Scripting Prerequisites

Function Calling Syntax

- Most functions take their arguments in the order in which they are listed in the documentation
- Some can *alternatively* called with all parameters passed in as name-value pairs (using the same names as in the help text, in CamelCase)
- If in doubt, pass them in by name – less chance of getting the order wrong, etc.
- It is usually a bad idea to try to mix positional and name-value arguments in one call – don't do it unless that's the default way to call the function
- **Example:**

```
bci_train(mydata,myapproach)
bci_train('Data',mydata,'Approach',myapproach)
```

Loading Data

- A data set (no matter what file format) is loaded using the function io_loadset()
- It is almost always enough pass in just the file name, as in the example:
  data = io_loadset('/somepath/somefile.xyz')

Defining an Approach

Defining a new Approach

- Defining an approach is the most complex area in scripting because a data structure must be constructed
- Since an approach is a particular instance of a BCI paradigm (used with custom parameters), an approach definition consists of:
  - The name of the paradigm (e.g., CSP, WindowMeans)
  - Optionally a list of arguments for the paradigm's calibrate() function
- The default way to specify an approach is as a cell array whose first element is the name of the paradigm and whose remaining elements are arguments to its calibrate() function
- **Example:**

```
appr = {'CSP','SignalProcessing',...,'FeatureExtraction',...};
```

Approach Parameters

- The parameters are a list of name-value pairs
- **Important:** The argument of an approach are not passed in a long ‘flat’ list, but they are organized in a hierarchy, i.e. some parameters have *named sub-parameters*
- **Example:**

```
app = {‘CSP’,’Prediction’,{‘MachineLearning’, …}};
```

MachineLearning is a sub-parameter of Prediction

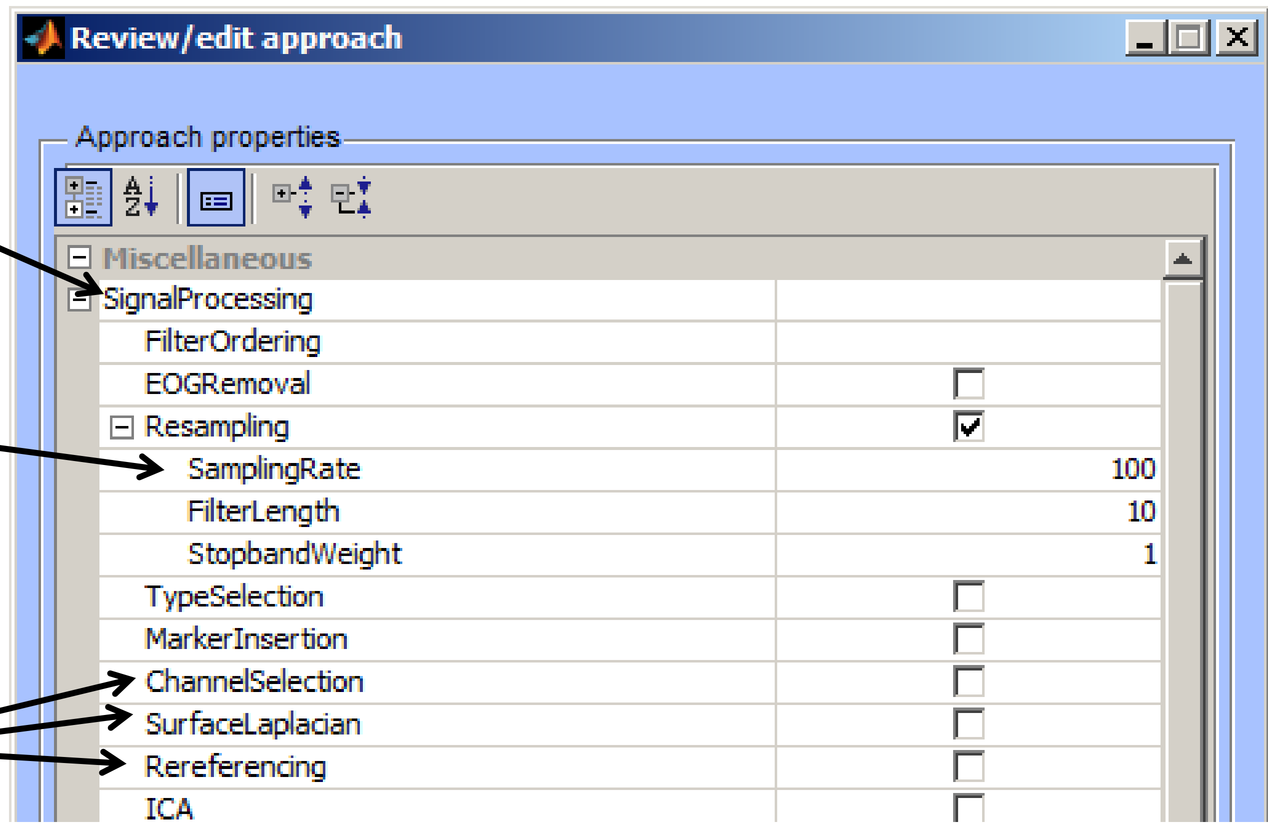
Prediction is a “top-level” parameter

Approach Parameters

- Which parameter names a BCI paradigm exposes is the business of the BCI paradigm
- However, practically all of them adhere to a uniform scheme of 2 top-level parameter names:
  - **SignalProcessing** is a top-level parameter that determines the signal processing stages that shall be used
  - **Prediction** is a top-level parameter that governs how the prediction function is being calibrated or applied

Correspondence With The GUI

- There is a 1:1 correspondence between the hierarchy of parameters that are specified in scripts and the layout of the parameter tree in the approach definition GUI



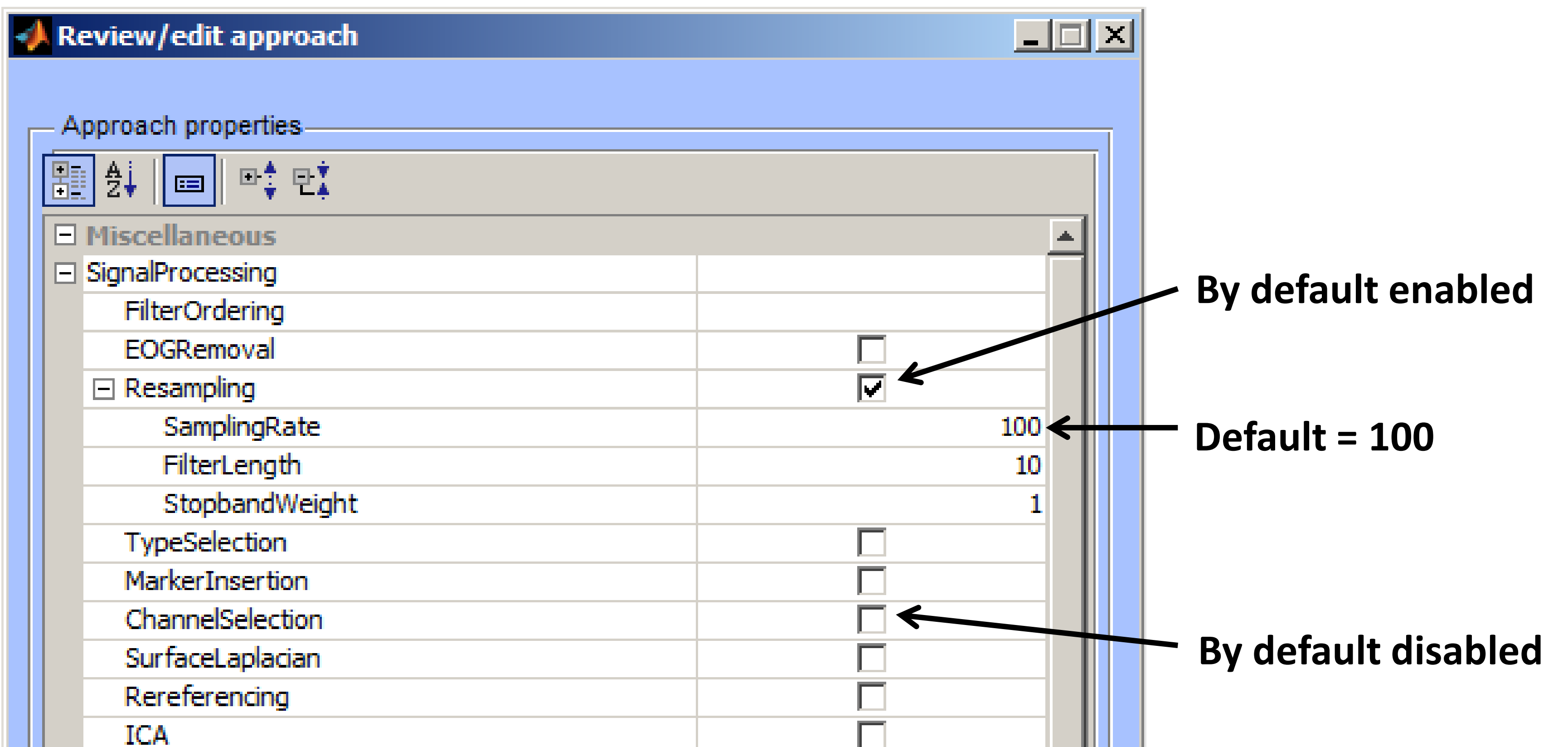
The SignalProcessing parameter

Sub-Parameter of Resampling (itself a sub-parameter of SignalProcessing)

Sub-parameters of SignalProcessing

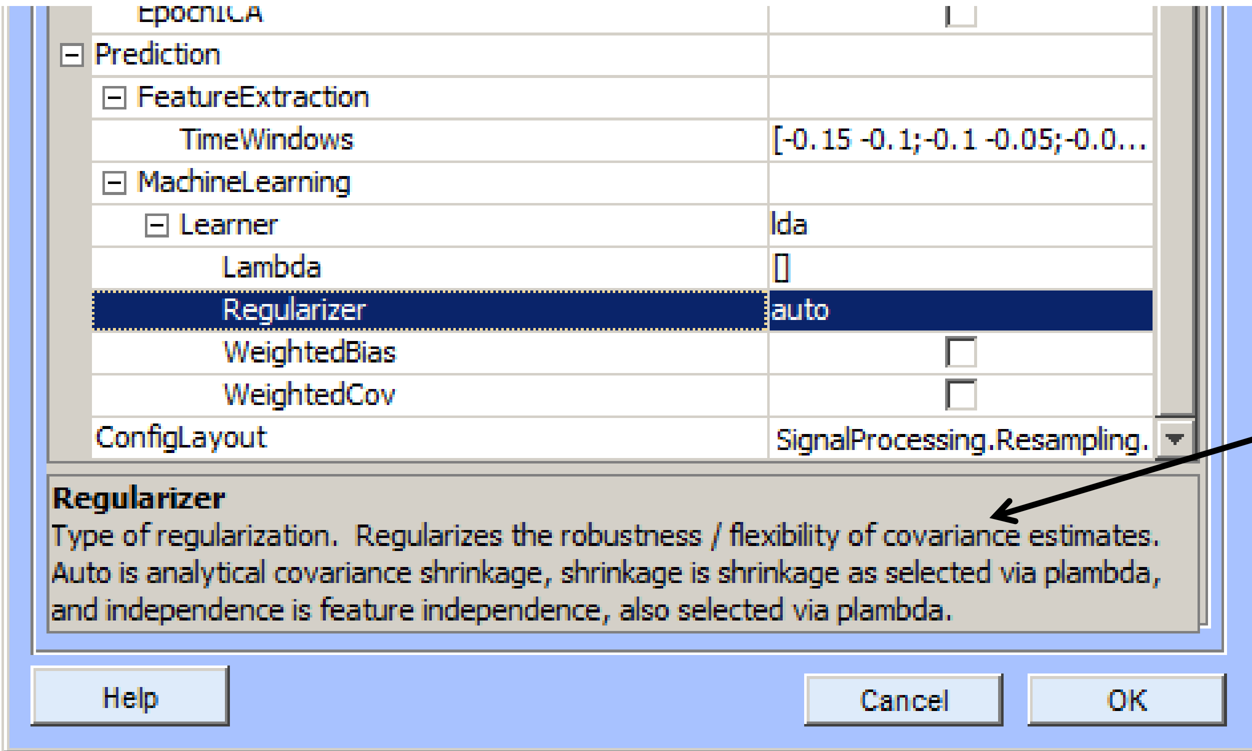
Default Values

- Each parameter has a default value (unless it makes *absolutely no sense*), which can also be looked up in the GUI



Parameter Help

- Each parameter has a help text, which is also visible in the GUI panel (at the bottom)



Help text for the selected parameter

The SignalProcessing Parameter

- Has one named sub-parameter for every signal processing plugin that can be used (these are found automatically)
- The name under which a given signal processing plugin appears is up to the plugin – they declare this property at the beginning of their code (you can look it up there)

```
% See also:
%   firpm, filter
%
%                                   Christian K
%                                   2010-04-17

if ~exp_beginfun('filter') return; end

declare_properties('name','FIRFilter', 'foll
```

Name of the sub-parameter as which this plugin shows up in the approach definition (below SignalProcessing)

The SignalProcessing Parameter

- The plugins that are listed under SignalProcessing are those in the directories:
  - code/filters (file names beginning with flt_)
  - code/dataset_editing (file names beginning with set_)
- The value assigned to a sub-parameter (e.g., FIRFilter) that is presented by a function (e.g., flt_fir.m) is by default a cell array of arguments to that function
- The arguments can be passed in any format accepted by the function, but preferably they should again be passed as name-value pairs to avoid confusion

Configuring Signal Processing Stages

- **Example:**

```
app={‘CSP’,’SignalProcessing’, ...
  {‘FIRFilter’,{‘Frequencies’,[7 8 14 15]}}};
```

- This example defines a CSP-based approach that uses a particular Frequencies value in its FIR filter
- The FIR filter is now also “enabled” if it was not before

Disabling Signal Processing Stages

- It is sometimes useful to disable a parameter that is enabled by default: This can be written (by convention) as follows:

```
app={'CSP','SignalProcessing',{'Resampling',[]};
```

- Note that these are [] brackets – using {} accidentally would still enable the filter, but passes an empty argument list to it!

Shortcuts for the Impatient

- BCILAB has the unhealthy habit of allowing *short forms for most things* – I recommend to avoid them whenever possible, but it helps recognizing them
- The most salient short-cut form is when a parameter that has sub-parameters is not assigned a cell array of arguments (like it should), but instead directly the value of the first sub-argument
- **Example:**

```
app={'CSP','SignalProcessing',{'Resampling',200}};
```

This number is assigned to the first sub-argument of the resampling filter (=the target sampling rate)

Shortcuts for the Impatient

- BCILAB has the unhealthy habit of allowing *short forms for most things* – I recommend to avoid them whenever possible, but it helps recognizing them
- The most salient short-cut form is when a parameter that has sub-parameters is not assigned a cell array of arguments (like it should), but instead directly the value of the first sub-argument
- **Example:**

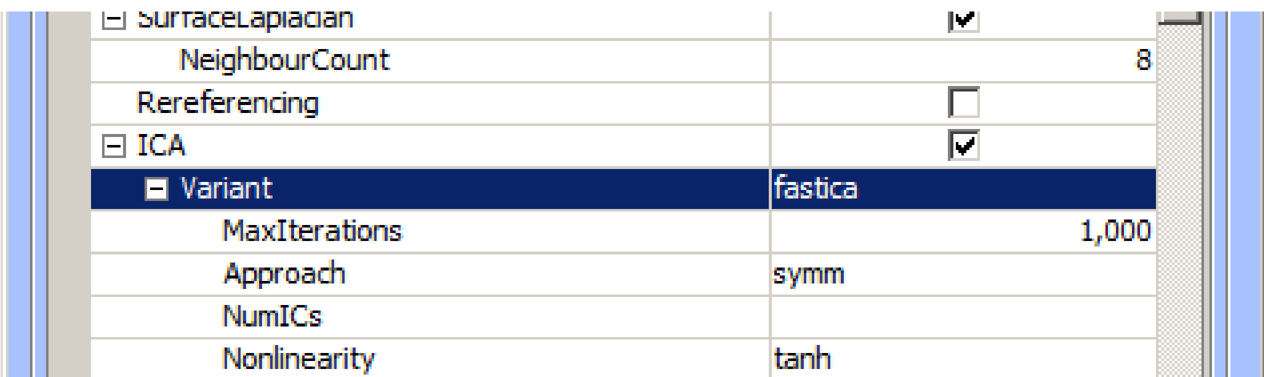
```
app={'CSP','SignalProcessing',{'Resampling',200}};
```

- **… is equivalent to:**

```
app={'CSP','SignalProcessing',...
  {'Resampling',{'SamplingRate',200}}};
```

Multi-Option Parameters

- The last kind of parameter that deserves mention are multi-option parameters, which consists of a *selection* argument (a string) and for each possible value a different list of sub-arguments
- An example are the different alternative variants supported by the ICA filter: amica, infomax, etc., all of which have algorithm-specific sub-arguments
- Below, the parameter named Variant is set to ‘fastica’, and the MaxIterations sub-parameter of Variant for the *fastica case* is set to 1000

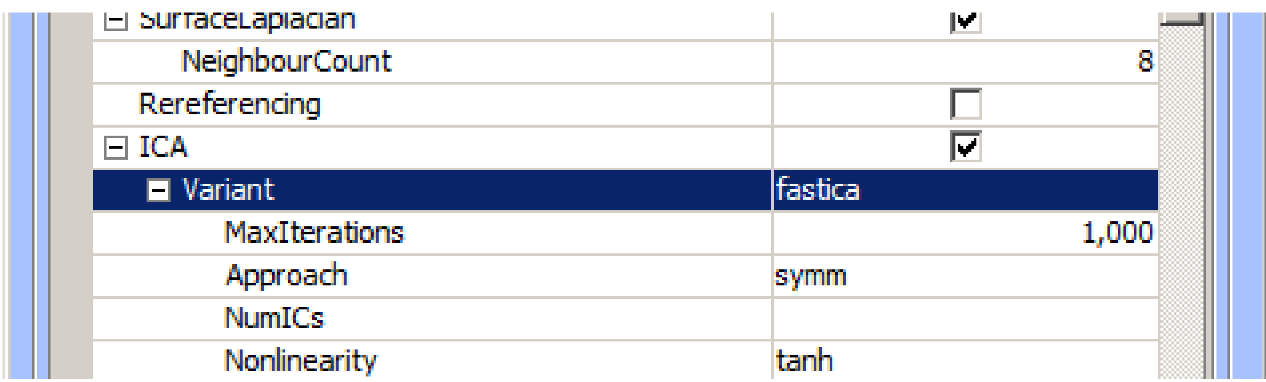


Multi-Option Parameters

- In scripts, multi-option parameters are written just like the overall approach definition: as a cell array whose first element is the name of the selection followed by name-value pairs for this case
- **Example:**

```
…,'Variant',{'fastica','MaxIterations',1000,'Approach','symm'}
```

- **… is equivalent to setting what is shown here in the GUI:**



Other Paradigm Parameters

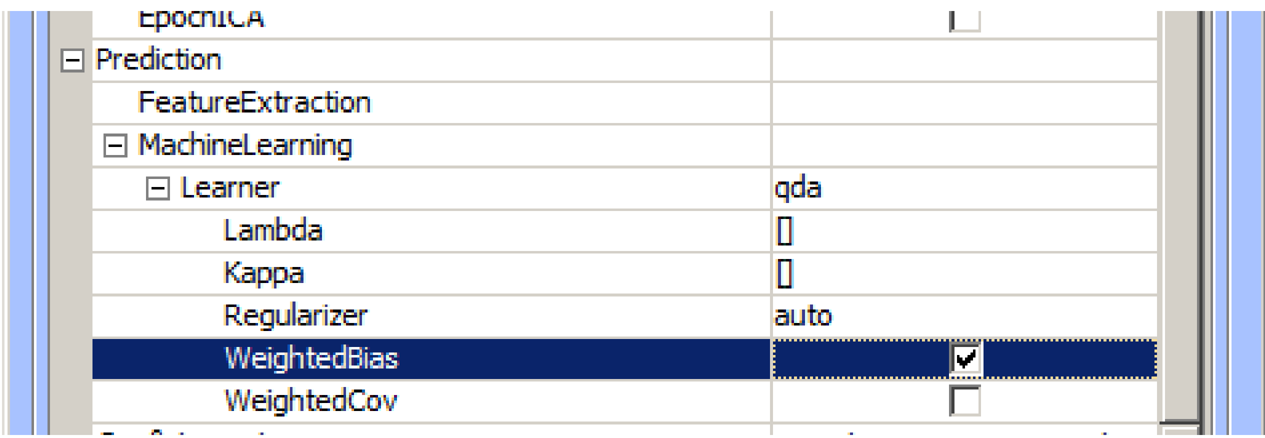
- The other parameters behave in exactly the same ways
- **Example:**
  - MachineLearning is a sub-parameter of Prediction, it has a Learner sub-parameter
  - Learner is a multi-option parameter with one case for each machine learning plugin (e.g., ‘lda’, ’qda’, ’logreg’, …)
  - The sub-parameters of the respective case are those that are exposed by the respective plugin function (e.g., ml_trainqda.m)

Configuring the Machine Learning Stage

- Thus, the following is a valid way to configure the machine learning function of a paradigm:

```
app={‘CSP’, ’Prediction’,{‘MachineLearning’, …
     {‘Learner’,{‘qda’ ‘WeightedBias’,true}}}};
```

- It corresponds to the following GUI setting:



Configuring the Machine Learning Stage

- Thus, the following is a valid way to configure the machine learning function of a paradigm:

```
app={‘CSP’, ’Prediction’,{‘MachineLearning’, …
     {‘Learner’,{‘qda’ ‘WeightedBias’,true}}}};
```

- Alternative shortcut form:

```
app={‘CSP’, ’Prediction’,{‘MachineLearning’, …
     {‘Learner’,‘qda’}}};
```

Instead of at least {‘qda’}

Remaining Script Workflows

Calibrating ("Training") a Model

- A new BCI model is created using a previously loaded data set (the training set) and a previously defined approach
- This is done using the function bci_train (the equivalent of the "Train new model…" dialog)
- **Example:**

```
raw = io_loadset('imag.set')
app = {'SpecCSP', ... };
[loss,model,stats] = bci_train('Data',raw,'Approach',app, ...
    'TargetMarkers',{'S  1','S  2'});
```

Calibrating a Model

- The bci_train function usually takes 3 inputs:
  - The data (Data parameter)
  - The approach (Approach parameter)
  - The description of how event types map onto class labels (TargetMarkers, same as in the GUI)
- The function returns three outputs:
  - The overall loss estimate (e.g. error rate)
  - The learned model
  - Statistics about the model and training process, including results of a cross-validation

Visualizing a Model

- Models are visualized using the function bci_visualize
- **Example:** bci_visualize(mymodel)

- This function can take extra arguments that are passed on to the responsible drawing function (but few drawing functions have arguments)

Applying a Model to Test Data

- For *offline application* to test data, the function bci_predict can be used – it applies the BCI model to each trial in the data and calculates loss statistics
- **Example:**

```
[outputs,loss,stats] = ...
    bci_predict('Data',mydata,'Model',mymodel);
```

- **Note:** the first output are the model's predictions for each trial in the data

Annotating Data with Continuous BCI Outputs

- The BCI output can be attached as an extra channel (or multiple channels, each representing the probability of class k) to a data set, using the function bci_annotate
- **Example:**

```
newset = bci_annotate('Data',mydata,'Model',mymodel)
```

Reading Real-Time Data

- Real-time data can be acquired from a device and written into a named workspace variable using the online reader plugins (run_read* functions)
- **Examples:**

```
run_readbiosemi(); # read from a BioSemi device

run_readdataset('MatlabStream','mystream','Dataset',myset);
```

Swartz Center for Computational Neuroscience

Sending Real-Time Outputs

- The outputs of a BCI model as applied to some stream(s) can be calculated in the background online and passed on to some destination – this is done using the online writer plugins (run_write*)
- These functions take usually the name of the model to use and the name(s) of the stream(s) to use
- **Example:**

```
run_writevisualization('Model','mymodel', ...
    'SourceStream','mystream')
```

Performing Batch Analyses

- Using bci_batchtrain, a single approach can be efficiently applied to a list of data sets or file names
- Also multiple approaches can be applied to one or more data sets in an automated manner
- Can not just train models but also make predictions and evaluate losses on test data sets
- **Example:**

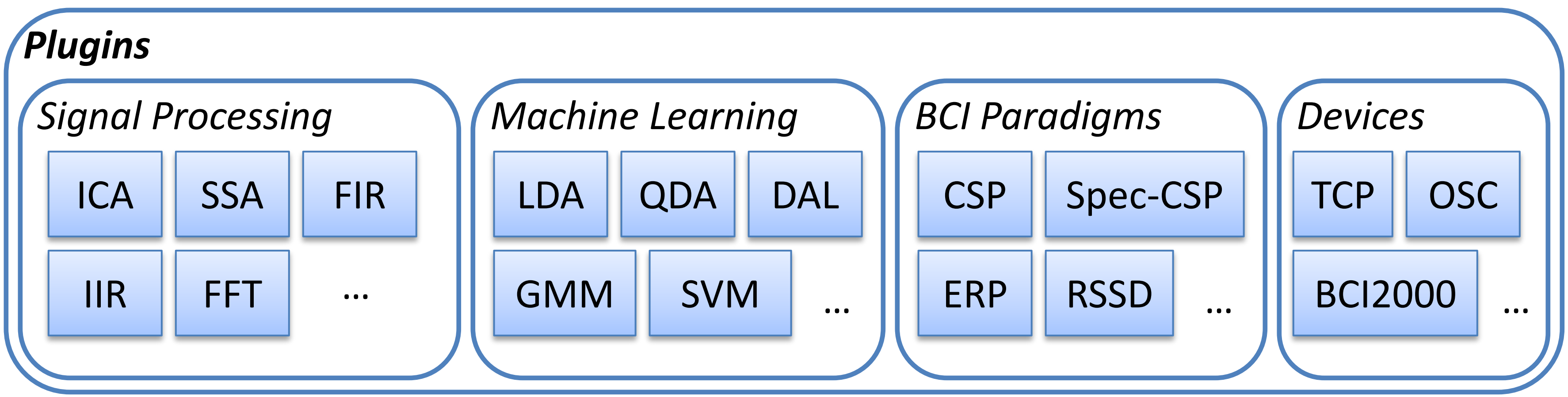
```
results = bci_batchtrain('Data',mydatasets, ...
    'Approaches',myapproaches,'TargetMarkers',mymarkers);
```

Parameter Searches

- It is possible to replace (practically) any value in an approach definition by a so-called "search range", i.e. a list of possible values to try automatically in a systematic manner
- A search range is specified by writing the expression search(value1, value2, …, valueN)
- Multiple search parameters in one approach lead to combinatorial grid search (slow!)
- **Example:**

```
app={'CSP','Prediction',{'FeatureExtraction',{ ...
    'PatternPairs',search(1,2,3)}}};
```

3 A Close Look at Components

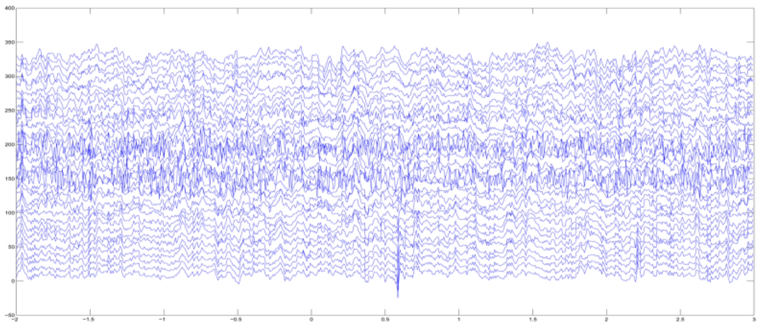


Component 1: Predictive Mapping

Central Predictive Mapping

- A BCI (with limited memory of the past) can be viewed as a mathematical function *f*:

y = f(**X**); **X**=



y= “subj. excited” (+1)
“subj. not excited” (-1)

- The functional form is arbitrary, for example

$$y = \mathrm{sign}(\mathrm{var}(\boldsymbol{W}\boldsymbol{X}) + b)$$

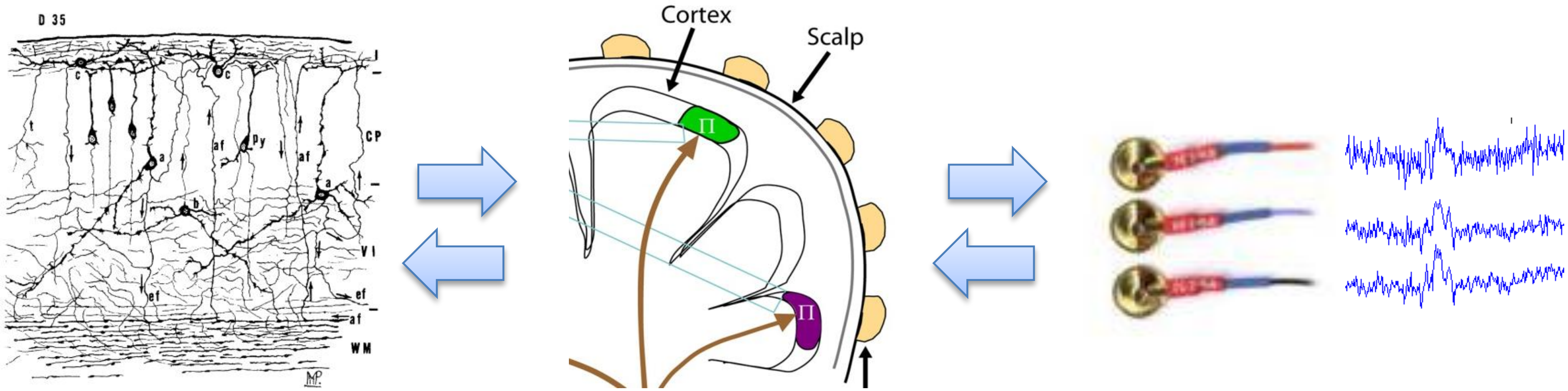
- The mapping involves free parameters, here **W** and b, and data from a *sliding window* **X**

Choice of a Functional Form

- Reflects the relationship between observation (data segment **X**) and desired output (cognitive state parameter y)

Choice of a Functional Form

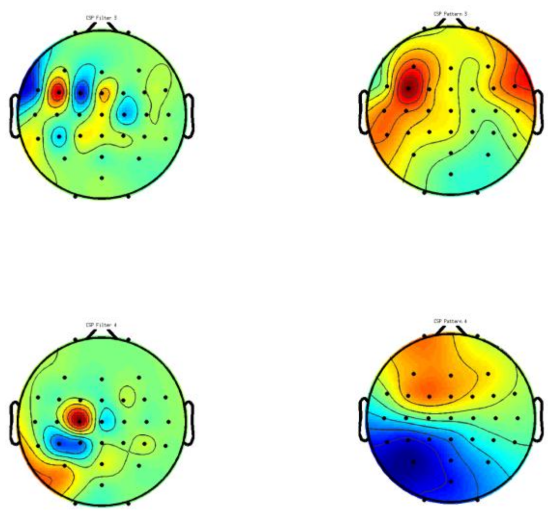
- Reflects the relationship between observation (data segment **X**) and desired output (cognitive state parameter y)
- Based on some assumed generative mechanism (forward model) – or ad hoc



- Remember: Functional form is the inverse mapping!

Key Ingredient: Spatial Filter

- Linear inverse of volume conduction effect between sources **S** and channels **X**
  $\boldsymbol{X} = \boldsymbol{AS}$ (forward)
  $\boldsymbol{S} = \boldsymbol{WX}$ (inverse)

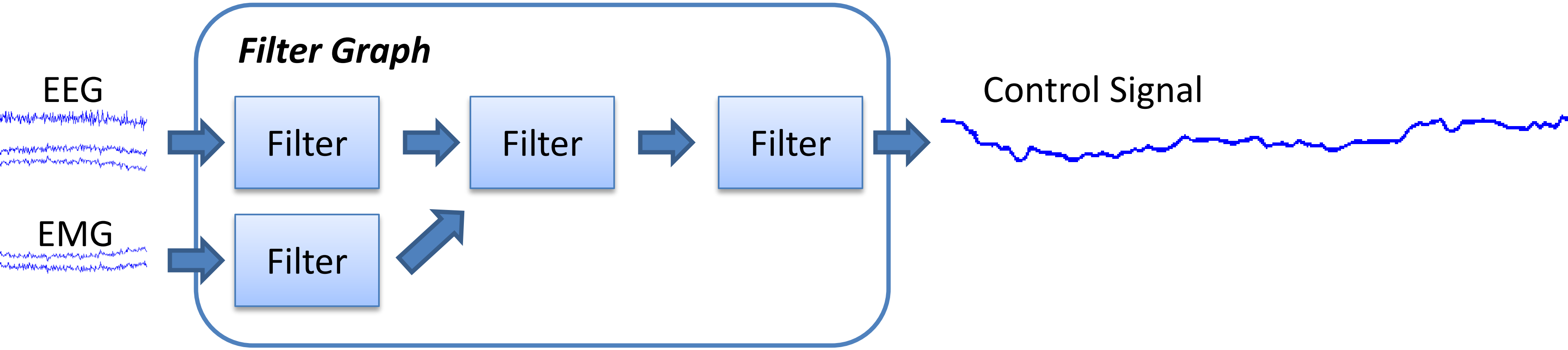


W **A=W^-1**

Component 2: Signal Processing

Role of Signal Processing

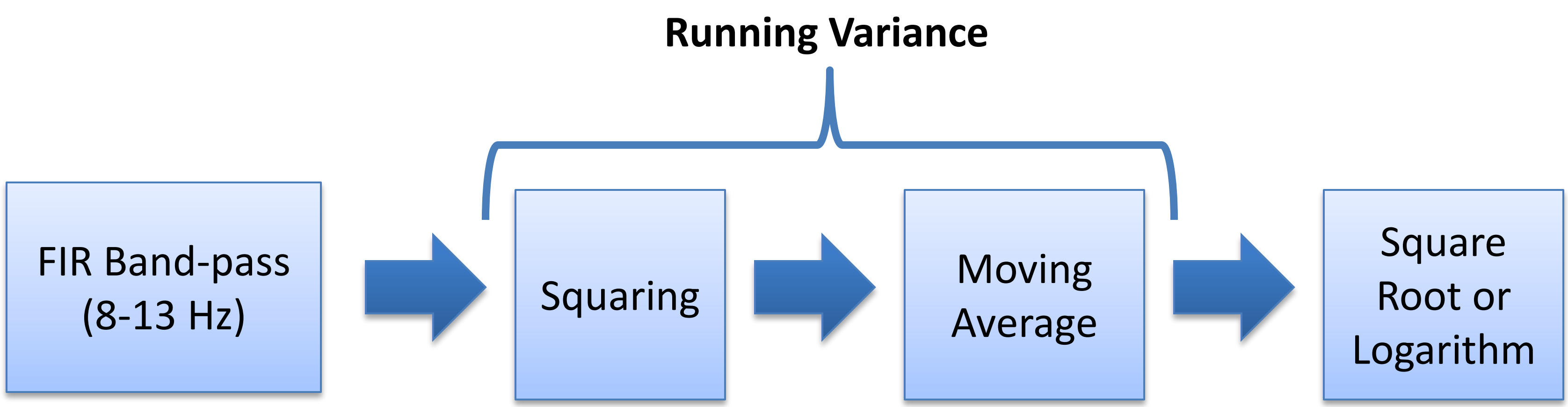
- BCILAB allows to implemented BCIs using a network of digital signal processing blocks ("filters")



- Relevant filter classes: *Spatial Filters, Temporal Filters, Spectral Filters, Spatio-Temporal Filters, Domain Transforms* (e.g. DFT)

Role of Signal Processing

- **Concrete Toy Example:** Feed the amplitude of a brain idle oscillation (e.g. 10 Hz alpha associated with relaxation) from one EEG channel back to the user/subject



$$y_i(n) = \sum_{k=0}^{m} b_k x_i(n-t)$$

$$y_i(n) = x_i(n)^2$$

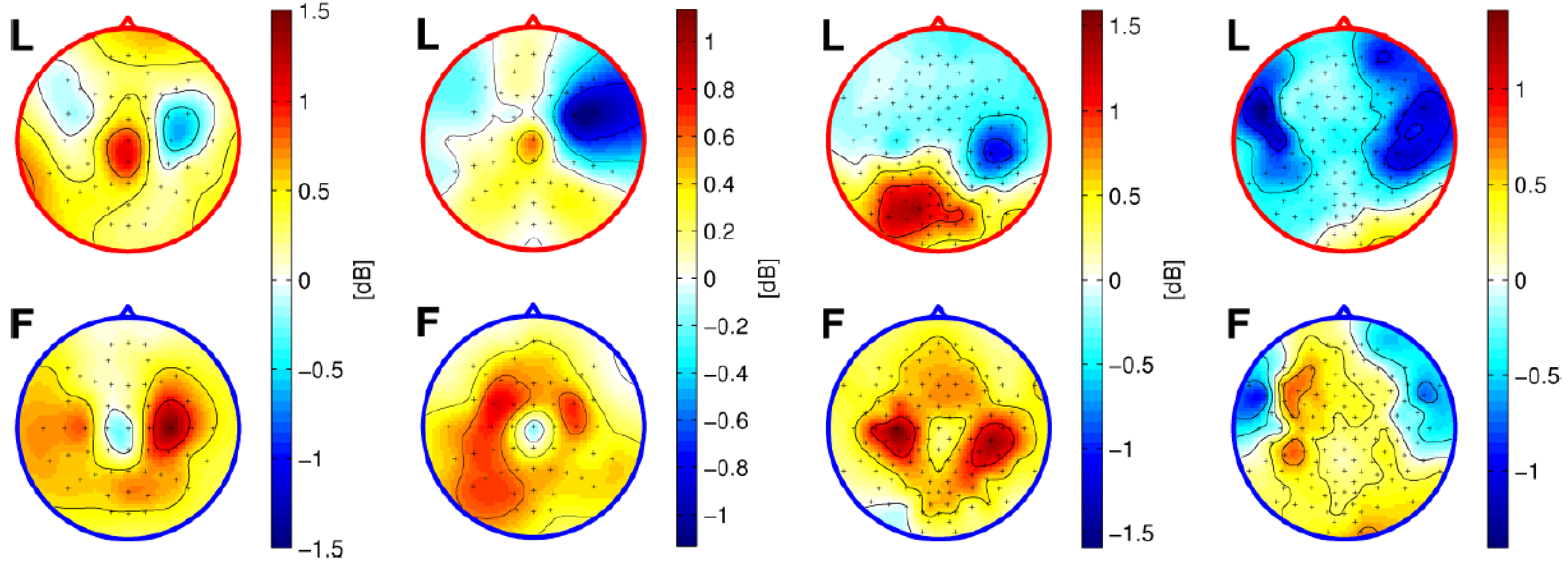
$$y_i(n) = \frac{1}{m-1} \sum_{k=0}^{m} x_i(n-k)$$

$$y_i(n) = \log x_i(n)$$

Component 3: Machine Learning

The Problem of Unknown Parameters

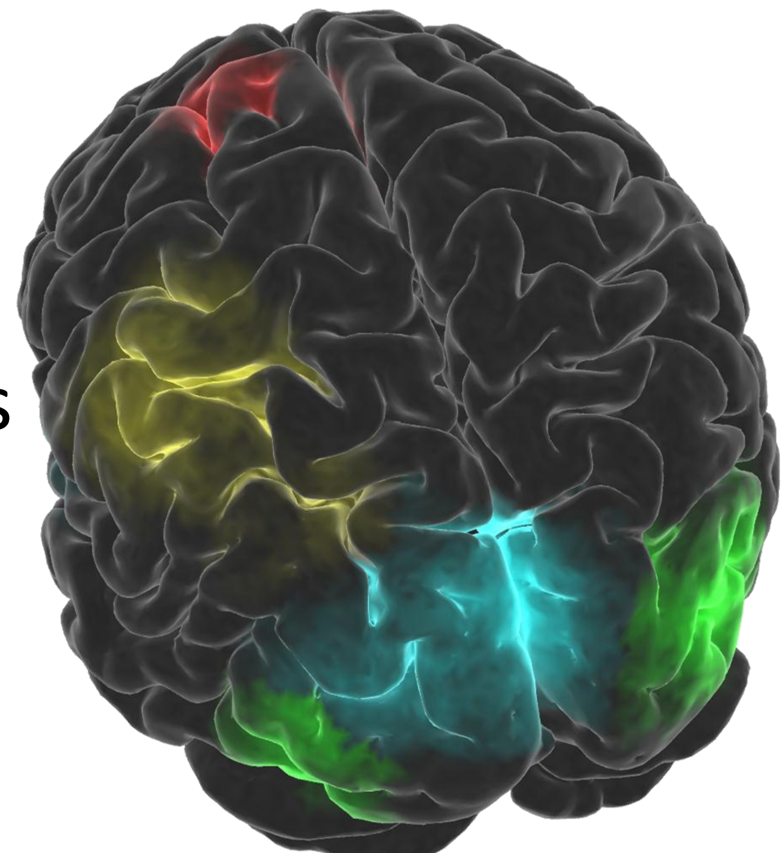
- Processing depends on unknown parameters (person-specific, task-specific, otherwise variable) – e.g., per-sensor weights as below:



Blankertz et al. 2007

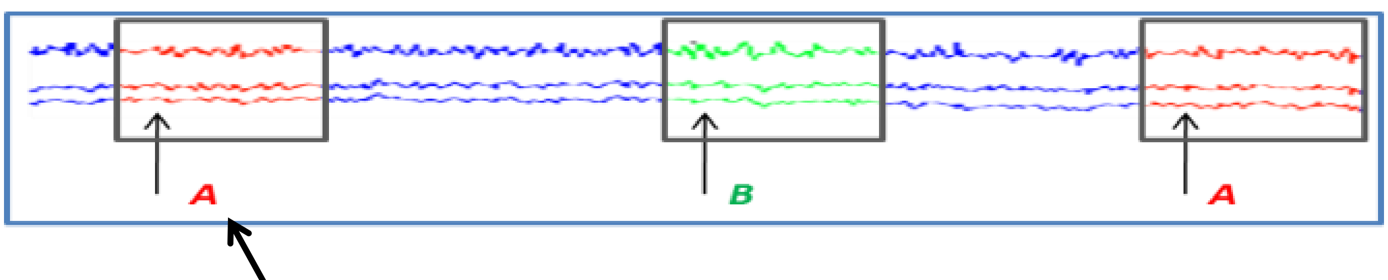
Reasons for Parameter Uncertainty

- Folding of cortex differs between any two persons
- Relevant functional map differs across individuals
- Sensor locations differ across recording sessions
- Brain dynamics are non-stationary at all time scales



Calibration Data

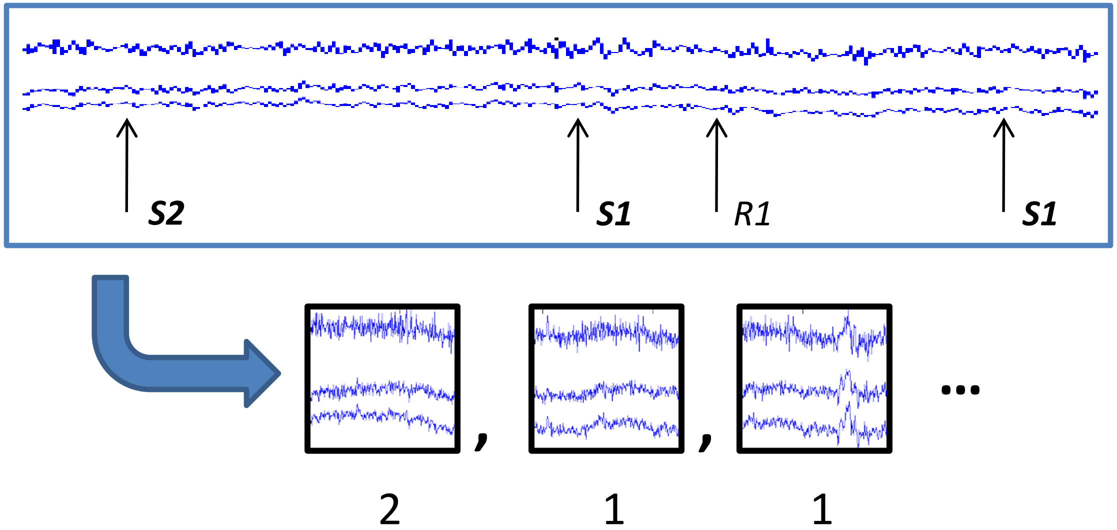
- Many possible kinds of data could be used
- Best known type of calibration data: ***example data***, i.e. examples of EEG of a person being excited, not excited, etc.
- Collected in a special *calibration recording* (before actual online use of the BCI)



"target markers" **in BCILAB**

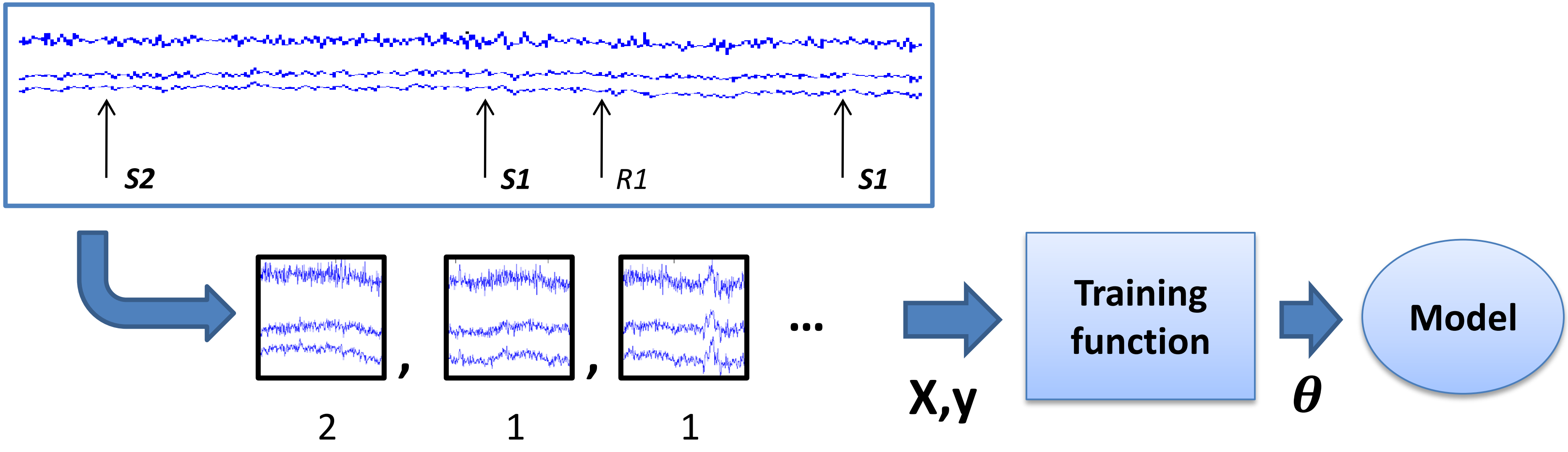
Machine Learning In Practice

- Often, one trial segment (sample) is extracted for every target marker in the calibration recording and is used as *training exemplar* $\boldsymbol{X}_k$
- Its associated label y_k can be deduced from the target marker



Machine Learning In Practice

- Often, one trial segment (sample) is extracted for every target marker in the calibration recording and is used as *training exemplar* $\boldsymbol{X}_k$
- Its associated label y_k can be deduced from the target marker

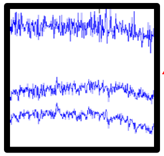


Component 4: Feature Extraction

Feature Extraction

- **Caveat:** Off-the-shelf machine learning methods often do *not work very well* when applied to raw signal segments of the calibration recording
  - too high-dimensional (too many parameters to fit)
  - too complex structure to be captured (too much modeling freedom, requires domain-specific assumptions)

1000s of degrees of freedom!



Feature Extraction

- **Typical Solution**: Introduce additional mapping (called *"feature extraction")* from raw signal segments onto feature vectors which extracts the *key features* of a raw observation
  - output is usually of lower dimensionality
  - hopefully statistically "better" distributed (easier to handle for machine learning)

Concrete Example Task

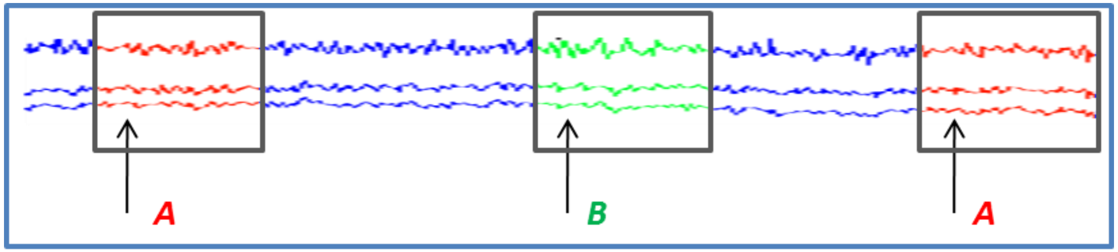
- **Flanker Task:** The experiment consists of a sequence of ca. 330 trials with inter-trial interval of 2s +/- 1.5s
- At the beginning of each trial, an arrow is presented centrally (pointing either left or right)
- The arrow is flanked by congruent or incongruent “flanker” arrows (preceding the center by a few ms):

← ← → ← ←

- The **subject is asked to press the left or right button, according to the central arrow direction, and makes frequent errors (ca. 25%)**

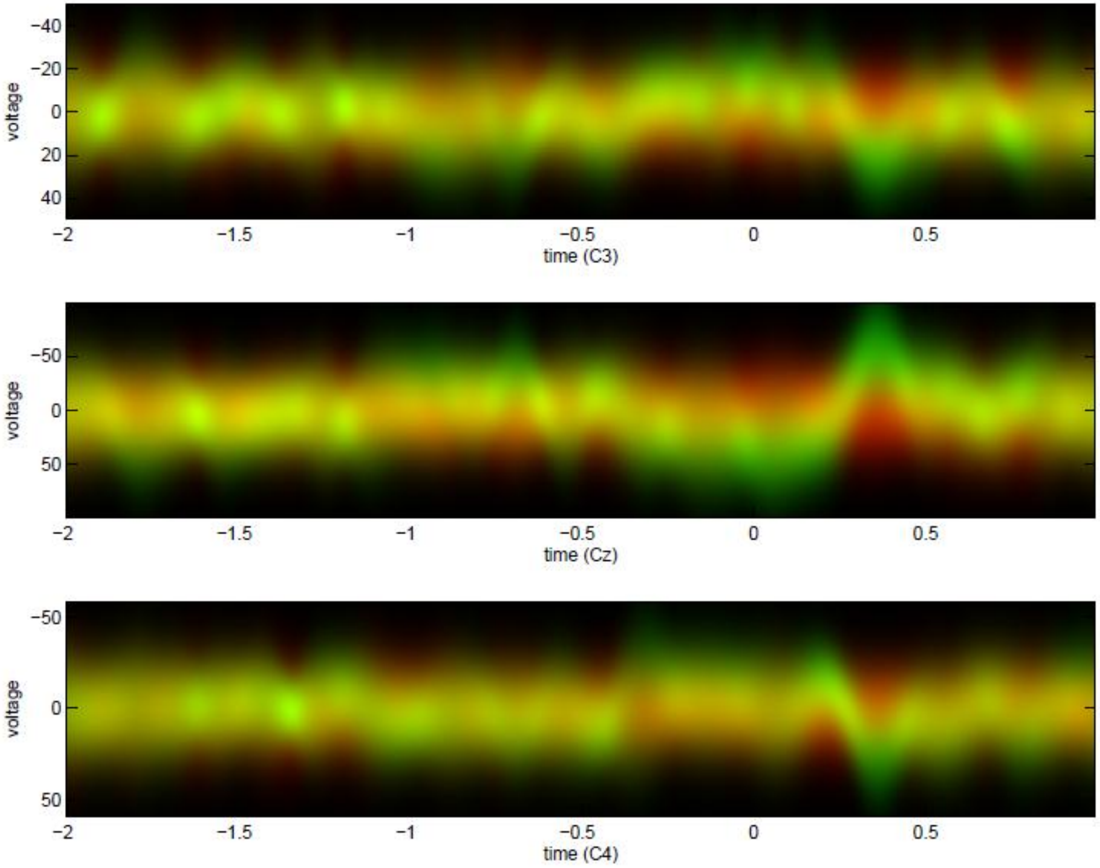
Approach

- Calibration recording is band-pass filtered between 0.5Hz and 15Hz
  - 0.5Hz lower edge removes drifts
  - 15Hz upper edge leaves enough room for sharp ERP features
- Epochs are extracted for each trial and label is set to A for incorrect trials and B for corrects

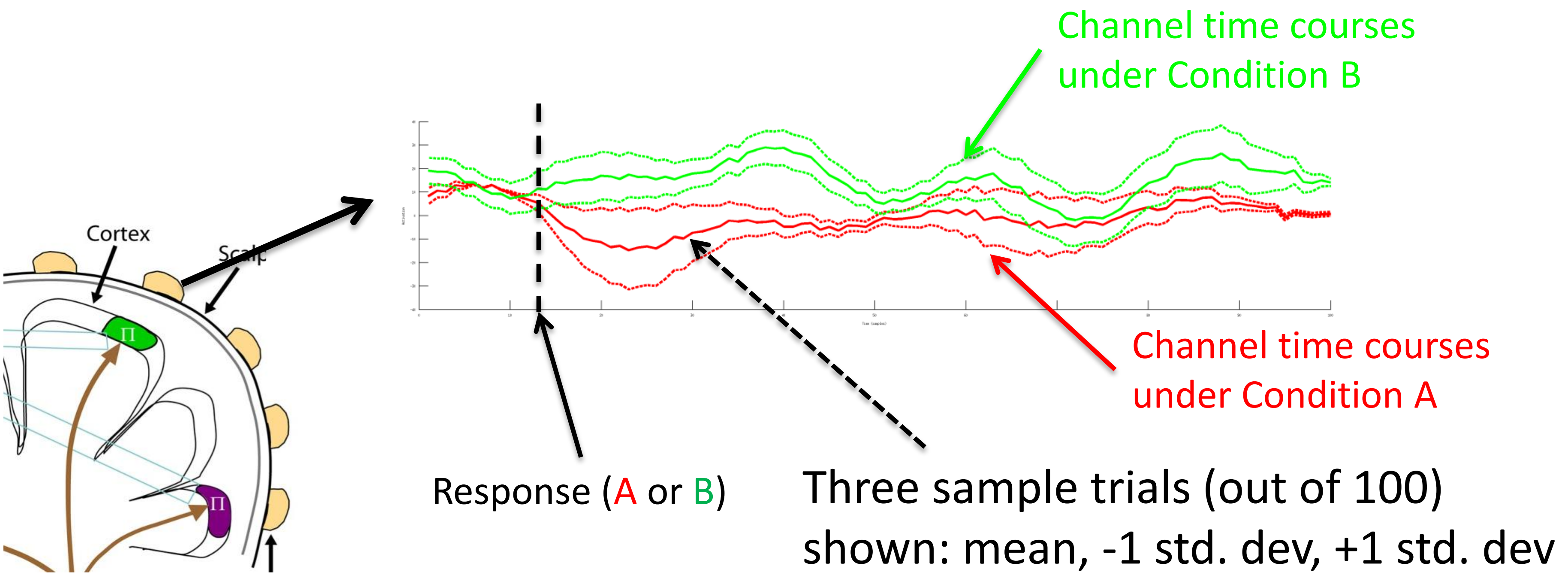


Actual Data

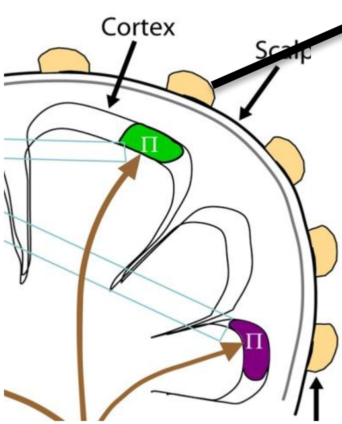
- Time courses for all trials super-imposed (color-coded by class) – but here different task

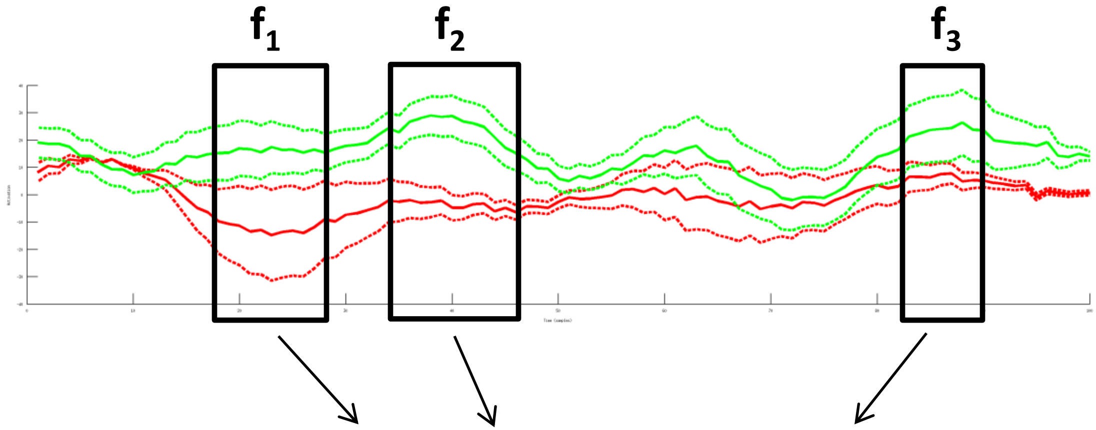


Extracted Epochs

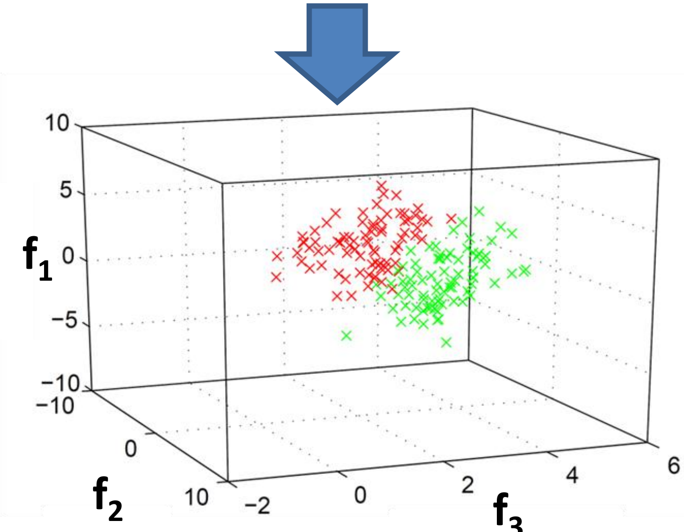


Extracting Linear Features



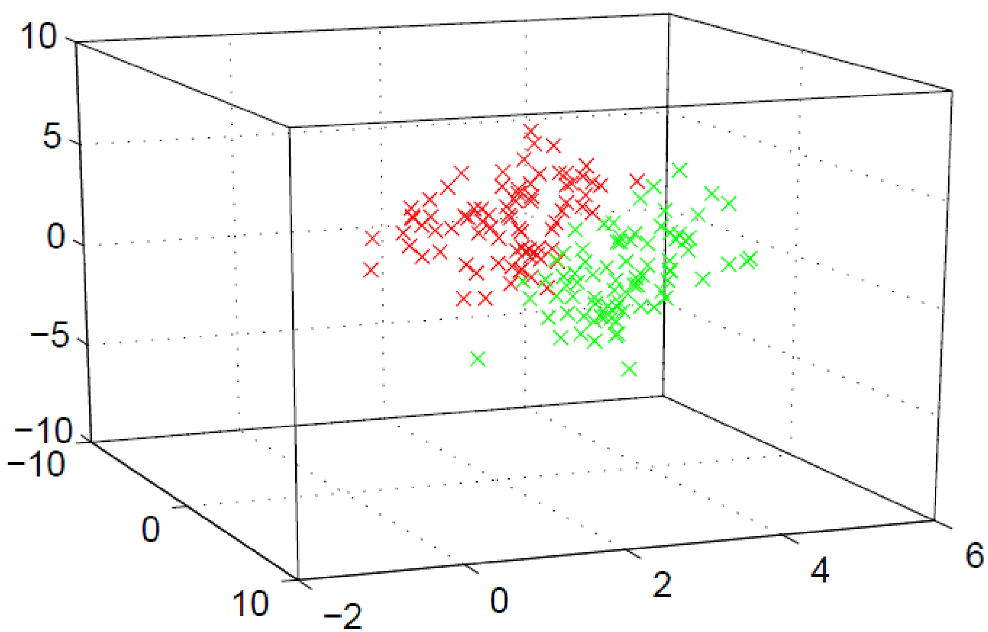


For each trial segment, calculate signal mean in 3 time sub-windows (→ 3-dim feature vector)



Resulting Feature Space

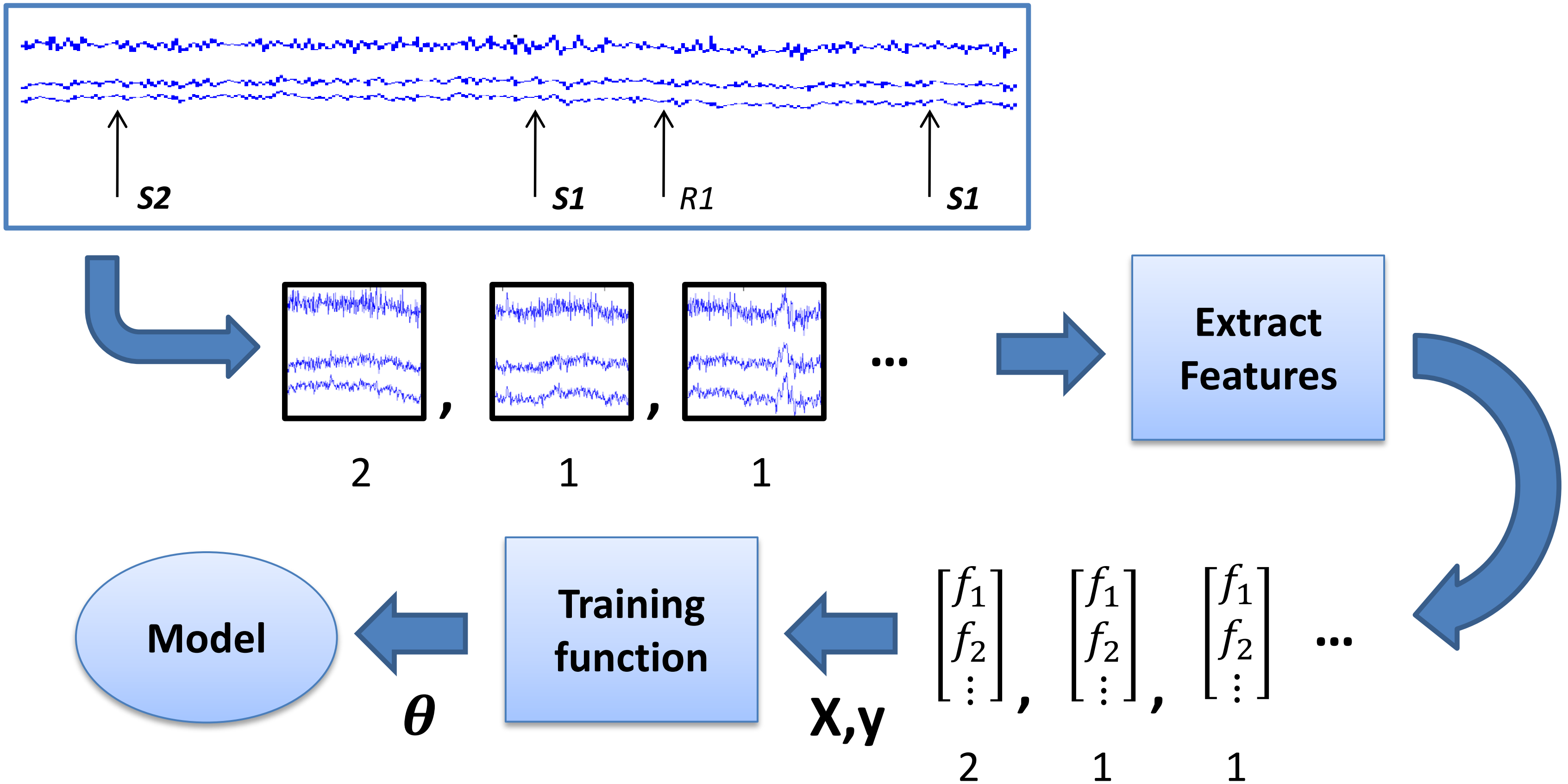
- Plotting the 3-element feature vectors for all error trials in red, and non-error trials in green, we obtain two distributions in a 3d space:



Note that across all channels this space has in fact 3 x #channels dimensions!

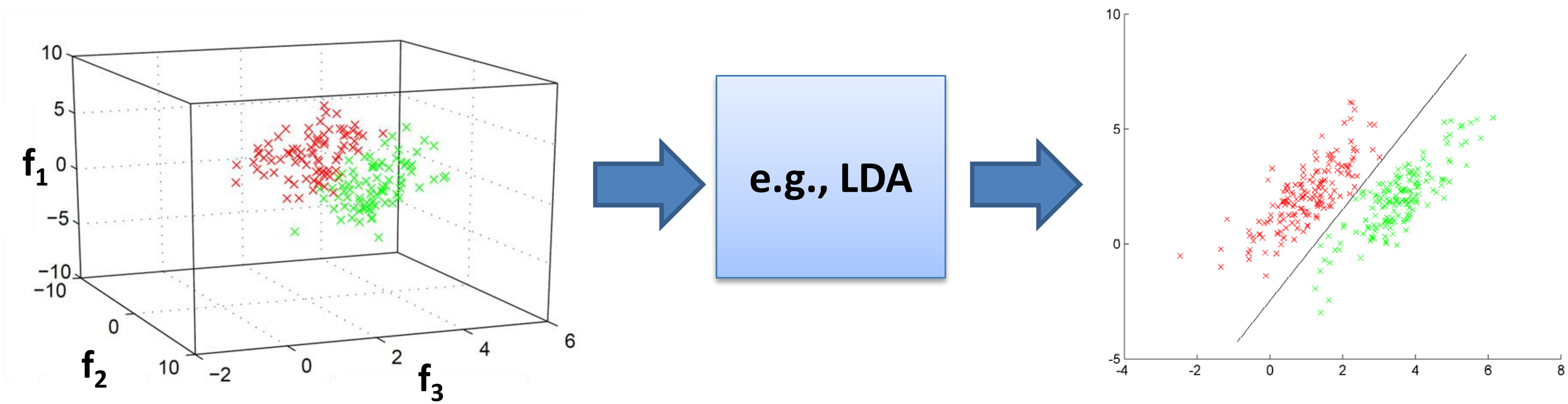
ML with Feature Extraction

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



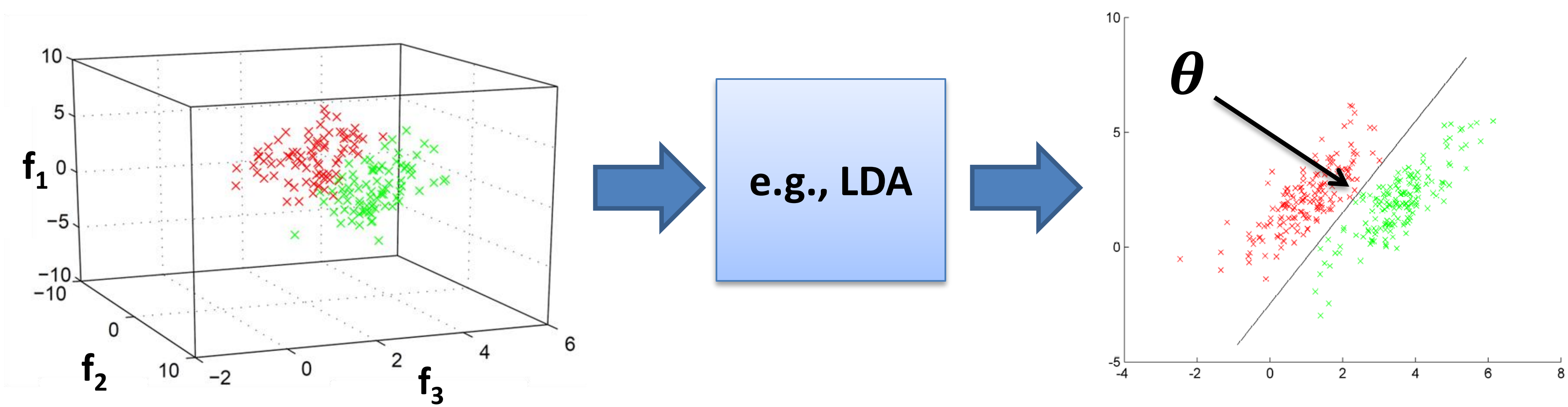
Machine Learning Continued

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



Machine Learning Continued

- The feature vectors are passed on to a machine learning function (e.g., Linear Discriminant Analysis)
- … which determines a parametric predictive mapping

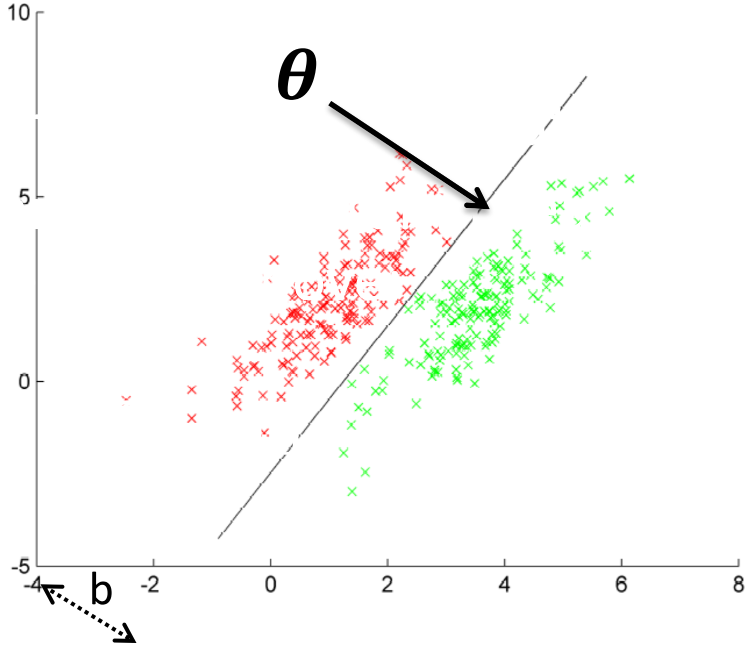


Simple 2-class LDA In a Nutshell

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

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

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



Resulting Predictive Mapping and Model

- LDA produces parameters of a linear mapping

$$y = \boldsymbol{\theta x} - b$$

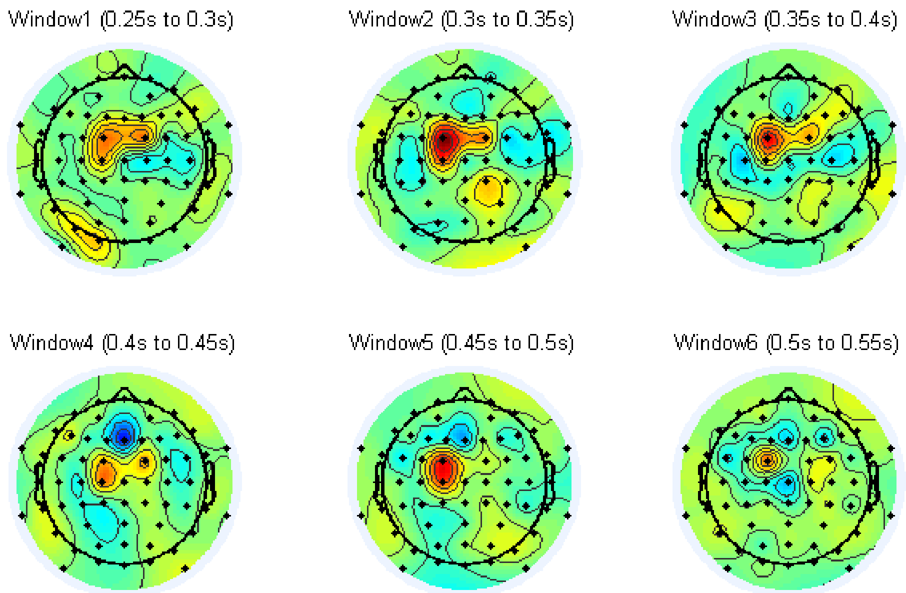
- For classification, the mapping is actually *non-linear*:

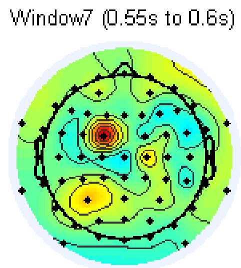
$$y = \mathrm{sign}(\boldsymbol{\theta x} - b)$$

- The learned model with its person-specific parameters here consists of $(\boldsymbol{\theta}, b)$; generally it could include adapted signal-processing parameters, feature-extraction parameters, etc.

Spatial Filters Visualized

- Topographically mapped, the following filters emerge:



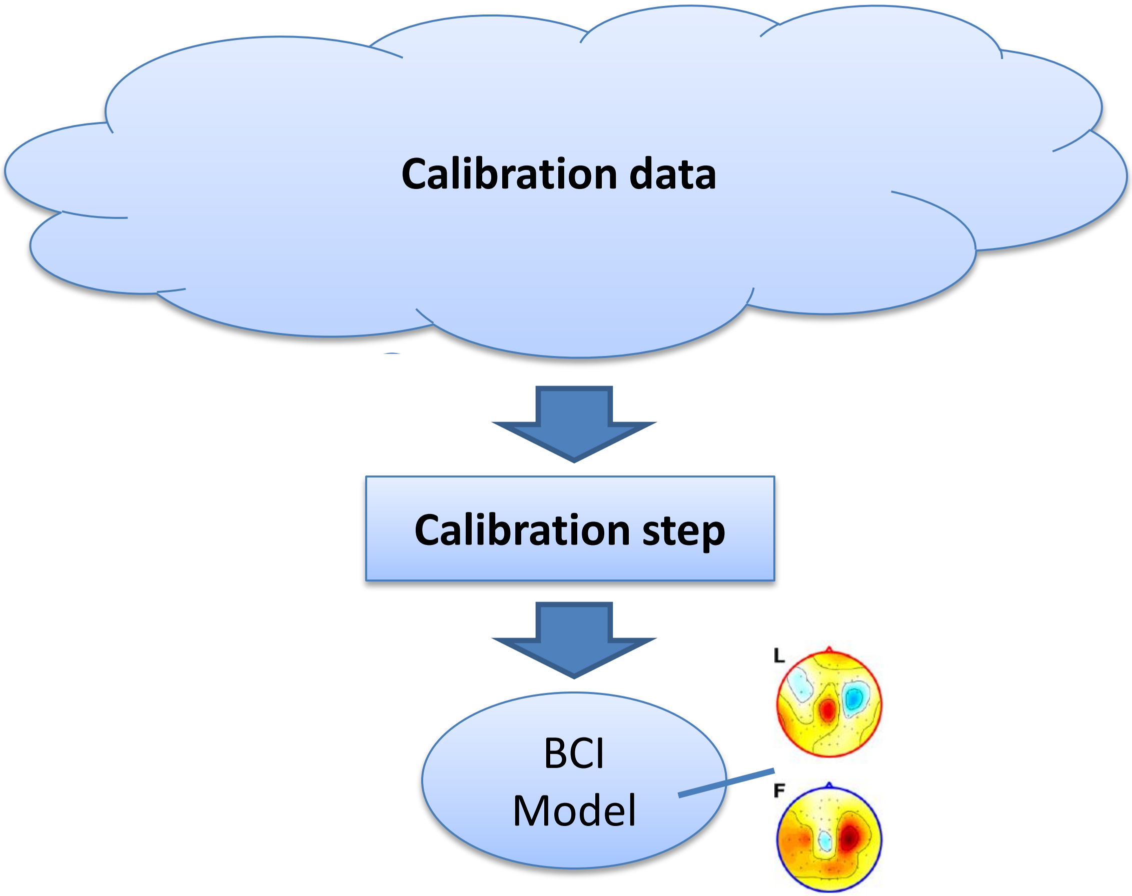


Note: This method (and its close relative using “shrinkage LDA” in particular) yield state-of-the-art Performance on ERPs.

Even More About Calibration Data

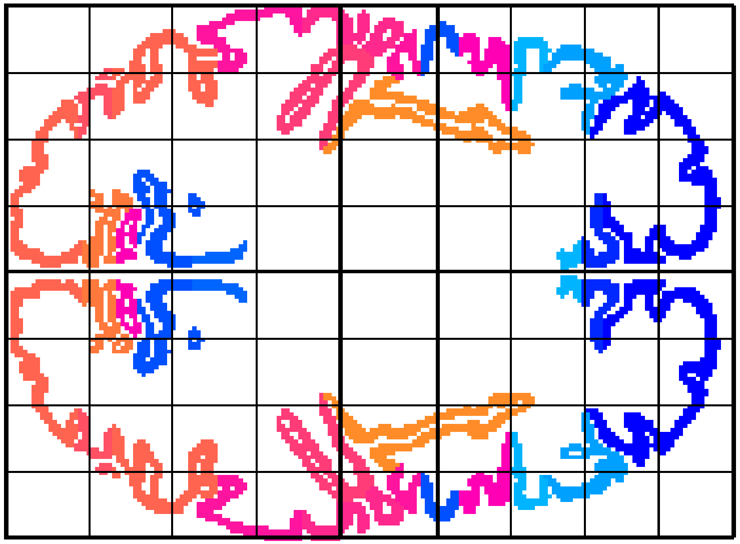
Model Calibration

- Can use *calibration / training data* to estimate parameters from, and a separate *calibration step*



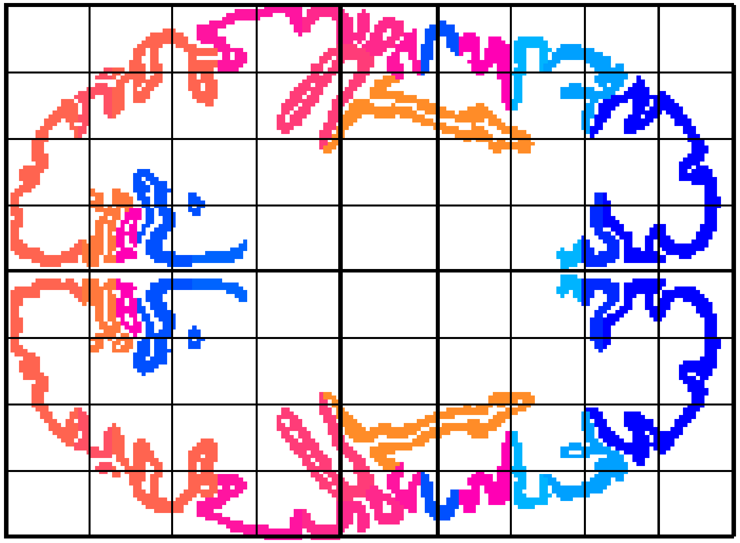
Prior Knowledge

- Prior knowledge is neuroscientific, such as:
  - Anatomical atlases (e.g. Talairach, LONI)
  - Functional atlases (if available)



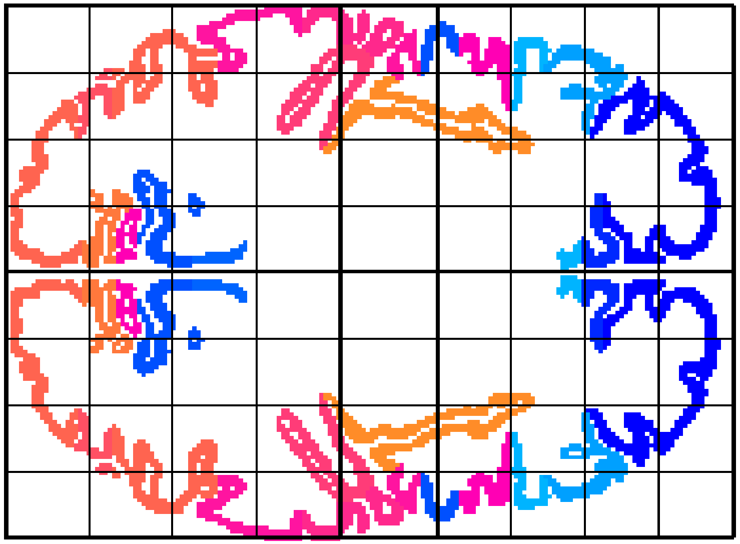
Prior Knowledge

- Prior knowledge is neuroscientific, such as:
  - Anatomical atlases (e.g. Talairach, LONI)
  - Functional atlases (if available)
  - Timing information (e.g. neural latencies, reaction times)

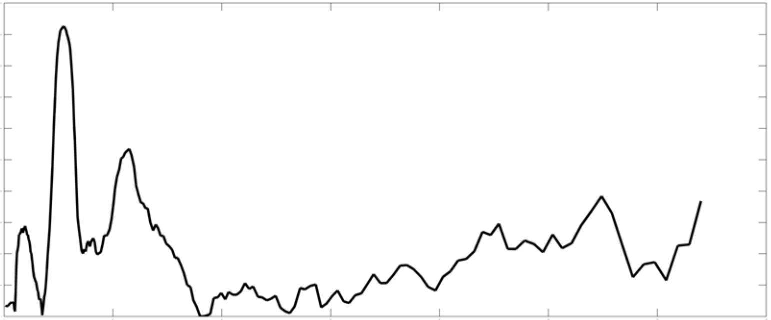


Prior Knowledge

- Prior knowledge is neuroscientific, such as:
  - Anatomical atlases (e.g. Talairach, LONI)
  - Functional atlases (if available)

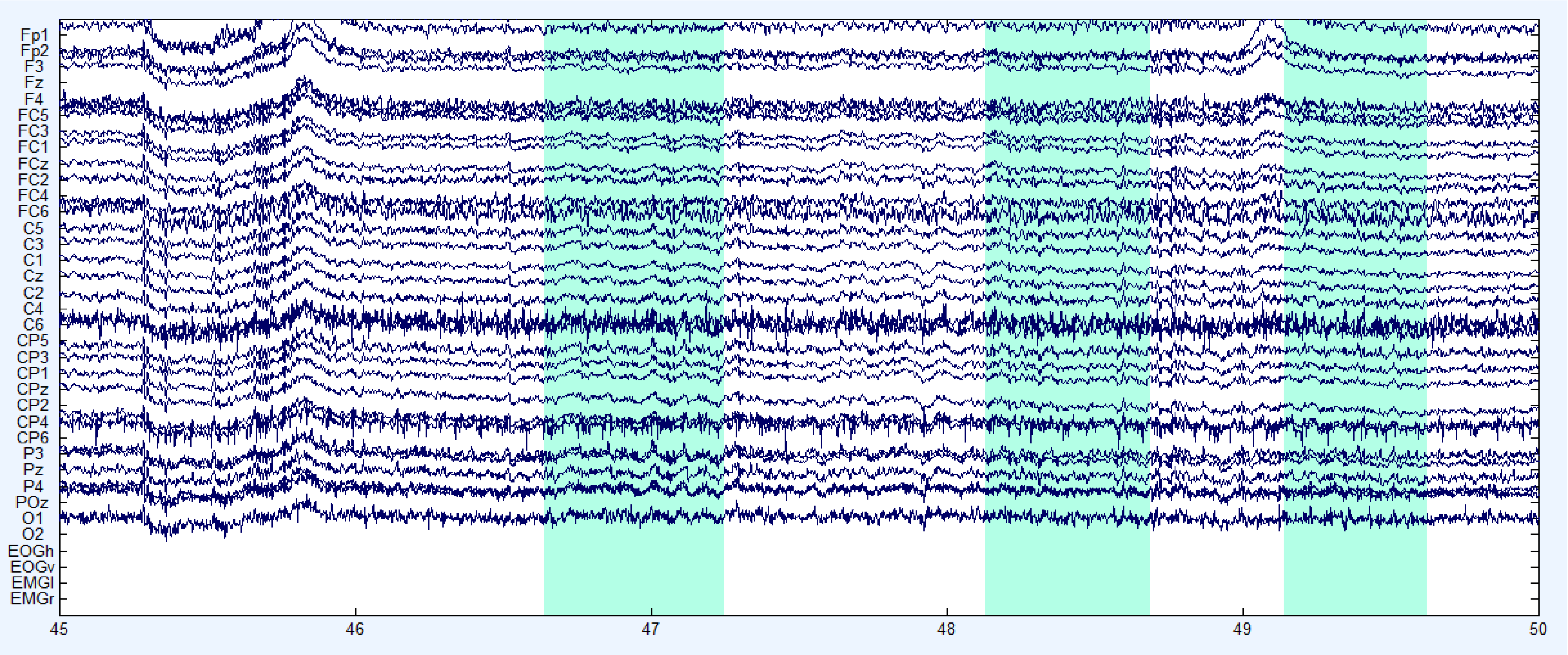


  - Timing information (e.g. neural latencies, reaction times)
  - Frequency bands of oscillatory processes (alpha, beta, theta, ...)



Calibration Data

- Example/calibration data is used to calculate optimal parameters of a BCI, and is *extremely important*



The Ideal Calibration Data

- Collected with the same/similar measurement apparatus as used for online runs
  - otherwise extra transformations and uncertainty incurred
- Comprises *multiple independent realizations / repetitions / trials* (to quantify variability)
  - one-shot learning (one exemplar) is *much* harder

The Ideal Calibration Data

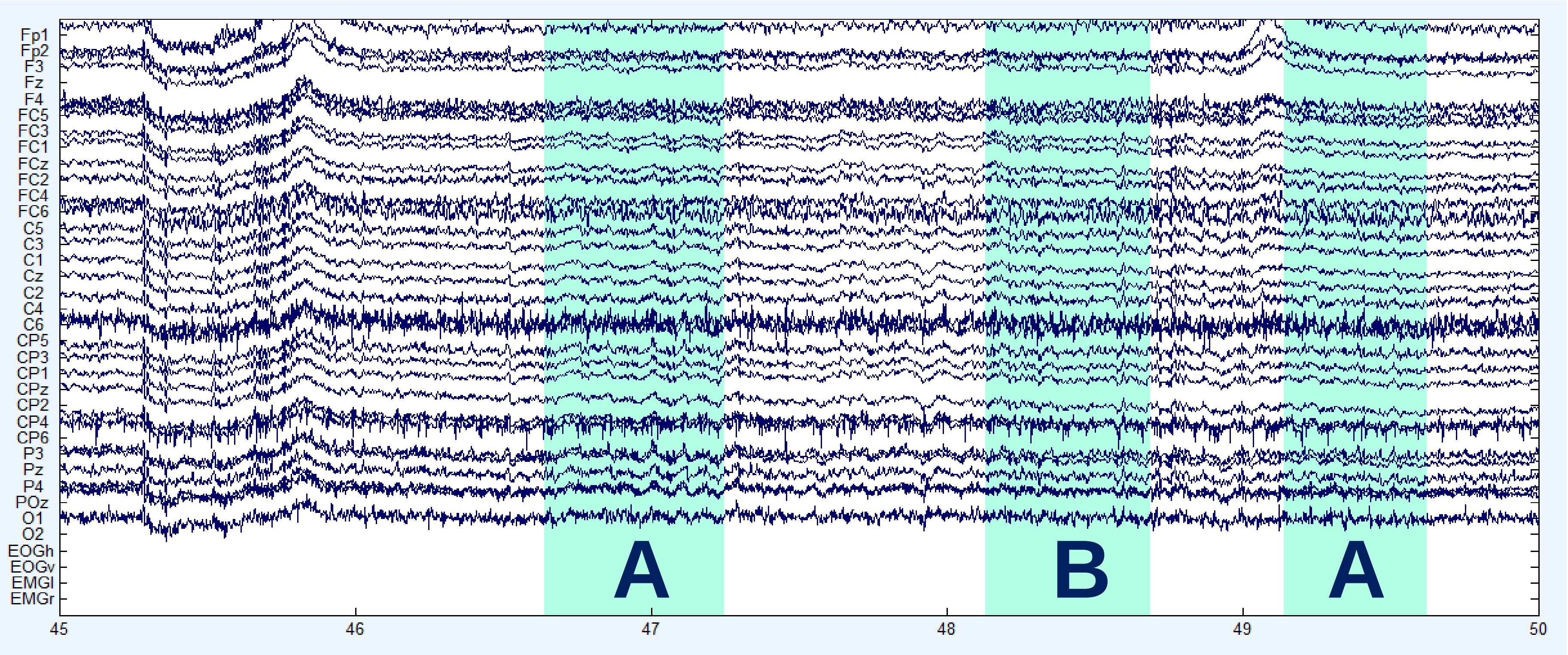
- Collected under conditions that are as close to those of the online runs as possible (i.e., drawn from the same statistical distribution)
  - Same person is preferable
  - Same sensor arrangement is preferable
  - Same session is preferable
  - Task parameters (stress level, …) should be similar
- Obviously a cost/benefit tradeoff:
  - Would trade off some performance for being able to reuse one recording for multiple sessions and persons

The Ideal Calibration Data

- If there is a systematic bias (e.g., different session), data should cover multiple realizations (e.g., multiple sessions) to capture variability
- A plain EEG recording is “unlabeled” (no knowledge about the association between raw observed signal and the cognitive state variable of interest)
- Labeled data (person is “surprised” / “not surprised”) is *far* more useful than unlabeled

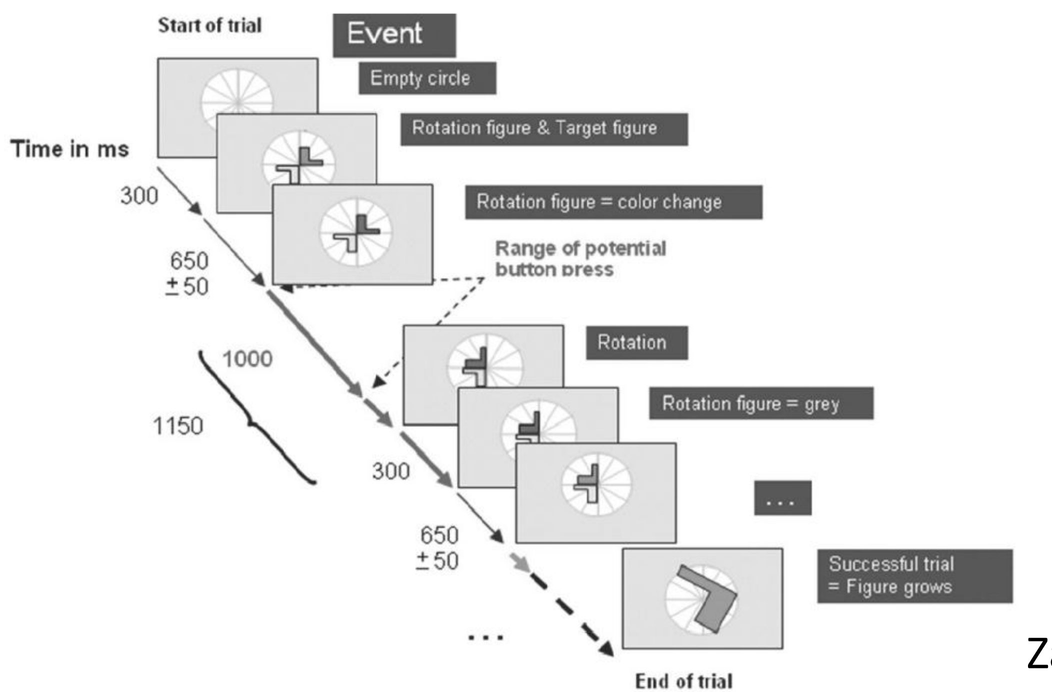
The Ideal Calibration Data

- Labels are assigned per realization (e.g., per trial) and *index the output that the BCI shall produce for this class of data*



Summary

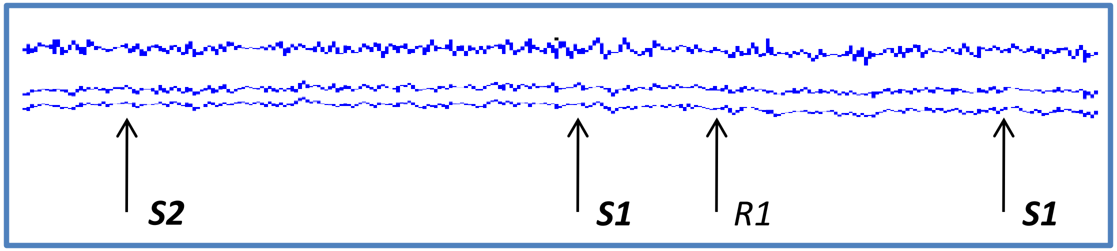
- The required data to calibrate a BCI resembles data produced by *controlled psychological experiments*



Zander et al., 2010

Summary

- Features
  - 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)
- Can also be used for offline performance tests



A Further Reading

These and Futher Slides:

ftp://sccn.ucsd.edu/pub/bcilab/

BCI Papers Worth Reading

- B. Blankertz, S. Lemm, M. Treder, S. Haufe, and K.-R. Mueller, "Single-trial analysis and classification of ERP components - A tutorial", NeuroImage, vol. 56, no. 2, pp. 814–825, May 2011.
- F. Lotte and C. Guan, “Regularizing common spatial patterns to improve BCI designs: unified theory and new algorithms,” IEEE Transactions on Biomedical Engineering, vol. 58, no. 2, pp. 355-362, Feb. 2011.
- R. Tomioka and K.-R. Mueller, A regularized discriminative framework for EEG analysis with application to brain-computer interface", NeuroImage, vol. 49, no. 1, pp. 415–432, 2010.
- B. Blankertz, G. Dornhege, M. Krauledat, K.-R. Mueller, and G. Curio, "The non-invasive Berlin brain-computer interface: Fast acquisition of effective performance in untrained subjects", NeuroImage, vol. 37, no. 2, pp. 539–550, Aug. 2007.
- M. Grosse-Wentrup, C. Liefhold, K. Gramann, and M. Buss, "Beamforming in noninvasive brain-computer interfaces", IEEE Trans. Biomed. Eng., vol. 56, no. 4, pp. 1209–1219, Apr. 2009.

BCI Surveys

- A. Bashashati, M. Fatourechi, R. K. Ward, and G. E. Birch, "A survey of signal processing algorithms in brain-computer interfaces based on electrical brain signals", J. Neural Eng., vol. 4, no. 2, pp. R32–R57, Jun. 2007.
- F. Lotte, M. Congedo, A. Lecuyer, F. Lamarche, and B. Arnaldi, "A review of classification algorithms for EEG-based brain-computer interfaces", J. Neural Eng., vol. 4, no. 2, pp. R1–R13, Jun. 2007.
- S. Makeig, C. Kothe, T. Mullen, N. Bigdely-Shamlo, Z. Zhang, K. Kreutz-Delgado, "Evolving Signal Processing for Brain–Computer Interfaces", Proc. IEEE, vol. 100, pp. 1567-1584, 2012.

Interesting Technical Papers

- D.P. Wipf and S. Nagarajan, "A Unified Bayesian Framework for MEG/EEG Source Imaging," NeuroImage, vol. 44, no. 3, February 2009.
- S. Haufe, R. Tomioka, and G. Nolte, "Modeling sparse connectivity between underlying brain sources for EEG/MEG," Biomedical Engineering, no. c, pp. 1-10, 2010.
- S. Boyd, N. Parikh, E. Chu, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," Information Systems Journal, vol. 3, no. 1, pp. 1-122, 2010.
- P. Zhao and B. Yu, "On Model Selection Consistency of Lasso," Journal of Machine Learning Research, vol. 7 pp. 2541-2563, 2006.

Technical Papers, ct'd

- J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Ng, "Multimodal Deep Learning," in Proceedings of the 28th International Conference on Machine Learning, 2011.
- K. N. Kay, T. Naselaris, R. J. Prenger, and J. L. Gallant, "Identifying natural images from human brain activity," Nature, vol. 452, no. 7185, pp. 352-355, Mar. 2008.
- O. Jensen et al., "Using brain-computer interfaces and brain-state dependent stimulation as tools in cognitive neuroscience," Frontiers in Psychology, vol. 2, p. 100, 2011.
- D.-H. Kim, N. Lu, R. Ma,. Y.-S. Kim, R.-H. Kim, S. Wang, J. Wu, S. M. Won, H. Tao, A. Islam, K. J. Yu, T.-I. Kim, R. Chowdhury, M. Ying, L. Xu, M. Li, H.-J. Cung, H. Keum, M. McCormick, P. Liu, Y.-W. Zhang, F. G. Omenetto, Y Huang, T. Coleman, J. A. Rogers, "Epidermal electronics," Science vol. 333, no. 6044, 838-843, 2011.

Researchers to Watch

- Klaus-Robert Mueller et al. (TU Berlin) – one of the leading BCI groups
  http://www.bbci.de/publications.html
- Marcel van Gerven et al. (Donders) – BCI and Neuroscience with a Bayesian approach
  https://sites.google.com/a/distrep.org/distrep/publications
- Ryota Tomioka (U Tokyo) – known for some technical achievements
  http://www.ibis.t.u-tokyo.ac.jp/RyotaTomioka
- Karl Friston et al. (UC London) – working on relevant underpinnings for neuroimaging (outside BCI)
  http://www.fil.ion.ucl.ac.uk/Research/publications.html
- Leading Statisticians and Machine Learners: Michael I. Jordan, Andrew Ng, Lawrence Carin, Zoubin Ghahramani, Francis Bach, Geoffrey Hinton, Ruslan Salakhutdinov, Yeh Whye Teh, David Blei, …