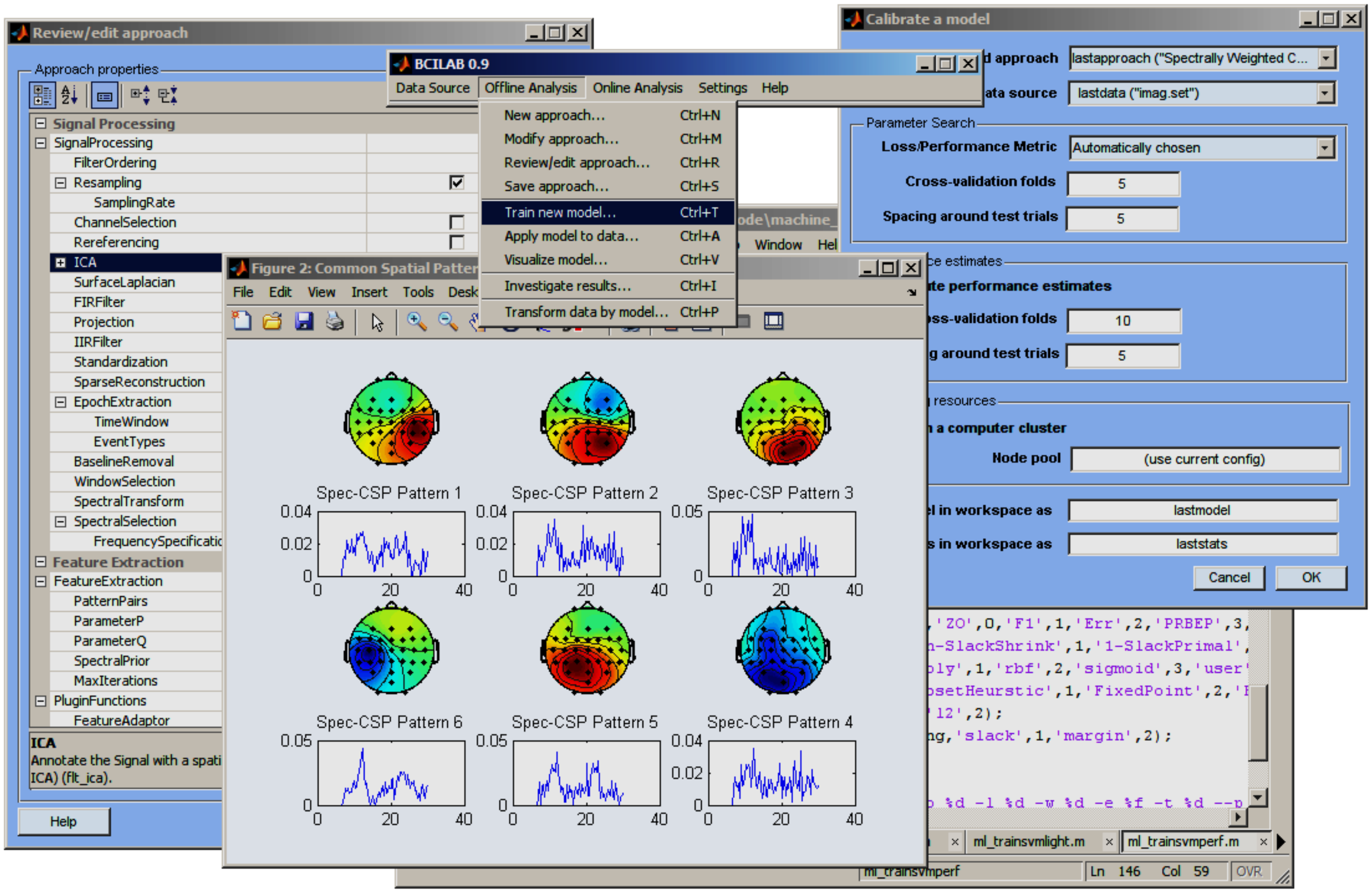
Source Information Flow Toolbox

"It makes you cool"



BCILAB Overview

What is BCILAB?



[http://sccn.ucsd.edu/**wiki/BCILAB**](http://sccn.ucsd.edu/wiki/BCILAB)

Idea & Purpose

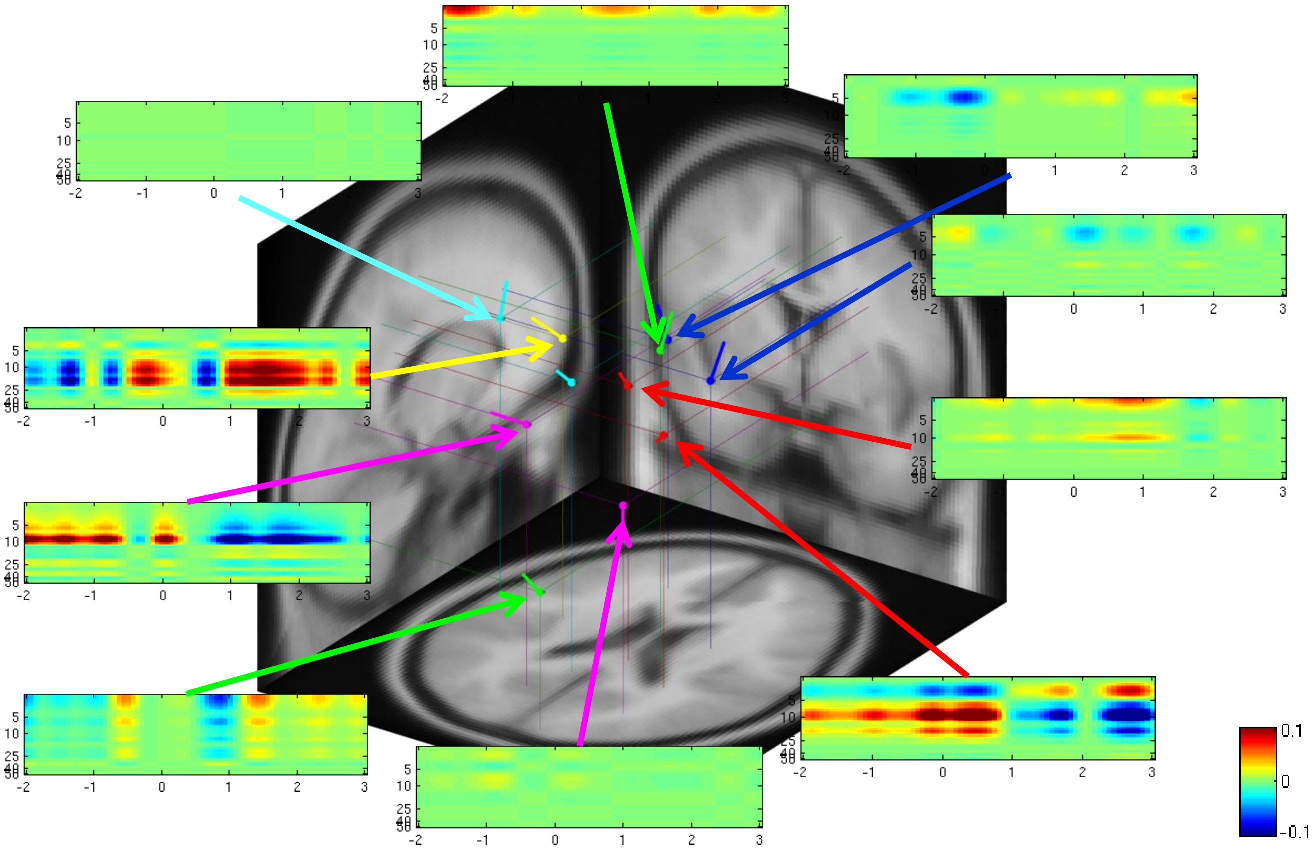
- Like EEGLAB, but for BCI (and/or cognitive state assessment)
  - Seeding a community
  - Strengthening links between BCI and Neuroscience
- SCCN's in-house tool for BCI problems
  - Main focus: Advanced cognitive monitoring
  - Part of a large US research program (CaN CTA)
  - Funded by ARL (and ONR, Swartz Foundation, …)

Research Directions

- **HCI**: User-state monitoring, intelligent assistive systems
- **Neuroscience**: Brain feedback experiments
- **Clinical**: Communication and control devices for the severely disabled
- **Entertainment**: Computer game controllers

Research Directions

- **Neuroscience**: also, *decoding models* of brain dynamics (exploratory research)



BCILAB's Niche

- State of the art
- Largest collection of machine learning & signal processing components in any open-source BCI package
  - Many standard components (CSP, LDA, SVM, …)
  - Many modern components (SBL, SSA, AMICA, HKL, DPGMM, LR-DAL, …)
  - Some novel components (OSR, RSSD, SSB, …)
- Next-generation framework
  - Fully probabilistic
  - Model inference from data corpora*
  - Anatomical priors, other neuroscience-aware features
  - Processing of parallel streams

(*: not yet in the current release)

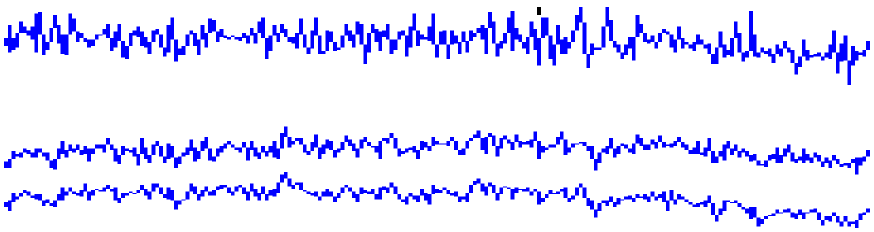
(Intangible) Aims

- Low entry barrier
  - *Developers*: Simple plugin framework, low overhead
  - *Experimenters*: User-friendliness, GUI, canned approaches
- Low usage friction
  - Flexible, unobstructive
  - Simple things easy to achieve, complex things possible
- Efficiency
  - No redundant computations (caching, …)
  - Parallel computation
  - Capable scripting (batch analysis, parameter search, …)
  - Automation

Theory, Terminology and BCILAB Equivalents

Signals

- We measure one or more (multi-channel, fixed-rate) *signals* of a person
  - EEG, ECoG, MEG, …
  - EMG, EOG, Gaze, MoCap, …

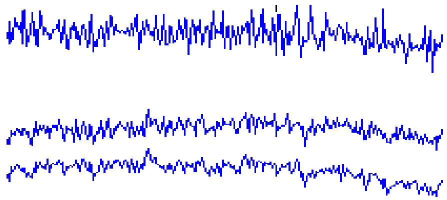


EEG

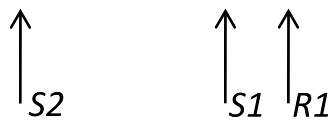
Signal Representations

Signal

.data



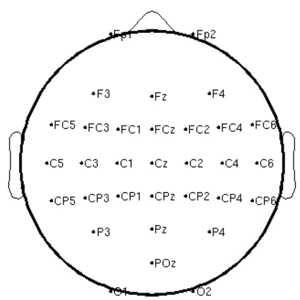
.event



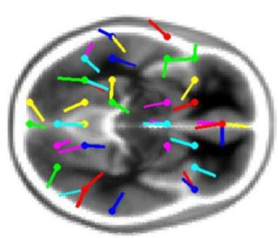
.srate *200Hz*

.xmin *0.0s*

.chanlocs



.dipfit



…

Signal Bundle

.streams

- Signal 1
- Signal 2
- ⋮
- Signal n

…

Dataset Collection

- Bundle 1
- Bundle 2
- ⋮
- Bundle n

Data Sources

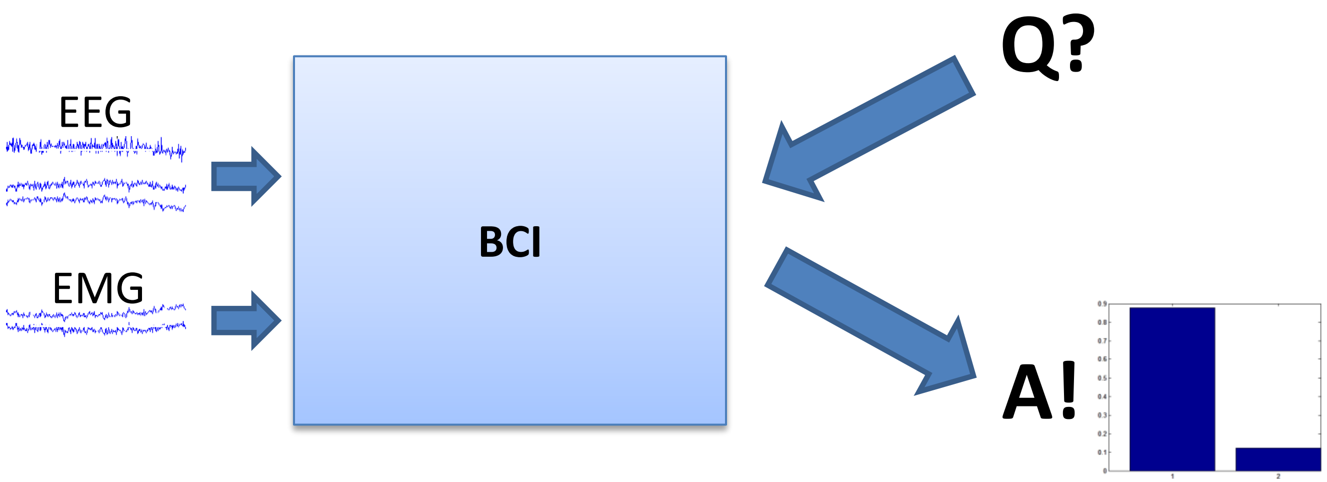
- Signals in BCILAB originate from *device plugins*
- Currently: BioSemi, TCP, OSC, BCI2000, DataRiver, playback, etc.
- More planned (e.g., BrainProducts, ANT, g.Tec, Emotiv, …)
- Quite easy to extend (typically a few 10s of LoC)

Basic Framework

- The goal is to build an oracle that consumes these signals and can answer (pre-defined) queries about cognitive state of the person
- BCIs fit into that framework

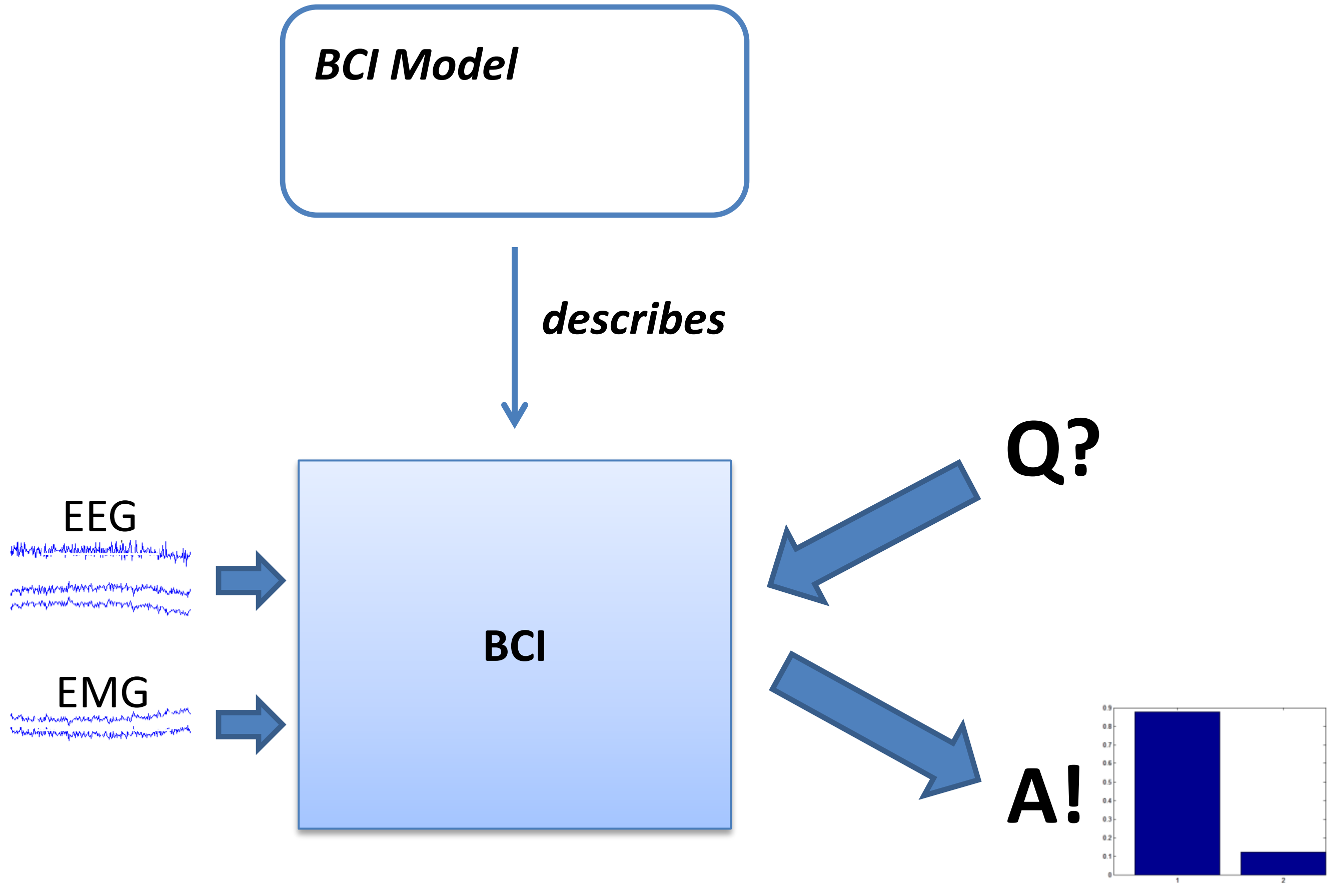
Basic Framework

- In BCILAB, outputs may be point estimates (scalar, vector) or probability distributions (discrete / continuous)
  - usually discrete prob. distributions



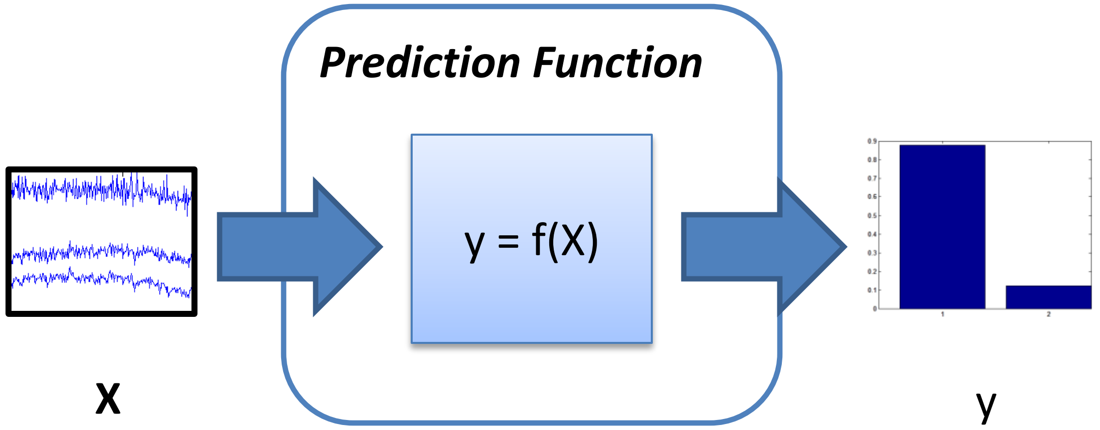
Basic Framework

- A BCI is specified/described by a “*BCI model*”



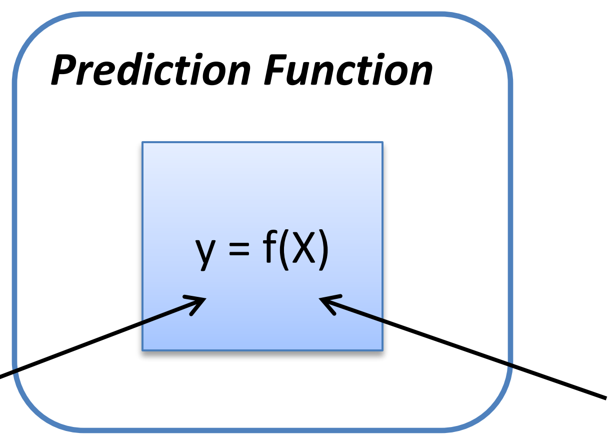
Model Structure

- All BCI models contain a mapping f**(X)** that maps a limited-length signal segment $\mathbf{X} \in \mathbb{R}^{C \times T}$ onto the output y
  - In BCILAB called the model's "*prediction function*"
  - May also accept segments from multiple signals



Model Structure

- Note: the prediction function involves a functional form and possibly some fixed parameters
  - Functional form reflects a relationship between measurements and cognitive state (inverse for some assumed generative mechanism)



Functional form, e.g.,
$y = w\,\mathrm{var}[\boldsymbol{WX}] + b$

Parameters, e.g.,
$\boldsymbol{W}, w, b$

Model Structure

- May also apply *signal processing methods* (here called *filters*) to the input signals (e.g., for computational efficiency or to leverage tools)
  - receive a signal and produce a transformed signal
  - online-capable and possibly adaptive / stateful
  - represented as plugins in BCILAB, more than 40 methods built in

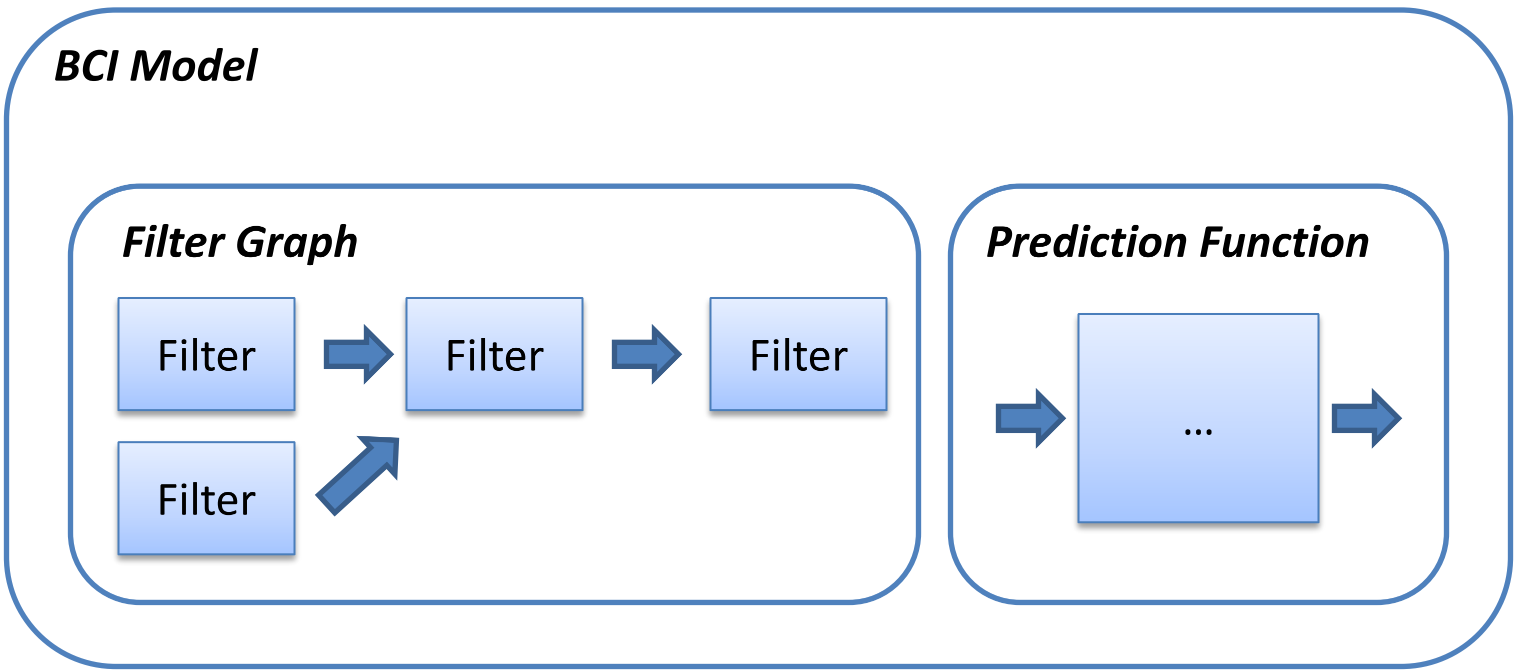
Major Filter Types

- Spatial filters (channel selection, surface Laplacian, ICA, CSP*, sparse recovery, …)
- Spectral filters (FIR, IIR, FFT)
- Epoch-based filters (windowing, Wavelet transform, Fourier transform, STFT, …)
- Miscellaneous (resampling, dipole fitting, …)

*: not implemented as a filter in BCILAB

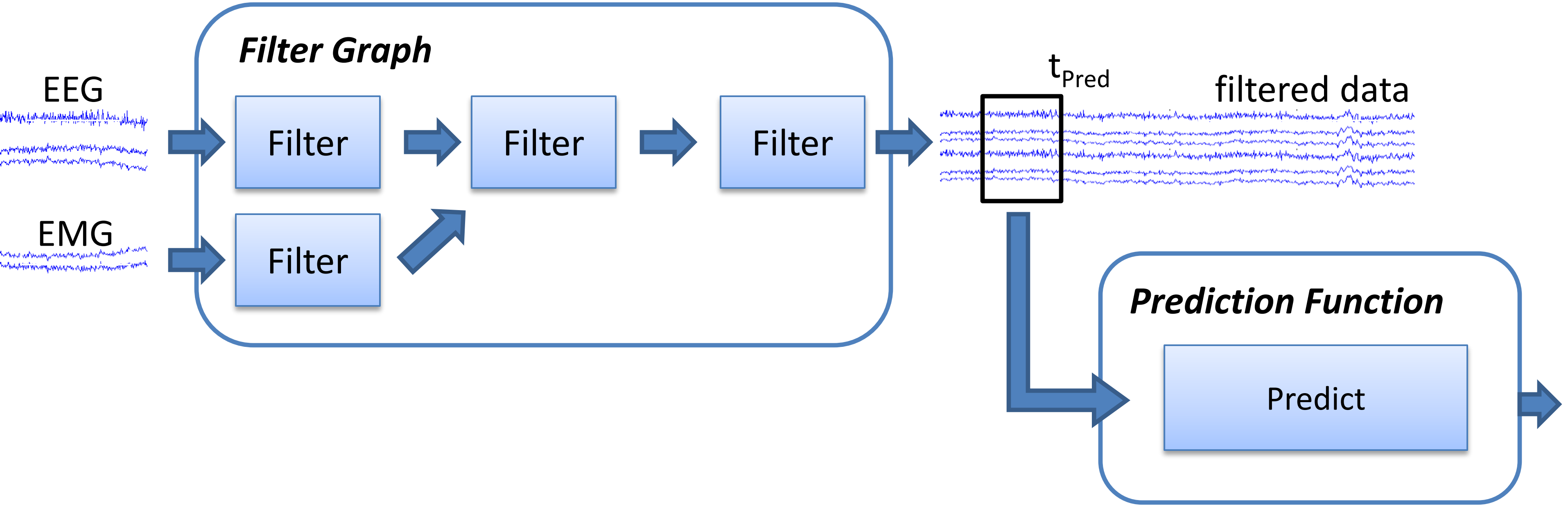
Model Structure

- Putting all together, a BCI model in BCILAB contains a *filter graph* and a prediction function
- Provides enough flexibility for most BCI designs



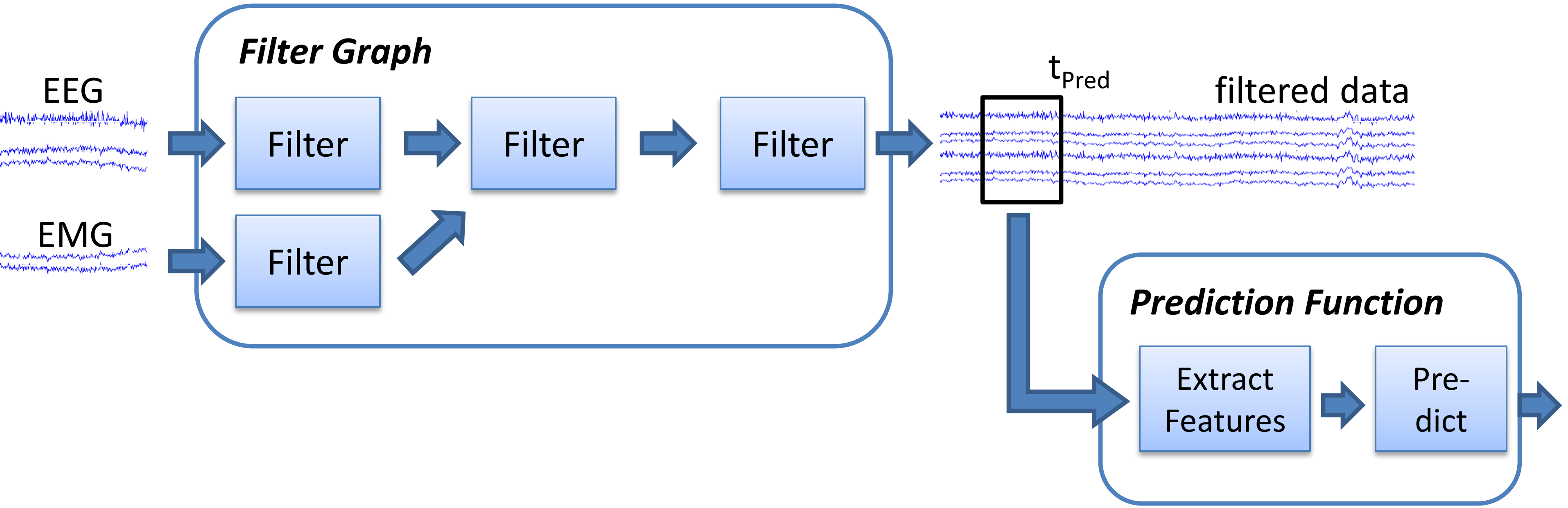
Online Data Flow

- In BCILAB, the filter graph receives all input samples, but the prediction function may be called on demand



Online Data Flow

- In BCILAB, the filter graph receives all input samples, but the prediction function may be called on demand
  - In most current BCIs, the prediction function consists of a dedicated “feature extraction” step and “prediction” step

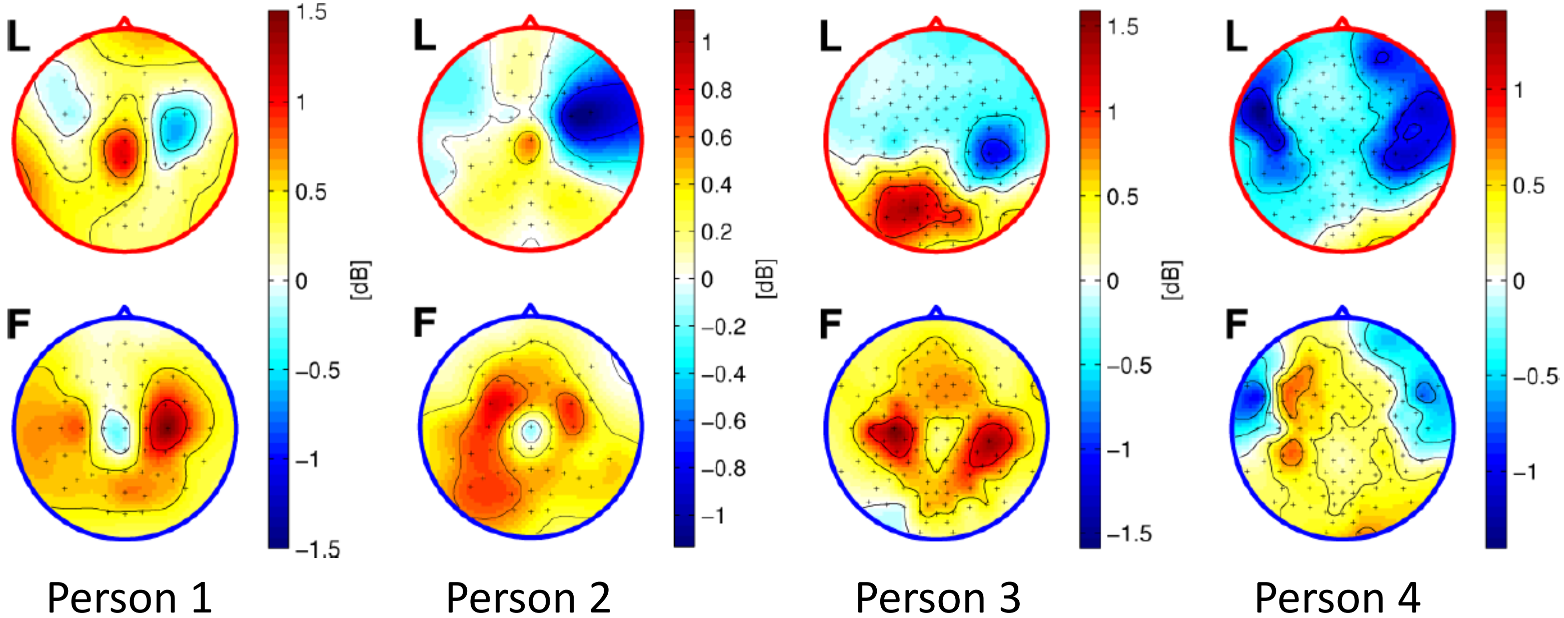


One Problem

- for most BCI questions and implementations, the parameters leading to best accuracy are *a priori* unknown!
  - Depend on hard-to-measure factors (e.g., brain functional map)
  - Depend on expensive-to-measure factors (e.g., brain folding)
  - Depend on highly variable factors (e.g., sensor placement, subject state)
  - Different for every person, task, montage, etc.

One Problem

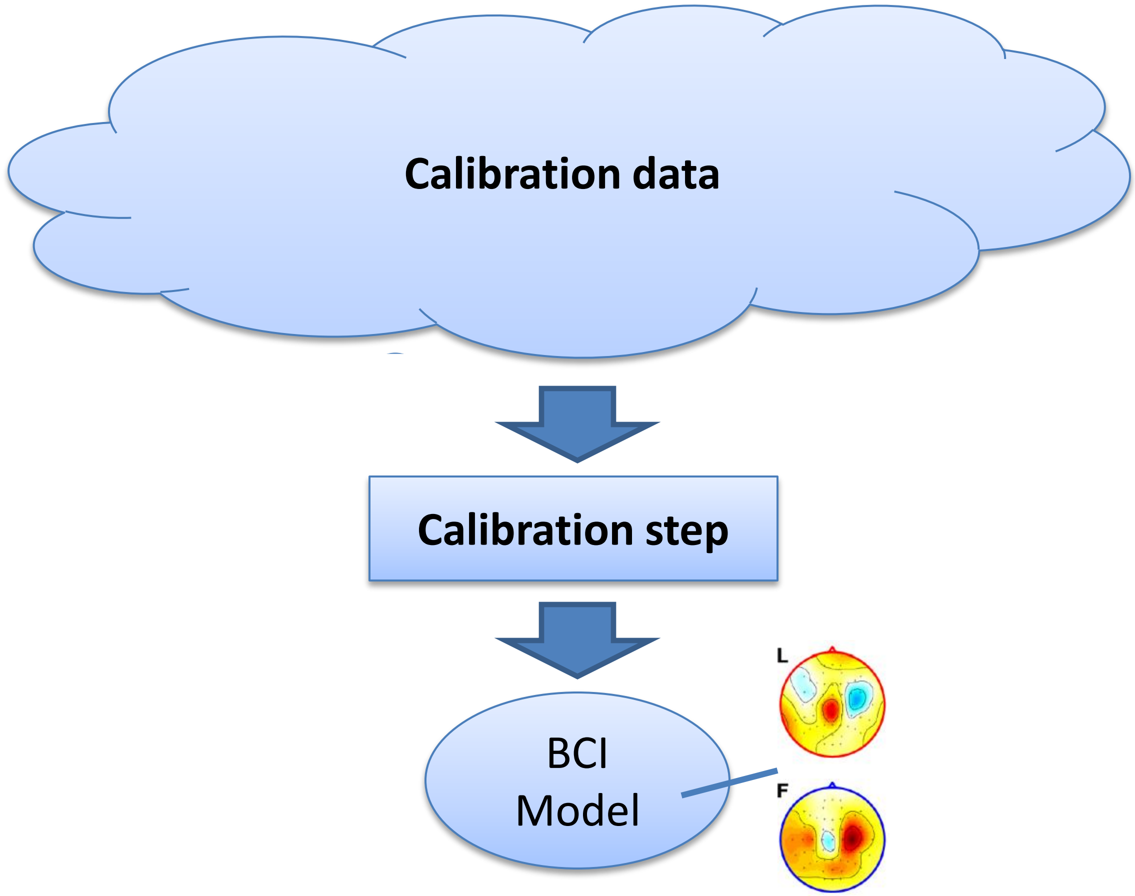
- Example per-channel parameters across four subjects:



(image: Blankertz et al. 2007)

Model Calibration

- Need *calibration / training data* to estimate parameters from and a separate *calibration step*



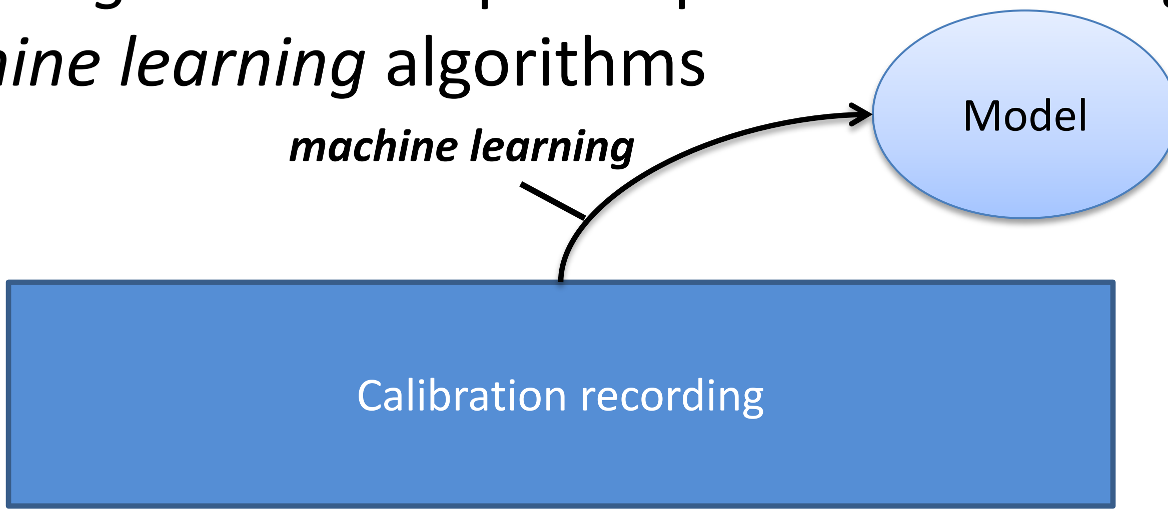
Model Calibration

- In theory many possibilities (e.g. MR scanner data + Beamforming)

Swartz Center for Computational Neuroscience

Model Calibration

- In theory many possibilities (e.g. MR scanner data + Beamforming)
- **Most successful way (so far)**: utilize data where both the BCI input (e.g. EEG) and desired output (cognitive state) is known – in BCILAB called "*calibration recording*" – and adapt BCI parameters using *machine learning* algorithms

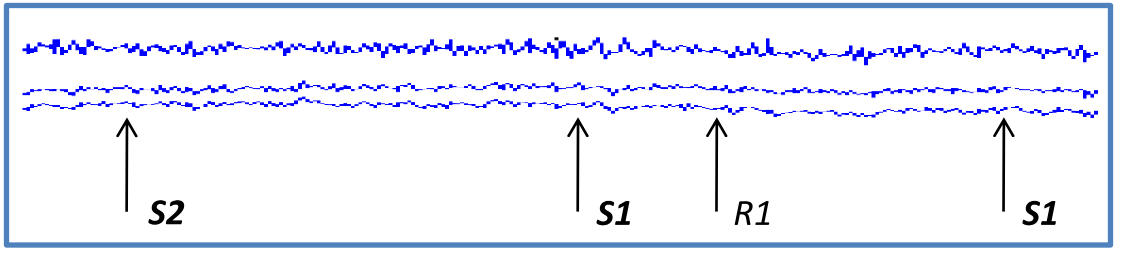


Calibration Recording

- Standard psychological experiment
  - continuous EEG (or other)
  - multiple trials/blocks (capturing variation)
  - randomized (eliminating confounds)

Calibration Recording

- Standard psychological experiment
  - continuous EEG (or other)
  - multiple trials/blocks (capturing variation)
  - randomized (eliminating confounds)
  - event markers to encode cognitive state conditions of interest, e.g., stimuli/responses (called "*target markers*" in BCILAB)

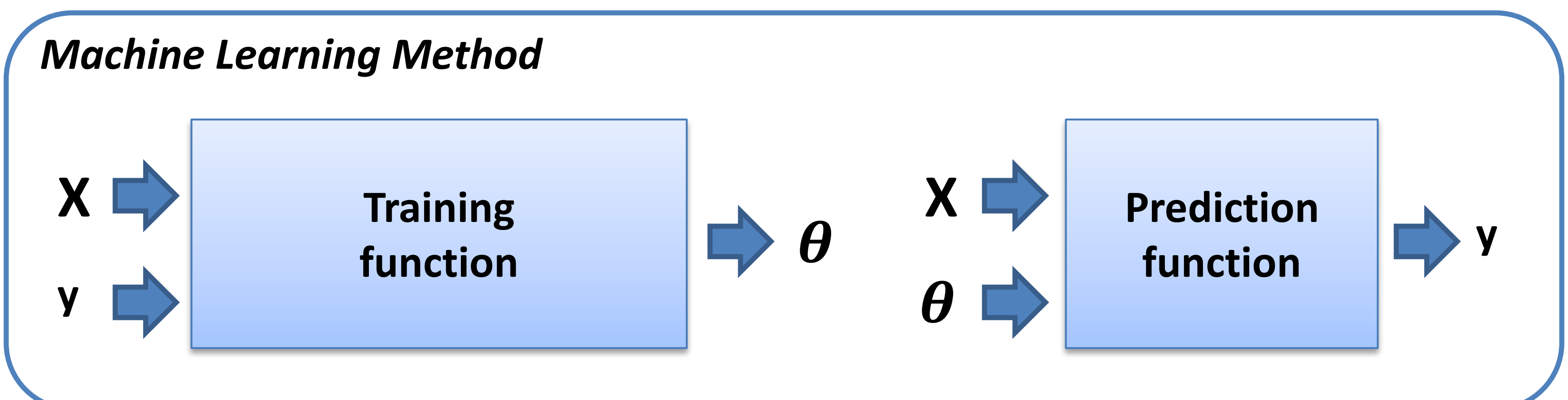


Machine Learning

- Large field with 100s of algorithms
- Most methods conform to a common interface of a *training function* and a *prediction function*
  - ***training function*** accepts a matrix of feature vectors (samples) $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ and target values $\boldsymbol{y} \in \mathbb{R}^{N \times D}$ with F the # of features per sample, N the # of samples, D the dimensionality of the target space (usually D=1); the output is a *model* with parameters $\boldsymbol{\theta}$
  - ***prediction function*** accepts a matrix of feature vectors $\boldsymbol{X}$, model parameters $\boldsymbol{\theta}$ and produces estimates of the corresponding target values $\boldsymbol{y}$

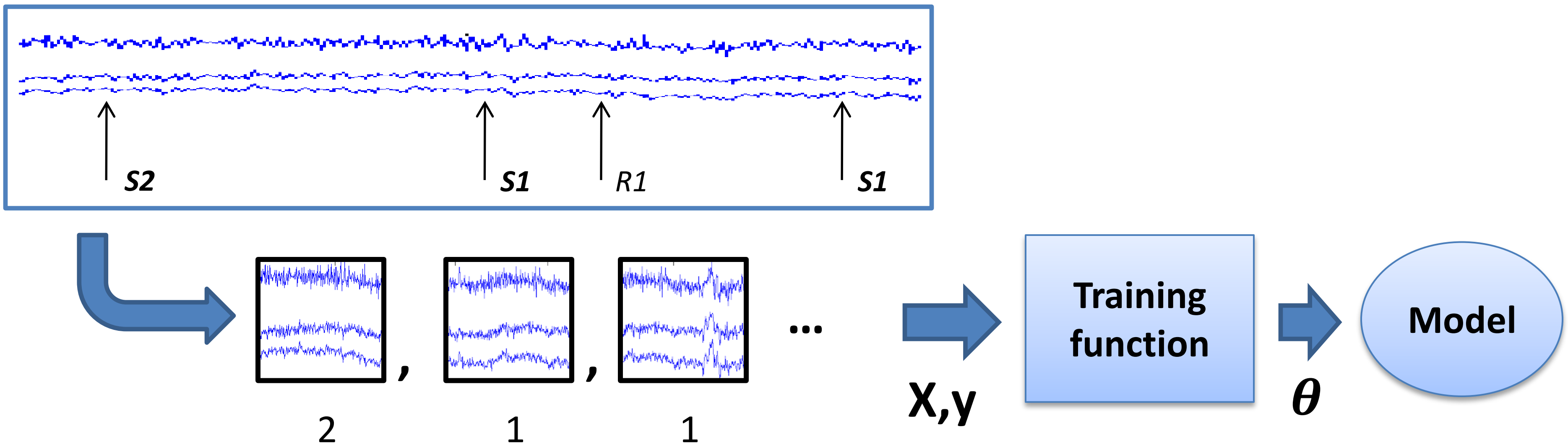
Machine Learning

- BCILAB has a plugin framework for machine learning (with >60 built-in algorithms)
- Supports some additional formats for **X**, e.g., matrix-valued features and **y** (e.g., common distributions)
- Training function also accepts additional parameters



Model Calibration, cont.

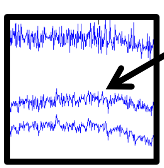
- In BCILAB, typically one trial segment (sample) is extracted for every target marker in the calibration recording



Feature Extraction

- **Problem:** Standard machine learning methods often do not work very well when applied to raw signal segments **X** of the calibration recording
  - too high-dimensional (too many parameters to fit)
  - too complex structure to be captured (too much modeling freedom)
  - (but note: different story for custom methods)

1000s of degrees of freedom

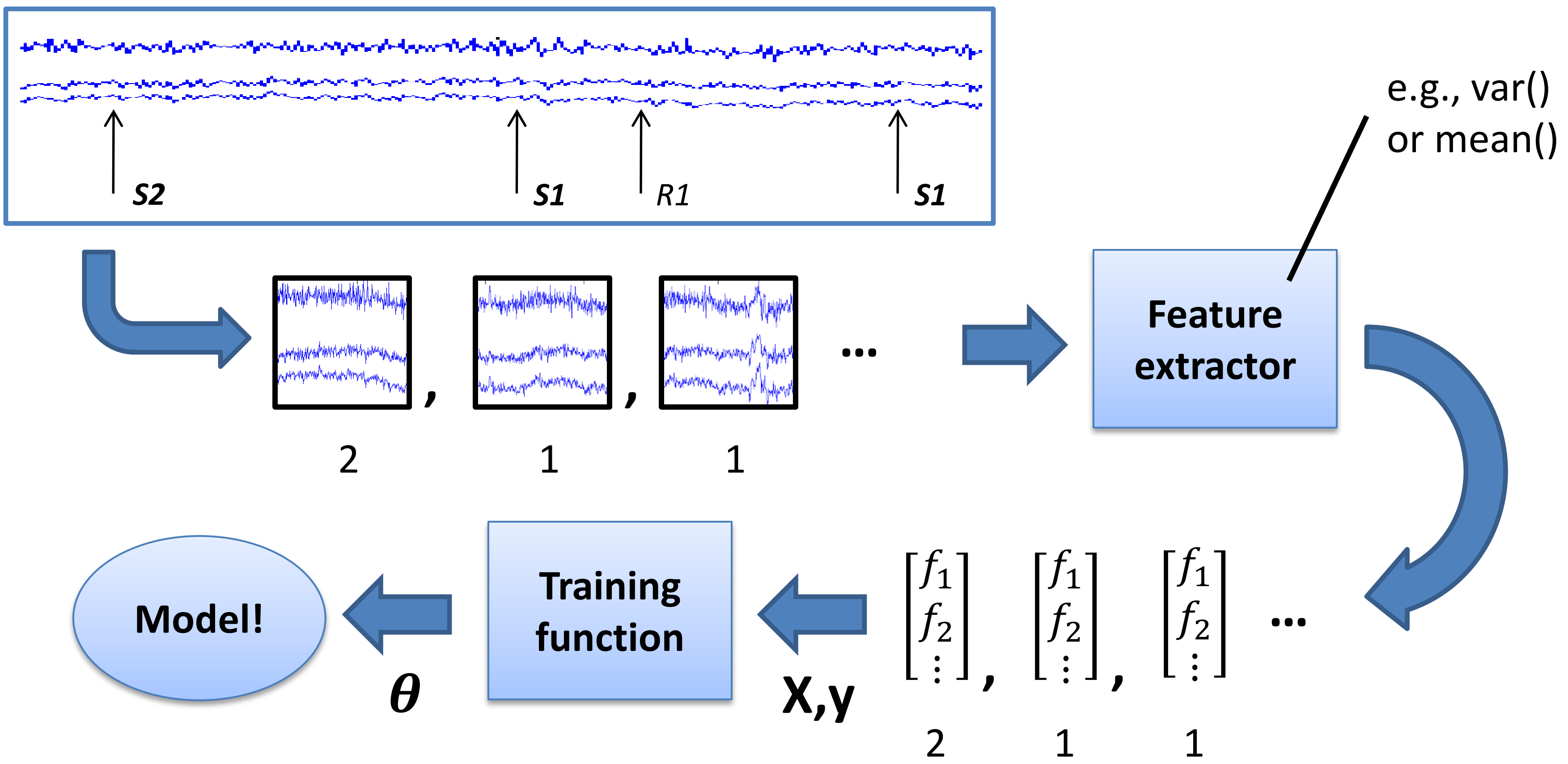


Feature Extraction

- **Solution**: Introduce additional mapping (called *"feature extraction")* from raw signal segments onto feature vectors
  - output is often of lower dimensionality
  - hopefully better distributed in the feature space (easy to handle for machine learning)

Model Calibration, cont.

- With feature extraction, the analysis process is as follows:

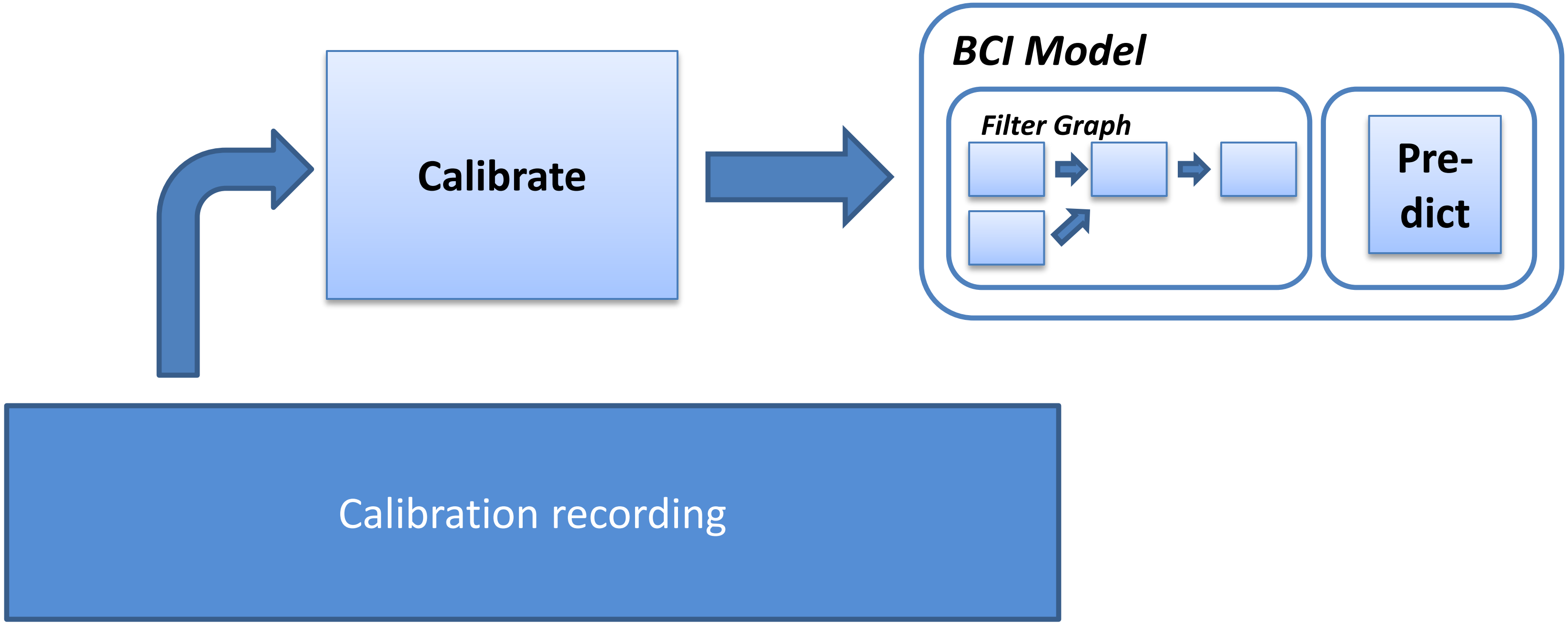


BCI Paradigms

- BCI paradigms are the coarsest plugin type in BCILAB and tie all parts of a BCI approach together
  - ***calibrate function***: accepts a calibration recording (with markers), additional parameters, and produces a *BCI model*; may utilize machine learning, signal processing and/or other methods
  - ***prediction function***: serves as the prediction function for the resulting BCI model
  - optionally additional code, e.g., for feature extraction and feature adaptation (feature learning and/or feature selection)

BCI Paradigms

- BCI paradigms are the coarsest plugin type in BCILAB and tie all parts of a BCI approach together
- Note: Multiple approaches can be realized with a single paradigm (using different parameters)



BCI Paradigm Plugins

- For event-related potentials
  - Time-window averages (explained next)
  - Wavelet features
  - First-order DAL
- For oscillatory processes
  - Common Spatial Patterns, Regularized CSPs, Spectrally weighted CSP, …
  - Channel-based approaches
  - ICA-based approaches (OSR, RSSD, …)
  - Second-order DAL
  - …

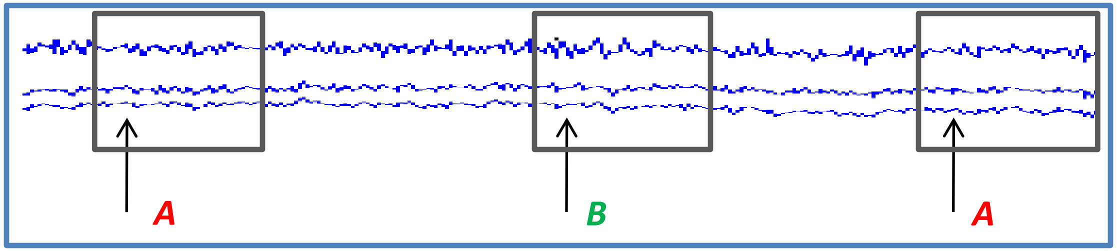
Concrete Approach for ERPs

Example Approach for ERPs

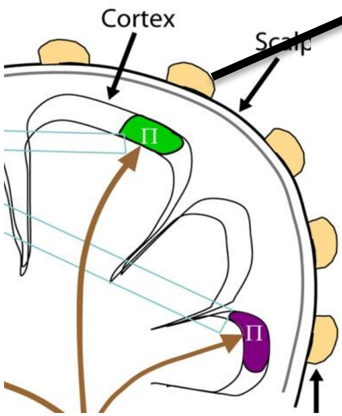
- Suppose a calibrate function with steps:
  1. Apply an IIR band-pass filter to raw data (0.5 - 15 Hz)
  2. Extract 0.8 s segments around stimuli in recording
  3. Extract features and run machine learning

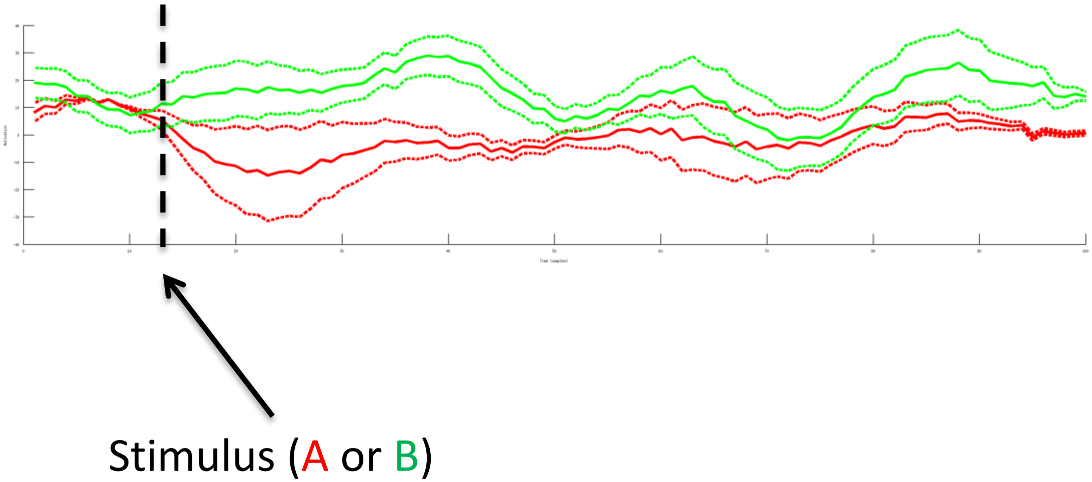
Example Approach for ERPs

- Suppose a calibrate function with steps:
  1. Apply an IIR band-pass filter to raw data (0.5 - 15 Hz)
  2. Extract 0.8 s segments around stimuli in recording
  3. Extract features and run machine learning
- Applied to a recording with 100 stimuli of type A and 100 stimuli of type B

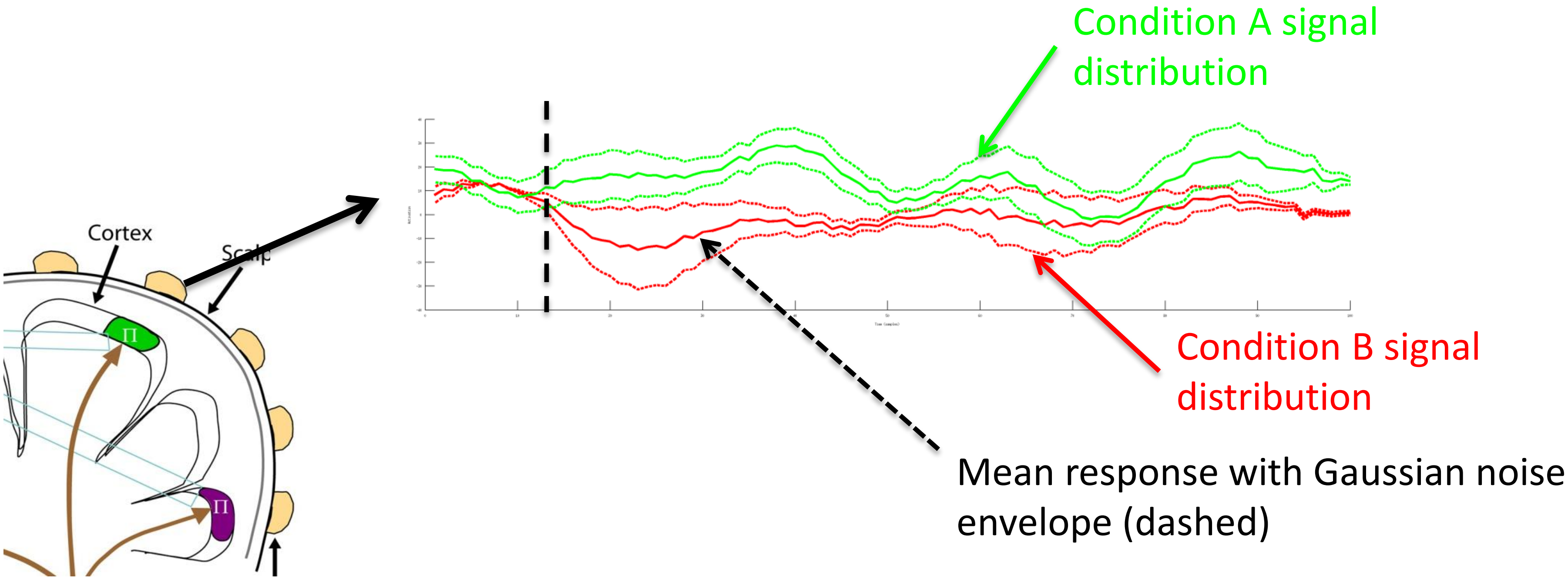


Resulting filtered segments

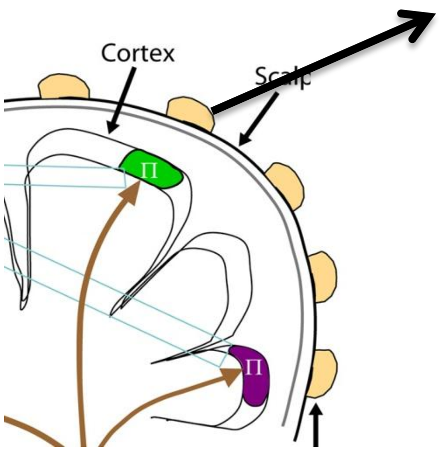


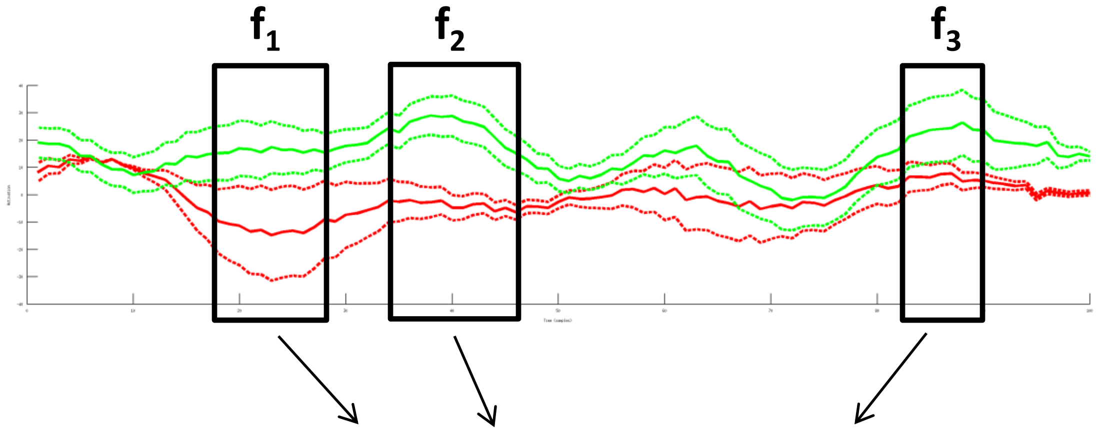


Resulting filtered segments

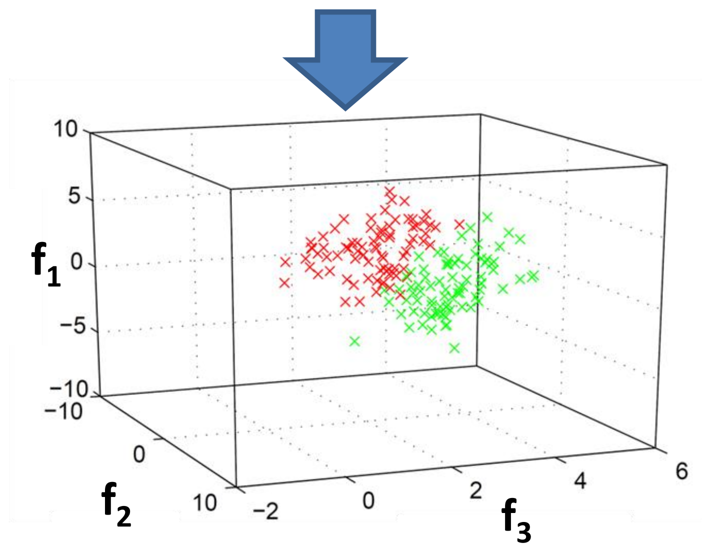


Extracting Features



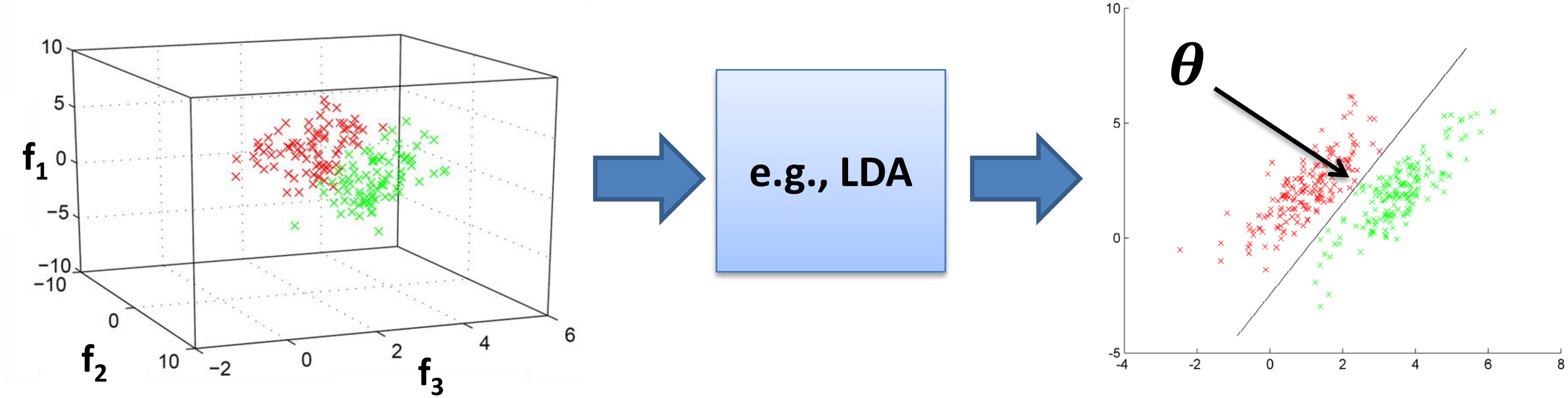


For each trial segment, calculate signal mean in three time windows (gives 3-dim feature vectors per trial)



Using Machine Learning

- The feature vectors are passed on to a machine learning function (e.g., Linear Discriminant Analysis)

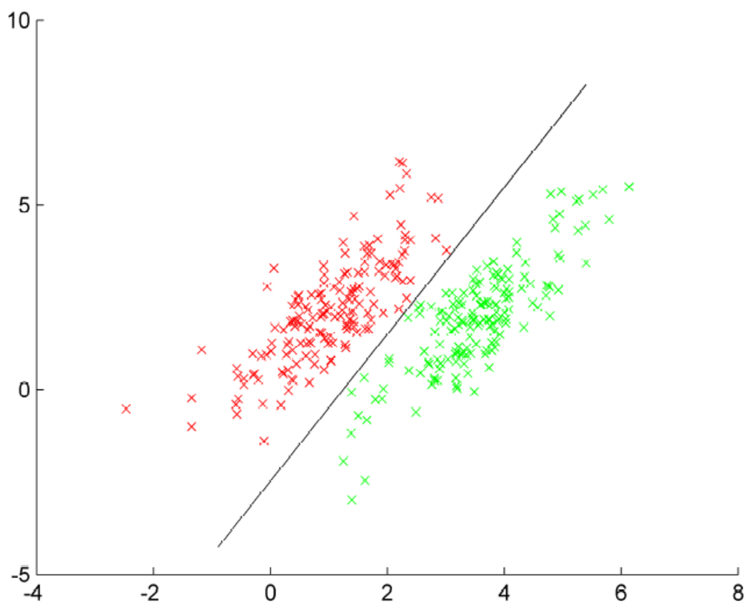


LDA In a Nutshell

- Given trial segments $\boldsymbol{x}_k$ (in vector form) in $\mathcal{C}_1$ and $\mathcal{C}_2$,

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|}\sum_{k\in\mathcal{C}_i} \boldsymbol{x}_k , \qquad \Sigma_i = \sum_{k\in\mathcal{C}_i} (\boldsymbol{x}_k - \boldsymbol{\mu}_i)(\boldsymbol{x}_k - \boldsymbol{\mu}_i)^\intercal$$

$$\boldsymbol{\theta} = (\Sigma_1 + \Sigma_2)^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1), \qquad \mathrm{b} = \boldsymbol{\theta}^\intercal(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)/2$$



LDA In a Nutshell

- Given trial segments $\boldsymbol{x}_k$ (in vector form) in $\mathcal{C}_1$ and $\mathcal{C}_2$,

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|}\sum_{k\in\mathcal{C}_i}\boldsymbol{x}_k\,, \qquad \Sigma_i = \sum_{k\in\mathcal{C}_i}(\boldsymbol{x}_k - \boldsymbol{\mu}_i)(\boldsymbol{x}_k - \boldsymbol{\mu}_i)^\intercal$$

$$\boldsymbol{\theta} = (\Sigma_1 + \Sigma_2)^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1), \qquad \mathrm{b} = \boldsymbol{\theta}^\intercal(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)/2$$

- **Caveat**: θ often high-dimensional but only few trials available
- Can use a regularized estimator instead, here using *shrinkage*; instead of Σ_i, we use $\tilde{\Sigma}_i$ above:

$$\tilde{\Sigma}_i = (1-\lambda)\Sigma_i + \lambda\boldsymbol{I}$$

Machine Learning Plugins

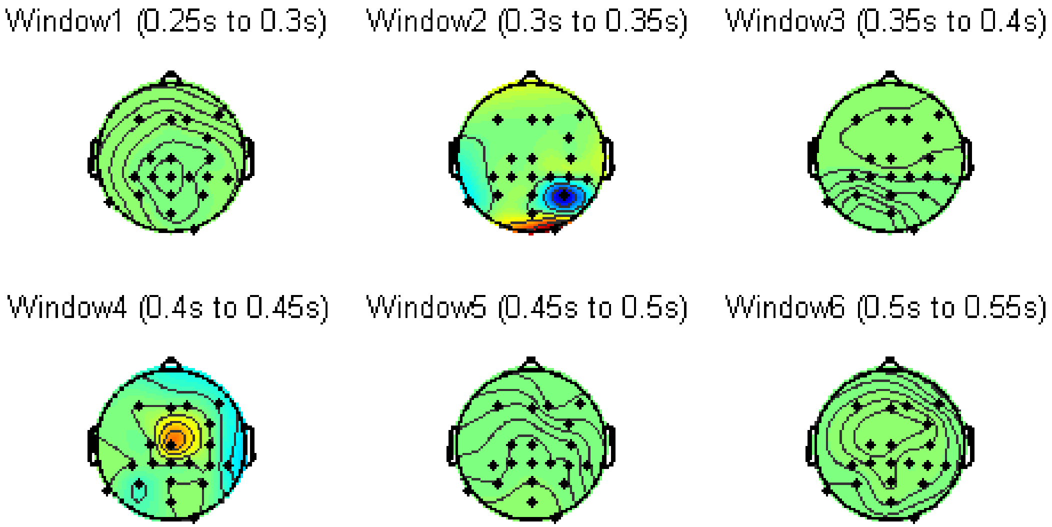
- Generative Models (LDA, RLDA, QDA, GMMs)
- Discriminative Models (SVMs, RVMs, GLMs, HKL, …)
- Custom Frameworks (convex optimization, graphical models, …)

Calibration: Summary

- Basic calibration in BCILAB typically involves:
  - Filtering the data (possibly adapting filters)
  - Extracting trial segments and features (if necessary)
  - Applying a machine learning function
  - Specifying the model structure (filter graph, prediction function, parameters)

Visualizing θ

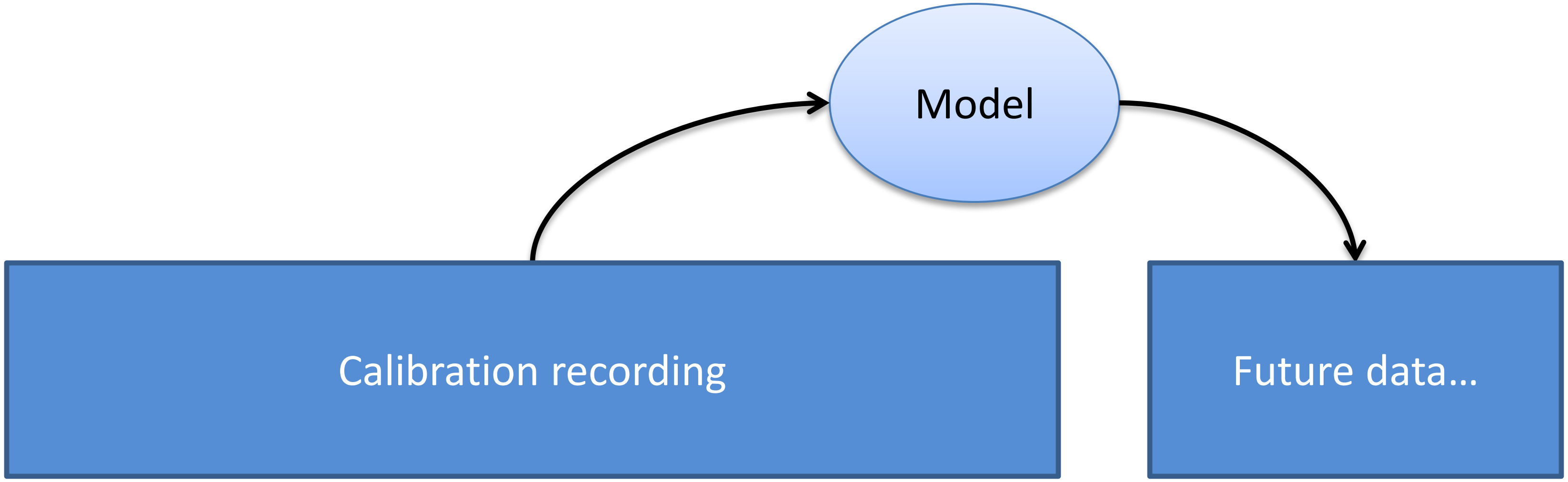
- Linear model weights can be visualized as a (color-coded) value per time window and channel
- Below: 6 windows, 21 channels, ERP task



Evaluating Model Performance

Offline Evaluation

- Given calibration data
- Estimate model parameters (spatial filters, statistics)
- Apply the model to new data (online / single-trial)
- Optionally: compare outputs with known state, compute loss statistics for the model / approach (e.g., mis-classification rate)



Offline Evaluation

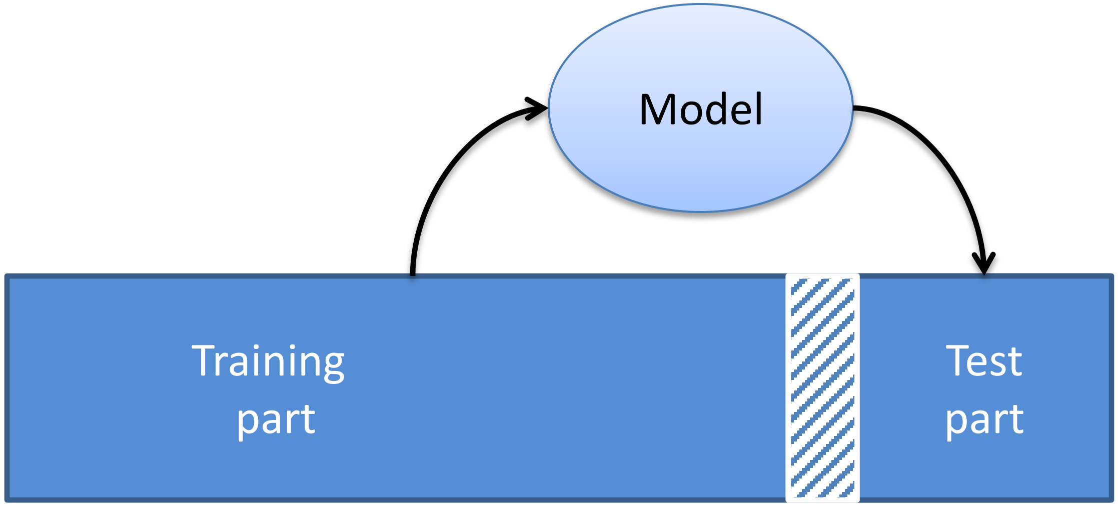
- Evaluation of computational approaches on a **single** data set?

?

Calibration recording

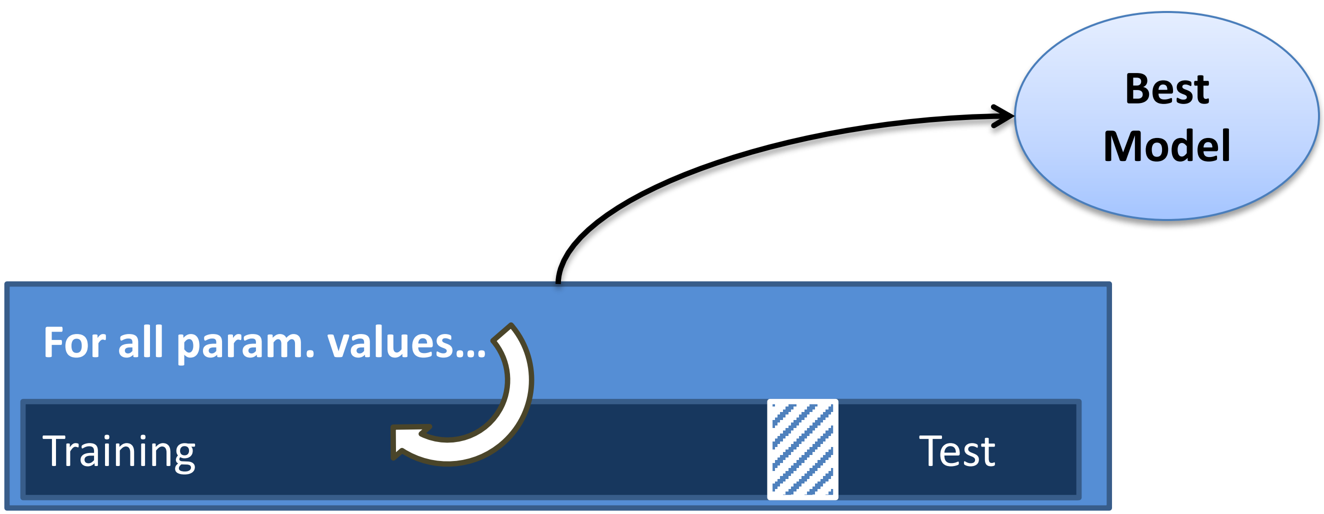
Offline Evaluation

- Evaluation of computational approaches on a single data set?
  - Can not test on the training data! (always on separate data)
  - Instead can split data set repeatedly into training/test blocks systematically, a.k.a. *cross-validation*



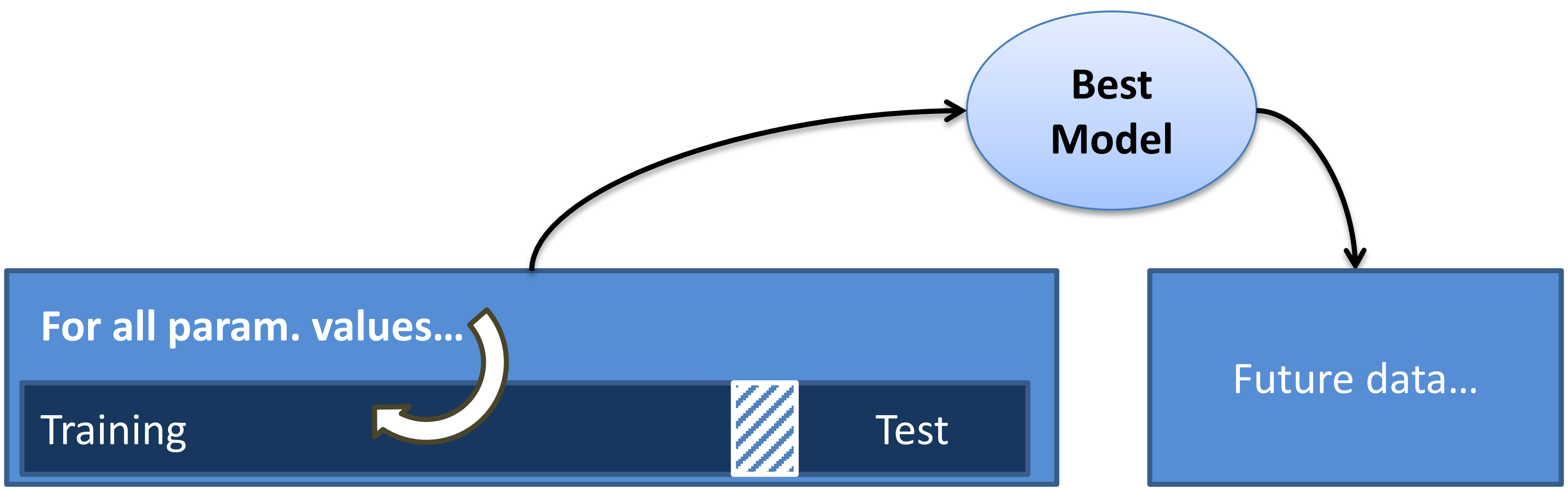
Resolving Free Parameters

- Can be done using cross-validation in a grid search (try all values of free parameters)
- **Caveat**: Resulting "optimal" numbers are *non-reportable* (cherry-picked!)



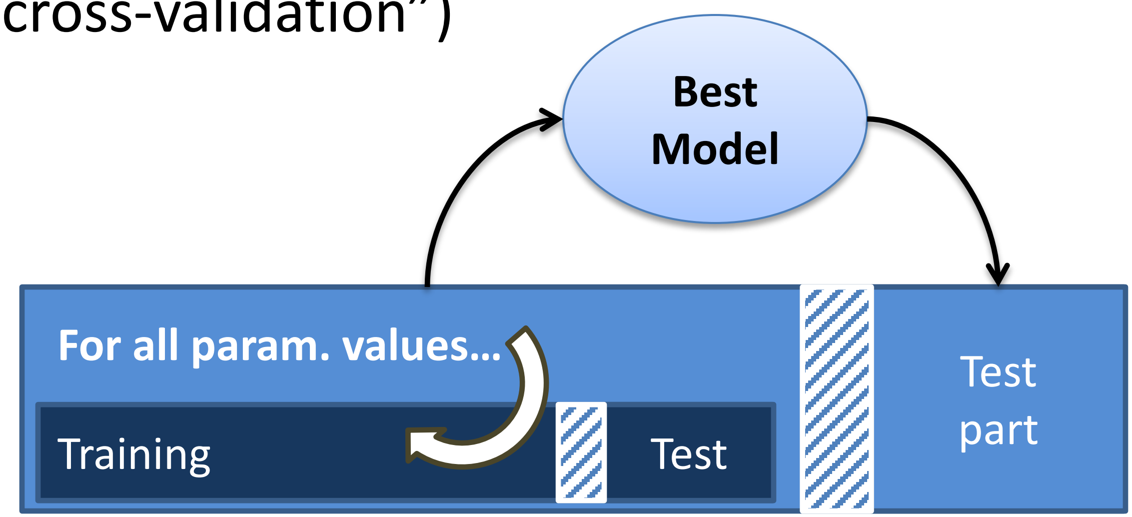
Resolving Free Parameters

- Can be done using cross-validation in a grid search (try all values of free parameters)
- **Caveat**: Resulting "optimal" numbers are *non-reportable* (cherry-picked!)
- But may test resulting best model on separate data



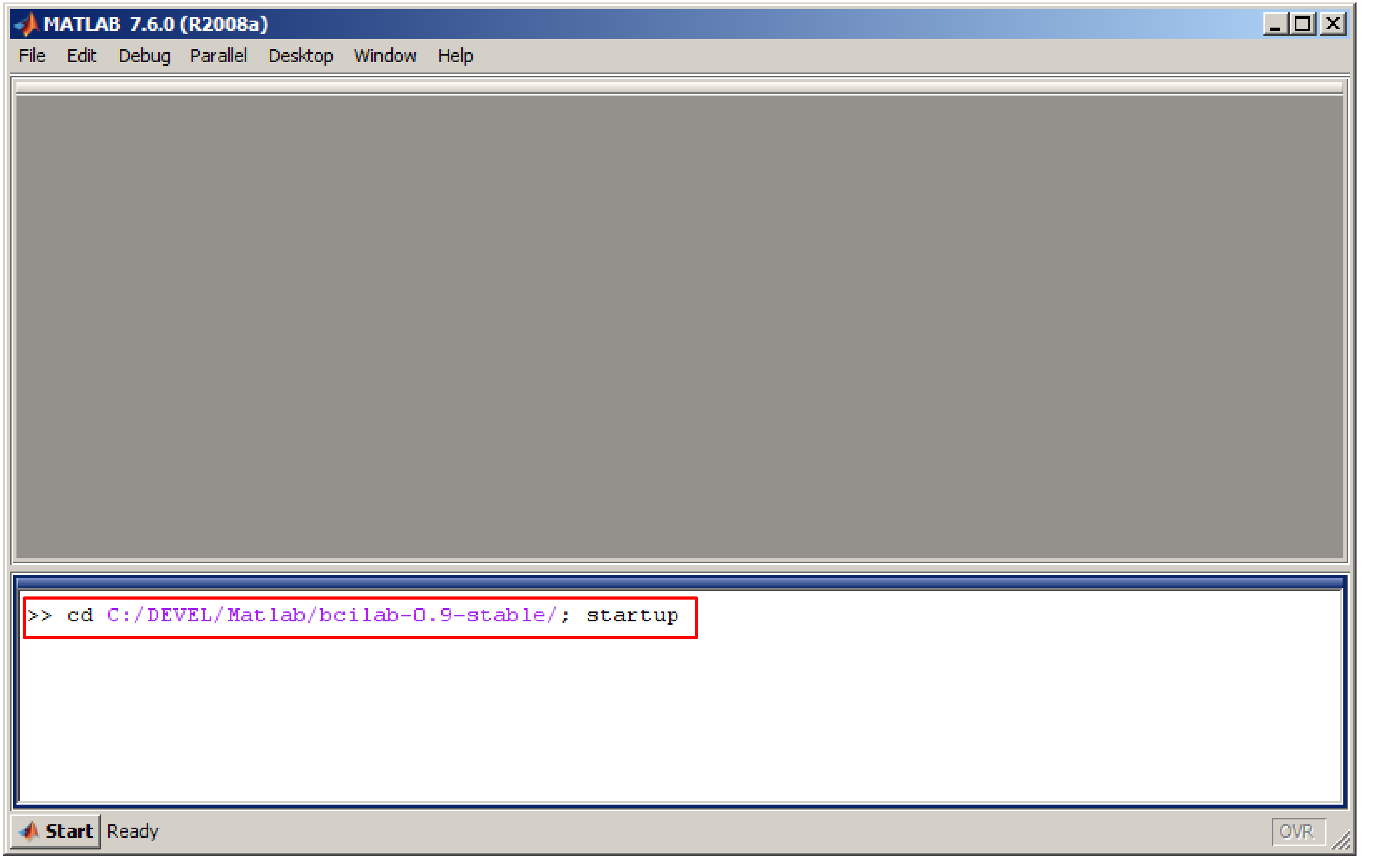
Resolving Free Parameters

- Can be done using cross-validation in a grid search (try all values of free parameters)
- **Caveat**: Resulting "optimal" numbers are *non-reportable* (cherry-picked!)
- But may test resulting best model on separate data
- **Or** run grid search *within* an outer cross-validation ("nested cross-validation")

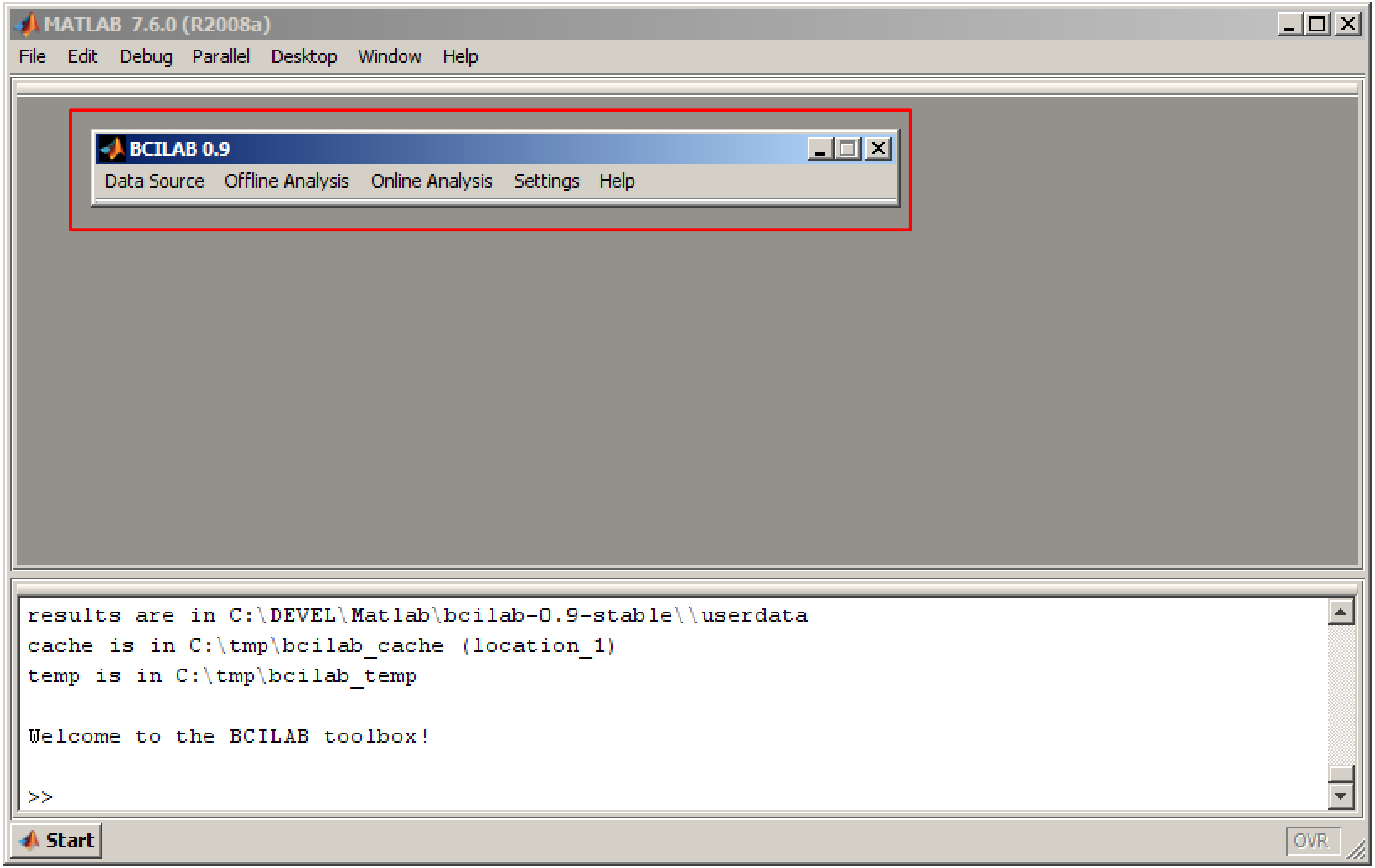


Next: Basic GUI Tour

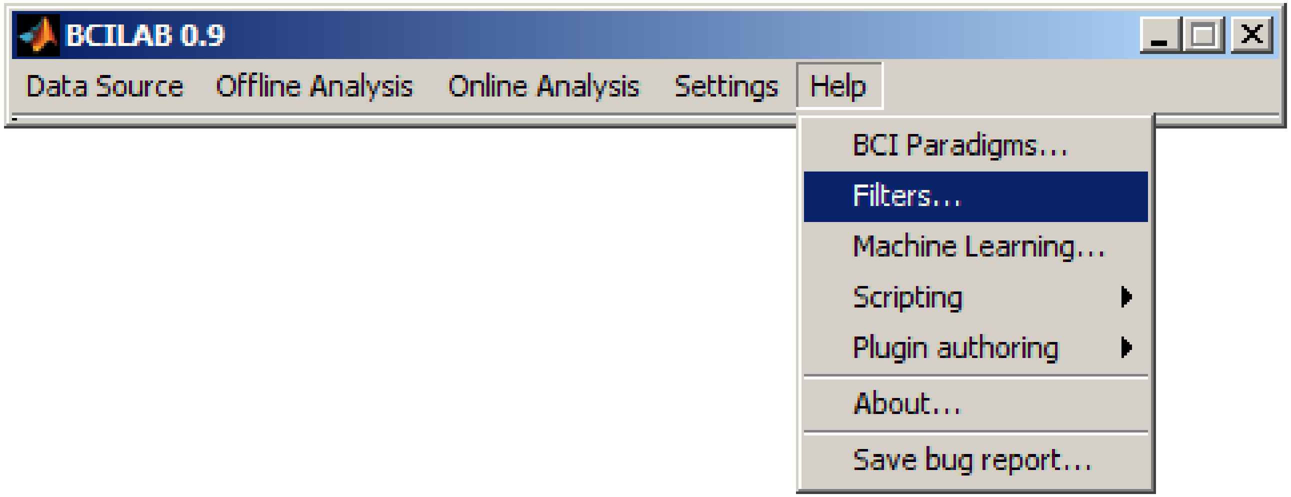
Startup


MATLAB 7.6.0 (R2008a)
File Edit Debug Parallel Desktop Window Help
>> cd C:/DEVEL/Matlab/bcilab-0.9-stable/; startup
Start Ready
OVR

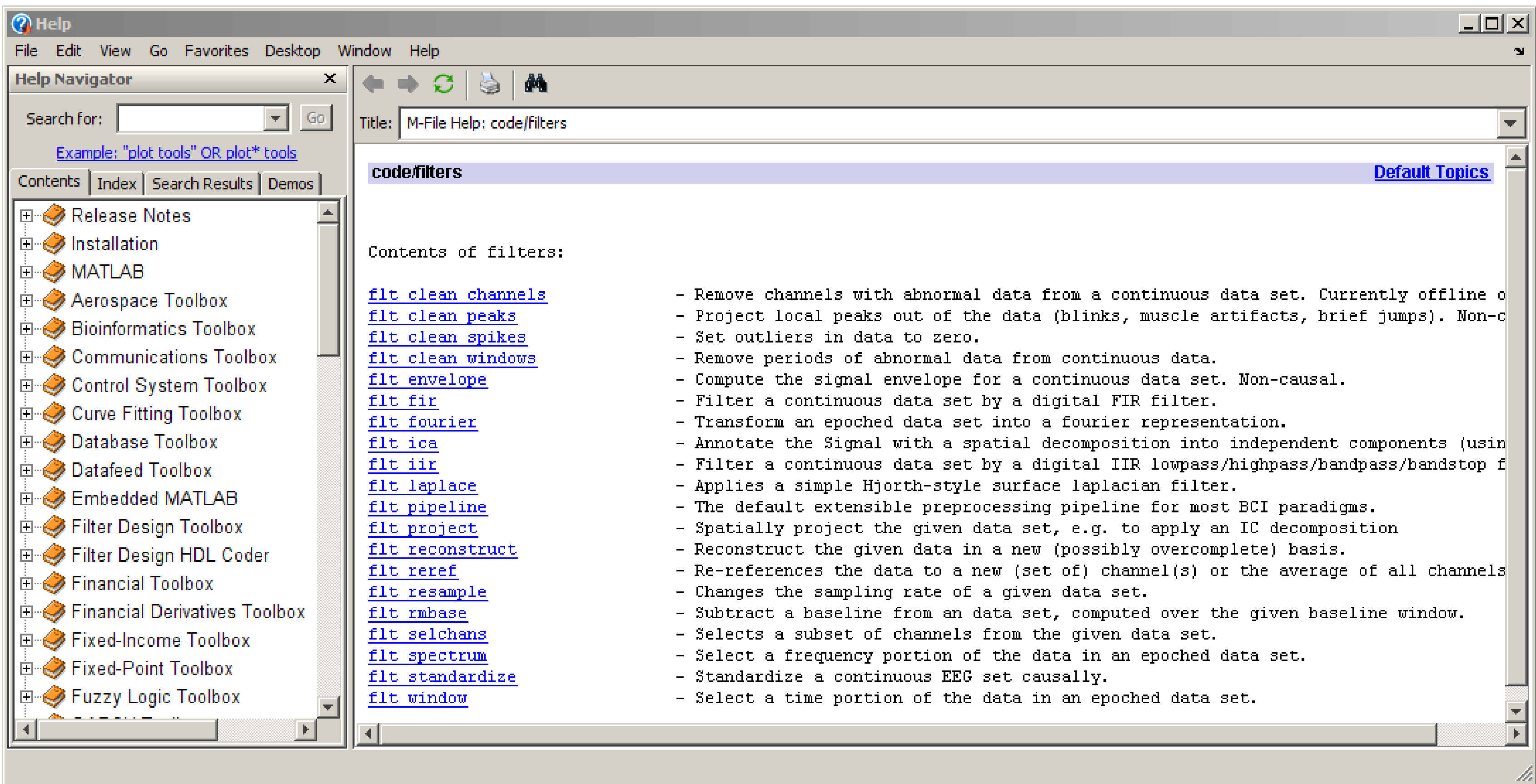
Toolbox GUI


MATLAB 7.6.0 (R2008a)
File Edit Debug Parallel Desktop Window Help
BCILAB 0.9
Data Source Offline Analysis Online Analysis Settings Help
results are in C:\DEVEL\Matlab\bcilab-0.9-stable\\userdata
cache is in C:\tmp\bcilab_cache (location_1)
temp is in C:\tmp\bcilab_temp
Welcome to the BCILAB toolbox!
>>
Start
OVR

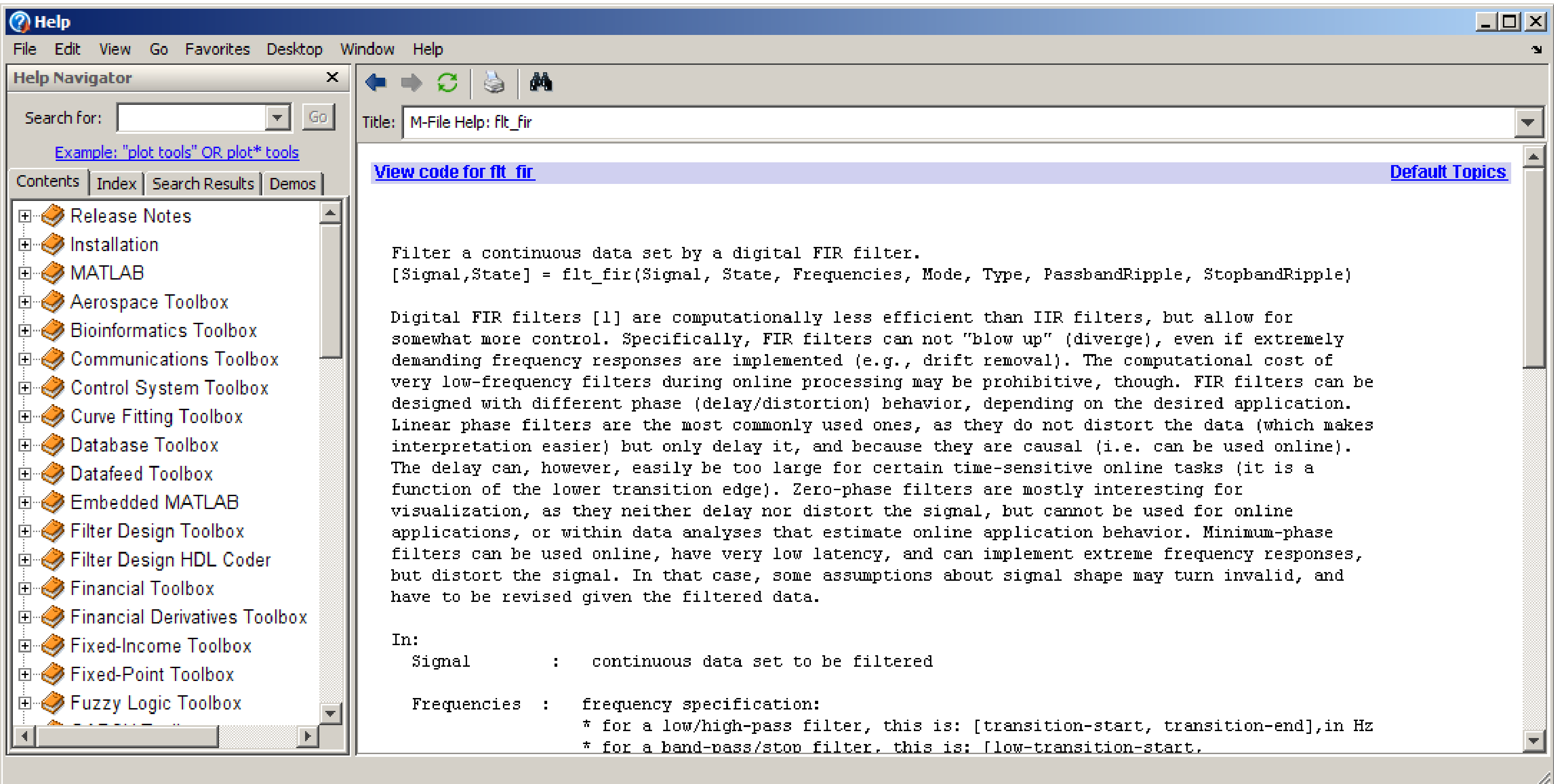
Getting Help (if Needed)



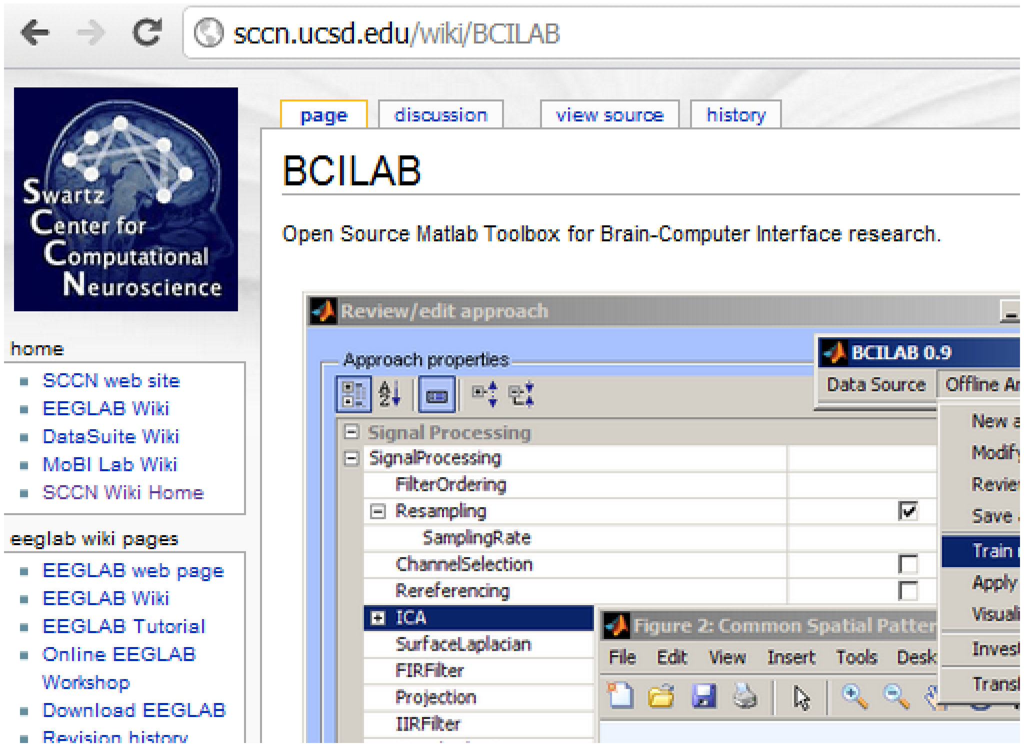
Getting Help (if Needed)



Getting Help (if Needed)



Getting Help (if Needed)



[http://sccn.ucsd.edu/**wiki/BCILAB**](http://sccn.ucsd.edu/wiki/BCILAB)

Use Case A

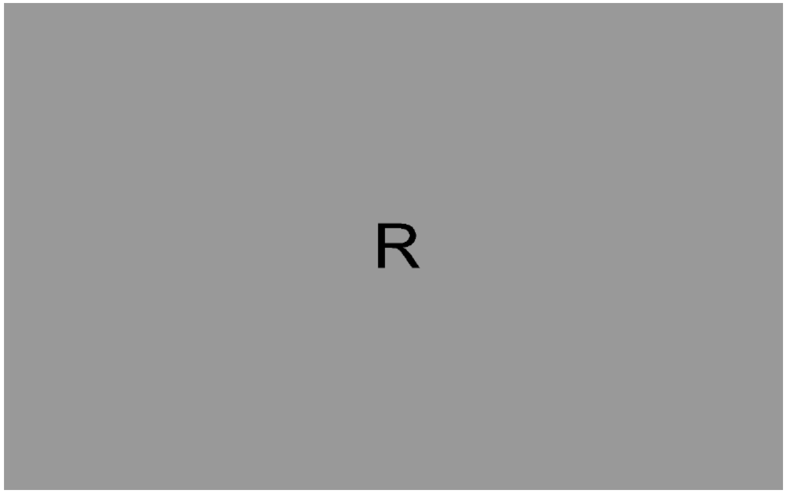
- You just recorded pilot data for some new study
- The idea is to try to estimate a certain aspect of cognitive state
- The question is what method works best, and what accuracies can be achieved

Use Case A

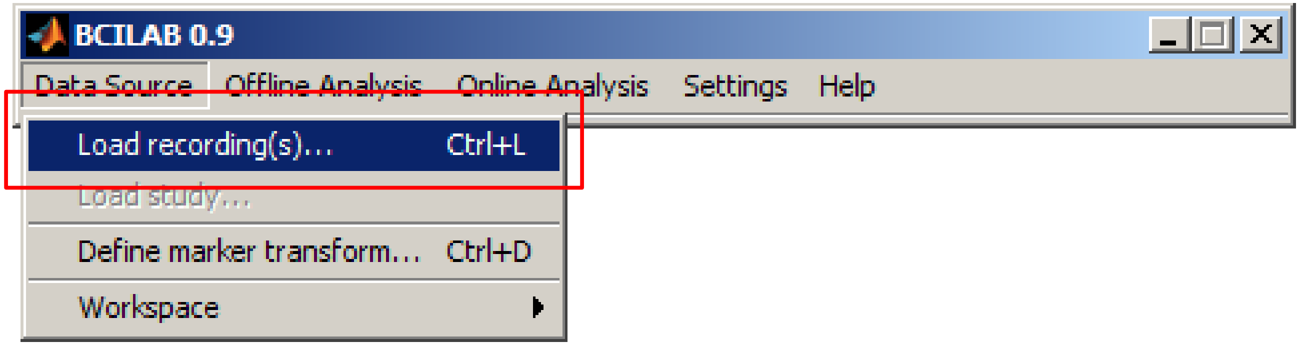
- Scenario: Subject is instructed to imagine a hand movement, either **left** hand or **right** hand (standard BCI case)
- Task: Estimate, from raw data, which hand movement was imagined
- Experimental data: EEG, 32 channels, 2 sessions (each ~30 min.)

Experimental task

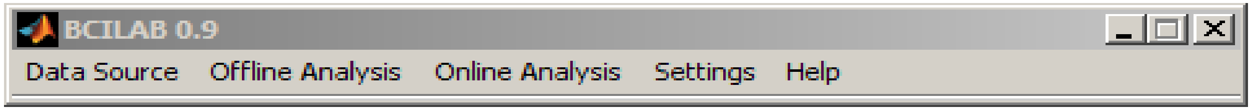
- 160 trials
- Randomized Instruction: L or R
- Displayed for 3s, followed by blank screen for 3.5s
- Sample:

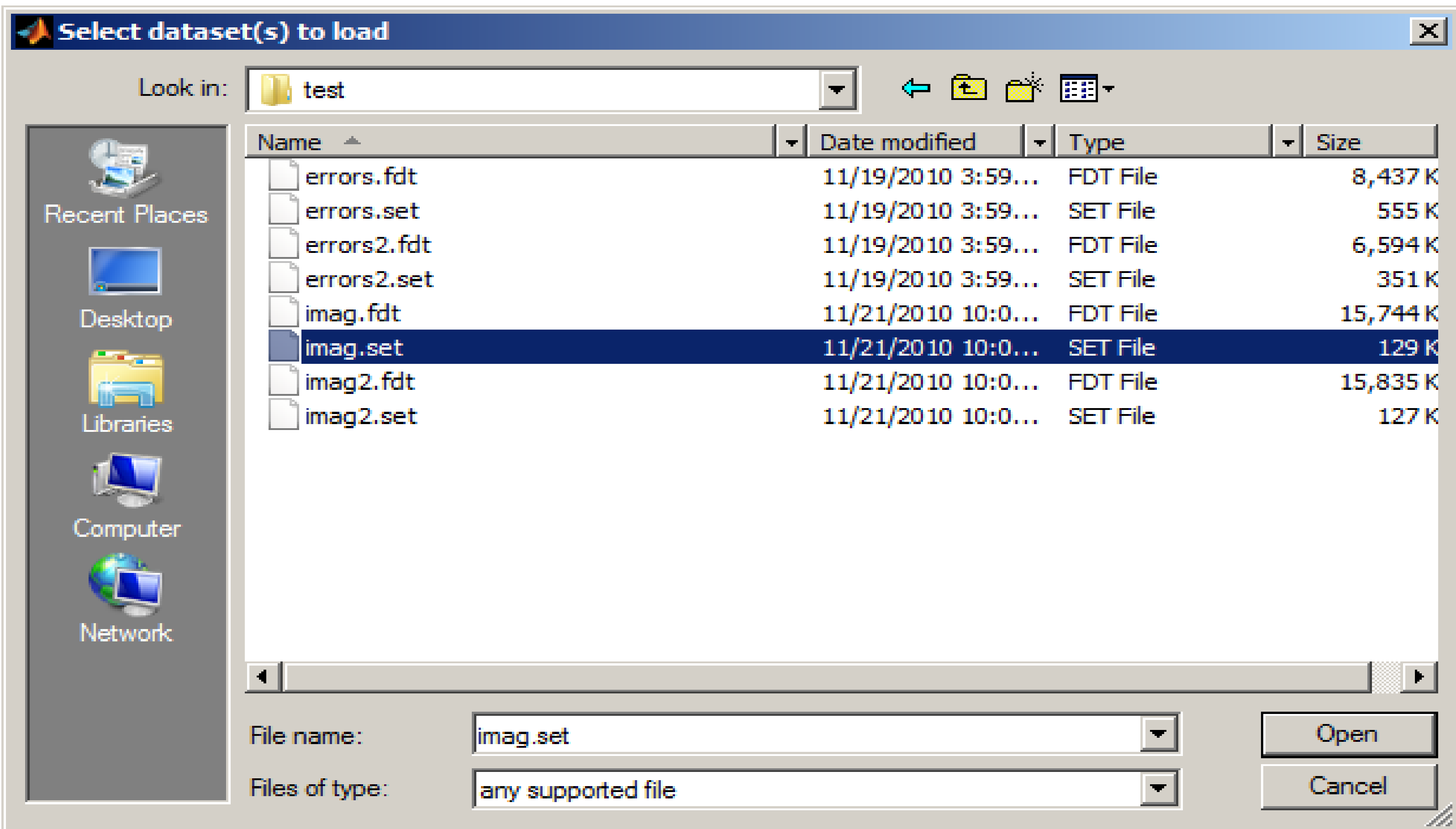


Loading the Data

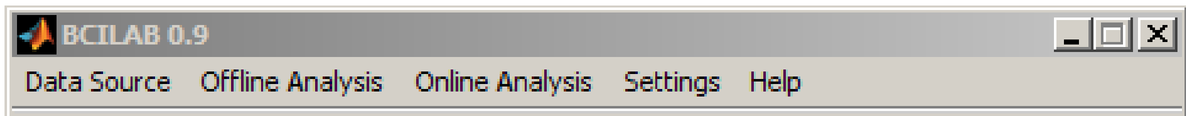


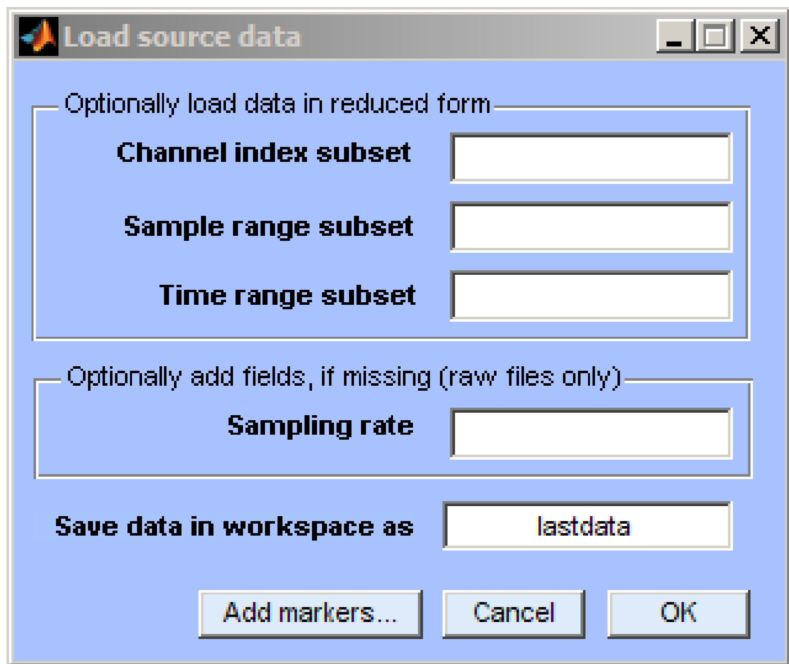
Loading the Data





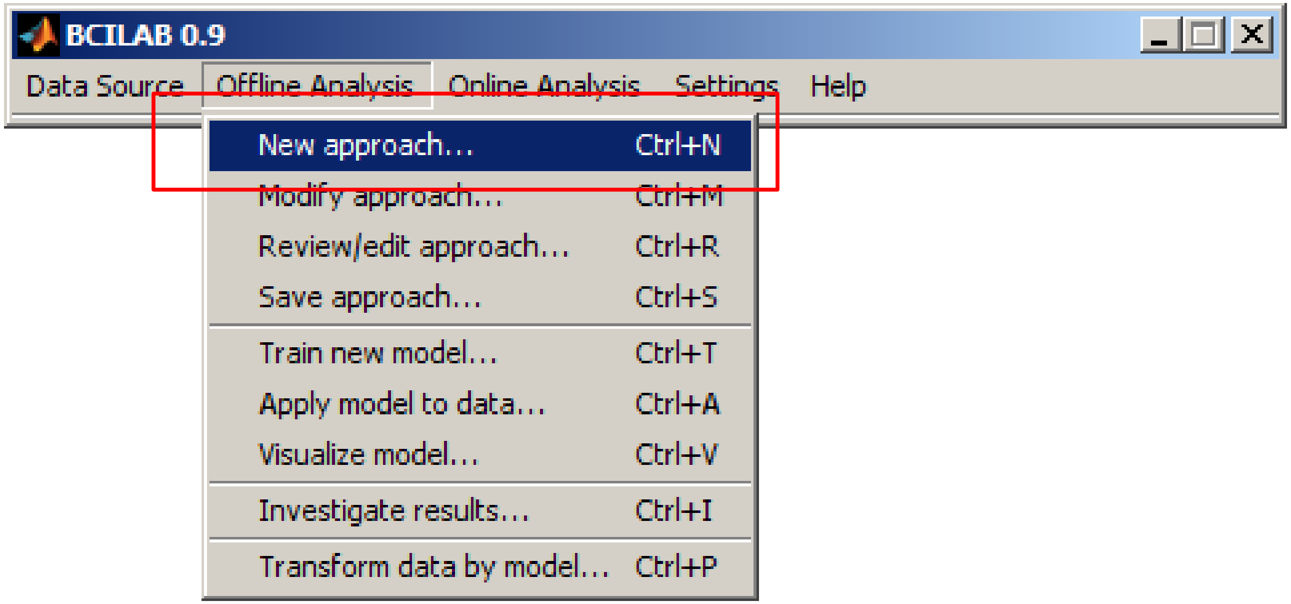
Loading the Data





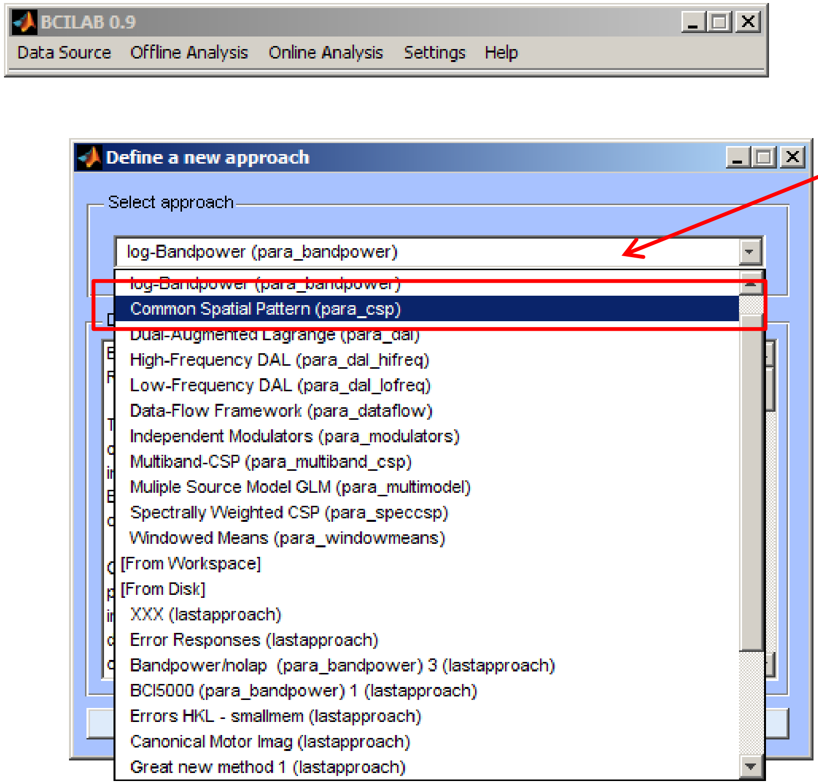
Defining an Approach

- An approach addresses both parts of the BCI problem: Mapping from observed signals to predictions, and learning the unknown parameters



Defining an Approach

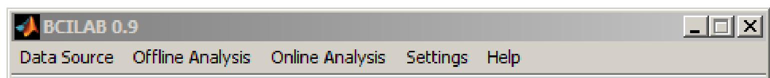
- You never start completely from scratch, but on the basis of what is known to work

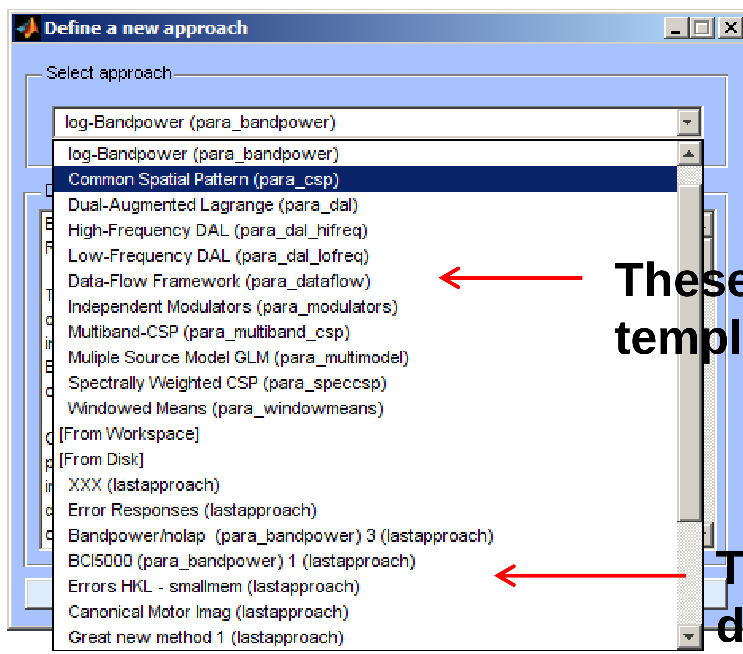


Common Spatial Pattern

Defining an Approach

- Some of these work best for oscillatory processes, others for ERP-like features, etc.

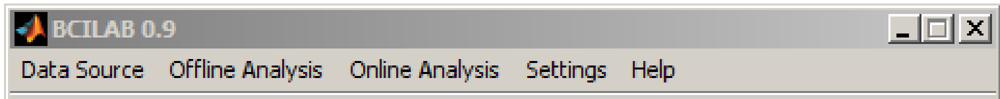


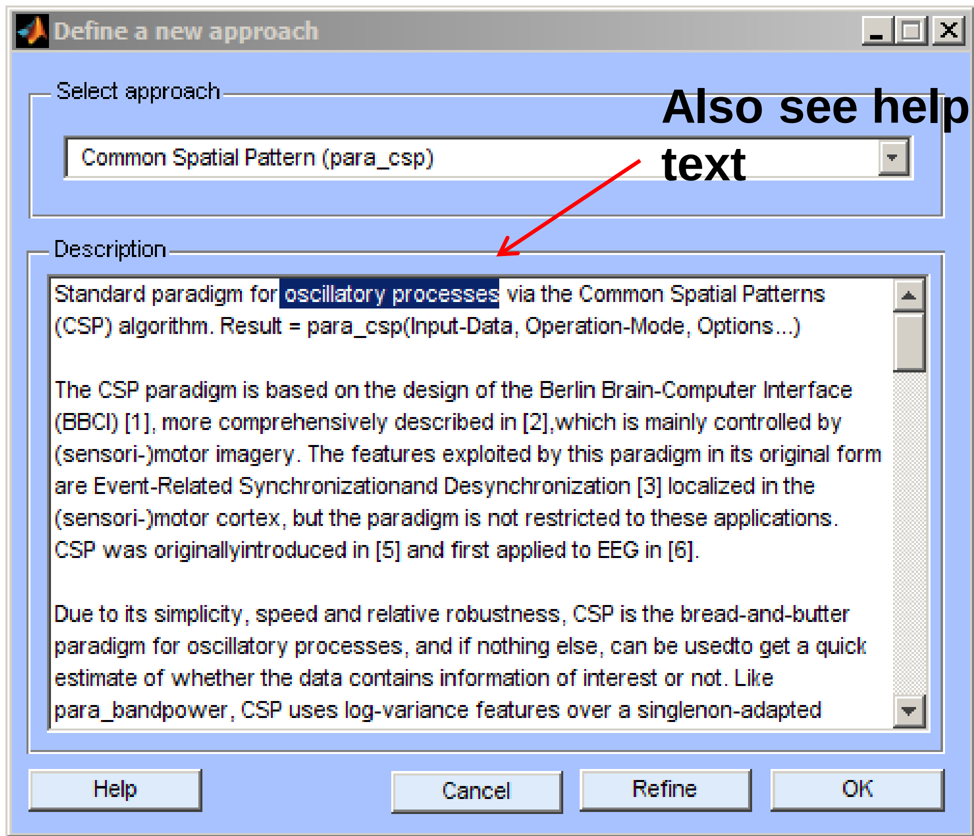


These are pre-defined templates

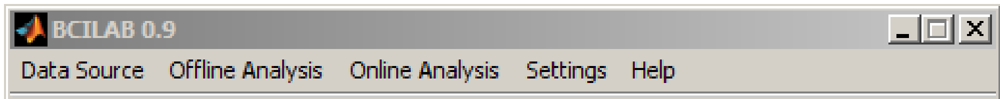
These are user-defined

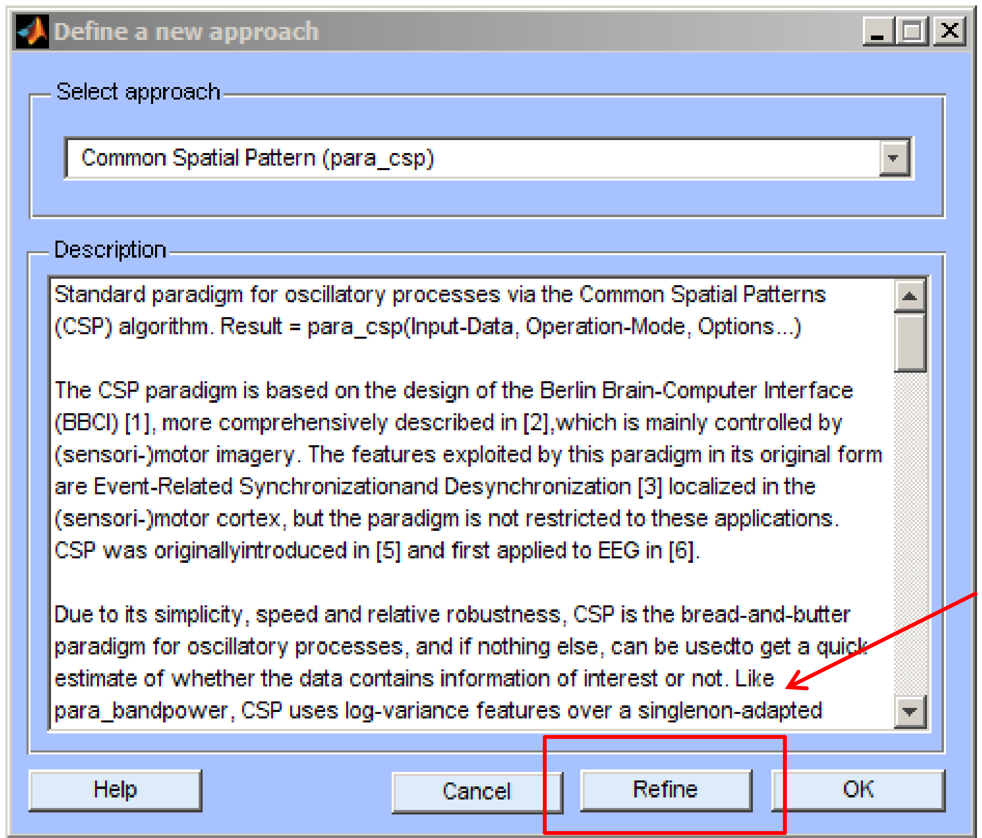
Defining an Approach





Defining an Approach

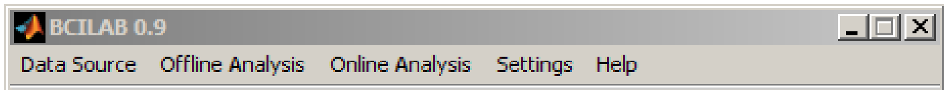


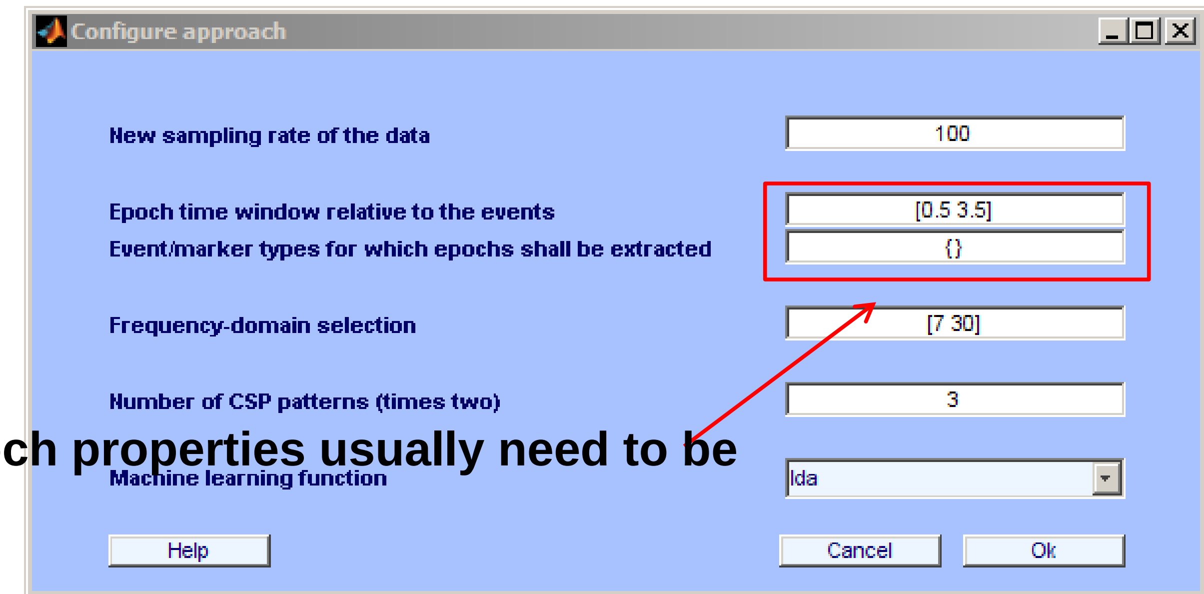


Adapt the template to your experiment

Configuring an Approach

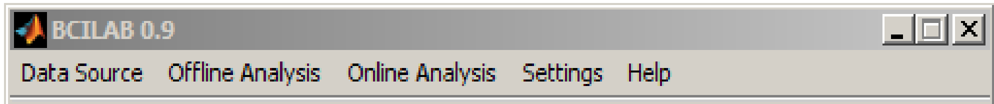
- Key properties can be configured in this dialog

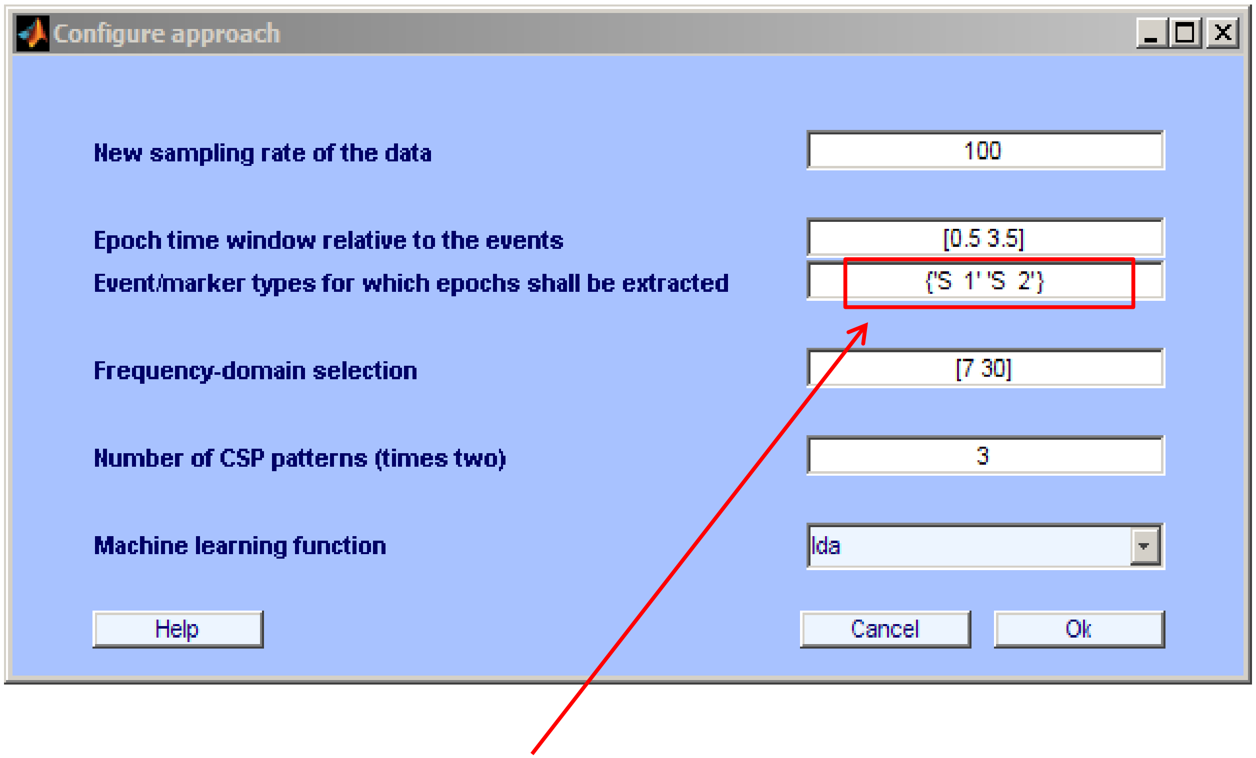




Trial epoch properties usually need to be adapted

Configuring an Approach

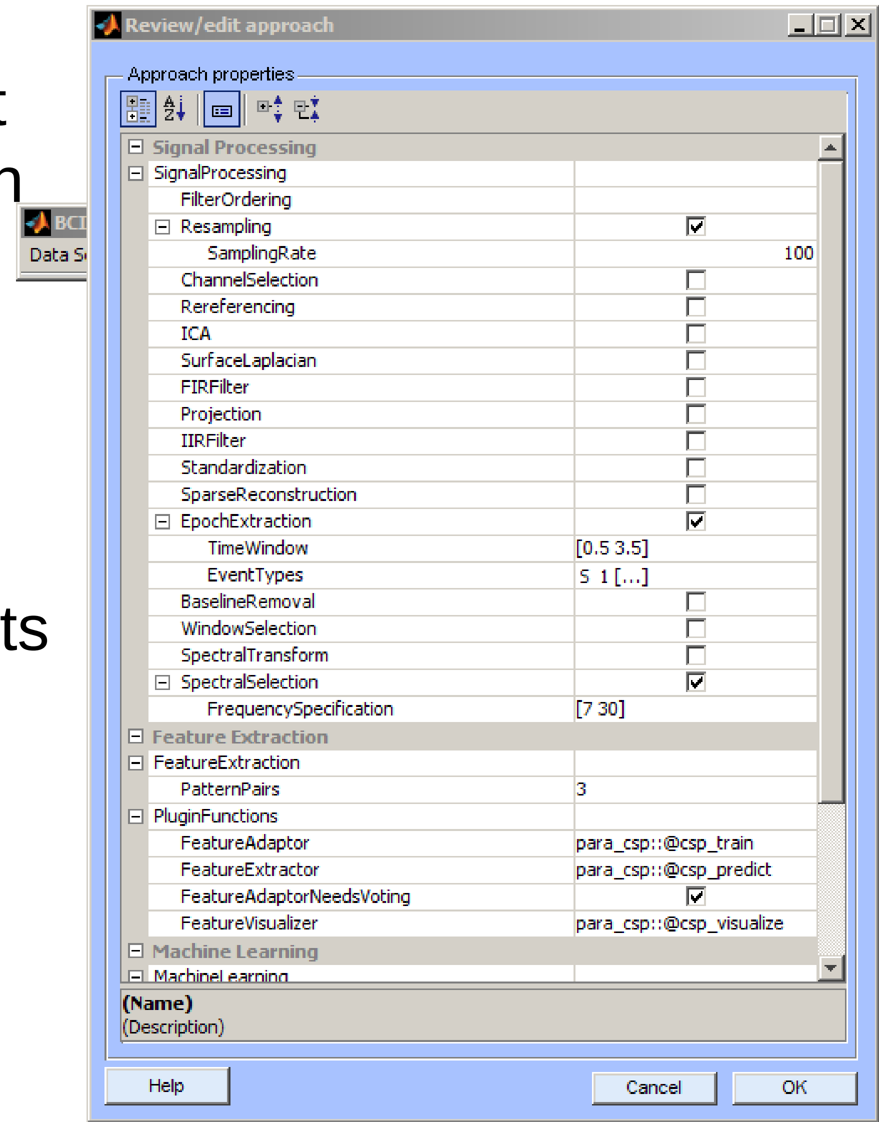




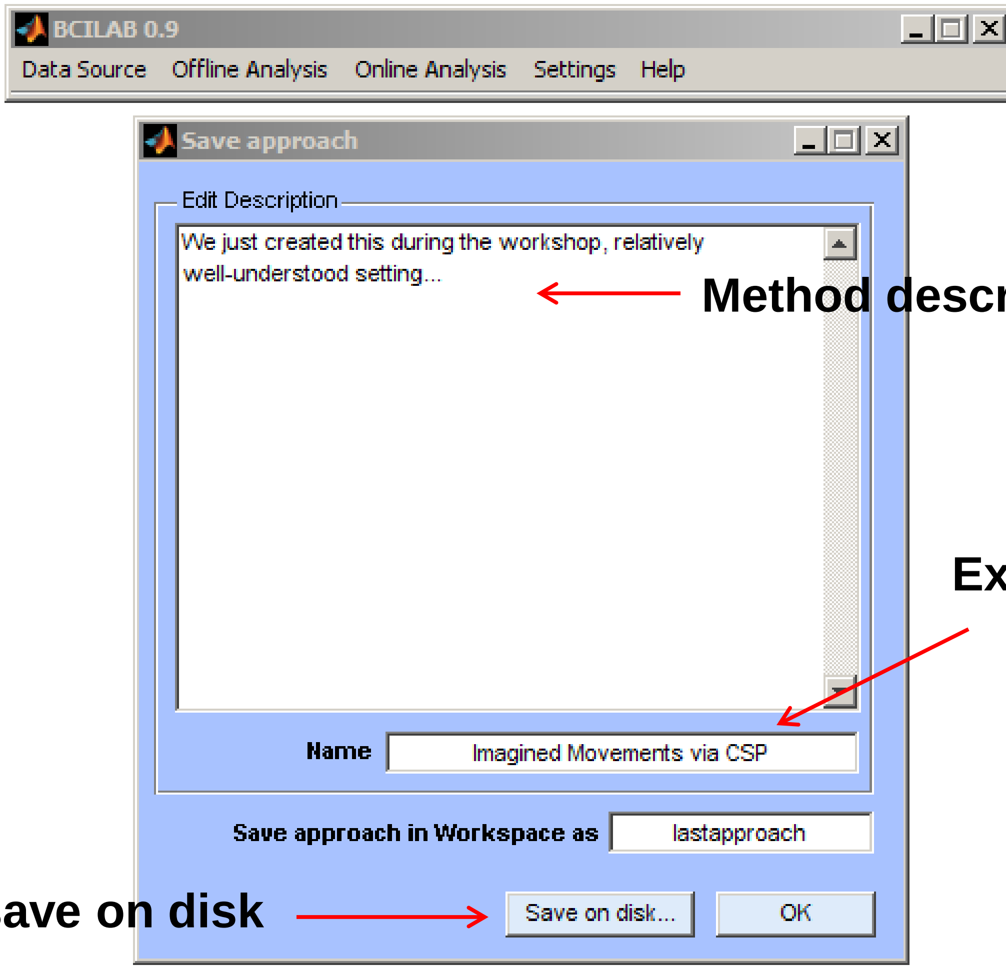
Also, target marker types in the data have to be specified

Reviewing/Editing an Approach

- The next panel allows to edit all properties of the approach
- Filter stages can be added and configured
- Feature extraction can be configured
- Machine learning components can be selected and configured
- For now, nothing to do



Saving the Approach



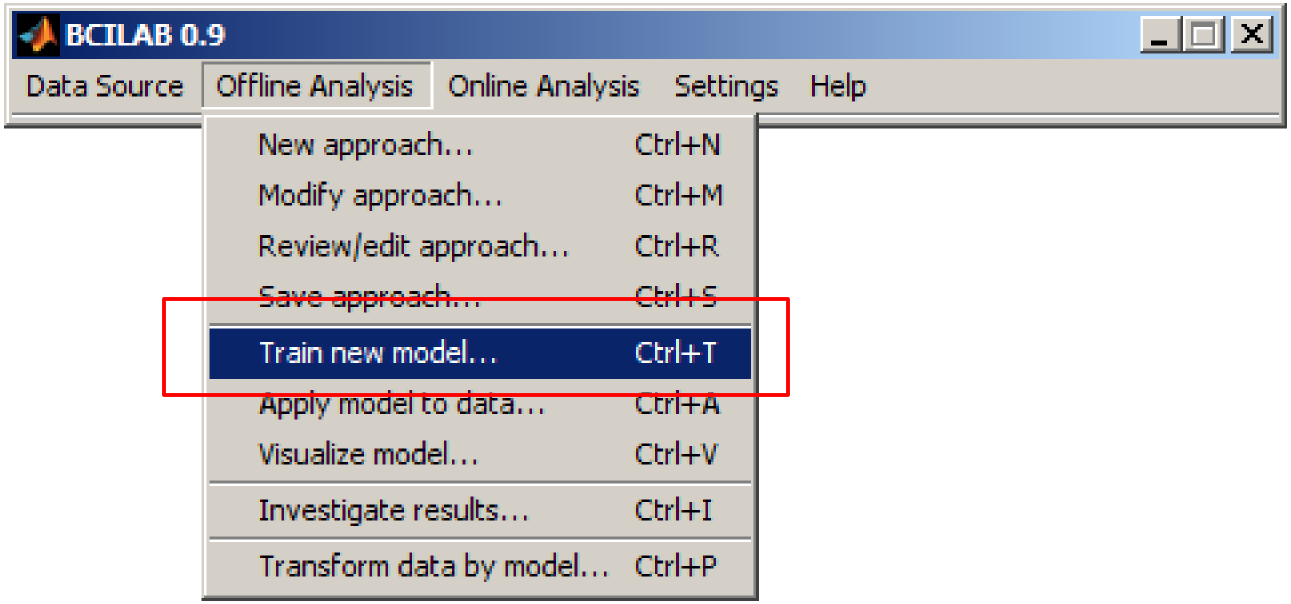
Method description

Executive summary

May save on disk

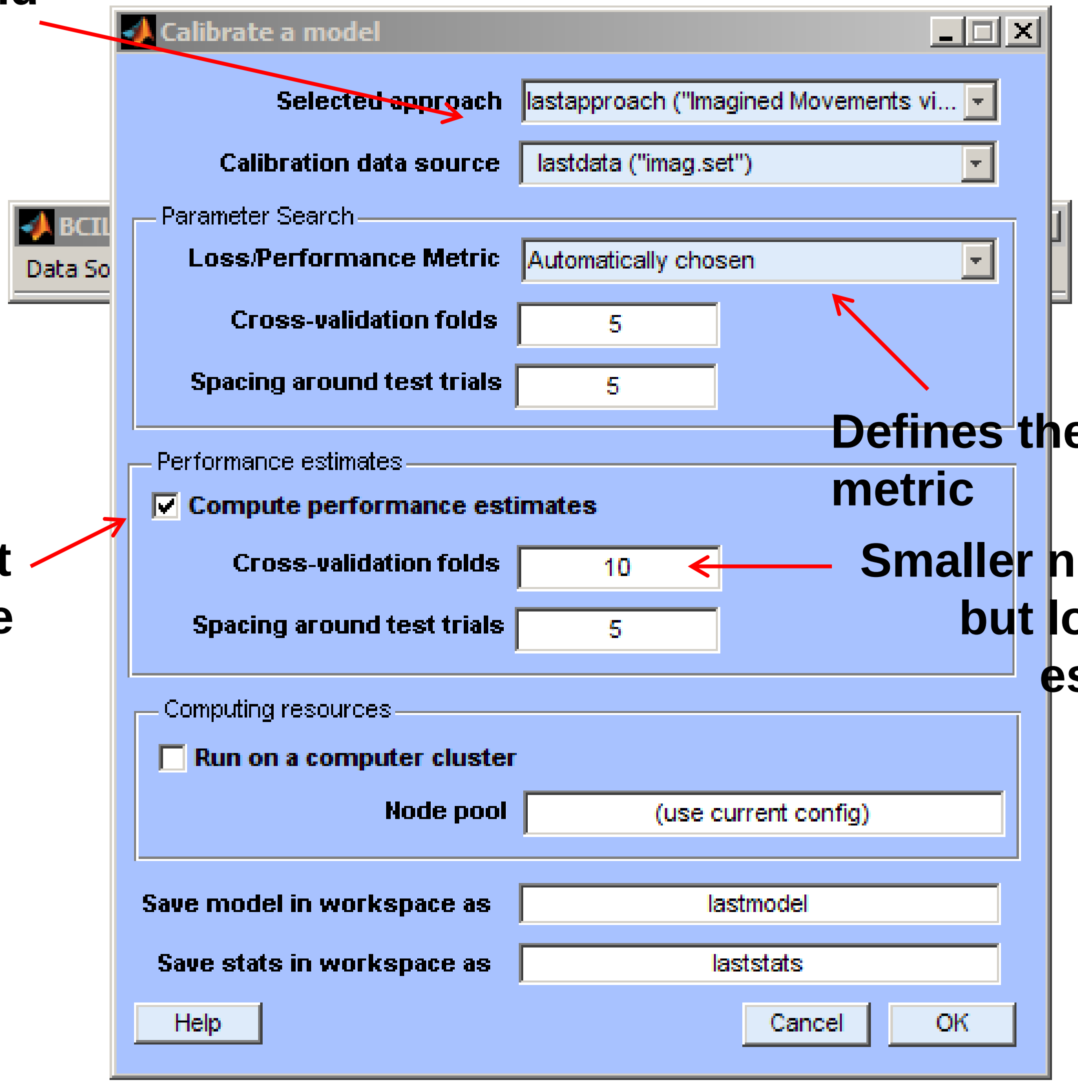
Learning a Predictive Model

- Put the method to the test...



Learning a Predictive Model

Approach and data to use

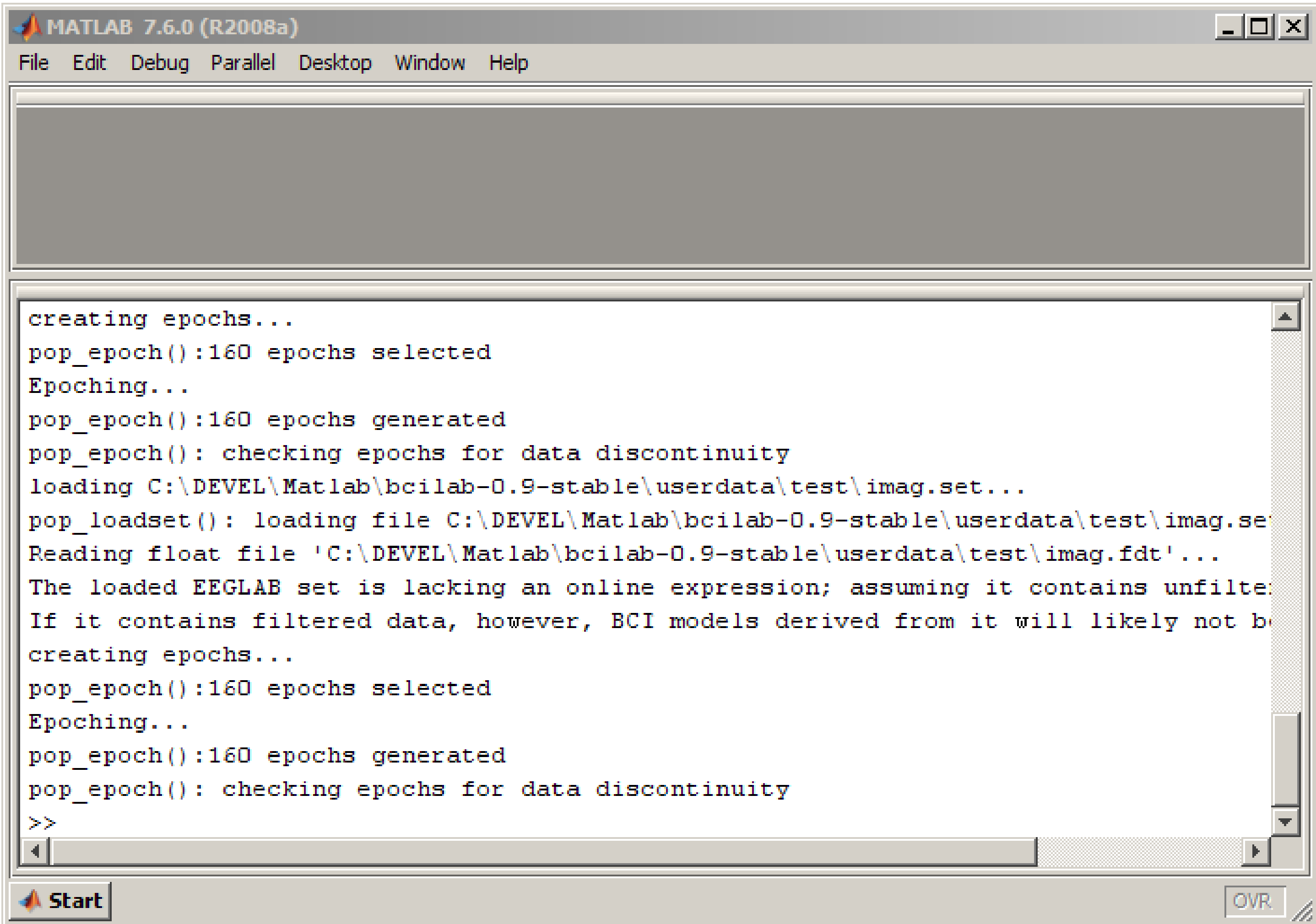


Defines the performance metric

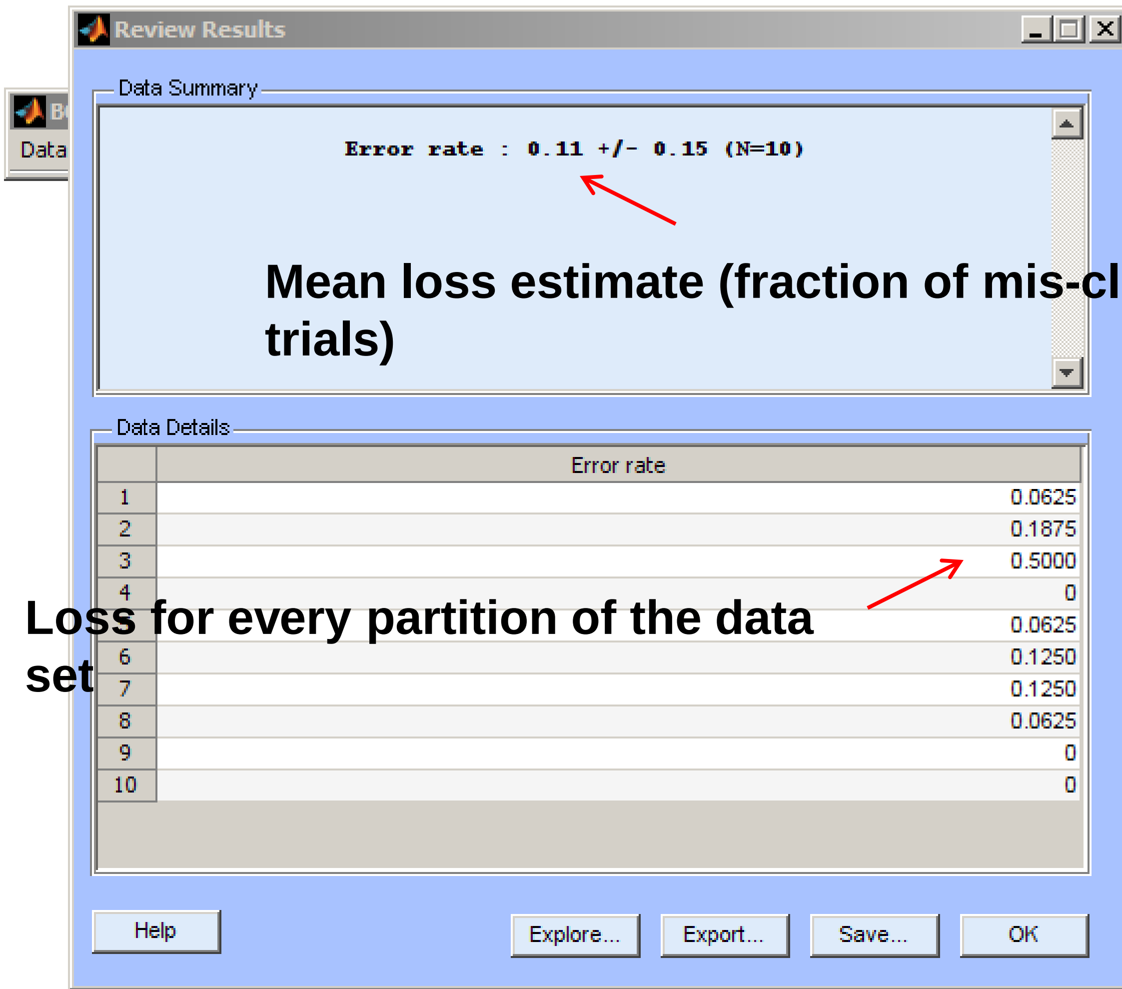
Check to get performance estimates

Smaller number: faster, but lower quality estimates

Waiting for Results



Reviewing Results



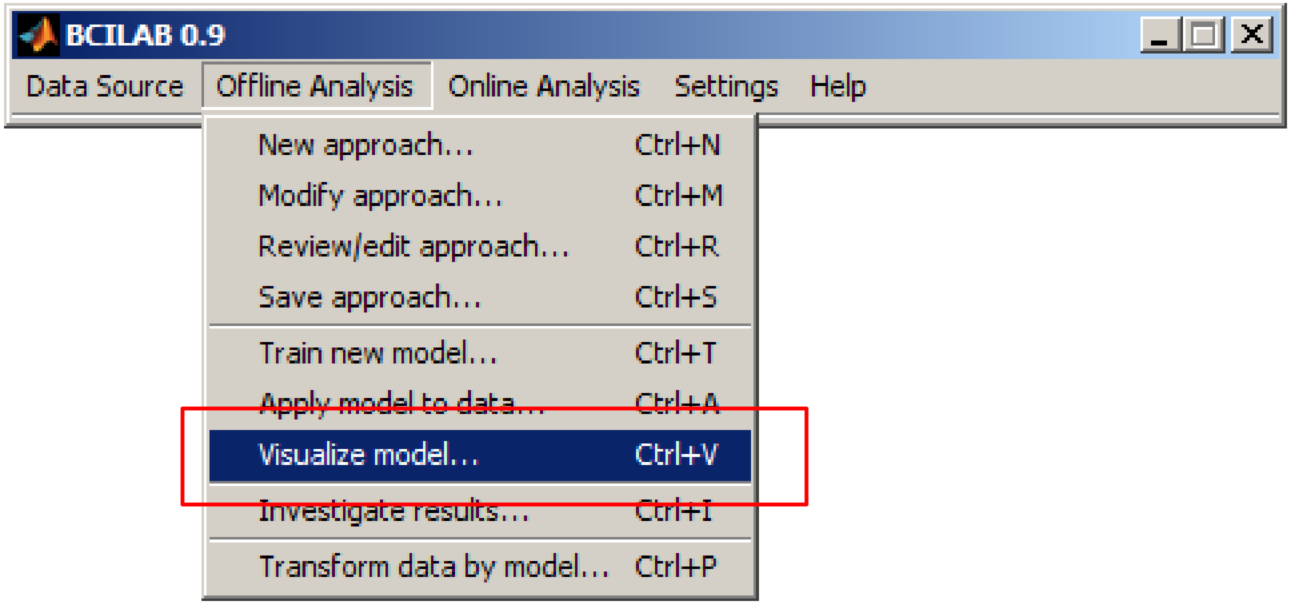
Mean loss estimate (fraction of mis-classified trials)

Loss for every partition of the data set

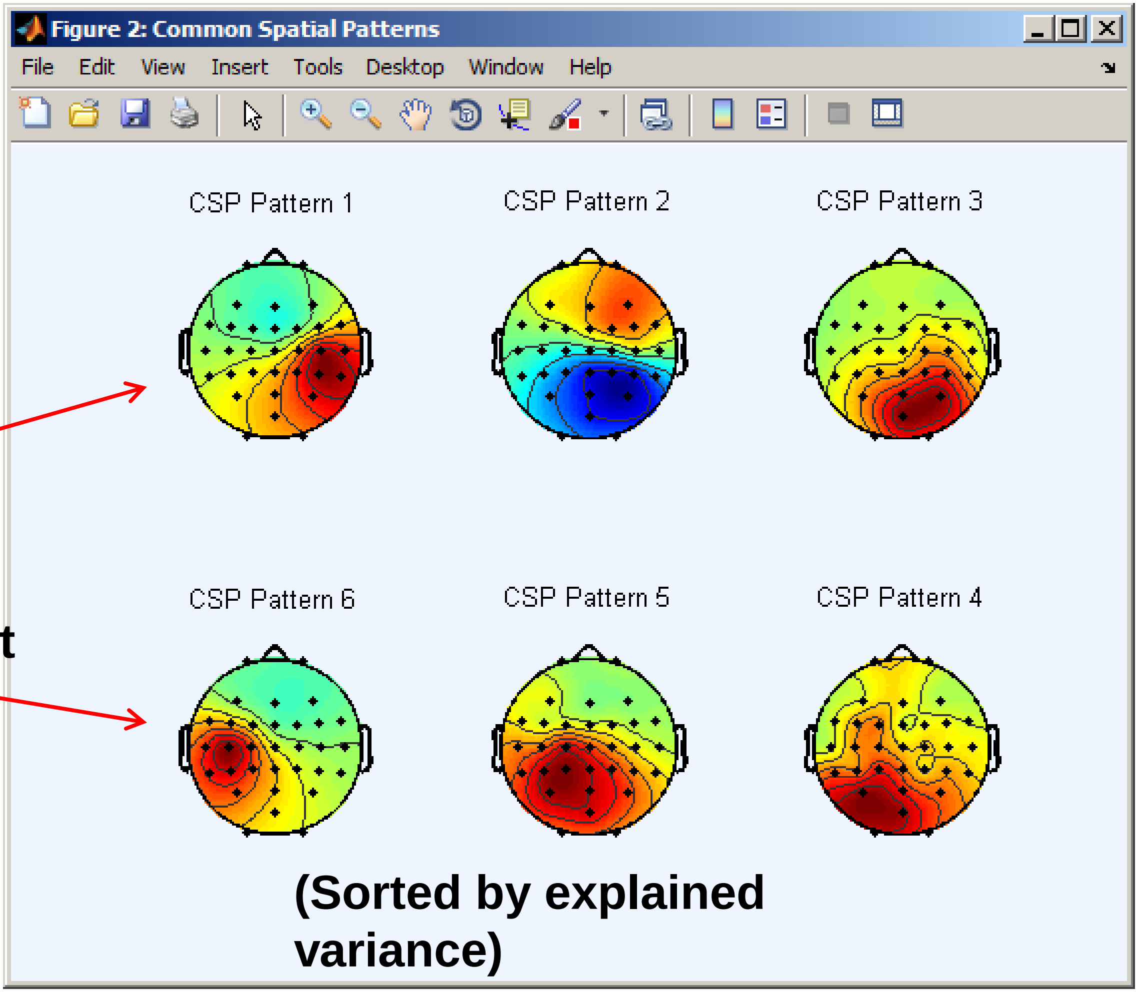
Reviewing Results

- 11% error rate is quite good for imagined movements; mean across studies & methods is probably closer to 25%
- chance level is here 50% (keep that in mind when evaluating)
- You may get multiple outputs (e.g., false positives, true positives, which show up in the table), depending on loss measure

Visualizing Model Properties



Visualizing Model Properties

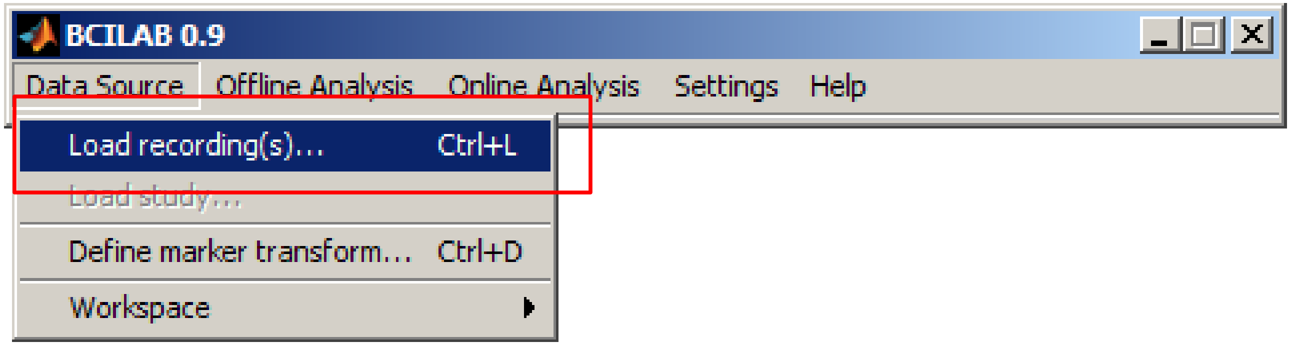


Motor cortex patterns, left hand

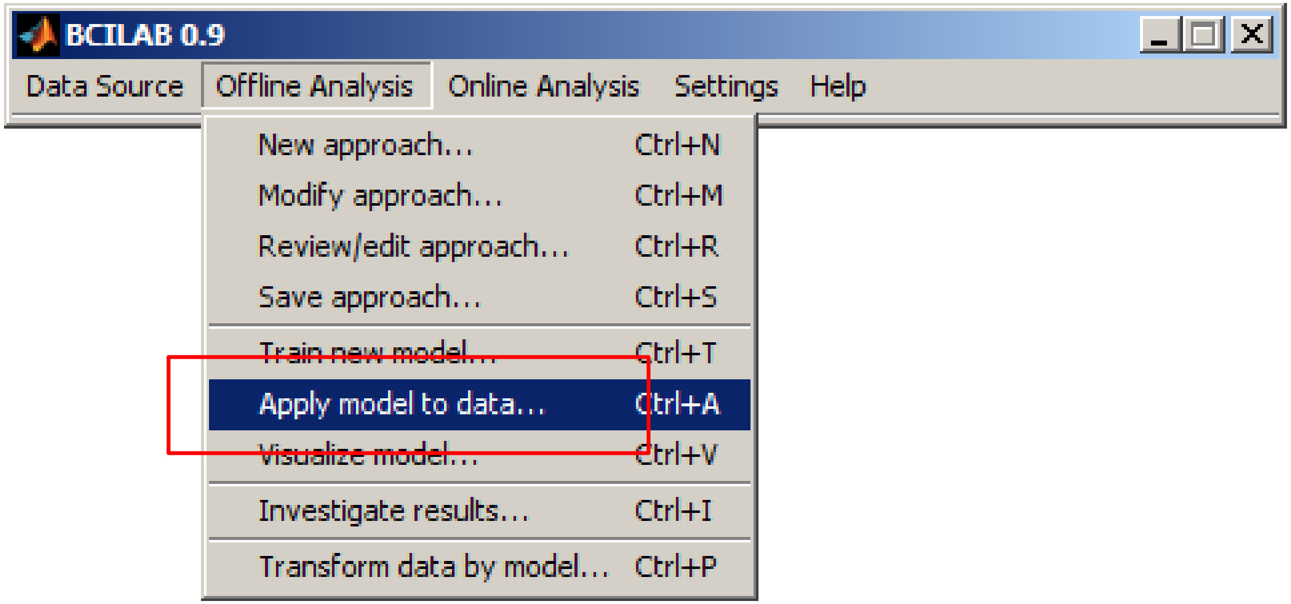
Motor cortex patterns, right hand

(Sorted by explained variance)

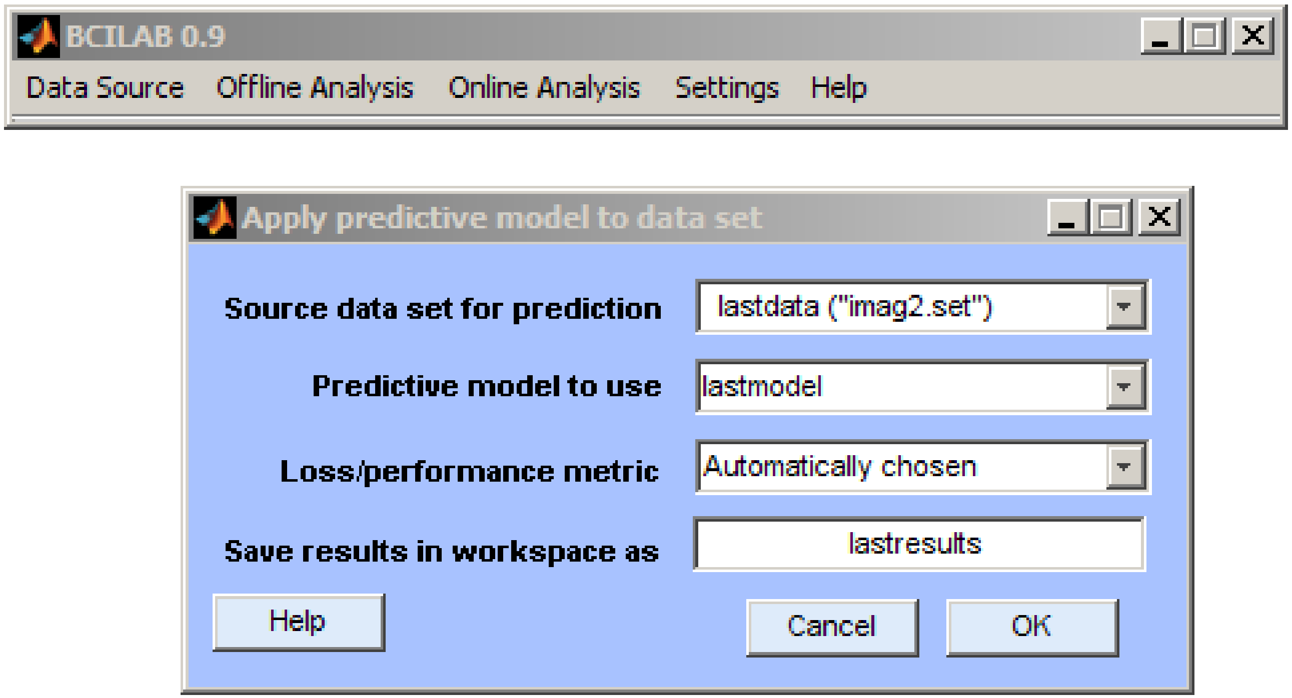
Apply Model to 2nd Session



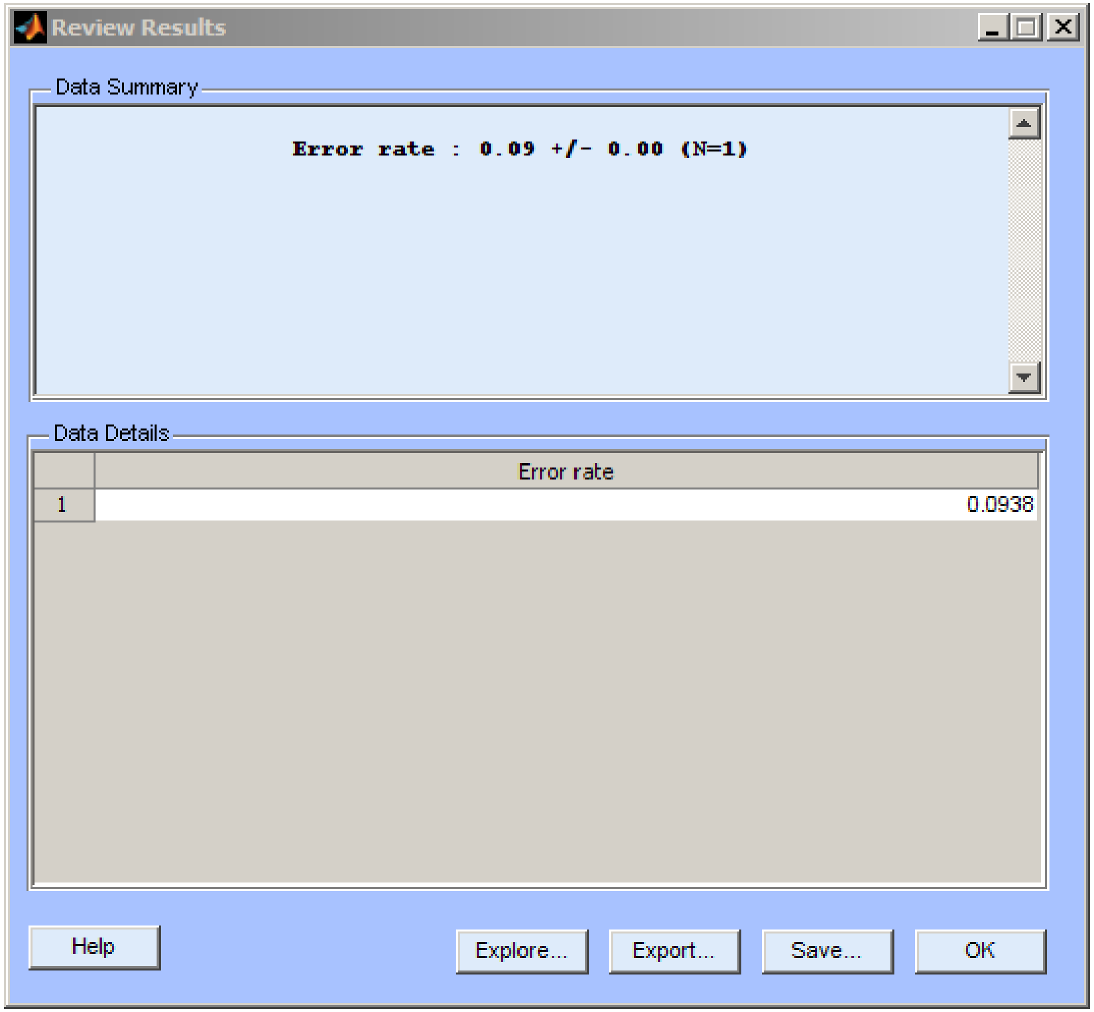
Apply Model to 2nd Session



Apply Model to 2nd Session

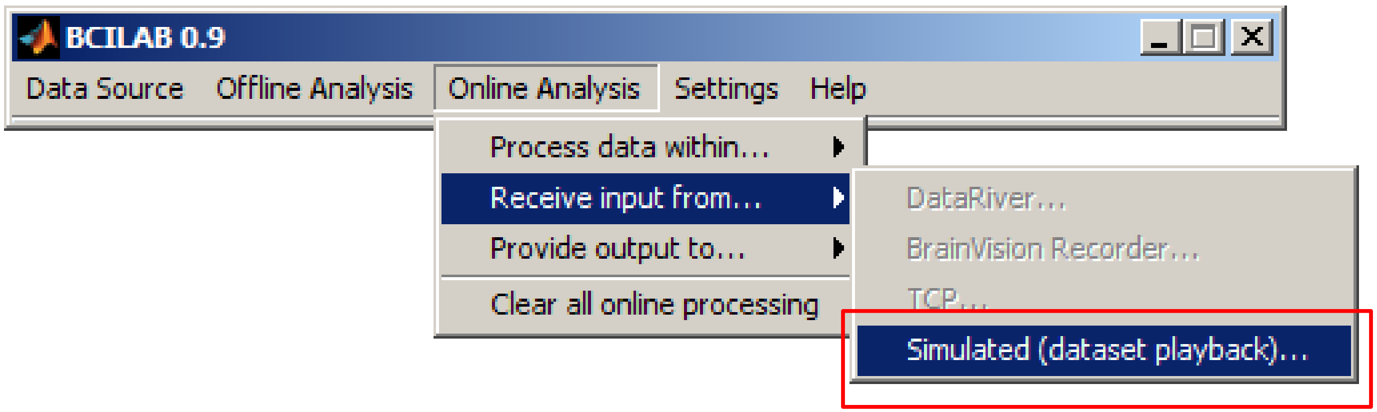


Reviewing Results



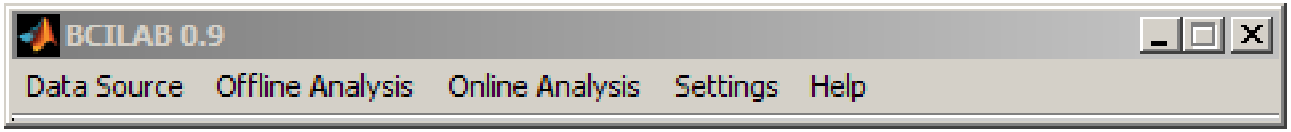
OR: Apply Model Online

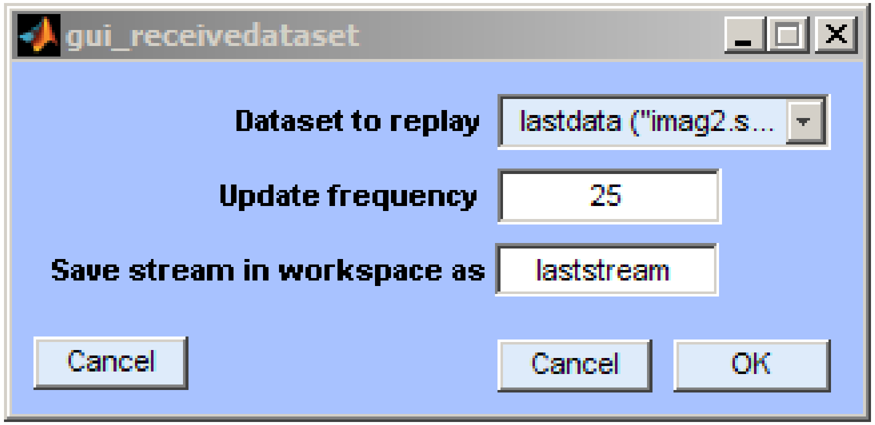
- (if you have a subject sitting next to you)
- Here: use a simulated data source (playing back the 2nd session)



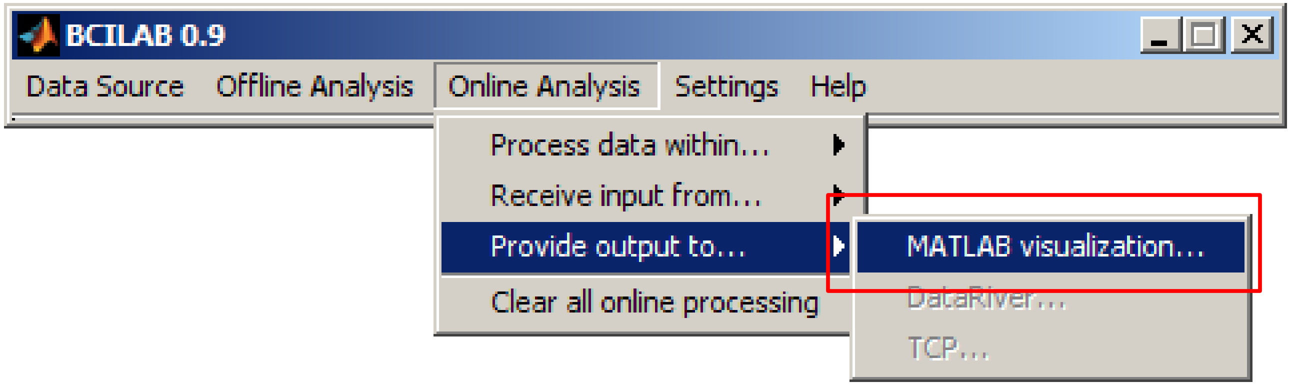
Online Application

- This adds a data feed process in the background



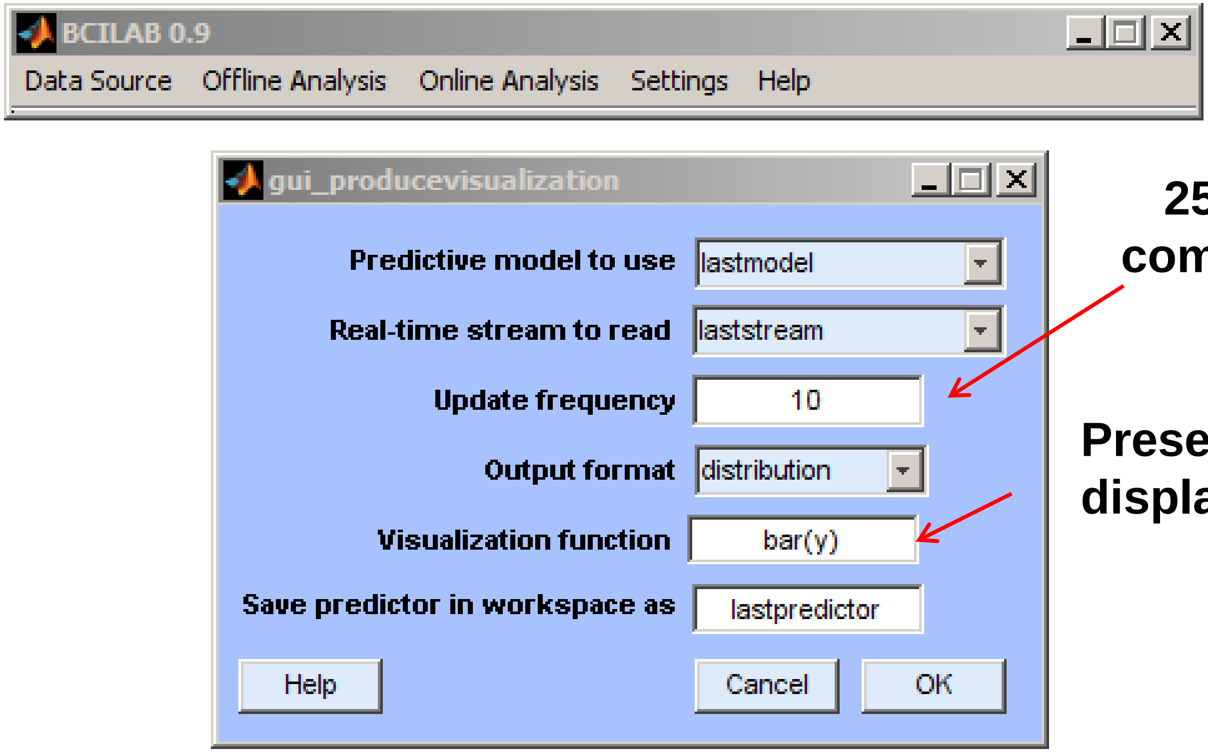


Online Application



Online Application

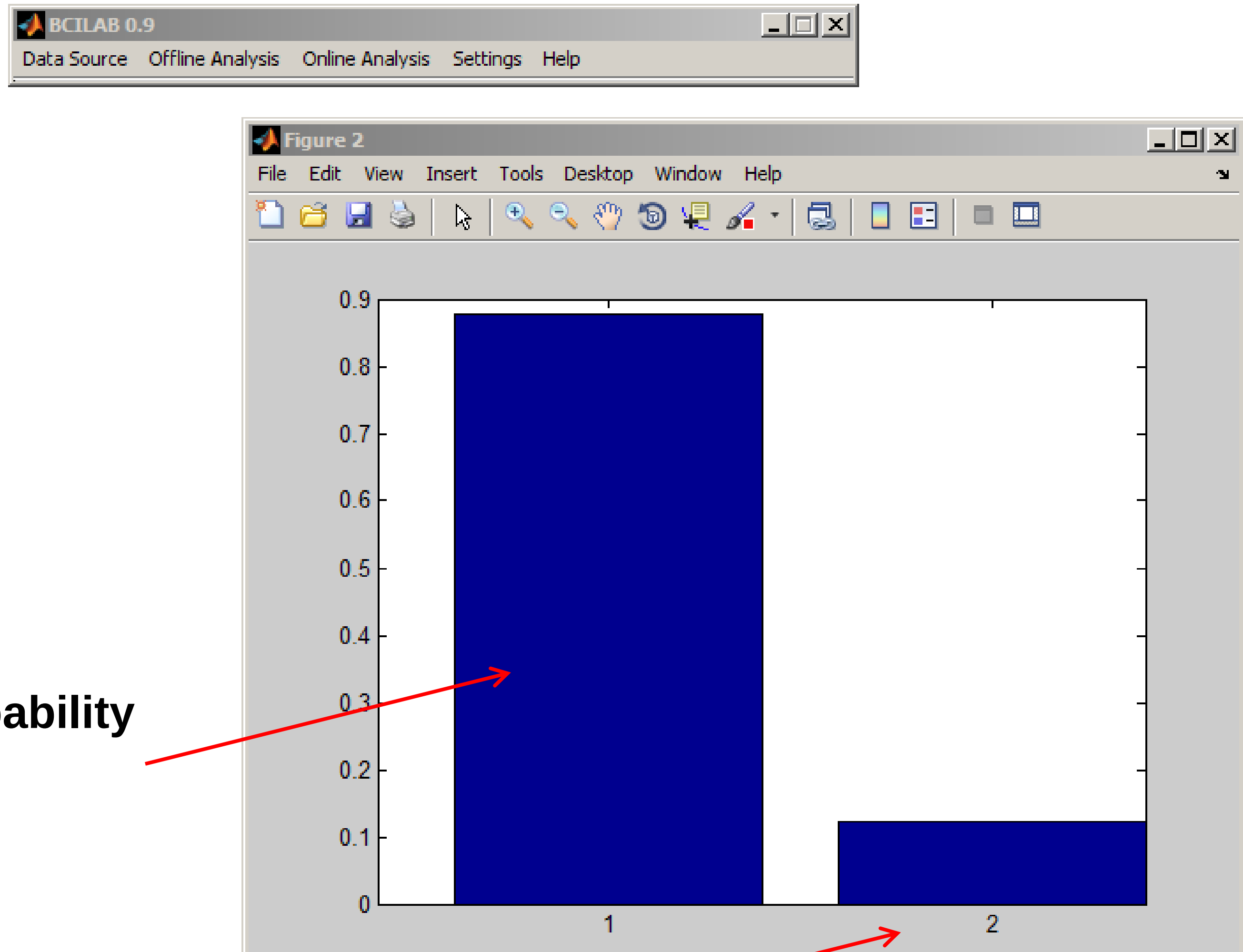
- This adds a real-time inference process in the background



25 Hz if your computer is fast enough

Presented via this display command

Real-time Output



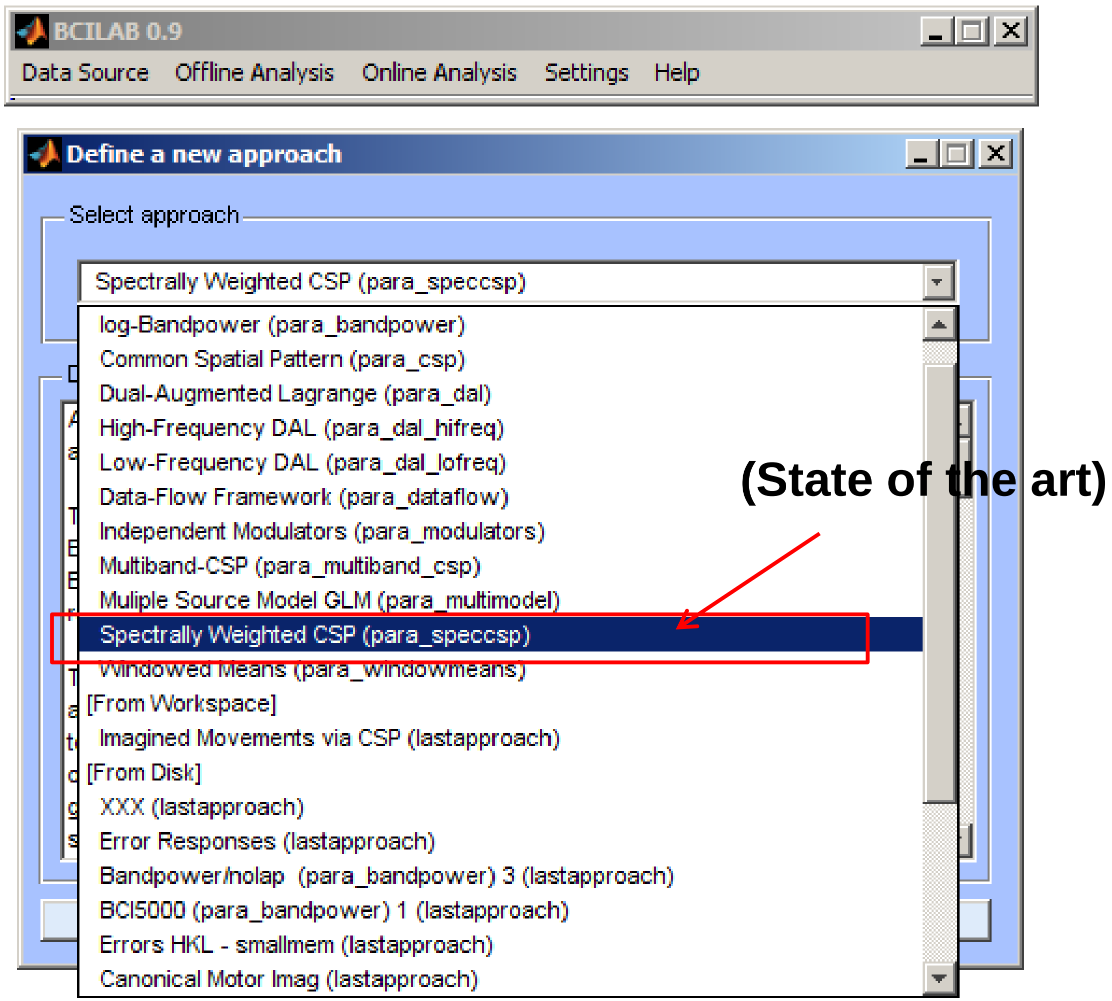
Current probability for class 1

Current probability for class 2

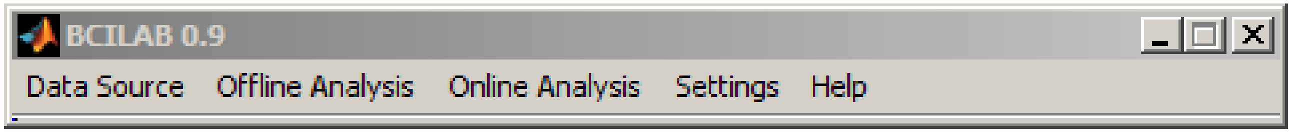
Real-time Output

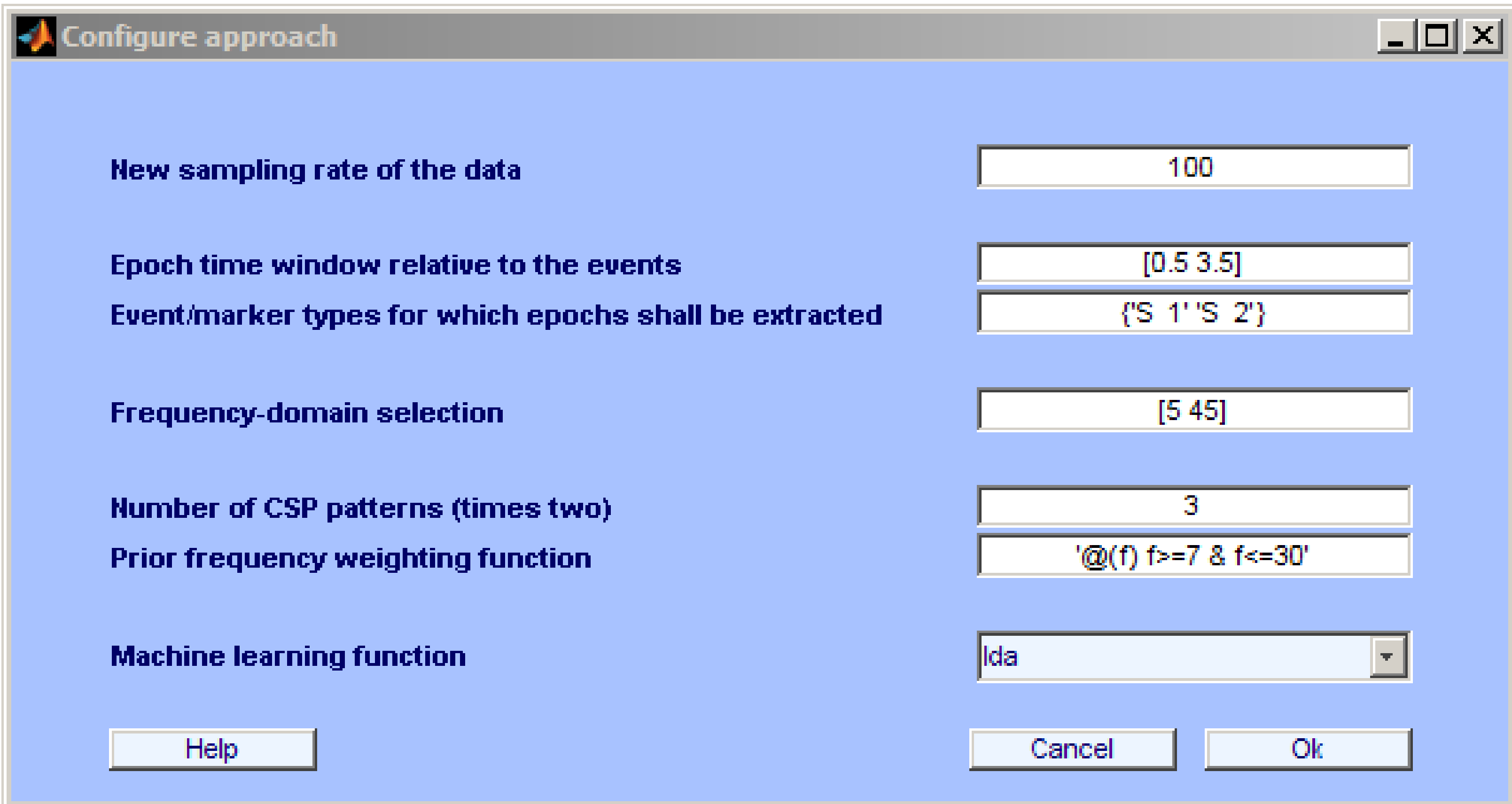
- If you have more classes, you get more bars
- You can also remap to other parameters (e.g. expected value)
- Note: the simple graphics command always renders into the current window

Using a More Ambitious Approach

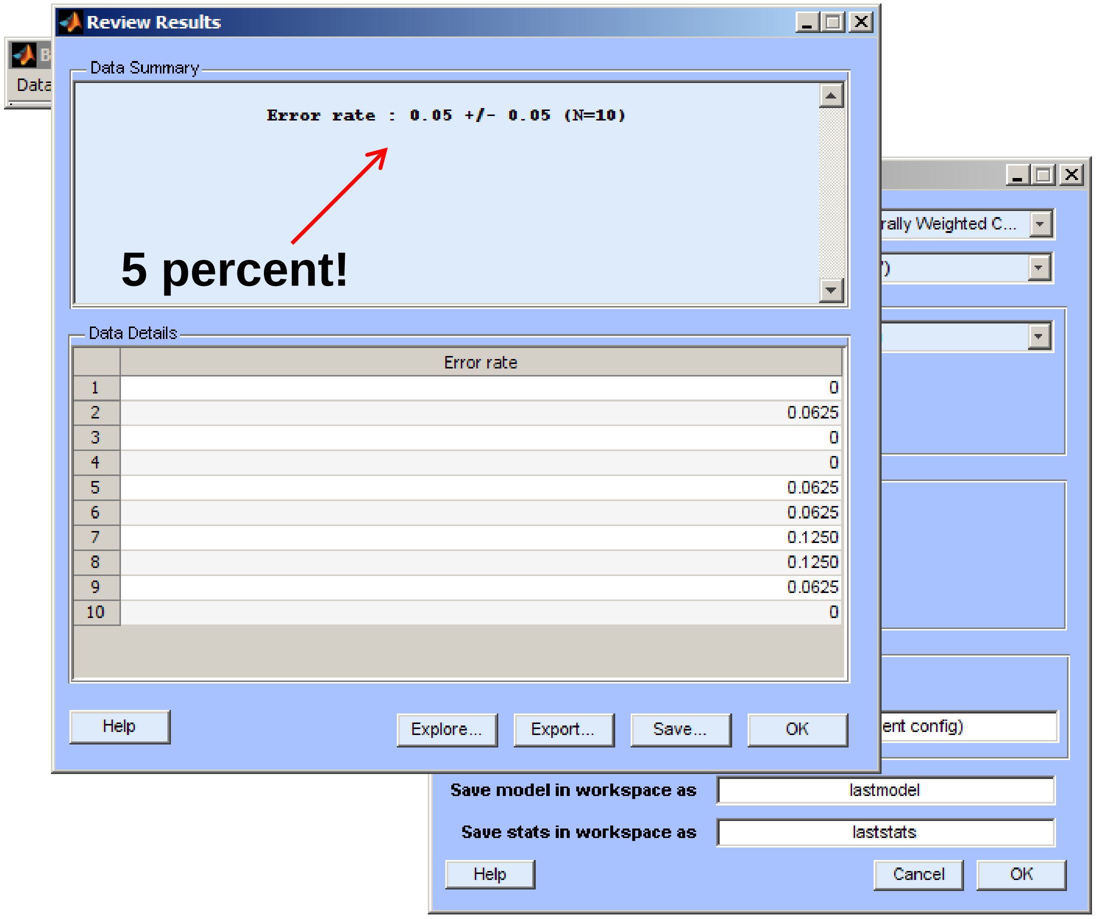


Using a More Ambitious Approach





Reviewing Results

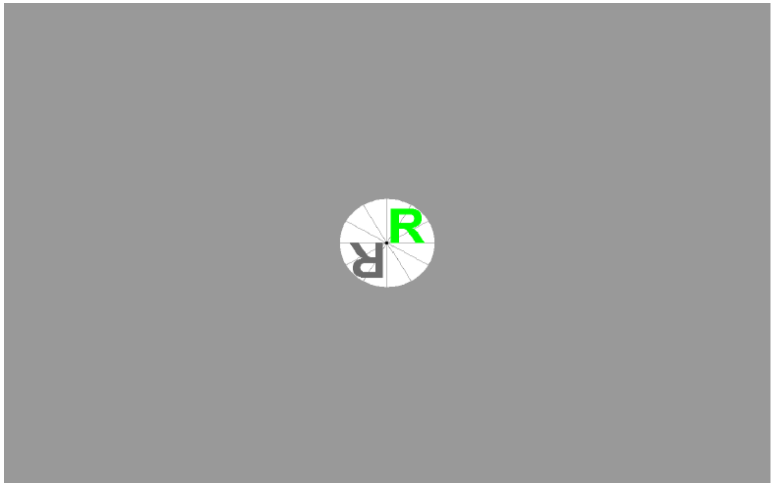


Use Case B

- Question: Can we predict whether the user perceives an event as being an error?
- Experimental data: EEG, 32 channels, 2 sessions

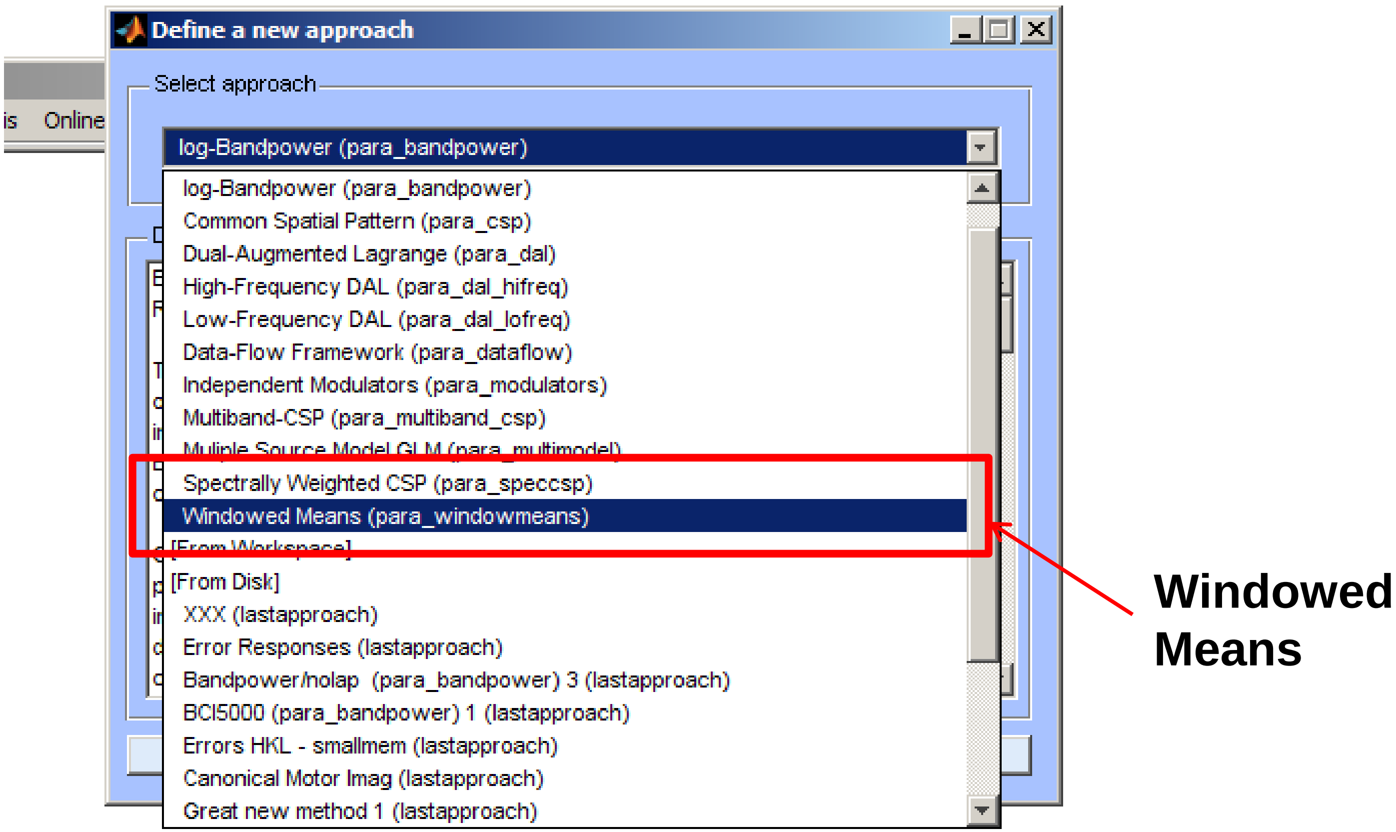
Experimental Task

- Experimental task: ~100 randomized trials, 3 types of stimuli:
  - expected/correct event: type 'S 11'
  - unexpected event A: type 'S 12'
  - unexpected event B: type 'S 13'
- Sample:

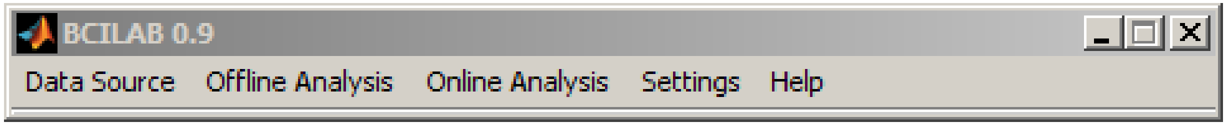


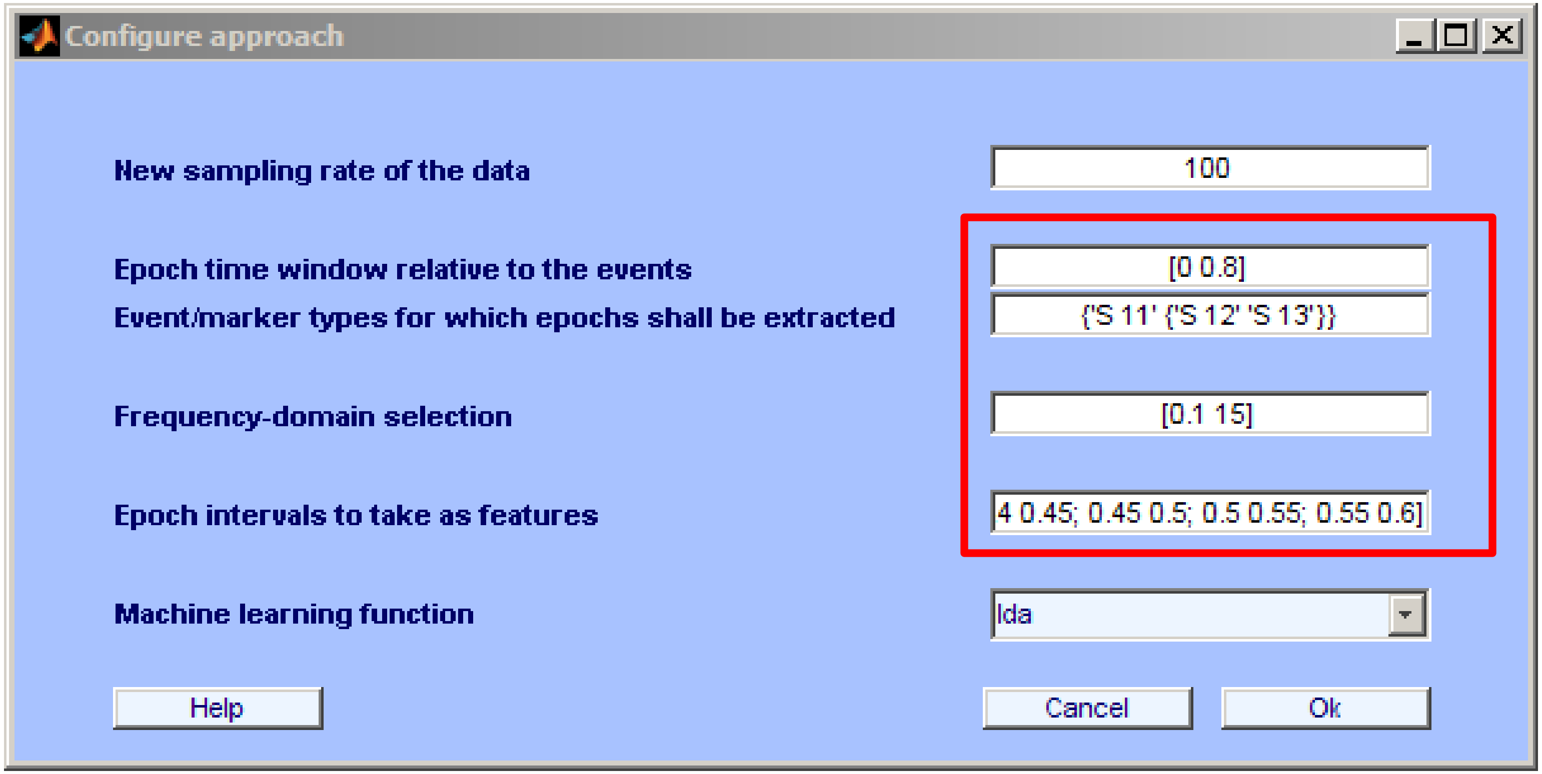
Define approach

- This time, an ERP-specific approach is needed

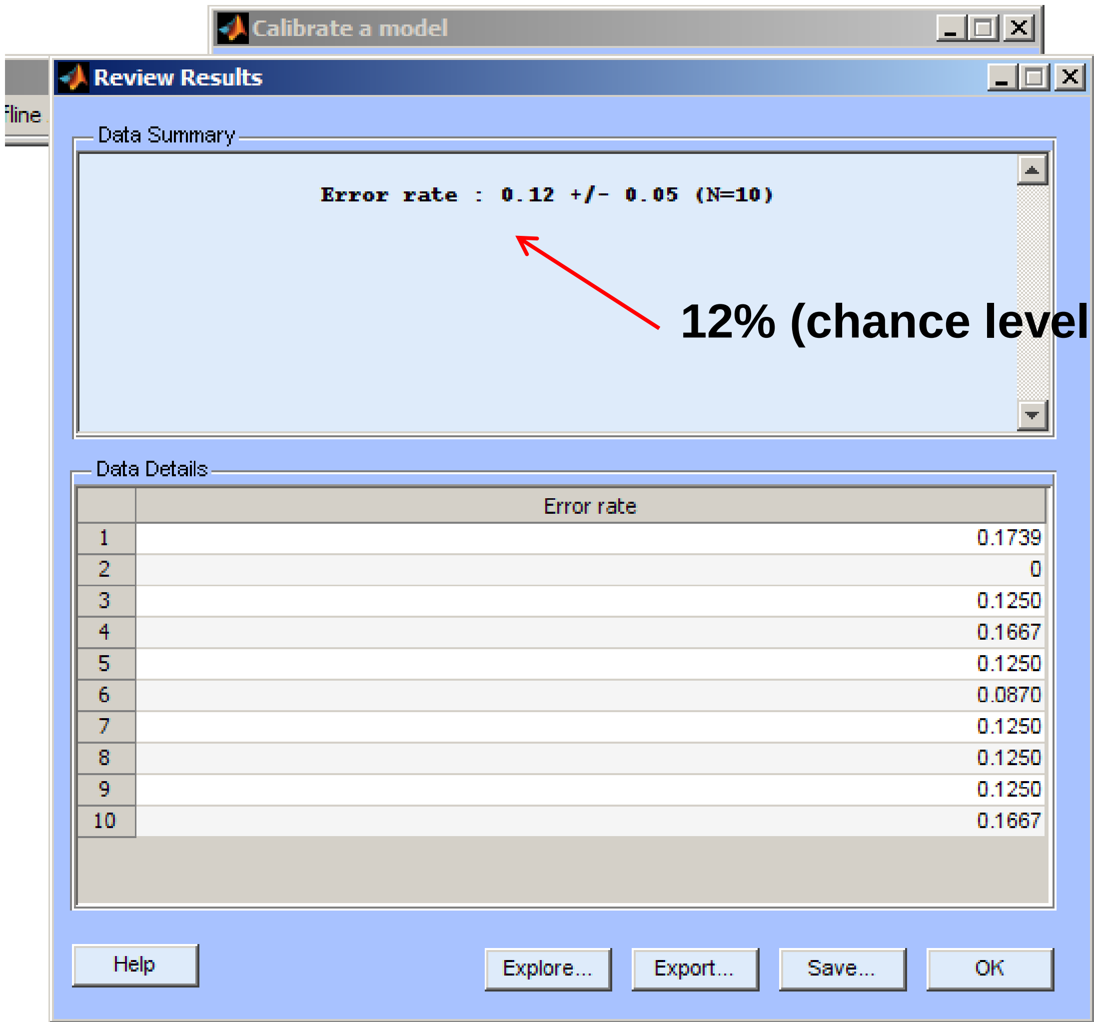


Major customizations



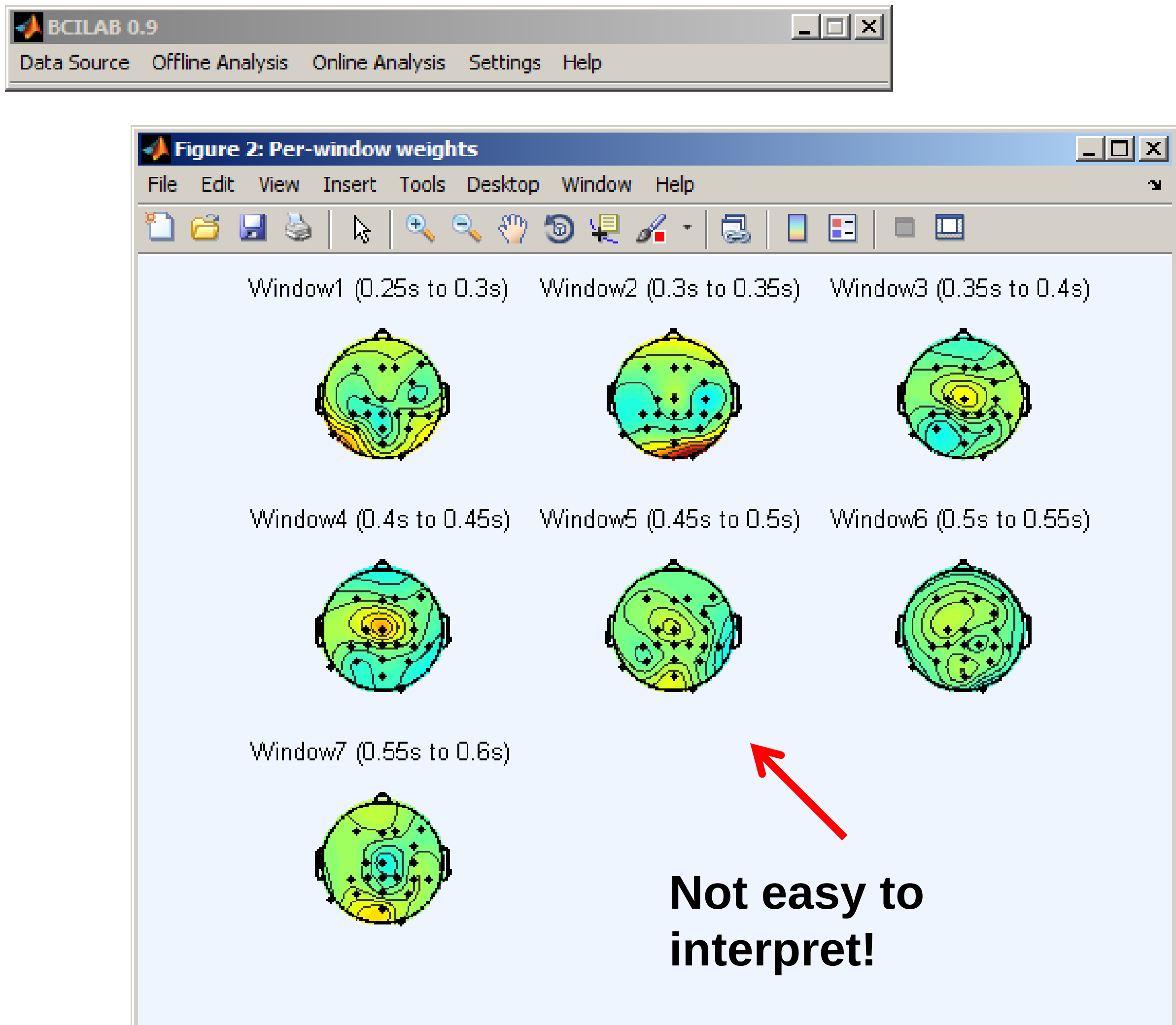


Train model, visualize

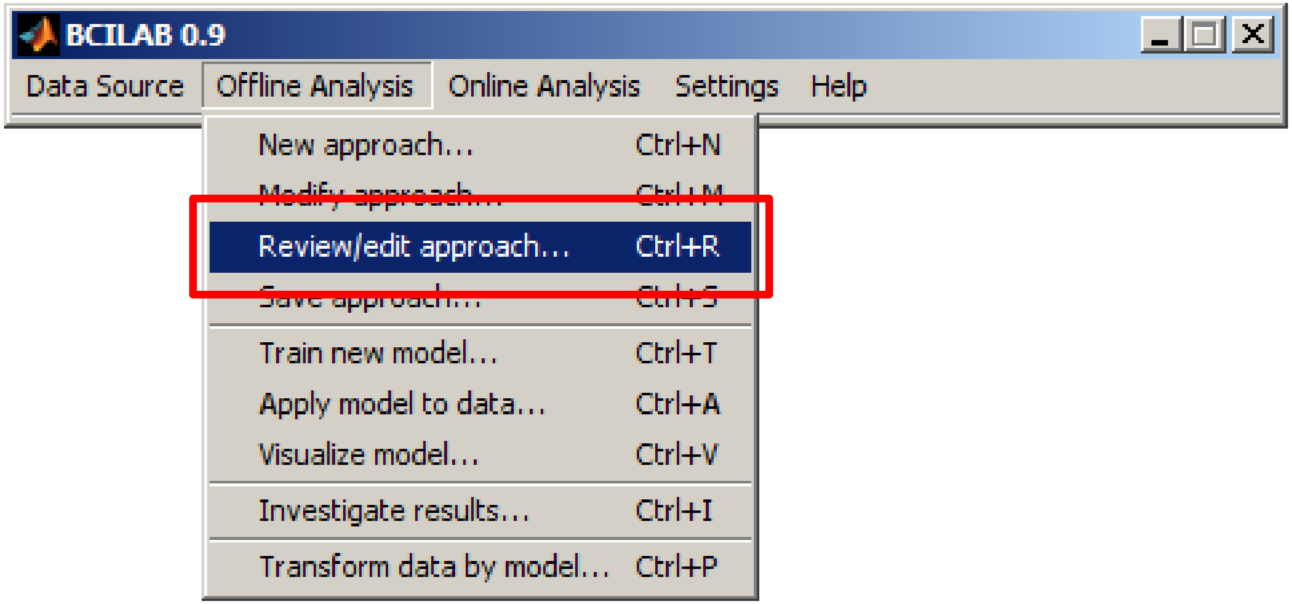


12% (chance level 33%)

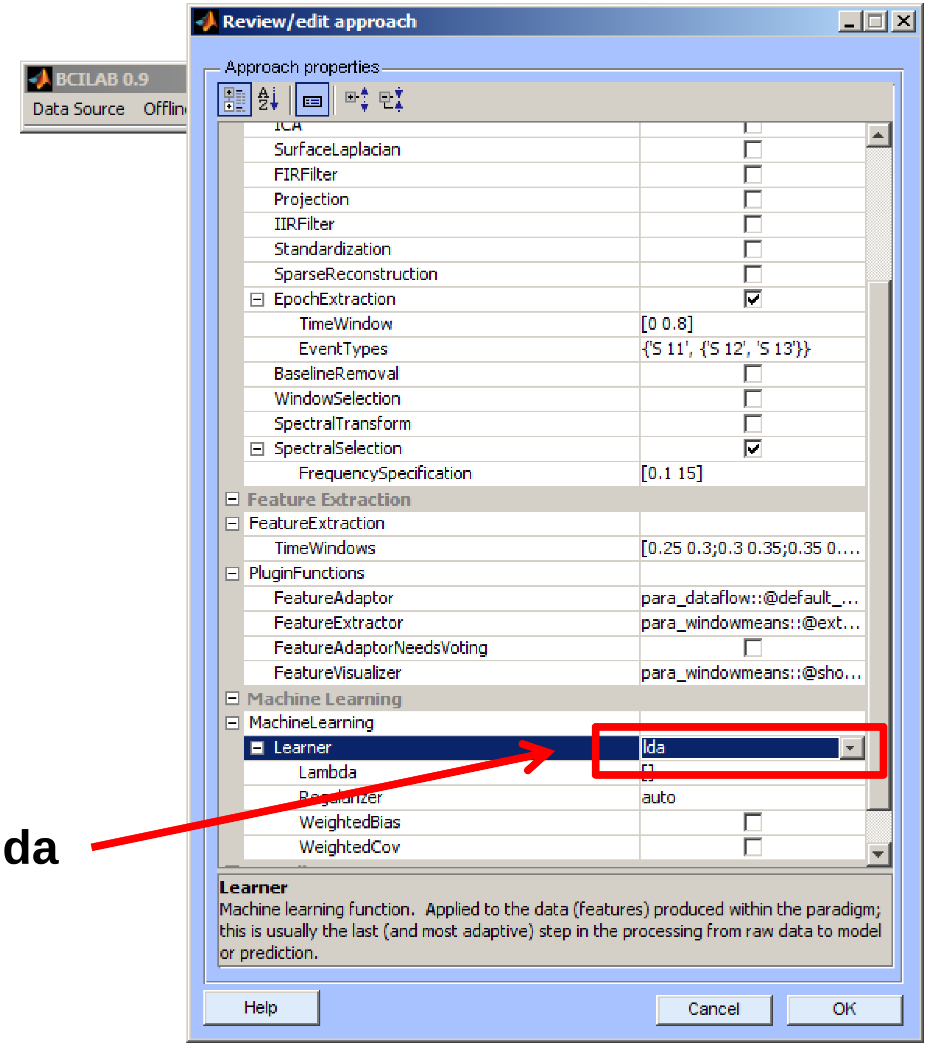
Train model, visualize



Using a Sparse Classifier



Using a Sparse Classifier

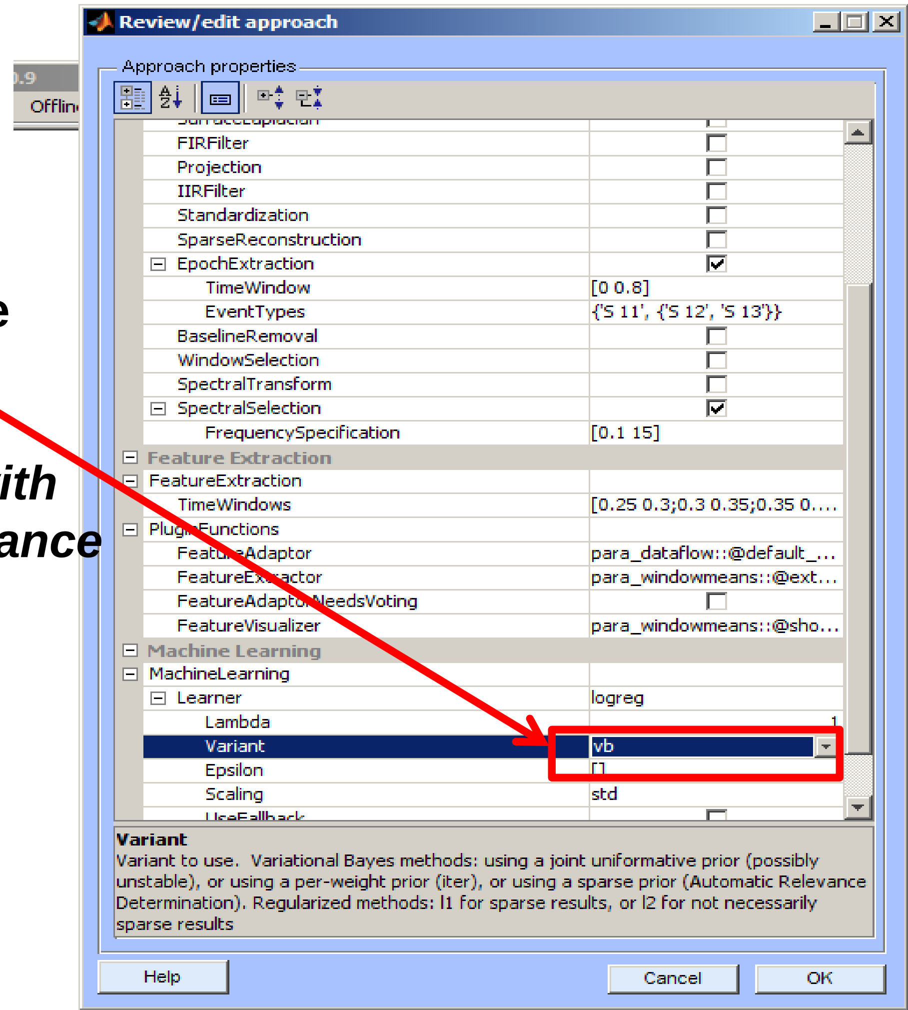


Selecting logreg instead of lda (for logistic regression)

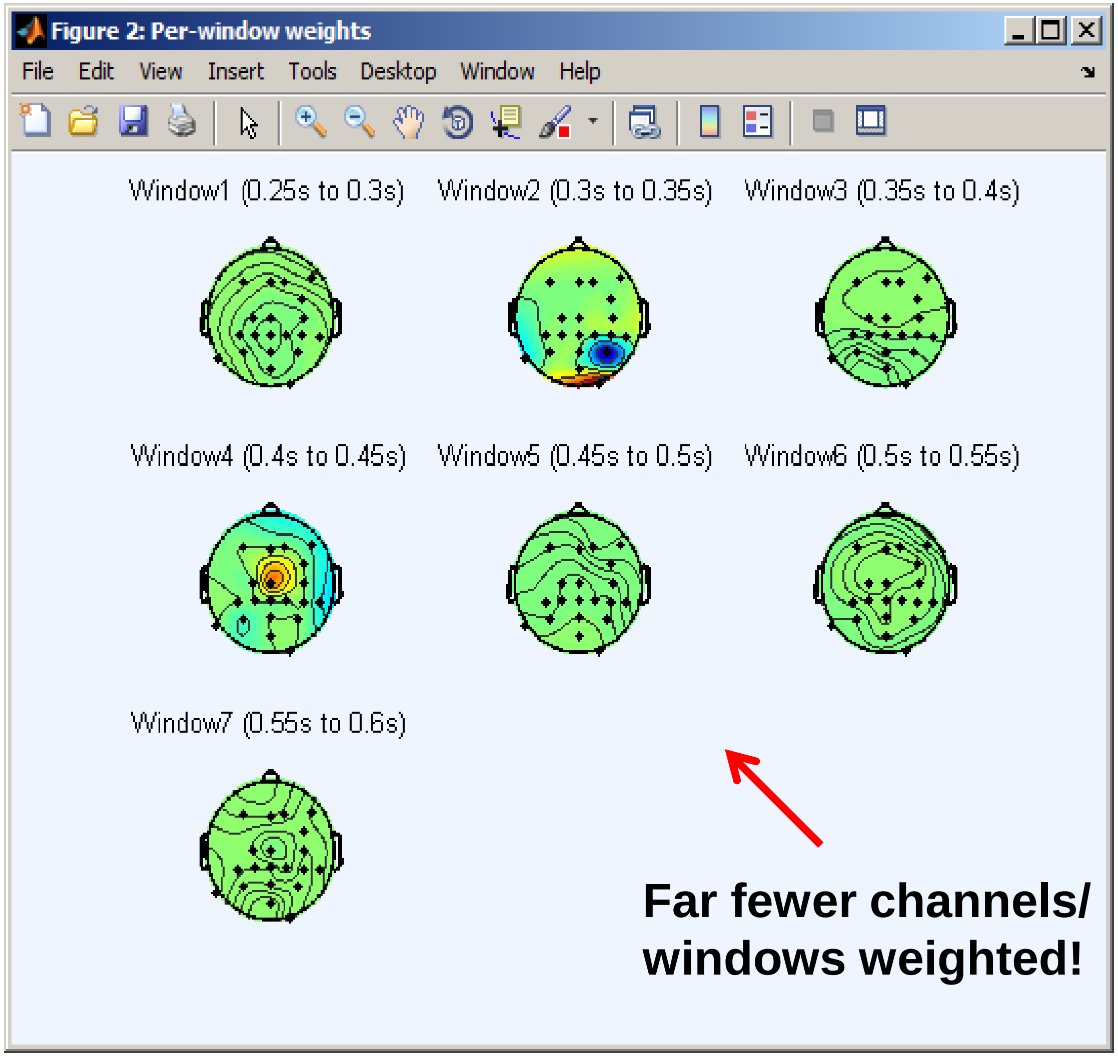
Using a Sparse Classifier

Selecting vb-ard Instead of vb for the “Variant” field

(yields “*sparse logistic regression with variational Bayesian automatic relevance determination*” as the classifier)



Training, Visualizing



Training, Visualizing

- Sparse classifiers can give more robust models (fewer channels / sources of errors used), and more interpretable models (only the most relevant features retained)

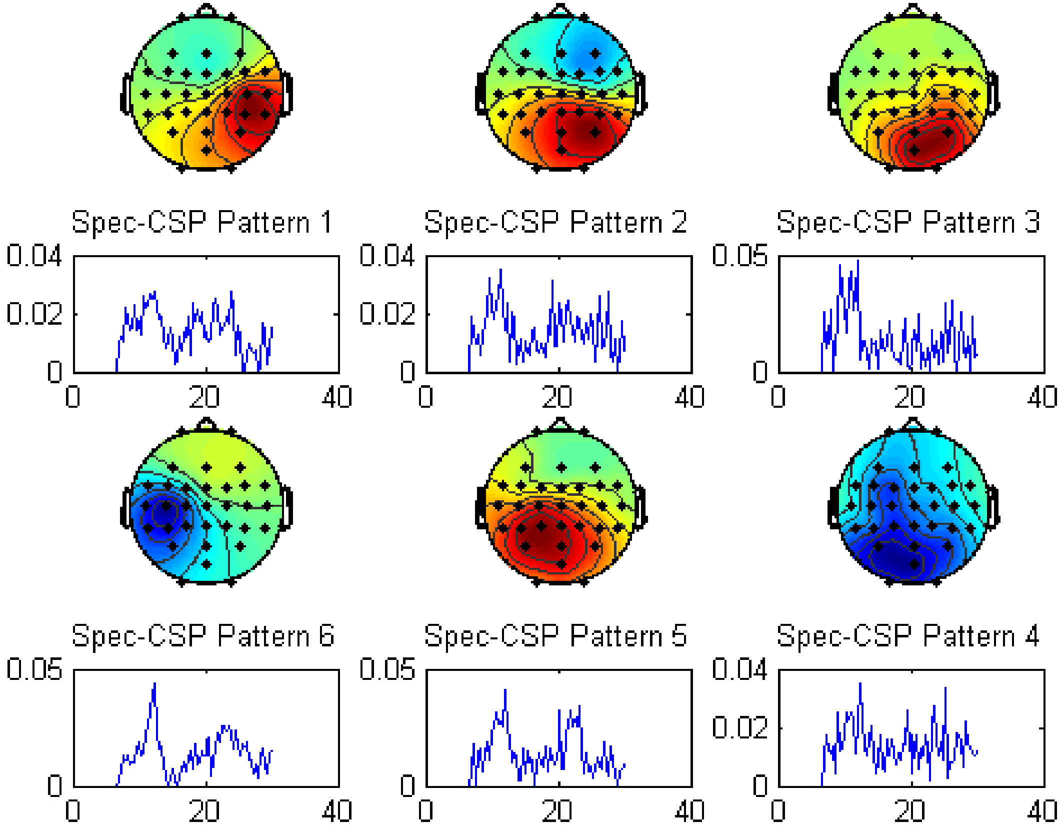
Next: Basic Scripting Tour

Scripting Examples

- Applying a Spec-CSP approach as seen in the GUI:

```
% load the data set
traindata = io_loadset('data:/tutorial/imag_movements1/calib/DanielS001R01.dat');

% define the approach
myapproach = {'SpecCSP' 'SignalProcessing',{'EpochExtraction', ...
    {'TimeWindow',[0 3.5],'EventTypes',{'StimulusCode_2','StimulusCode_3'}}}};

% learn a predictive model
[trainloss,lastmodel,laststats] = bci_train({'data',traindata,'approach',myapproach}); %#ok<>

% visualize results
bci_visualize(lastmodel)
```

Visualization Output

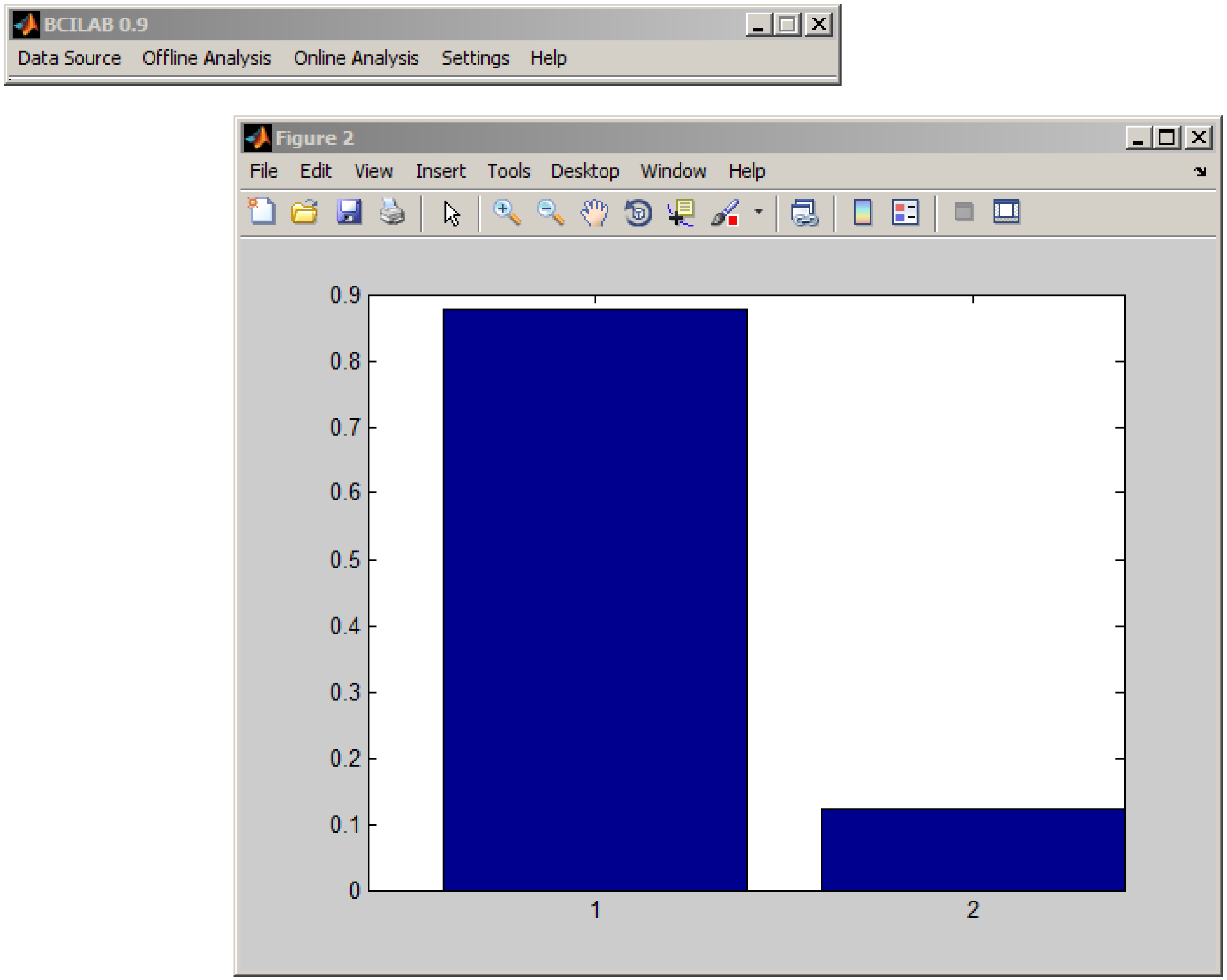


Scripting Examples

- Running the resulting model in real time on some data:

```
% load feedback session
testdata = io_loadset('data:/tutorial/imag_movements1/feedback/DanielS001R01.dat');

% play it back in real time
run_readdataset('Dataset',testdata);

% process data in real time using lastmodel, and visualize outputs
run_writevisualization('Model',lastmodel, 'VisFunction','bar(y)');
```

Script Output



Scripting Examples

- Doing a parameter search and nested cross-validation

```
%% --- train an alternative model with parameter search ---
% (over possible values for the number of pattern pairs, using CSP; note: this takes quite some time!)
% (the number of pattern pairs found optimal should be 3 in this case)

% load the data set (BCI2000 format)
traindata = io_loadset('data:/tutorial/imag_movements1/calib/DanielS001R01.dat');

% define approach
myapproach = {'CSP' ...
    'SignalProcessing',{'EpochExtraction',{'TimeWindow',[0 3.5], ...
                         'EventTypes',{'StimulusCode_2','StimulusCode_3'}}}, ...
    'FeatureExtraction',{'PatternPairs',search(1,2,3)}};

% learn model; here, using only a 5x outer cross-validation as it is otherwise too slow
[trainloss,lastmodel,laststats] = bci_train({'data',traindata,'approach',myapproach, ...
    'eval_scheme',{'chron',5,5}});

% visualize results
bci_visualize(lastmodel);
```

Search over different alternatives

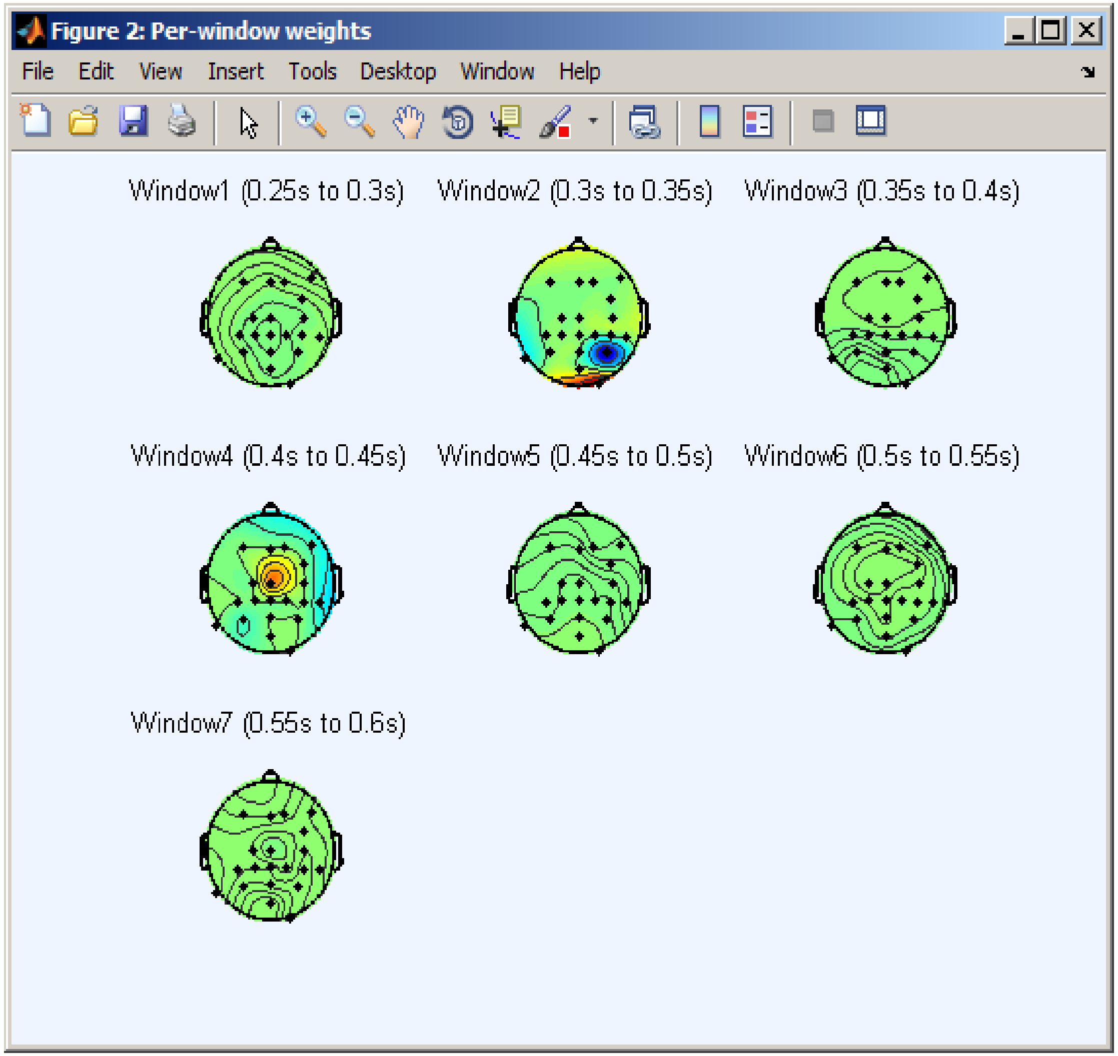
Also: Custom cross-validation scheme

Scripting Examples

- Running the advanced ERP analysis (with sparse classifier):

```
% define markers; here, two groups of markers are being defined; the first group represents class 1
% (correct responses), and the second group represents class 2 (incorrect responses).
mrks = {{'S101','S102'},{'S201','S202'}};

% define ERP windows of interest; here, 7 consecutive windows of 50ms length each are being
% specified, starting from 250ms after the subject response
wnds = [0.25 0.3;0.3 0.35;0.35 0.4; 0.4 0.45;0.45 0.5;0.5 0.55;0.55 0.6];

% define load training data (BrainVision format)
traindata = io_loadset('data:/tutorial/flanker_task/12-08-001_ERN.vhdr');

% define approach
myapproach = {'Windowmeans' ...
    'SignalProcessing', {'EpochExtraction',{'TimeWindow',[0 0.8],'EventTypes',mrks}, 'SpectralSelection',[0.1 15]}, ...
    'FeatureExtraction',{'TimeWindows',wnds}, ...
    'MachineLearning',{'Learner', {'logreg', [],'Variant','vb-ard'}}};

%learn model
[trainloss,lastmodel,laststats] = bci_train({'data',traindata,'approach',myapproach});

% visualize results
bci_visualize(lastmodel)
```

Visualization Output



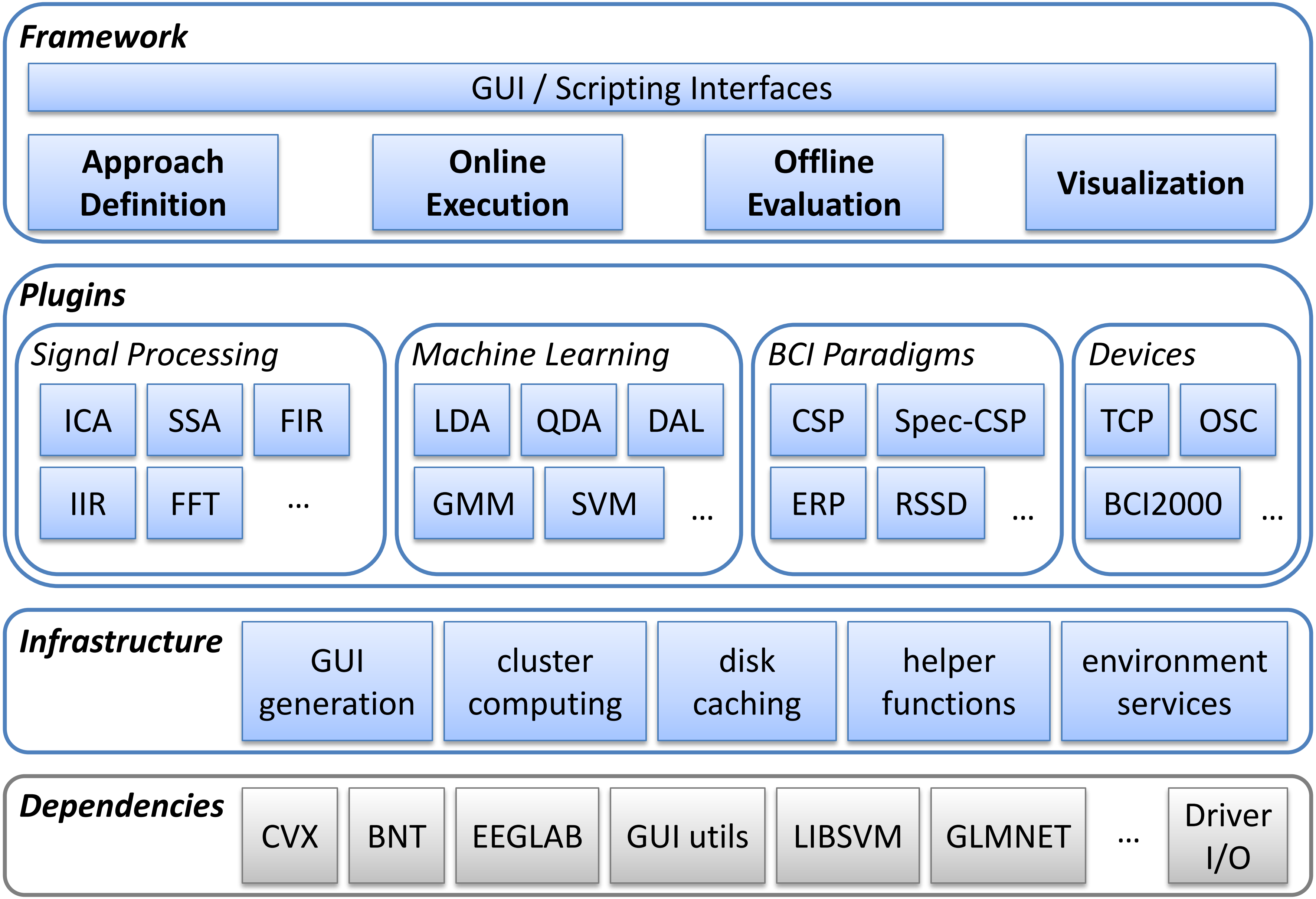
Scripting Examples

- Running a batch analysis for 3 modern approaches and 136 data sets (upcoming version only):

```
% define markers; here, two groups of markers are being defined; the first group represents class 1
% (correct responses), and the second group represents class 2 (incorrect responses).
mrks = {{'S101','S102'},{'S201','S202'}};
wnds = [0.25 0.3;0.3 0.35;0.35 0.4; 0.4 0.45;0.45 0.5;0.5 0.55;0.55 0.6];

% define approaches
approaches.wmeans_lda = {'Windowmeans' 'flt',{'events',mrks,'epoch',[0 0.8],'spectrum',[0.1 15]},'fex',{'wnds',wnds}};
approaches.wavelet_lars = {'Dataflow' 'flt',{'events',mrks,'epoch',[0 0.8],'spectrum',[0.1 15],'wavelet','on'},...
    'ml',{'learner',{'logreg',[],'variant','lars'}}};
approaches.dal = {'DAL_Lofreq','SignalProcessing',{'Resampling',60,'IIRFilter','off','FIRFilter',[0.1 0.5 18 21], ...
    'EpochExtraction',{'EventTypes',mrks,'TimeWindow',[-0.2 0.65]}},'MachineLearning',{'Learner',{'dal',2.^(8:-0.125:1)}}};

% run a batch analysis...
results = bci_batchtrain('Datasets','/data/projects/grainne/ERN/*.vhdr','Approaches',approaches,'RetainExistingResults',true);
```

Sample Plugins

BCILAB Architecture



FFT Filter

```
function signal = flt_fft(varargin)
% Apply an FFT to each epoch of an epoched signal (Example).
% Signal = flt_fft(Signal, LogPower)
%
% This is example code to transform a signal into the power domain, or log-power domain. A
% fully-featured version of this is flt_fourier.
%
% In:
%   Signal :   Epoched data set to be processed
%
%   LogPower : whether to take the logarithm of the power (instead of the raw power) (default: false)
%
% Out:
%   Signal  :   processed data set
%
%                                 Christian Kothe, Swartz Center for Computational Neuroscience, UCSD
%                                 2011-01-19

if ~exp_beginfun('filter') return; end

% requires epoched data, works best on spatially filtered data
declare_properties('name','EpochedFFT', 'depends','set_makepos', 'follows',{'flt_project','flt_window'}, 'independent_channels',true);

% declare arguments
arg_define(varargin,...
    arg_norep({'signal','Signal'}), ...
    arg({'do_logpower','LogPower'}, false, [], 'Compute log-power. Taking the logarithm of the power in each frequency band is easier to

% apply FFT and cut mirror half of the resulting samples
tmp = fft(signal.data,[],2);
tmp = tmp(:,1:signal.pnts/2,:);

% take signal power or log(power)
if do_logpower
    signal.data = log(abs(tmp));
else
    signal.data = abs(tmp);
end

exp_endfun;
```

Kernel SVMs (via SVMperf)

```
arg_define([0 3],varargin, ...
    arg_norep('trials'), ...
    arg_norep('targets'), ...
    arg({'cost','Cost'}, search(2.^(-5:2:15)), [], 'Regularization parameter. Reasonable range: 2.^(-5:2:15), greater is stronger. By default, it is average
    arg({'ptype','Type'}, 'classification', {'classification','regression','ranking'}, 'Type of problem to solve.','cat','Core Parameters'), ...
    arg({'kernel','Kernel'}, 'rbf', {'linear','rbf','poly','sigmoid','user'}, 'Kernel type. Linear, or Non-linear kernel types: Radial Basis Functions (gene
    arg({'g','RBFScale','gamma'}, search(2.^(-16:2:4)), [], 'Scaling parameter of the RBF kernel. Should match the size of structures in the data; A reasonal
    arg({'d','PolyDegree'}, uint32(3), [], 'Degree for the polynomial kernel.','cat','Core Parameters'), ...
    arg({'etube','EpsilonTube','tube'}, 0.1, [], 'Epsilon tube width for regression.','cat','Core Parameters'), ...
    arg({'rbalance','CostBalance','balance'}, 1, [], 'Relative cost of per-class errors. The factor by which training errors on positive examples outweight
    ...
    arg({'s','SigmoidPolyScale'}, 1, [], 'Scale of sigmoid/polynomial kernel.','cat','Miscellaneous'), ...
    arg({'r','SigmoidPolyBias'}, 1, [], 'Bias of sigmoid/polynomial kernel.','cat','Miscellaneous'), ...
    arg({'u','UserParameter'}, '1', [], 'User-defined kernel parameter.','cat','Miscellaneous','type','char','shape','row'), ...
    arg({'bias','Bias'}, false, [], 'Include a bias term. Only implemented for linear kernel.','cat','Miscellaneous'), ...
    arg({'scaling','Scaling'}, 'std', {'none','center','std','minmax','whiten'}, 'Pre-scaling of the data. For the regulariation to work best, the features
    arg({'clean','CleanUp'}, false, [], 'Remove inconsistent training examples.','cat','Miscellaneous'), ...
    arg({'epsi','Epsilon','eps'}, 0.1, [], 'Tolerated solution accuracy.','cat','Miscellaneous'), ...
    arg({'verbose','Verbose'}, false, [], 'Show diagnostic output.','cat','Miscellaneous'));

if is_search(cost)
    cost = 1; end
if is_search(g)
    g = 0.3; end

% find the class labels
classes = unique(targets);
if length(classes) > 2
    % in this case we use the voter
    model = ml_trainvote(trials,targets,'1v1',@ml_trainsvmlight,@ml_predictsvmlight,varargin{:});
else
    % scale the data
    sc_info = hlp_findscaling(trials,scaling);
    trials = hlp_applyscaling(trials,sc_info);

    % remap target labels to -1,+1
    targets(targets==classes(1)) = -1;
    targets(targets==classes(2)) = +1;

    % rewrite sme string args to numbers
    ptype = hlp_rewrite(ptype,'classification','c','regression','r','ranking','p'); %#ok<*NODEF>
    kernel = hlp_rewrite(kernel,'linear',0,'poly',1,'rbf',2,'sigmoid',3,'user',4);

    % build the arguments
    args = sprintf('-z %s -c %f -v %d -w %f -j %f, -b %d -i %d -e %f -t %d -d %d -g %f -s %f -r %f -u %s', ...
        ptype,cost,verbose,etube,rbalance,bias,clean,epsi,kernel,d,g,s,r,u);

    % run the command
    model = svmlearn(trials,targets,args);
    model.sc_info = sc_info;
    model.classes = classes;
end
```

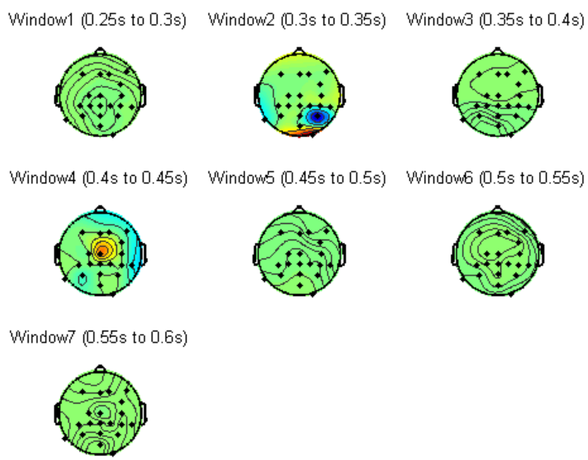
CSP Paradigm

```
classdef ParadigmCSP < ParadigmDataflowSimplified
    methods

        function defaults = preprocessing_defaults(self)
            defaults = {'FIRFilter',{'Frequencies',[6 8 28 32],'Type','minimum-phase'}, 'EpochExtraction',[0.5 3.5], 'Resampling',10
        end

        function [model,needsvoting] = feature_adapt(self,varargin)
            arg_define(varargin, ...
                arg_norep('signal'), ...
                arg({'patterns','PatternPairs'},3,[],'Number of CSP patterns (times two).','cat','Feature Extraction','type','expres

            if signal.nbchan < patterns
                error('CSP requires at least as many channels as you request output patterns. Please reduce the number of pattern pa
            for k=1:2
                trials{k} = exp_eval(set_picktrials(signal,'rank',k),1);
                covar{k} = cov(reshape(trials{k}.data,size(trials{k}.data,1),[])');
                covar{k}(~isfinite(covar{k})) = 0;
            end
            [V,D] = eig(covar{1},covar{1}+covar{2});
            model.filters = V(:,[1:patterns end-patterns+1:end]);
            P = inv(V);
            model.patterns = P([1:patterns end-patterns+1:end],:);
            model.chanlocs = signal.chanlocs;
            needsvoting = true;
        end

        function features = feature_extract(self,signal,featuremodel)
            features = zeros(size(signal.data,3),size(featuremodel.filters,2));
            for t=1:size(signal.data,3)
                features(t,:) = log(var(signal.data(:,:,t)'*featuremodel.filters)); end
        end

        function layout = dialog_layout_defaults(self)
            layout = {'flt.srate','flt.epoch','flt.fir.fspec','flt.fir.ftype',[],'pred.fad.patterns',[],'pred.ml.learner'};
        end
    end
end
```

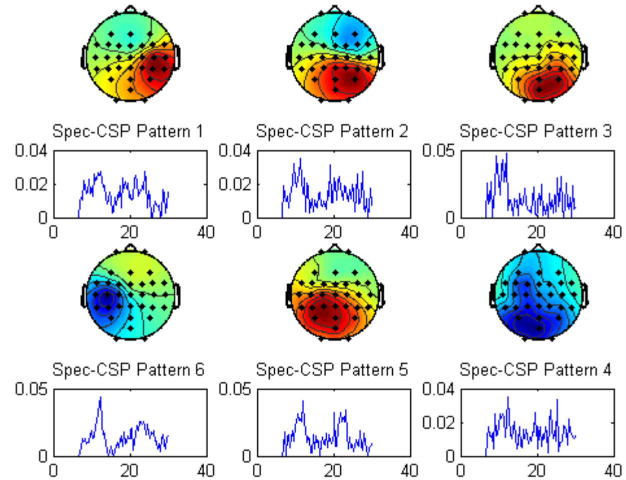
Ongoing Work

- Better domain-specific assumptions in BCI approaches (moving beyond off-the-shelf components)
  - e.g., expressed as general convex optimization, Bayesian inference
- Integration of (quantitative) prior knowledge
  - Anatomical (or even functional) priors from brain atlases, etc.
- Integration of larger data sources
  - Multiple recordings, multiple subjects, …
- Better exploitation of multiple modalities
  - Hybrid BCIs, general cognitive state assessment, …
- More hardware devices! ☺

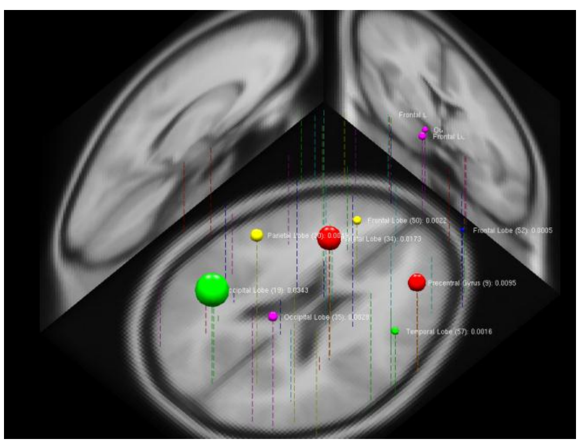
Teaser: Some Model Types



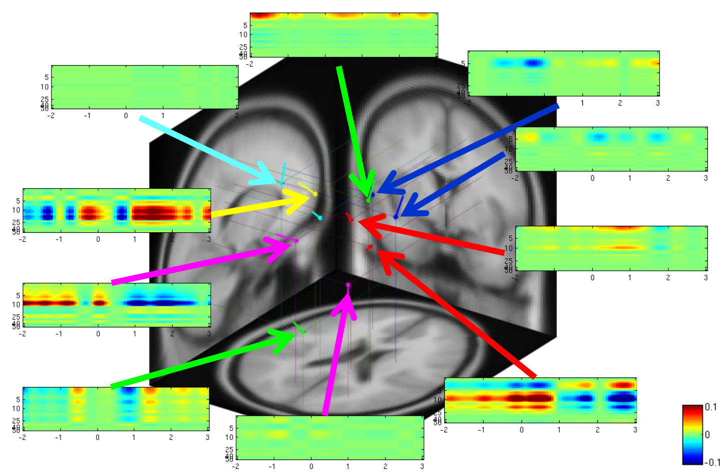
VB-ARD (on ERPs)



Spec-CSP



OSR



RSSD

Thanks!

Questions?

http://sccn.ucsd.edu/**wiki/BCILAB**