BDTB Manual

Ver. 1.0

2011/08/03

Introduction

This is the *Brain Decoder Toolbox (**BDTB**)* manual.
BDTB performs "**decoding**" of brain activity, by learning the difference between brain activity patterns among conditions and then classifying the brain activity based on the learning results.

BDTB is a set of Matlab functions.
BDTB is OS-independent.
BDTB was tested on Matlab R2010a, using Windows 7 Professional.

Some functions of *BDTB* rely on functions of "SPM5".
BDTB can use "LIBLINEAR", "LIBSVM", "OSU-SVM", and "SLR" as classifiers.
You can obtain them from the following sources.

- SPM5
  http://www.fil.ion.ucl.ac.uk/spm/software/spm5/

- LIBLINEAR
  http://www.csie.ntu.edu.tw/~cjlin/liblinear/

- LIBSVM
  http://www.csie.ntu.edu.tw/~cjlin/libsvm

- OSU-SVM
  http://svm.sourceforge.net/download.shtml

- SLR
  http://www.cns.atr.jp/~oyamashi/SLR_WEB.html

Copyright

Flow of Decoding

BDTB decodes brain activities in the following flow.

1. Create Mat File
   a. Load experimental design
   b. Load brain activity information
   c. Save the data into a mat file

2. Decoding
   a. Run preprocessing for the data
   b. Learn statistical models
   c. Classify the data by using learned models

Create Mat File

A mat file containing the brain activity information and the experimental design is created in this step.
The brain activities and the experimental design should be written in a *BDTB*-specific structure.

The format of the *BDTB*-specific structure D is shown below.

Structure D（Data）

field name	description	format
data	Brain activity information	Real-numbered matrix, [time (sample) x space(channel / voxel)]
label	Conditions, such as kinds of stimuli or movements	Real-numbered matrix, [time x # of label-type]
label_type	Types of conditions	Cell string matrix, [1 x # of label-type]
label_def	Name of each condition	Cell string hierarchy, [1 x # of label-type][1 x # of conditions]
design	Experimental design - numbers of runs, sessions, or blocks	Whole-numbered matrix, [time x # of design-type]
design_type	Types of experimental design	Cell string matrix, [1 x # of design-type]
stat	Statistics of samples	Real-numbered matrix, [# of stats-type x space]
stat_type	Types of statistics	Cell string matrix, [1 x # of stats-type]
xyz	Coordinate value of each channel / voxel	Real-numbered matrix, [3(x,y,z) x space]
roi	Channel- / voxel-assignment of each ROI	Boolean matrix, [# of ROI x space]
roi_name	Name of each ROI	Cell string matrix, [1 x # of ROI]

Decoding

The structure D in the pre-made mat file is read, and data is divided between training data and test data.
Training data is used for learning of the statistical models.
Test data is used for classification by the learned models.

BDTB has some useful functions like cross-validation and filters.
In order to use them, the brain activity information and the experimental design should be contained in the structure D (see Create Mat File), and the parameters of functions should be in the structure P.

The format of structure P is outlined below.

field name	description	format
<function name> . <parameter name>	Parameter value	Structure named by function name
(examples)		
selectChanByTvals.num_chans	200	
selectChanByTvals.tvals_min	3.2	
reduceOutliers.std_thres	4	

※ Please refer to List of Functions and help of each function for information about parameters.

Sample Program

You can download sample programs and data from the following web site (http://www.cns.atr.jp/dni/download/brain-decoder-toolbox/).

The sample programs run Create Mat File and Decoding according to Flow of Decoding.

The sample data was obtained from the following experiment.

Subject : a man
Setting of MRI : 1.5 T MRI(Shimadzu-Marconi), FOV 192 mm, 64 x 64 matrix, 3 x 3 x 3 mm, 50 slices, TE 50 ms, FA 90 deg, TR 5 s

Subject was instructed to continue making the gesture of rock-paper-scissors (RPS) in the MRI according to the visual and temporal cues.
The visual cues indicated which hand gesture should be made.
The temporal cues (beeping sounds) were given in 1 sec intervals to inform the subject about the timing of the repetition.
Each run had a 20 sec rest period at the beginning and at the end to relax the subject.
Between the rest periods, the subject repeated making the requested gesture that was changed in 20 sec intervals as below.

```
Run 1 :  rest  S  R  P  R  P  S  rest
Run 2 :  rest  S  P  R  R  S  P  rest
Run 3 :  rest  R  S  P  R  P  S  rest
Run 4 :  rest  P  R  S  R  S  P  rest
Run 5 :  rest  S  P  R  R  S  P  rest
Run 6 :  rest  P  S  R  P  R  S  rest
Run 7 :  rest  P  R  S  R  P  S  rest
Run 8 :  rest  S  P  R  S  R  P  rest
Run 9 :  rest  S  R  P  R  P  S  rest
Run10 :  rest  R  S  P  S  P  R  rest
```

make_fmri_mat.m

This function reads the brain activity information and the experimental design, puts them into the structure D, and creates a mat file.
For information about the format of structure D, please refer to Create Mat File.

Following is a description about each parameter.

P.sbj_id = 'SS100511'

ID of subject, specified by his/her initials in capital letters and the date of the experiment (YYMMDD format)

P.paths.to_lib = ''
P.paths.to_dat = ''

Paths to root directory of *BDTB* and data
If empty, you can specify them while this function is running.

P.fMRI.TR = 5

TR of MRI

P.fMRI.begin_vols = 3

Number of the first file in each run
When the number is different in each run, this value is a vector, [1 x # of runs].
(In this sample, this value is 3 because the first 2 samples in all runs have been deleted.)

P.fMRI.run_names = { 'a' ,' b' ,' c' ,' d' ,' e' ,' f' ,' g' ,' h' ,' i',' j' }

Letters that indicate the run number (※1)

P.fMRI.base_file_name = ['r' P.sbj_id]

Base name of fMRI files (※1)

P.prtcl.labels_runs_blocks = ... (1:rest, 2:Rock, 3:Scissors, 4:paper)

Condition / label (whole number) of each block, separated in a cell matrix per run, and per label type (※2)

P.prtcl.labels_type = { 'rock-paper-scissors' }

Type of condition / label

P.prtcl.labels_def = { 'rest' ,' rock' ,' scissors' ,' paper' }

Name of each condition / label, separated in a cell matrix per label type

P.prtcl.samples_per_label = {4}

Number of samples for each label specified in 'labels_runs_blocks', separated in a cell matrix per label type

When the number is the same in all runs and all labels, this value is a scalar; when the number is different in each label but the same in all runs, this value is a vector [1 x # of labels]; when the number is different in each label and each run, this value is a cell matrix [1 x # of run][1 x # of labels]. (※2)

P.prtcl.samples_per_block = 4

Number of samples for each block

When the labels and the blocks are the same, this value is the same as 'samples_per_label'.

P.rois.spm_ver = 5

Version of SPM that is used for ROI creation, because axis definition is dependent on the version of SPM

P.rois.roi_set = 'roi'

Name of ROI set

P.rois.roi_dir = 'roi/'

Name of directory that has ROI files

P.rois.roi_files = { 'M1_RHand' ,' SMA_RHand' ,' CB_RHand' ,...}

Names of ROI files (don't need '.mat')

When ROI isn't used (all voxels are used), specify an empty cell matrix.

P.stats.stat_dir = 'roi/'

Name of directory that has files containing statistics

P.stats.stat_files = {{ 'VOX_CB_RHand.mat' ,' VOX_M1_RHand.mat' ,...}}

Names of files containing statistics

When you want to summarize statistics in multiple files, group the file names in the same cell.

P.stats.stat_type = { 'tval' }

Name of statistics type

P.output.verbose = 0

Print detail level, 0 (no printing) ～ 2 (output all)

P.output.save_ver = 7

Format of mat file that will be created

P.output.file_name = [P.sbj_id '_fmri_' P.rois.roi_set '_v' …]

Name of mat file that will be created

※1 In this sample, the name of the fMRI file is

[prefix][subject ID][run letter][file number in 4 figures] .hdr / .img

So, [run letter (a, b, …)] is used for 'run_names', and [prefix][subject ID] is used for 'base_file_name'.

※2 D.label, the labels corresponding to samples, is made by a combination of 'labels_runs_blocks' and 'samples_per_label'.

When 'samples_per_label' is a scalar:

D.label is made by repeating each label of 'labels_runs_blocks' 'samples_per_label' number of times.

When 'samples_per_label' is a vector:

D.label is made by repeating the i-th label in each run of 'labels_runs_blocks' 'samples_per_label(i)' number of times.

When 'samples_per_label' is a cell matrix:

D.labels is made by repeating the i-th label in the j-th run of 'labels_runs_blocks' 'samples_per_label{j}(i)' number of times.

decode_basic.m

This function reads the structure D (see Create Mat File), applies filters to the data, and calculates the decoding accuracy by using cross-validation.

Following is a description of each parameter.

P.script_name = mfilename
P.date_time = datestr(now,' yyyy-mm-dd HH:MM:SS')
Name of this function and time at when this function is run

P.paths.to_lib = ''
P.paths.to_dat = ''
Paths to root directory of *BDTB* and data
If empty, you can specify them while this function is running.

P.procs1 = {...}
Names of functions that will be applied to data before cross-validation (※3)
Functions are applied in the specified order.

P.procs2 = {...}
Names of functions that will be applied to data in cross-validation (※3)
Functions are applied in the specified order.

P.<function name>.<parameter name> = ...
Parameters of filters (※4)

P.models = { 'libsvm_bdtb' }
Names of models
When multiple models are specified, they are used in parallel.

P.<model name>.<parameter name> = ...
Parameters of models (※4)

※3 The filters specified in 'procs1' are applied to data before cross-validation.
This means that they are applied to the data that hasn't yet been divided between training data and test data.
So you should notice that the information you can't obtain from training data only shouldn't be used. (Double dipping)
The filters specified in 'procs2' are applied to training data and test data individually in cross-validation.
The parameters calculated in the training session can be used in the testing session.

※4 Please refer to List of Functions and help of each function and model for information about parameters.

List of Functions

This is the list of functions *BDTB* contains.

Filters

averageBlocks	Average data in each block
averageLabels	Average data in each label
balanceLabels	Balance number of samples among labels
convertLabel	Converting labels
detrend_bdtb	Detrend on data along time dimension
highPassFilter	High-pass filter
normByBaseline	Normalize data by its baseline
poolSample	Average data in each label
reduceOutliers	Reduce outlier values of data
removeBlockSample	Remove samples in each block
selectBlockSample	Select samples in each block
selectChanByTvals	Select channels based on t-value
selectConds	Select samples corresponding to labels
selectLabelType	Select label type from multiple labels
selectTopFvals	Select data based on F-value
shiftData	Shift data along time dimension
zNorm_bdtb	Normalize data by z-score

Models

liblinear_bdtb	Perform LIBLINEAR
libsvm_bdtb	Perform LIBSVM
slr_lap_bdtb	Perform SLR-LAP-1vsR
slr_var_bdtb	Perform SLR-VAR-1vsR
smlr_bdtb	Perform Multinomial SLR
svm11lin_bdtb	Perform OSU-SVM

Evaluation

crossValidate	Perform leave-one-out cross-validation
validate	Perform validation

averageBlocks Average data in each block

[D, pars] = averageBlocks(D, pars)

Average data in each block

Input :

D.data	– brain activity information
D.label	– condition of each sample
D.design	– design matrix of experiment (to get block information)
D.design_type	– name of each design type (to find 'block')

Optional :

pars.begin_off	– number of samples to remove from the beginning of each block (default: 0)
pars.end_off	– number of samples to remove from the end of each block (default: 0)
pars.target_labels	– labels for which data samples are averaged (default: all labels)
pars.verbose	– print detail level (default: 1)

Output :

D.data	– block averaged data
D.label	– labels for averaged data
D.design	– design matrix of averaged data

averageLabels Average data in each label

[D, pars] = averageLabels(D, pars)

Average data in each continuous label

Input :

D.data	– brain activity information
D.label	– condition of each sample

Optional :

pars.begin_off	– number of samples to remove from the beginning of each block (default: 0)
pars.end_off	– number of samples to remove from the end of each block (default: 0)
pars.target_labels	– labels for which data samples are averaged (default: all labels)
pars.verbose	– print detail level (default: 1)

Output :

D.data	– averaged data
D.label	– labels for averaged data
D.design	– design matrix of averaged data

balanceLabels Balance number of samples among labels

[D, pars] = balanceLabels(D, pars)

Equalize the number of samples among labels

Input :

D.data	– brain activity information
D.label	– condition of each sample

Optional :

pars.method	– equalizing-method 1: averaging, 2: adjust to min (default), 3: adjust to max
pars.doTest	– should we balance in testing? 0: no, 1: yes (default)
pars.mode	– 1: training, 2: testing
pars.verbose	– print detail level (default: 0)

Output :

D.data	– data matching the new labels
D.label	– new balanced labels

convertLabel Converting labels

[D, pars] = convertLabel(D, pars)

Converting labels according to the given table

Input :

D.label — condition of each sample

pars.list — conversion table of labels, {[org1, new1], [org2, new2], …} format

Output :

D.label — converted labels

detrend_bdtb　　Detrend on data along time dimension

[D, pars] = detrend_bdtb(D, pars)

Detrend on data along the time dimension

Input :

D.data	—	brain activity information

Optional :

D.design	—	design matrix of experiment (to get run information)
D.design_type	—	name of each design type (to find 'run')
pars.sub_mean	—	subtract the mean?
		0: no (default), 1: yes
pars.method	—	detrend-method
		linear: subtract linear fit (default)
		constant: subtract just the mean
pars.breaks	—	break points for piecewise detrend
		[begin1, begin2, …; end1, end2, …] format
pars.break_run	—	use runs as breaks?　0: no, 1: yes (default)
pars.verbose	—	print detail level (default: 1)

Output :

D.data	—	detrended data

※ This filter is applied to data along the time dimension, so the data should be continuous along the time dimension.
If there are gaps in time like the time between runs, you should specify them in pars.breaks, or use run information obtained from D.design.

highPassFilter　　High-pass filter

[D, pars] = highPassFilter(D, P)

Apply high-pass filter to data

Input :

D.data	−	brain activity information

Optional :

D.design	−	design matrix of experiment (to get run information)
D.design_type	−	name of each design type (to find 'run')
pars.dt	−	sampling interval [sec] (default: 2)
pars.cutoff	−	cut-off frequency [sec] (default: 128) or list of cut-off frequency (e.g. [128, 128, …])
pars.app_dim	−	dimension along which this process will be applied 1: across time (default), 2: across space
pars.linear_detrend	−	perform 'detrend' before high-pass filtering? 0: no, 1: yes (default)
pars.breaks	−	break points for piecewise filtering [begin1, begin2, …; end1, end2, …] format
pars.break_run	−	use runs as breaks?　0: no, 1: yes (default)
pars.verbose	−	print detail level (default: 1)

Output :

D.data	−	high-pass filtered data

※ This filter is applied to data along the time dimension, so the data should be continuous along the time dimension.
If there are gaps in time like the time between runs, you should specify them in pars.breaks, or use run information obtained from D.design.

normByBaseline　　Normalize data by its baseline

[D, pars] = normByBaseline(D, pars)

Calculate the baseline of data along the time dimension, and normalize the data by it

Input :

D.data	—	brain activity information
D.label	—	condition of each sample

Optional :

D.design	—	design matrix of experiment (get run information)
D.design_type	—	name of each design type (find 'run')
pars.base_conds	—	label that is used for calculation of the baseline (default :1)
pars.zero_thres	—	threshold below which the baseline is considered zero
pars.breaks	—	break points for piecewise normalization [begin1, begin2, …; end1, end2, …] format
pars.break_run	—	use runs as breaks?　0: no, 1: yes (default)
pars.verbose	—	print detail level (default: 1)
pars.mode	—	baseline normalization mode 0: subtraction of and division by the mean (i.e. % signal change, default) 1: only division by the mean 2: only subtraction of the mean 3: subtraction of the mean and division by the std (i.e. z-score)

Output :

D.data	—	normalized data

※ This filter is applied to data along the time dimension, so the data should be continuous along the time dimension.
If there are gaps in time like the time between runs, you should specify them in pars.breaks, or use run information obtained from D.design.

poolSample　　Average data in each label

[D, pars] = poolSample(D, pars)

Pool and average data with same labels

This filter can average data in other blocks depending on parameters.

Input :

D.data	−	brain activity information
D.label	−	condition of each sample
D.design	−	design matrix of experiment (to get run and block information)
D.design_type	−	name of each design type (to find 'run' and 'block')
pars.nPool	−	number of pooling samples
pars.poolLabel	−	target labels to pool

Optional :

pars.poolSep	−	pool samples in other blocks?　0: no, 1: yes (default)
pars.useResid	−	use residual samples? 0: delete 1: add last block (default) 2: make one more block

Output :

D.data	−	averaged data
D.label	−	labels for averaged data
D.design	−	design matrix of averaged data

※ This filter permutes labels and samples.
(In each run, nontarget labels get moved to the front, and target labels are all moved to the back of the data set.)
So, please notice when this filter is applied to data.

reduceOutliers Reduce outlier values of data

[D, pars] = reduceOutliers(D, pars)

Reduce outlier values of data along the time / space dimension

Input :

D.data	—	brain activity information

Optional :

D.design	—	design matrix of experiment (to get run information)
D.design_type	—	name of each design type (to find 'run')
pars.app_dim	—	dimension along which reduction will be applied 1: across time (default), 2: across space
pars.remove	—	remove channels including outliers? 0: clip outliers, 1: remove instead of clip outliers
pars.method	—	method to find outliers 1: max std deviation only 2: constant 'min_val', 'max_val' only 3: both (default)
pars.std_thress	—	number times std for threshold cut off and clip (default: 3)
pars.num_its	—	number of iteration (default: 10)
pars.max_val	—	absolute max value (default: inf)
pars.min_val	—	absolute min value (default: -inf)
pars.breaks	—	break points for piecewise normalization [begin1, begin2, …; end1, end2, …] format
pars.break_run	—	use runs as breaks? 0: no, 1: yes (default)
pars.verbose	—	print detail level (default: 1)

Output :

D.data	—	data reduced outlier values
D.xyz	—	X, Y, Z-coordinate values within the selected channels
D.stat	—	statistics within the selected channels
D.roi	—	ROI information within the selected channels

※ This filter is applied to data along the time dimension, so the data should be continuous along the time dimension.
If there are gaps in time like the time between runs, you should specify them in pars.breaks, or use run information obtained from D.design.

※ D.xyz, D.stat, and D.roi will be updated according to the selected channels when you set the parameter to remove channels including outliers.

removeBlockSample　　Remove samples in each block

[D, pars] = removeBlockSample(D, pars)

Remove the specified number of samples in each block

Input :

D.data	– brain activity information
D.label	– condition of each sample
D.design	– design matrix of experiment (to get block information)
D.design_type	– name of each design type (to find 'block')
pars.begin_off	– start index offset of samples to be removed from the beginning of each block (default: 0)
pars.end_off	– end index offset of samples to be removed from the end of each block (default: 0)

Optional :

pars.target_labels	– labels for which data samples are removed (default: all labels)
pars.verbose	– print detail level (default: 1)

Output :

D.data	– removed data
D.label	– labels for removed data
D.design	– design matrix of removed data

selectBlockSample　　Select samples in each block

[D, pars] = selectBlockSample(D, pars)

Select the specified number of samples in each block

Input :

D.data	—	brain activity information
D.label	—	condition of each sample
D.desing	—	design matrix of experiment (to get block information)
D.design_type	—	name of each design type (to find 'block')
pars.inds	—	indices of samples to be selected for each block

Optional :

pars.target_labels	—	labels for which data samples are selected (default: all labels)
pars.verbose	—	print detail level (default: 1)

Output :

D.data	—	selected data
D.label	—	labels for selected data
D.design	—	design matrix of selected data

selectChanByTvals Select channels based on t-value

[D, pars] = selectChanByTvals(D, pars)

Select the specified number / ratio of channels whose t-values are within the specified region of t-values

Input :

D.data	– brain activity information
D.stat	– statistics of each sample (to get t-value)
D.stat_type	– name of each statistics type (to find 'tval')

Optional :

pars.num_chans	– number of channels to select (whole number), or percent of existing ones (decimal, less than 1) (default: all channels)
pars.tvals_min	– min value of t-values range to use (default: -inf)
pars.tvals_max	– max value of t-values range to use (default: inf)
pars.verbose	– print detail level (default: 1)

Output :

D.data	– data within the selected channel
D.xyz	– X, Y, Z-coordinate values within the selected channel
D.stat	– statistics within the selected channel
D.roi	– ROI information of selected channel

selectConds Select data corresponding to labels

[D, pars] = selectConds(D, pars)

Select samples corresponding to labels that match the specified ones

Input :

D.data	– brain activity information
D.label	– condition of each sample
pars.conds	– conditions to be selected from labels

Optional :

pars.verbose	– print detail level (default: 1)

Output :

D.data	– data corresponding to matched labels
D.label	– condition labels matching 'conds'

selectLabelType　　Select label type from multiple labels

[D, pars] = selectLabelType(D, pars)

Select a label type that is used for analysis from multiple labels

Input :

D.label	– condition of each sample
pars.target	– target label type index to be selected

Output :

D.label	– labels of the selected label type
D.label_type	– selected label type
D.label_def	– name of each condition in selected label type

※ The structure D can contain multiple label types.
But you should select only one label type to be used for analysis.

selectTopFvals　　Select data based on F-value

[D, pars] = selectTopFvals(D, pars)

Calculate F-values, and select the specified number / ratio of channels / samples whose F-values are within the specified region of F-values

Input :

D.data	—	brain activity information
D.label	—	condition of each sample

Optional :

pars.inds_fvals	—	indices of data ordered by F-values (made in training, used in testing)
pars.fvals	—	F-values matching 'inds_fvals' (descending)
pars.mode	—	1: training (make 'inds_fvals'), 2: testing (use 'inds_fvals')
pars.app_dim	—	application dimension 1: across time, 2: across space (default)
pars.num_comp	—	number of F-values to select (whole number) or percent of existing ones (decimal, less than 1) (default: all)
pars.fvals_min	—	min value of F-values range to use (default: -inf)
pars.fvals_max	—	max value of F-values range to use (default: inf)
pars.verbose	—	print detail level (default: 1)

Output :

D.data	—	selected data

※ This filter should be applied to data after division between training data and test data, as the selected data is defined based on the F-values that are calculated only from training data.
If not, the information obtained from the test data is included in F-values. (Double dipping)

shiftData　　Shift data along time dimension

[D, pars] = shiftData(D, pars)

Shift data along the time dimension in each run
This filter changes the relation between data and labels.

Input :

D.data	–	brain activity information
D.design	–	design matrix of experiment (to get run and block information)
D.design_type	–	name of each design type (to find 'run' and 'block')
pars.shift	–	number of time samples to shift data

Optional :

pars.verbose	–	print detail level (default: 1)

Output :

D.data	–	shifted data
D.label	–	labels for shifted data
D.design	–	design matrix of shifted data

※ In case of fMRI, the hemodynamic delay must be considered, so the data should be shifted.

※ This filter is applied to data along the time dimension, so the data should be continuous along the time dimension.
The information about gaps in time is obtained from D.design.

zNorm_bdtb　　Normalize data by z-score

[D, pars] = zNorm_bdtb(D, pars)

Normalize data by z-score along the time / space dimension

Input :

D.data	– brain activity information

Optional :

pars.mode	– 1: training (make mean, std), 2: testing (use mean, std)
pars.smode	– over-riding static mode? 0: no (default), 1: yes (always calculate mean, std)
pars.app_dim	– dimension to normalize along 1: time, 2: space (default)
pars.sub_mean	– subtract mean? 1: yes (default), 2: no
pars.verbose	– print detail level (default: 0)
pars.mu	– mean (calculated in training, used in testing)
pars.sd	– standard deviation (calculated in training, used in testing)

Output :

D.data	– normalized data

liblinear_bdtb Perform LIBLINEAR

[result, pars] = liblinear_bdtb(D, pars)

Use “LIBLINEAR” as the statistical model

Input :

D.data	– brain activity information
D.label	– condition of each sample

Optional :

pars.model	– training result (made in training, used in testing)
pars.mode	– 1: training, 2: testing
pars.verbose	– print detail level (default: 0)
pars.ops	– parameters for “LIBLINEAR”
	Please refer to README of “LIBLINEAR”

Output :

result.model	– ‘liblinear_bdtb’
result.pred	– predicted labels
result.label	– defined labels
result.weight	– weight and bias

libsvm_bdtb　　Perform LIBSVM

[result, pars] = libsvm_bdtb(D, pars)

Use “LIBSVM” as the statistical model

Input :

D.data	– brain activity information
D.label	– condition of each sample

Optional :

pars.model	– training result (made in training, used in testing)
pars.mode	– 1: training, 2: testing
pars.verbose	– print detail level (default: 0)

LIBSVM pars : – parameters for “LIBSVM”

pars.kernel	Please refer to README of “LIBSVM”
pars.cost	
pars.gamma	
pars.coef	
pars.degree	
pars.prob	

Output :

result.model	– ‘libsbm_bdtb’
result.pred	– predicted labels
result.label	– defined labels
result.dec_val	– decision values
result.weight	– weight and bias

slr_lap_bdtb　　Perform SLR-LAP-1vsR

[result, pars] = slr_lap_bdtb(D, pars)

Use “SLR-LAP(sparse logistic regression with Laplace approximation)-1vsR” as the statistical model

Input :

D.data	—	brain activity information
D.label	—	condition of each sample

Optional :

pars.conds	—	conditions to be tested
pars.mode	—	1: training, 2: testing
pars.verbose	—	print detail level (default: 0)

SLR pars :　— parameters for “SLR”

pars.scale_mode　　Please refer to README of “SLR”

pars.mean_mode

SLR pars for test :

pars.weight

pars.ix_eff

pars.norm_scale

pars.norm_base

pars.norm_sep

SLR pars for train :

pars.nlearn

pars.ax0

pars.amax

Output :

result.model	—	‘slr_lap_bdtb’
result.pred	—	predicted labels
result.label	—	defined labels
result.dec_val	—	decision values
result.weight	—	weight and bias

slr_var_bdtb　　Perform SLR-VAR-1vsR

[result, pars] = slr_var_bdtb(D, pars)

Use “SLR-VAR(sparse logistic regression with variational approximation)-1vsR” as the statistical model

Input :

D.data	—	brain activity information
D.label	—	condition of each sample

Optional :

pars.conds	—	conditions to be tested
pars.mode	—	1: training, 2: testing
pars.verbose	—	print detail level (default: 0)

SLR pars :　— parameters for “SLR”

pars.scale_mode　　Please refer to README of “SLR”

pars.mean_mode

SLR pars for test :

pars.weight

pars.ix_eff

pars.norm_scale

pars.norm_base

pars.norm_sep

SLR pars for train :

pars.nlearn

pars.ax0

pars.amax

Output :

result.model	—	‘slr_var_bdtb’
result.pred	—	predicted labels
result.label	—	defined labels
result.dec_val	—	decision values
result.weight	—	weight and bias

smlr_bdtb Perform Multinomial SLR

[result, pars] = smlr_bdtb(D, pars)

Use “Multinomial SLR” as the statistical model

Input :

D.data	—	brain activity information
D.label	—	condition of each sample

Optional :

pars.conds	—	conditions to be tested
pars.mode	—	1: training, 2: testing
pars.verbose	—	print detail level (default: 0)

SLR pars : — parameters for “SLR”

pars.scale_mode
pars.mean_mode

Please refer to README of “SLR”

SLR pars for test :

pars.weight
pars.ix_eff
pars.norm_scale
pars.norm_base
pars.norm_sep

SLR pars for train :

pars.nlearn
pars.ax0
pars.amax

Output :

result.model	—	‘smlr_bdtb’
result.pred	—	predicted labels
result.label	—	defined labels
result.dec_val	—	decision values
result.weight	—	weight and bias

svm11lin_bdtb　　Perform OSU-SVM

[result, pars] = svm11lin_bdtb(D, pars)

Use “OSU-SVM” as the statistical model

Input :

D.data	– brain activity information
D.label	– condition of each sample

Optional

pars.weight	– weight (calculated in training, used in testing)
pars.mode	– 1: training, 2: testing
pars.num_boot	– number of bootstrap samples 0: no bootstrapping >0: number of samples <0: use ‘-num_boot x length(labels)’
pars.verbose	– print detail level (default: 0)

Output :

result.model	– ‘svm11lin_bdtb’
result.pred	– predicted labels
result.label	– defined labels
result.dec_val	– decision values
result.weight	– weight and bias

※ This model can be used on 32-bit only, because the distributed mex file is for 32-bit only.

crossValidate　　Perform leave-one-out cross-validation

[result, P] = crossValidate(D, P, procs, models)

Perform leave-one-out cross-validation

Input :

D.data	– brain activity information
D.label	– condition of each sample
D.design	– design matrix of experiment (to get the basis of grouping)
procs	– names of the processing functions to be called
models	– names of the model functions to be called

Optional :

P.<function>	– parameters of 'procs' and 'models'
P.crossValidate.fold_ind	– index of D.design means which design is used as the basis of grouping(fold) (default: 1)
P.crossValidate.res_train	– return training result also? 0: no (default), 1: yes
P.crossValidate.verbose	– print detail level (default: 1)

Output :

result{}.model	– names of used models
result{}.pred	– predicted labels
result{}.label	– defined labels
result{}.dec_val	– decision values
result{}.weight	– weights and bias
result{}.freq_table	– frequency table
result{}.correct_per	– percent correct

※ The data is divided into some groups based on the experimental design specified by 'fold_ind'.
When the design of run is specified, leave-'one run'-out cross-validation is performed, and when the design of block is specified, leave-'one block'-out cross-validation is performed.

validate Perform validation

[result, P] = validate(D_tr, D_te, P, procs, models)

Validate test data by the statistical models trained using training data

Input :

D_tr.data	– brain activity information for training
D_tr.label	– conditions of each sample for training
D_te.data	– brain activity information for testing
D_te.label	– conditions of each sample for testing
procs	– names of the processing functions to be called
models	– names of the model functions to be called

Optional :

P.<function>	– parameters of 'procs' and 'models'
P.validate.res_train	– return training result also? 0: no (default), 1: yes
P.validate.verbose	– print detail level (default: 1)

Output :

result{}.model	– names of used models
result{}.pred	– predicted labels
result{}.label	– defined labels
result{}.dec_val	– decision values
result{}.weight	– weights and bias
result{}.freq_table	– frequency table
result{}.correct_per	– percent correct

History

Ver. 1.0 2011/08/03

Contact

Satoshi Murata
Research Engineer in ATR Intl. Computational Neuroscience Labs
satoshi-m@atr.jp