

PRoNTo Manual

The *PRoNTo* Development Group
(and honorary members)

John Ashburner
Carlton Chu
Andre Marquand
Janaina Mourao-Miranda
Joao M. Monteiro
Christophe Phillips
Anil Rao
Jonas Richiardi
Jane Rondina
Maria J. Rosa
Jessica Schrouff
Konstantinos Tsirlis
Tong Wu



This is a beta version and is subject to change.

Machine Learning & Neuroimaging Laboratory
Centre for Medical Image Computing
Computer Science Department, UCL
90 High Holborn, Holborn, London WC1V 6LJ, UK

September 28, 2021

<http://www.mlnl.cs.ucl.ac.uk/pronto>

Contents

1	Introduction	11
1.1	Background	11
1.2	Methods	12
1.2.1	Inputs and preprocessing	13
1.2.2	Machine learning algorithms	13
1.3	Installing & launching the toolbox	14
1.3.1	Installation	14
1.3.2	Launching and batching	14
1.3.3	Troubleshooting	15
1.4	What's new?	16
1.4.1	Version 3.0	16
1.4.2	Version 2.1	18
1.4.3	Version 2.0	19
1.4.4	Version 1.1	19
1.4.5	Version 1.0	20
1.5	How to cite	20
1.6	PRoNTo History	20
1.7	Main contributors	21
1.8	Acknowledgements	23
I	Description of PRoNTo tools	25
2	Data & Design	27
2.1	Introduction	27
2.2	The PRT directory	28
2.3	Groups	29
2.4	Subjects/Samples	29
2.5	Modalities/runs	29
2.5.1	Select by samples	30
2.5.2	Select by subjects	32
2.6	Masks	35
2.7	Review data and design	35
2.7.1	HRF correction	36
2.8	Load, Save and Quit	37
2.9	Data & Design output	37
2.10	Batch interface	38
2.11	PRT structure	39
2.11.1	Introduction	39
2.11.2	Changes	40

3 Prepare feature set	41
3.1 Introduction	41
3.2 Feature extraction and pre-processing	42
3.3 Prepare feature set	43
3.3.1 NIfTI and .mat data	44
3.3.2 MEEG data	45
3.4 Batch interface	47
3.5 PRT structure	47
4 Model Specification and Estimation	49
4.1 Introduction	49
4.2 Model specification	49
4.3 Feature set	50
4.4 Model type / pattern recognition algorithm	51
4.4.1 Classification	51
4.4.2 Regression	52
4.4.3 Hyper-parameter optimization	53
4.5 Cross-validation	54
4.6 Model estimation	57
4.7 Batch interface	58
4.8 Model: Specify from	60
4.9 Important changes from PRoNTo v3.0	60
5 Display Model Performance	63
5.1 Introduction	63
5.2 Launching results display	63
5.3 The main results display window	64
5.4 Measuring model performance	65
5.4.1 Classification	65
5.4.2 Regression	65
5.4.3 Permutation testing	67
5.5 Visualizing the model performance	67
5.5.1 Classification	68
5.5.2 Regression	71
5.5.3 Influence of the hyper-parameter on performance	73
6 Computing Feature and Region Contributions	75
6.1 Introduction	75
6.2 Feature weights	76
6.3 Atlas-based weights	77
6.4 Batch interface	77
7 Display weights	79
7.1 Introduction	79
7.2 Displaying weights	80
7.2.1 Select image to display	81
7.2.2 Weights map	83
7.2.3 Anatomical image	83
7.2.4 Additional plots	83
8 List of input files	87

II Batch interfaces	89
9 Data & Design	91
9.1 Directory	91
9.2 Groups	91
9.2.1 Group	91
9.3 Masks	94
9.3.1 Modality	94
9.4 Review	94
10 Feature set/Kernel	95
10.1 Load PRT.mat	95
10.2 Feature/kernel name	95
10.3 Data format	95
10.3.1 Nifti	95
10.3.2 MEEG	96
10.3.3 .mat	97
11 Model: Specify new	99
11.1 Load PRT.mat	99
11.2 Model name	99
11.3 Feature sets	99
11.3.1 Feature set name	99
11.4 Model Type	99
11.4.1 Classification	99
11.4.2 Regression	105
11.5 Cross-validation type	109
11.5.1 Leave one subject out	109
11.5.2 k-folds CV on subjects	109
11.5.3 Leave one subject per group out	109
11.5.4 k-folds CV on subjects per group	109
11.5.5 Leave one block out	109
11.5.6 k-folds CV on blocks	109
11.5.7 Leave one block per class out	110
11.5.8 k-folds CV on block per class	110
11.5.9 Leave one run/session out	110
11.5.10 Custom	110
11.6 Include all scans	110
11.7 Data operations	110
11.7.1 Mean centre features	110
11.7.2 Other Operations	110
12 Model: Run	111
12.1 Load PRT.mat	111
12.2 Model name	111
12.3 Do permutation test?	111
12.3.1 No permutation test	111
12.3.2 Permutation test	111
III Practical Tutorials	113
13 Block design fMRI dataset	115
13.1 GUI analysis	115
13.1.1 Data & Design	116
13.1.2 Prepare feature set	119
13.1.3 Model: Specify new	120
13.1.4 Model: Specify from (optional step)	122

13.1.5	Model: Run	123
13.1.6	Display model (optional step)	123
13.1.7	Display results	124
13.1.8	Compute weights (optional step)	126
13.1.9	Display weights	126
13.2	Batch analysis	126
13.2.1	Data & Design	127
13.2.2	Feature set / Kernel	128
13.2.3	Model: Specify new	130
13.2.4	Model: Specify from (optional step)	132
13.2.5	Model: Run	132
13.2.6	Compute weights (optional step)	132
14	Regression dataset	135
14.1	GUI analysis	135
14.1.1	Data & Design	135
14.1.2	Prepare feature set	136
14.1.3	Model: Specify new	136
14.1.4	Model: Specify from	137
14.1.5	Display results	137
14.2	Batch analysis	139
14.2.1	Data & Design	139
14.2.2	Feature set / Kernel	139
14.2.3	Model: Specify new (KRR)	140
14.2.4	Model: Run (KRR)	141
14.2.5	Model: Specify and Run (RVR and GPR)	142
14.3	Removing confounds (optional)	142
14.4	Within- and between- subject regression	143
15	Multiple Kernel Learning example	145
15.1	GUI analysis	146
15.1.1	Data & Design	146
15.1.2	Prepare feature set	146
15.1.3	Model: Specify new	147
15.1.4	Model: Specify from	147
15.1.5	Model: Run	148
15.1.6	Display model (optional step)	148
15.1.7	Display results	149
15.1.8	Compute weights	150
15.1.9	Display weights	150
15.2	Batch analysis	151
15.2.1	Data & Design	151
15.2.2	Feature set / Kernel	151
15.2.3	Model: Specify new	152
15.2.4	Model: Run	154
15.2.5	Compute weights (optional step)	154
16	Removing confounds: a classification example	155
16.1	Introduction	155
16.2	GUI analysis	156
16.2.1	Data & Design	156
16.2.2	Prepare feature set	157
16.2.3	Model: Specify new	158
16.2.4	Model: Specify from	159
16.2.5	Model: Run	159
16.2.6	Display results	160
16.3	Batch analysis	160

16.3.1	Data & Design	160
16.3.2	Feature set / Kernel	161
16.3.3	Model: Specify new	162
16.3.4	Model: Run	163
16.3.5	Compute weights	163
16.3.6	Display weights	164
16.4	Effects of removing covariates	164
17	Multi-modal face recognition example	167
17.1	GUI analysis	168
17.1.1	Data & design	168
17.1.2	Prepare feature set	172
17.1.3	Model: Specify new	175
17.1.4	Model: Specify from	175
17.1.5	Model: Run	177
17.1.6	Display results	178
17.1.7	Compute weights	178
17.1.8	Display weights	178
17.1.9	Using an atlas with .mat	180
17.2	Batch analysis	181
17.2.1	Data & Design	182
17.2.2	Feature set / Kernel	185
17.2.3	Model: Specify new	186
17.2.4	Model: Run	188
17.2.5	Compute weights	190
17.2.6	Display results & weights	190
17.2.7	Using an atlas with .mat	190
18	Classification of semi-simulated ECoG data	191
18.1	GUI analysis	192
18.1.1	Data & design	192
18.1.2	Prepare feature set	192
18.1.3	Model: Specify new	194
18.1.4	Model: Specify from	195
18.1.5	Model: Run	195
18.1.6	Display results	196
18.1.7	Compute weights	196
18.1.8	Display weights	196
18.2	Batch analysis	197
18.2.1	Data & Design	197
18.2.2	Feature set / Kernel	199
18.2.3	Model: Specify new	200
18.2.4	Model: Run	202
18.2.5	Display results	202
18.2.6	Compute & Display weights	203
19	Non-kernel machine example	205
19.1	GUI analysis	205
19.1.1	Data & Design	205
19.1.2	Prepare feature set	205
19.1.3	Model: Specify new	206
19.1.4	Model: Specify from	206
19.1.5	Model: Run	206
19.1.6	Display results	207
19.1.7	Compute weights	207
19.1.8	Display weights	207
19.2	Batch analysis	208

19.2.1	Data & Design	208
19.2.2	Feature set / Kernel	208
19.2.3	Model: Specify new	208
19.2.4	Model: Specify from	208
19.2.5	Model: Run & Display results	210
19.2.6	Compute & Display weights	210
20	Within-subject Regression	213
20.1	GUI analysis	214
20.1.1	Data & Design	214
20.1.2	Prepare feature set	215
20.1.3	Model: Specify new	215
20.1.4	Model: Run	216
20.1.5	Display results	217
20.2	Batch analysis	218
20.2.1	Data & Design	218
20.2.2	Feature set / Kernel	218
20.2.3	Model: Specify new	219
20.2.4	Model: Run & Display results	220
21	New Machine Tutorial	221
21.1	Introduction	221
21.2	prt_new_machine.m	222
21.2.1	Inputs	222
21.2.2	Outputs	223
21.3	How to import and test your new machine in Batch	223
21.4	How to import your new machine in GUI	227
21.4.1	prt_defaults.m	227
21.4.2	prt_get_machine.ui.m	228
21.4.3	prt_ui_copy_model.m	228
21.4.4	prt_plot_nested_cv.m	229
21.4.5	prt_weights_*.m	229
21.4.6	Running your new machine	229
IV	Advanced topics	231
22	Developer's manual	233
22.1	Introduction	233
22.2	PRoNTo folder structure	234
22.3	Data & Design	235
22.3.1	PRT fields created	235
22.3.2	Files created	235
22.3.3	GUI behaviour	235
22.3.4	Batch behaviour	236
22.3.5	Functions called	236
22.4	Prepare feature set	237
22.4.1	PRT fields created	237
22.4.2	Files created	237
22.4.3	GUI behaviour	238
22.4.4	Batch behaviour	238
22.4.5	Functions called	238
22.5	Model: Specify new/from	239
22.5.1	PRT fields created	240
22.5.2	Files created	241
22.5.3	GUI behaviour	241
22.5.4	Batch behaviour	241

22.5.5	Functions called	242
22.6	Model: Run	243
22.6.1	PRT fields created	245
22.6.2	Functions called	245
22.7	Compute weights	246
22.7.1	PRT fields created	247
22.7.2	Files created	247
22.7.3	Functions called	247
23	PRoNTo functions and the PRT structure	249
23.1	List of PRoNTo functions	249
23.2	The PRT structure	249
V	Appendix	251
24	Appendix	253
24.1	One data file per subject	253
24.2	Compute atlas for connectivity matrix	255
24.3	Connectivity matrix from MEEG	257
24.4	Connectivity ROI weights	258
VI	Bibliography	265

Chapter 1

Introduction

Contents

1.1	Background	11
1.2	Methods	12
1.2.1	Inputs and preprocessing	13
1.2.2	Machine learning algorithms	13
1.3	Installing & launching the toolbox	14
1.3.1	Installation	14
1.3.2	Launching and batching	14
1.3.3	Troubleshooting	15
1.4	What's new?	16
1.4.1	Version 3.0	16
1.4.2	Version 2.1	18
1.4.3	Version 2.0	19
1.4.4	Version 1.1	19
1.4.5	Version 1.0	20
1.5	How to cite	20
1.6	PRoNTo History	20
1.7	Main contributors	21
1.8	Acknowledgements	23

1.1 Background

Advances in neuroimaging techniques have radically changed the way neuroscientists address questions about functional anatomy, especially in relation to behavioural and clinical disorders. Many questions about brain function, previously investigated using intracranial electrophysiological recordings in animals can now be addressed non-invasively in humans. Such studies have yielded important results in cognitive neuroscience and neuropsychology. Amongst the various neuroimaging modalities available, Magnetic Resonance Imaging (MRI) has become widely used due to its relatively high spatial and temporal resolution, and because it is safe and non-invasive. By selecting specific MRI sequence parameters, different MR signals can be obtained from different tissue types, giving images with high contrast among organs, between normal and abnormal tissues and/or between activated and deactivated brain areas. MRI is often sub-categorized into structural MRI (MRI) and functional MRI (fMRI). Positron Emission Tomography (PET) is another example of neuroimaging modality which measures metabolic processes. Examples of other of imaging modalities that measure brain signals are ElectroEncephaloGraphy (EEG) recordings and MagnetoEncephaloGraphy (MEG) recordings. Neuroimaging data are inherently multivariate, since each measure (scan or recording) contains information from thousands of locations (e.g. voxels in MRI or electrodes in EEG). Considering that most brain functions are distributed processes involving a network of brain regions, it would seem desirable to use the spatially distributed information contained in the data to give a better understanding of brain functions in normal and abnormal conditions.

The typical analysis pipeline in neuroimaging is strongly rooted in a mass-univariate statistical approach, which assumes that activity in one brain region occurs independently from activity in other regions. Although this has yielded great insights over the years, specially in terms of function localization, and continues to be the tool of choice for data analysis, there is a growing recognition that the spatial dependencies among signal from different brain regions should be properly modelled. The effect of interest can be subtle and spatially distributed over the brain - a case of high-dimensional, multivariate data modelling for which conventional tools may lack sensitivity.

Therefore, there has been an increasing interest in investigating this spatially distributed information using multivariate pattern recognition approaches, often referred to as multi-voxel pattern analysis (MVPA) (see [18], [11] and [19]). Where pattern recognition has been used in neuroimaging, it has led to fundamental advances in the understanding of how the brain represents information and has been applied to many diagnostic problems. For the latter, this approach can be used to predict the group membership of the patient scanned (healthy vs. patients or disease A vs. B) and can provide the discriminating pattern leading to this classification. Pattern recognition techniques can also be used to identify relationships between patterns of brain structure or activity and continuous measures such as age or a clinical score. Such information can then be used to predict individual-level measures for new individuals (i.e. regression models).

Several active areas of research in machine learning are crucially important for the difficult problem of neuroimaging data analysis: modelling of high-dimensional multivariate time series, sparsity, regularisation, dimensionality reduction, causal modelling, and ensembling to name a few. However, the application of pattern recognition approaches to the analysis of neuroimaging data is limited mainly by the lack of user-friendly and comprehensive tools available to the fundamental, cognitive, and clinical neuroscience communities. Furthermore, it is not uncommon for these methods to be used incorrectly, with the most typical case being improper separation of training and testing datasets.

Note: PRoNTo (Pattern Recognition for Neuroimaging Toolbox) is first and foremost a machine learning tool used for neuroimaging analyses, so throughout the manual we mostly refer to examples that focus on neuroimaging data. That being said, since from the current release PRoNTo supports MEEG (SPM) data as well as any .mat file, one can use PRoNTo to analyze for example MEEG data in sensor space (2D, or even 1D), or yet any kind of .mat file, such as behavior data. However, it is best to use PRoNTo primarily for neuroimaging data which is its original purpose or to build multi-modal predictive models including imaging and non-imaging information. So from the current release, we will refer to our data with the general term ‘samples’ and ‘features’, instead of referring to ‘voxels’ as our features. The reason is that from the current release we have expanded our functionalities to include data that are not 3D, and therefore they do not have ‘voxels’ as features.

1.2 Methods

PRoNTo (Pattern Recognition for Neuroimaging Toolbox) is a toolbox based on pattern recognition techniques for the analysis of neuroimaging data. Statistical pattern recognition is a field within the area of machine learning which is concerned with automatic discovery of regularities in data through the use of computer algorithms, and with the use of these regularities to take actions such as classifying the data into different categories [3]. In PRoNTo, brain images are treated as spatial patterns and statistical learning models are used to identify statistical properties of the data that can be used to discriminate between experimental conditions or groups of subjects (classification models) or to predict a continuous measure (regression models).

PRoNTo is MATLAB-based and includes six main modules: ‘Data & Design’, ‘Prepare feature set’, ‘Model: Specify new’, ‘Model: Specify from’, ‘Model: Run’ and ‘Compute weights’. For a specific model PRoNTo can display results in terms of performance as well as the model’s weights. Additional review options enable the user to review information about the data, features and models. All modules were implemented using a graphical user interface (GUI) and the MATLAB Batch System. Using the MATLAB Batch System the user can run each module as batch jobs, which enables a very efficient analysis framework. All information about the data, experimental design, models and results are saved in a structure called PRT. PRoNTo also creates additional files during the analysis that are described in details in the next chapters.

The toolbox code is distributed for free, but as copyright software under the terms of the GNU General Public License as published by the Free Software Foundation.

1.2.1 Inputs and preprocessing

In terms of neuroimaging modalities, PRoNTo accepts NIfTI files, which are files mostly designed to analyse structural and functional MRI as well as PET. In case of NIfTI files, it assumes that the neuroimaging data has been previously pre-processed using SPM (<http://www.fil.ion.ucl.ac.uk/spm/>) or a similar software for neuroimaging analysis. In general, raw fMRI data should be previously corrected for movement artefact (realigned) and time difference in slice acquisition (slice time correction), mapped to a common template (normalized) and spatially smoothed. The normalisation and spatial smoothing steps might not be necessary for single subject analysis. In addition, the General Linear Model (GLM) can be applied as a pre-processing step for pattern recognition analysis. In this case, the GLM coefficients (e.g. beta or contrast images from SPM) will correspond to the spatial patterns. Using beta images should be preferred instead of raw data in the case of event-related designs with short inter-stimulus time and/or event duration to better take into account the Haemodynamic Response Function (HRF). **Important note:** Beta images output by SPM contain NaNs (Not a Number) values in some voxels/features. For better performance of the model, a mask should be created to exclude the corresponding voxels/features from the analysis. Practically, a mask should be built (e.g. using SPM imcalc batch or a script provided in ‘utils’) to specify which voxels/features have scalar values (0 and 1 in mask) or NaN. The updated mask should then be input in the Data & Design window. To ease this extra preprocessing step, a script is provided in the PRoNTo folder *utils*. It takes as inputs the images (i.e. beta images in NIfTI format), the mask to update (e.g. SPMnoeyes.nii provided in the PRoNTo folder *mask*) and the directory to save the updated mask. Inputs are optional, and just typing in the MATLAB command: `prt_utils_update_mask` will ask for the different inputs using file and path selectors.

Raw structural MRI data should be previously mapped to a common template (normalized) and spatially smoothed. Raw PET data should be realigned, normalized and smoothed. PET data is also usually scaled. This operation can be performed before hand or during the building of the feature set.

Another type of neuroscience data supported by PRoNTo is electrophysiological data. These can be 1D or 2D EEG, MEG, LFP or ECoG. PRoNTo accepts this type of data in the MEEG data format, which is the standard data format created and used by SPM. PRoNTo assumes that this data format represents a within-subject design and it will read all the information automatically. The data has to contain epoched trials, either one per stimulus or one per condition. Only one MEEG file can be entered per subject and modality.

The third data format supported by PRoNTo is .mat files. For example, 1D or 2D psychometric data, or connectivity matrices, can be built in .mat files, according to PRoNTo specifications, and analyzed. The variables need to be numerical (no categorical inputs) and saved as an array or vector in the first variable of the file, with no restriction on the dimensions.

A limitation PRoNTo has is that for some computational reasons it cannot accept the standard format that is usually used for data of this type, i.e. #samples/subjects X #features. Instead, it required one .mat file per sample/subject. PRoNTo provides scripts where one can easily do these operations. Furthermore, the dimensionality of the data needs to be consistent across samples, i.e. same number of dimensions and variables in the same order.

Finally, the model weights can be displayed only for 1D and 2D MEEG or .mat inputs. The display has not been specialized for MEEG as there are many much better tools to explore such files, either directly in SPM or converting back to your favourite software.

1.2.2 Machine learning algorithms

In PRoNTo different pattern recognition algorithms correspond to different machines. The machine library in PRoNTo v3.0 includes three pattern classification algorithms: Support Vector Machine ([5], [17]), Gaussian Process Classifier (binary and multiclass, [22], [14]) and L1 Multi-Kernel Learning [20]. There are also five pattern regression algorithms available: Kernel Ridge Regression [26], Relevance Vector Regression [28], Gaussian Process Regression [22], linear epsilon-insensitive SVM (ϵ -SVM) regression, and L1 Multi-Kernel regression

[20]. From the current release there is also the option of using non-kernel machines in addition to the kernel ones. So there are 5 available non-kernel classification algorithms: Binary SVM using L1 and L2 loss, Multiclass SVM, as well as Logistic Regression using L1 and L2 loss ([4, 6]). Finally, for regression there is only the linear epsilon-insensitive SVM (ϵ -SVM) at the moment.

PRoNTo was developed with the aim of facilitating the interaction between machine learning and the neuroimaging communities. On one hand the machine learning community should be able to contribute to the toolbox with novel published machine learning models. On the other hand, the toolbox should provide a variety of tools for the neuroscience and clinical neuroscience communities, enabling them to ask new questions that cannot be easily investigated using existing statistical analysis tools.

1.3 Installing & launching the toolbox

In order to work properly, PRoNTo requires 2 other softwares:

- A recent version of MATLAB. We used versions 7.5 (R2007b) to 9.4 (R2018a) to develop PRoNTo, and PRoNTo will not work with earlier versions¹. The statistics toolbox of MATLAB is also required. Note: Windows and Mac users may encounter compiler issues when using R2016b and R2017b. Files may need to be compiled again. Users can install the MATLAB required compiler from Add-Ons, or by manually downloading MinGW.
- SPM12 [15] installed on your computer². Users are recommended to have the latest updates by typing: spm_update (check if there are new updates available) and spm_update update (start updating files). **Important note:** if you already have a version of SPM on your computer, please update it. Various bugs, especially in terms of weight visualization, arise from out of date SPM versions.

PRoNTo latest public version can be downloaded, after registration, from the following address: <http://www.mlnl.cs.ucl.ac.uk/pronto/prtsoftware.html>.

1.3.1 Installation

After downloading the zipped file containing PRoNTo, the installation proceeds as follow:

1. Uncompress the zipped file in your favourite directory, for example C:\PRoNTo\;
2. Launch MATLAB;
3. Go to the “File” menu → “Set path”;
4. Click on the “Add folder” button and select the PRoNTo folder, i.e. C:\PRoNTo\ if you followed the example;
5. Click on save.

Some routines, in particular the ‘machines’, are written in C++ (.cpp files) for increased efficiency. We are trying to provide these compiled routines for the usual Operating Systems (OS’s) such as: Windows XP (32 bits), Windows 7/10 (64 bits), Mac OS 10 and Linux (32 and 64 bits). If your OS is not listed or routines do not work properly then you should compile the routines for your specific OS.

1.3.2 Launching and batching

Once installed, there are three ways to call up PRoNTo functionalities. To launch the toolbox GUI, just type `prt` or `pronto` at the MATLAB prompt and the main GUI figure will pop up, see Fig. 1.1. From there on simply click on the processing step needed (see Part I of this manual). Most functions of PRoNTo have been integrated into the `matlabbatch` batching system [8] (like SPM8) and the batching GUI is launched from the main GUI by clicking on the Batch button (see Part II of this manual). Of course most tools can also be called individually by calling them directly from the MATLAB prompt, or for scripting in a .m file (see Part IV of this manual).

¹Any later MATLAB version should work, *in principle*.

²SPM12 can be dowloaded from the following website: <http://www.fil.ion.ucl.ac.uk/spm/software/>. You should install it in a suitable directory, for example C:\SPM12\, then make sure that this directory is on the MATLAB path. No need to include the subdirectories!

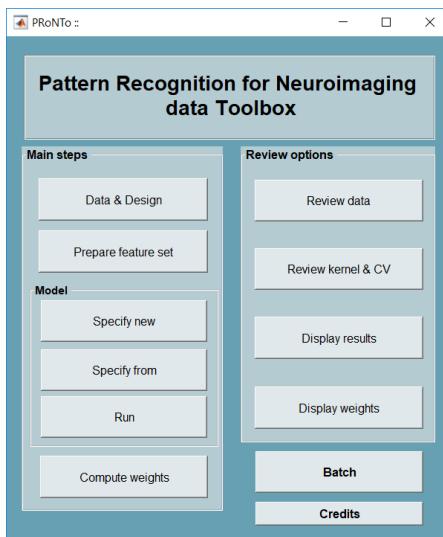


Figure 1.1: Main GUI interface: each button launches a specific processing step.

1.3.3 Troubleshooting

MEX files are provided for 64 bit Windows, MacOS and Linux (Ubuntu/Debian) systems. However, if your system specifications do not align with those above, making new MEX files is necessary. In such cases you will most likely hit an error. Please follow the instructions below in case this happens.

Compiling LIBSVM

Some problems when using SVMs might also arise due to LIBSVM, in which case, you might need to compile it on your own. The first thing that needs to be done is to download the desired LIBSVM version (usually the latest one) from the following website: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>. Then, the process will depend on your operating system.

If the steps described below do not work, please refer to the README file that comes with LIBSVM.

Microsoft Windows

- Make sure you have a C++ compiler installed. If not, you can install Microsoft Visual C/C++;
- Copy the libsvm folder to the ‘machines’ directory of your PRoNTo installation (e.g. C:\PRoNTo\machines\);
- Open a DOS command window and change to the libsvm folder in the previous step (`cd C:\PRoNTo\machines\libsvm-3.17`). If the environment variables of VC++ have not been set, run the following command:
`C:\Program Files\Microsoft Visual Studio 10.0\VC\bin\vcvars32.bat`. This command might be different, depending on the path of your Visual Studio installation;
- In the libsvm folder run the command: `nmake -f Makefile.win clean all`
- If no errors appear, open MATLAB;
- Change to the ‘matlab’ folder inside the libsvm folder (e.g. C:\PRoNTo\machines\libsvm-3.17\matlab\);
- Run `make` in the MATLAB Command Window. If there are no errors, you have just successfully compiled LIBSVM to be used with MATLAB.

Remember, if you want to use the version that you have just compiled, you have to add the libsvm folder to your path in MATLAB. If you have more than one libsvm folder inside the ‘machines’ folder, please remove one of them from the MATLAB path. You should only have one libsvm folder in your path.

Unix (Mac OS or Linux)

- Make sure you have a C++ compiler installed. If you are using Mac OS, please install ‘Xcode’. On Linux systems, you should already have ‘gcc’ installed;
- Copy the libsvm folder to the ‘machines’ directory of your PRoNTo instalation (e.g. `/home/<username>/PRoNTo/machines/`);
- Open a terminal window and change to the ‘machines’ directory: `cd PRoNTo/machines/`
- Compile libsvm by running the following command: `make`
- If no errors appear, open MATLAB;
- Change to the ‘matlab’ folder inside the libsvm folder (e.g. `PRoNTo/machines/libsvm-3.17/matlab/`);
- Run `make` in the MATLAB Command Window. If there are no errors, you have just successfully compiled LIBSVM to be used with MATLAB.

Remember, if you want to use the version that you have just compiled, you have to add the libsvm folder to your path in MATLAB. If you have more than one libsvm folder inside the ‘machines’ folder, please remove one of them from the MATLAB path. You should only have one libsvm folder in your path.

Compiling GPML

In case there are compiler problems when using GPs due to GPML, you will need to compile it on your own. The instructions to do so are approximately the same as with LIBSVM.

- Inside MATLAB, change to the ‘util’ folder inside the gpml folder (e.g. `PRoNTo/machines/gpml/gpml-matlab-v3.5-2014-12-08/util/`);
- Run `make` in the MATLAB Command Window. If there are no errors, you have just successfully compiled GPML to be used with MATLAB.

1.4 What’s new?

1.4.1 Version 3.0

This version of the toolbox has a new layout, and includes multiple new features and functionalities. Below is a list of selected major changes and new functionalities:

File formats

PRoNTo now accepts multiple ‘Data formats’, i.e. NIfTI (as before), .mat and MEEG SPM data.

- **MEEG:** These can be 1D or 2D EEG, MEG, LFP or ECoG files converted to the MEEG SPM data format and can be entered when choosing ‘Select by Subject’. PRoNTo assumes that this file format represents a within-subject design that it will read automatically. The data has to contain epoched trials, either one per stimulus or one per condition. Only one MEEG file can be entered per subject and modality.
- **.mat:** Any set of variables can be saved in a .mat and used in PRoNTo. The variables need to be numerical (no categorical inputs) and saved as an array or vector (no restriction on dimension) in the first variable of the file. One .mat needs to be provided per sample. The dimensionality of the data needs to be consistent across samples (i.e. same number of dimensions and variables in the same order). The model weights can be displayed only for 1D and 2D MEEG or .mat inputs. The display has not been fine-tuned for MEEG as there are many tools available to display such files (either directly in SPM or converting back to your favourite software).

Combine multiple types of data

In previous versions of PRoNTo, modalities could only be combined if they had the same number of features. The framework has been modified to allow the combination of modalities with different number of features at the model level. The option for ‘building one kernel per modality’ in the ‘Prepare feature set’ module has been removed. This means that, at the feature set step, only runs of a same data format can be concatenated as samples. At the model step, multiple feature sets can now be included in a model. These will be either concatenated (if a single kernel machine is selected) or used in multiple kernel settings during model estimation.

Important note: when combining multiple feature sets, the number (and the order) of the samples need to be identical across feature sets. This means that if there are 3 conditions in the MEEG data saved as ‘condB’, ‘condA’, ‘condC’, then it is not enough for the equivalent beta images for fMRI to include the conditions ‘condA’, ‘condB’ and ‘condC’. These also need to be entered in the exact same order! It might not throw an error but will lead to poor performance of the model.

Subsampling classes

PRoNTo now allows to subsample classes to match the number of samples in the smallest class (or as close as possible) while taking into account the stratified structure of the data (e.g. blocks of scans in fMRI) as well as potential pooling of multiple conditions (i.e. it will subsample equally from all pooled conditions). Related to this option, a new module has been implemented: ‘Specify from’. This module (batch and GUI) allows to choose a model that has been previously specified as a ‘basis’ for a new model. Some fields of this ‘basis’ model will be copied into the new model, including class or regression sample selection and outer cross-validation. This ensures that, when performing subsampling, the exact same samples are considered in the ‘basis’ model and in the new model to ease performance comparison.

Important note: Subsampling uses the standard random number generator implementations of MATLAB. Different issues have been reported about the generation of different random sequences across different Operating Systems, or even different versions of the same Operating Systems due to various reasons, which could affect your results and their reproducibility. If total reproducibility is a must, the user is advised to set their own random number generator. This can be done by modifying the script *prt-model.m* in line #338 where the standard random number generator is set.

Machines

- **Kernel machines:** PRoNTo interfaces LIBSVM. However, only the SVM algorithm was used so far. PRoNTo now interfaces LIBLINEAR as well and uses more algorithms from both libraries. In addition, the ‘custom’ machine can now be optimized given hyper-parameters. PRoNTo also allows the optimization of more than one parameter (entered as a cell array of values that is then transformed into a grid).
- **Non-kernel machines:** The non-kernel route has also been enabled, i.e. the ‘Use kernel’ radio button can be ticked off to use primal formulations instead of kernel machines. This is intended for samples with a low number of features. Please note that using these algorithms on typical neuroimaging datasets (i.e. # features >> # samples), will likely lead to time-consuming estimations and poor performance.

Regression targets

- **Per subject:** Multiple regression targets can be specified at the Data & Design level. When choosing which samples to use for the regression model, the user can also specify which target to use. This avoids the need for multiple PRT files when multiple regression targets might be of interest.
- **Per trial:** Regression targets can be input along the conditions of an experimental design to perform within-subject regression.

Cross-validation

A ‘Leave-One-Block-per-Class-Out’ cross-validation scheme was added with its k-folds counterpart to allow balanced train and test sets when performing within subject cross-validation.

Other important changes

- **Model performance estimation:** Model performance was computed based on the concatenation of predictions across folds. This way to estimate model performance can however lead to over-optimistic model performance estimations and further reflects a model that was not estimated (instead, we estimated multiple models). We have hence modified the code to compute the average of performance across folds. This is also reflected in the ‘Display results’ window where some plots have been replaced at the model level (e.g. replacing the overall confusion matrix by a ‘balanced accuracy distribution’ across folds in the plot). In general, the results will reflect more the average but also the deviation of the results across folds. We encourage the users to report both values. Furthermore, there has also been a change in the way we compute the normalized MSE. Until PRoNTo v2.1, the MSE was normalized by dividing it with the range of values ($\max(\text{targets}) - \min(\text{targets})$). The main disadvantage of this was that in the case of spurious outliers, the results were not representative of how good the model fit the data. So from PRoNTo v3.0 the MSE is normalized by dividing the MSE with the *variance* of the target values.
- **Gaussian Process Regression:** Until PRoNTo v2.1, the covariance matrix used was $k(x, z) = (x' * z) / t2$ with $t2$ being the optimizable hyperparameter controlling the scaling of the latent function. After some empirical results showing that this covariance matrix was overparameterized, from PRoNTo v3.0 we change the covariance matrix to $k(x, z) = (x' * z)$, which has no hyperparameters.
- **Hyper-parameter optimization:** This change only applies when multiple values of the hyper-parameter lead to the same maximum value of model performance. In previous versions, PRoNTo was choosing the median value. In v3.0, PRoNTo identifies the largest stable region (if the ‘Image Processing’ toolbox from MATLAB is present) and chooses the center of gravity of this region in the hyper-parameter grid as the best hyper-parameter. This change should have limited impact on the results.
- **Increased flexibility:** Many checks and errors have been simplified or discarded to allow more flexible analyses. It however comes with the downside that it is easier to make mistakes in the analysis. Please make sure you double-check every step and regularly load the PRT to verify the input information.

MATLAB important changes

We noticed a change between MATLAB versions R2017b and R2018a in the MATLAB **norm** function which is used by PRoNTo when normalizing samples. And hence, users should be aware that when normalizing samples, results might vary between MATLAB versions up to R2017b and from version R2018a and onwards.

1.4.2 Version 2.1

This version of the toolbox is released mainly to provide a bug-fixed version of v2.0. One new functionality is added: removing confounds. A detailed description of this procedure can be found at http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/research/Research_Notes/RN_17_09_ARJMM.pdf.

Below is a list of selected bug fixes that might affect your use of the software or previously obtained results:

- **HRF delay and overlap:** In the Data&Design module users can define parameters to approximate the HRF shape (namely HRF delay and overlap). In earlier versions, the values could only be entered once and applied to all modalities. This has now been fixed to allow different values for different modalities, in the batch and in the GUI.
- **Freeze the Review Data window:** In relation to the HRF delay and overlap bug fix, the Review data is now ‘on wait’ until closed. This means that no operation (in PRoNTo or in MATLAB) can be done until the window is closed. This ensures that values input in the Data review GUI are passed to the main Data and Design window.
- **ROC and AUC:** There was a bug in how we computed the area under the curve (AUC) and plotted the receiver operating characteristic (ROC) curve that we have now fixed. This means that different AUC values and ROC plots might be observed, if comparing previously obtained results in older versions and in v2.1. We hence recommend re-computing AUC and re-plotting ROC if needed.

- **k-fold CV:** Before v2.1, when using k-fold cross validation (CV) and the number of samples was not divisible by k, we re-allocated the remainder to the last fold by default. For example, a 4 fold CV with 43 samples would lead to 4 folds of size 10,10,10 and 13 respectively. In v2.1, the remainder is evenly re-allocated to each fold, starting from the first. The above example would then lead to folds of size 11, 11, 11 and 10. This change affects all k-fold CV scheme and should then be taken into consideration if comparing results from older versions with results from v2.1.

1.4.3 Version 2.0

This version of the toolbox (2015) aims at providing multiple new functionalities, including:

- **Build one kernel per modality:** When there are multiple modalities in the same dataset, it is now possible to build one kernel per modality and use them in the same model later on. If this option is not chosen, the selected modalities can still be concatenated as additional samples/examples or sessions (if they have the same features) or used separately.
- **Build one kernel per region:** It is also possible to specify an atlas, comprising regions of interest (ROIs) as defined by values in the atlas ranging from 1 to *number of ROIs*. For each anatomically defined ROI, a kernel will be built taking into account the multivariate pattern within the region. It is possible to use this feature in combination with the previous one (e.g. one kernel per region and per modality, or concatenate modalities as additional samples and build one kernel per region).
- **New machines:** The list of available machines was extended, and now includes L1 Multi-Kernel Learning (MKL, classification and regression). The latter corresponds to a hierarchical model defined by weights at two levels: the feature level and the kernel level (i.e. ROI and/or modality).
- **Flexible cross-validation:** A GUI is provided to manually specify a custom cross-validation matrix. It allows to either specify a basis (e.g. leave-one-subject-out), load a .mat or specify the number of folds. Then the user can, for each fold, select which examples are part of the training set, test set, or won't be used. The resulting matrix can be saved for further use.
- **Nested cross-validation for hyperparameter optimization:** It is now possible to optimize the hyperparameter(s) of some machines (e.g. the soft-margin parameter, C, in SVM) using a nested cross-validation (CV) framework. The number of folds in the nested cross-validation, used only to estimate the value of the hyperparameter leading to the highest performance, does not need to be the same as the 'outer' cross-validation (i.e. the one estimating the final model performance). For example, to decrease computational expenses, the nested CV can be a 4-fold CV while the outer CV can be a leave-one-out.
- **Display results:** The display of the results was divided into two modules (Display Results and Display Weights). This allows to review the model performance in one window, with all the statistics. A new graph displaying the effect of the hyperparameter (if optimized) was also included.
- **Weights per ROI:** For each model it is possible to build images representing the weights per feature and also images summarising the weights per regions of interest as defined by an atlas. If an MKL model was built on ROIs, the contribution of each ROI (regional weight) is explicitly derived. On the other hand, if a simple kernel model was selected (e.g. SVM on the whole brain), the weights per feature will be averaged (in absolute value) within each region, as defined by an atlas specified by the user. In both cases, an additional image, with weights per ROI, is created and saved.
- **Display weights:** The weights of each model can be displayed at the feature level in this window. If weights per ROI were derived (either summarized or from an MKL on ROIs), weights per region can be displayed as an image, as well as in a sorted list of regions. The same applies for MKL models on multiple modalities. A histogram is also displayed representing the contribution/weight of each ROI/modality to the model. The table can be exported as text for future use in publications/communications.

1.4.4 Version 1.1

In 2012, PRoNTo v1.1 was released mainly to provide bug fixes for version v1.0. Two features were also added:

- automatic compiling of the machines (in particular: no more issues with SVM, nor with MATLAB toolboxes and paths).

- k-folds Cross-Validation (CV): specify the number of folds or set it to 1 for half-half CV (train on first half, test on second).

1.4.5 Version 1.0

Launched in 2011, this version of PRoNTo allows to perform all the analysis steps, from Data & Design to computing the weights for three classification machines (SVM, binary and multi-class GP) and two regression machines (KRR, SVR).

1.5 How to cite

Please cite:

- Schrouff J, Rosa MJ, Rondina JM, Marquand AF, Chu C, Ashburner J, Phillips C, Richiardi J, Mourao-Miranda J. [PRoNTo: Pattern Recognition for Neuroimaging Toolbox](#). Neuroinformatics, 2013, 11(3), 319-337.
- Schrouff J, Mourao-Miranda J, Phillips C, & Parvizi J. [Decoding intracranial EEG data with multiple kernel learning method](#). Journal of neuroscience methods, 2016, 261, 19-28.

When using PRoNTo analyses in any type of publication. In addition, when using Multiple Kernel Learning to combine signals from different Regions of Interest (ROIs) or modalities, one should also refer to the following publication:

- Schrouff J, Monteiro, JM, Portugal L, Rosa MJ, Phillips C, Mourao-Miranda J. [Embedding Anatomical or Functional Knowledge in Whole-Brain Multiple Kernel Learning Models](#). Neuroinformatics, 2018, 16(1), 117-143.

And finally when using a posteriori weight summarization with atlas-defined regions of interest one should also refer to:

- Schrouff J, Cremers J, Garraux G, Baldassarre L, Mourao-Miranda J, Phillips C. [Localizing and Comparing Weight Maps Generated from Linear Kernel Machine Learning Models](#). International Workshop on Pattern Recognition in Neuroimaging (PRNI), 2013, 124-127, DOI: 10.1109/PRNI.2013.40.

1.6 PRoNTo History

The first version of the ‘Pattern Recognition for Neuroimaging Toolbox’, aka. PRoNTo, was developed in 2011 by an international team of researchers (the PRoNTo development group) led by Prof Janaina Mourao-Miranda and supported by the European Union through the PASCAL Harvest programme ([PASCAL2](#)).

The original development group included Janaina Mourao-Miranda, Christophe Phillips, Jessica Schrouff, John Ashburner, Maria Joao Rosa, Jonas Richiardi, Andre Marquand, Jane Rondina and Carlton Chu. The motivation to develop PRoNTo stemmed from the unmet need of a flexible pattern recognition analysis framework that would accommodate different types of neuroimaging data, could address various research questions, and be safely used by neuroimagers who are non-expert in machine learning. As the team included SPM developers (John Ashburner and Christophe Phillips) PRoNTo took advantage of many of the SPM existing functions, such as those for file handling and image display, as well as the batch system ([Glauch V](#)).

Over the years, new contributors have joined the development group and PRoNTo was further extended to include new functionalities, such as flexible cross-validation frameworks, option to remove effect of confounds, and atlas based multiple kernel learning (PRoNTo versions 1.1, 2.0, 2.1). In particular, Jessica Schrouff has been a major contributor of PRoNTo from its inception till version 3.0. PRoNTo’s latest version (PRoNTo v3.0) now accepts multiple data formats and enables the building of multimodal predictive models. The latest versions of PRoNTo have been supported by the [Wellcome Trust](#).

Currently, the PRoNTo project is led by Prof Janaina Mourao-Miranda and co-supervised by Prof Christophe Phillips. Konstantinos Tsirlis supports the PRoNTo developments and is in charge of its management and maintenance. Contributions (e.g. new features, debugging, manual, demonstration data) are mostly provided by Prof Janaina Mourao-Miranda's group and collaborators but external contributions are welcome.

1.7 Main contributors

PRoNTo is developed by the Machine Learning & Neuroimaging Laboratory, Computer Science department, University College London, UK (<http://www.mlnl.cs.ucl.ac.uk>) and associated researchers.

The main contributors, in alphabetical order, are:

Dr. John Ashburner is a Professor of Imaging Science at the Wellcome Trust Centre for Neuroimaging at the University College London Institute of Neurology. He is mainly interested in modelling brain anatomy from MR scans, and more recently in applying pattern recognition methods to make predictions about individual subjects. He is a co-developer of the SPM software (intra- and inter-subject registration, tissue classification, visualization and image file formats), which is used internationally by thousands of neuroimaging researchers. He has a Web of Science h-index of 98. He did not contribute any actual code to PRoNTo, but he did attend many of the meetings;

Dr. Carlton Chu is a research scientist at Google DeepMind. Before joining DeepMind, he was a research fellow in brain imaging at the National Institute of Mental Health (NIMH), NIH. He received the B.Eng. degree (1st class Honours) from Auckland University, in 2002 and the master of Biomedical Engineering from University of New South Wales, in 2004. Carlton obtained a PhD in Neuroimaging method from University College London in 2009, working in the statistical methods group at the Wellcome Trust Centre for Neuroimaging, creators of the famous "SPM" program. There he developed innovative pattern recognition methods to automatically detect the early stages of neurodegenerative diseases such as Alzheimer's and Huntingdon's from structural brain images. In 2007, Carlton won the first prize in the 2nd Pittsburgh Brain Activity Interpretation Competition (PBAIC), a prestigious international competition involving the application of machine learning to the problem of classification of brain activity. He led a small research team to victory, acclaim from peers in the field, and the \$10K first prize. His current research interests include image segmentation using convolutional neural networks and applications of deep-learning. Carlton was involved in the development of PRoNTo v1.0;

Dr. Andre Marquand is a Principal Investigator and an assistant professor at the Donders Institute for Brain Cognition and Behaviour. His research focuses on the application of probabilistic machine learning techniques to neuroimaging data, particularly for clinical applications. His recent work includes the development of multi-class, multi-task and multi-modality pattern classification methods that offer many advantages over current techniques including more sensitive and specific detection of disease effects. He did his PhD in Clinical Neuroscience, King's College London, UK. Andre was involved in the development of PRoNTo v1.0;

Prof Janaina Mourao-Miranda is a Wellcome Trust Senior Research Fellow and a Professorial Research Associate at the Centre for Medical Image Computing (CMIC), Computer Science Department, UCL. Over the past years her research has involved developing and applying pattern recognition methods to analyze neuroimaging data, in particular brain activation and structural patterns that distinguish between controls and patients. Her current research focuses on developing machine-learning models to investigate complex relationships between neuroimaging data and multidimensional descriptions of mental health disorders. Janaina has been leading the PRoNTo project since 2011 and has contributed to PRoNTo v1.0, v1.1, v2.0, v2.1 and v3.0;

Dr. Joao M. Monteiro is a former MPhil/PhD Student at University College London under the supervision of Prof John Shawe-Taylor and Dr. Janaina Mourao-Miranda. His research focused on the application of unsupervised machine learning methods to neuroimaging. He contributed to PRoNTo v1.1, v2.0 and v2.1;

Dr. Christophe Phillips is Research Director at the GIGA Cyclotron Research Centre in vivo imaging and Associate Professor at the Department of Electrical Engineering and Computer Science, University of Liège, Belgium. His research focuses on the processing of multi-modal neuroimaging data. Recent work within the field of "brain decoding" aimed at distinguishing between levels of consciousness in unresponsive

patients or between typical and atypical Parkinson Disease patients using Positron Emission Tomography (PET) imaging, as well as tracking mnesic traces in trained healthy subjects with fMRI. Christophe contributed to PRoNTo v1.0, v1.1, v2.0, v2.1 and v3.0;

Dr. Anil Rao was a post-doctoral researcher in Centre for Medical Image Computing (CMIC) and Max Planck University College London Centre for Computational Psychiatry and Ageing Research at the University College London. He contributed to version 2.1, in particular to the removing confounds functionality.

Dr. Jonas Richiardi is currently Clinical Research Lead in the Department of Radiology, Lausanne University Hospital, with a joint affiliation to Siemens Healthcare Switzerland's Advanced Clinical Imaging Technology group. He was previously a Marie Curie Fellow with a project on 'Modelling and Inference on brain Networks for Diagnosis', jointly affiliated to the FINDlab at Stanford University and LabNIC at the University of Geneva. His research interests include the combination of imaging modalities with other biological information sources including genomic data, learning with graphs, machine learning for neuroimaging, brain connectivity / resting-state data analysis, interpretability of brain decoding results, and functional biomarkers. Jonas contributed to PRoNTo v1.0 and v1.1;

Dr. Jane Rondina is a research fellow at the University College London Institute of Neurology. Previously, she was a post-doctoral research associate at the Centre for Neuroimaging Sciences, King's College London. In the past years, her research has involved application of pattern recognition methods to neuroimaging data and development of a stability-based method for feature selection and mapping in neuroimaging. Her current research focuses on prognosis and prediction of treatment response, mainly addressing approaches to combine complementary information from different imaging modalities and other sources of data (clinical, demographic and genetic). She contributed to the development of PRoNTo v1.0 and v2.0;

Dr. Maria J. Rosa is an Advanced Support Engineer at MathWorks. Before joining Mathworks, she was an imaging scientist at IXICO, plc. Before, she was a Post-Doctoral Research Fellow at the Institute of Psychiatry, King's College London (KCL) and a Wellcome Trust post doctoral research associate at the Centre for Computational Statistics and Machine Learning (CSML), UCL. Maria's main area of work is the development and application of machine learning and multivariate methods to neuroimaging data. She did her PhD at the Wellcome Trust Centre for Neuroimaging, UCL. She contributed to PRoNTo v1.0, v1.1, v2.0 and materials for v3.0;

Dr. Jessica Schrouff got her PhD from the University of Liège, under the supervision of Dr. C. Phillips. She was a post-doctoral researcher at the Laboratory of Behavioral and Cognitive Neuroscience, Stanford University before obtaining a Marie Curie fellowship with University College London. Her research applied machine learning techniques to neuroimaging data (both cognitive and clinical neuroscience) and investigated the limitations of such models, especially in terms of interpretation. Jessica now pursues related work at Google Health. She contributed to PRoNTo versions 1.0, 1.1, 2.0, 2.1 and was a major contributor to v3.0;

Konstantinos Tsirlis has a background in Physics, Neuroscience and Machine Learning. His M.Sc. Thesis was in Causal Structure Learning where he implemented a novel hybrid method for learning the causal structure of continuous data in the presence of latent confounders. Before joining CMIC he was involved in various computational and experimental neuroscience projects, and two causal discovery and bioinformatics projects. He is currently a Research Software Engineer in the lab, focusing on developments in PRoNTo v3.0, as well as some machine learning projects.

Dr. Tong Wu got her PhD from the University of Queensland in Australia, under supervision of Dr. Tianzi Jiang, Dr. David Reutens and Dr. Simone Bossard. She was a post-doctoral researcher in Centre for Medical Image Computing (CMIC) at University College London. Her research focus during PhD was resting-state mouse fMRI in both wild type mice and a schizophrenia mouse model. She contributed to PRoNTo v2.1 and was involved in PRoNTo v3.0.

We also want to thank the Laboratory of Behavioral and Cognitive Neuroscience at Stanford University for their early adoption of v3.0 and their feedback, as well as the students and post-docs for their help in testing the software and writing this manual: Fabio Ferreira, Dr Agoston Mihalik, Rafael Ramos, Dr. Juan E. Arco, Dr. Liana Lima Portugal, Liane Canas, Dr. Ana Regia Neves, Rochelle Silva, Dr. Orlando Fernandes Junior, Dr. Qinquan Gao. Finally, we would like to thank Prof Leticia de Oliveira from the Fluminense Federal University in Brazil for her suggestions to the manual and helping us extensively test PRoNTo v3.0.

1.8 Acknowledgements

PRoNTo is the deliverable of a Pascal Harvest project coordinated by Dr. J. Mourao-Miranda and its development was possible with the financial and logistic support of

- [PASCAL2](#) and its HARVEST programme.
- The [Department of Computer Science](#), University College London.
- The [Wellcome Trust](#) under grants no. WT086565/Z/08/Z and no. WT102845/Z/13/Z.
- The Fonds de la Recherche Scientifique ([FNRS](#)), Belgium.
- Fundação para a Ciência e Tecnologia ([FCT](#)), Portugal.
- Swiss National Science Foundation (PP00P2-123438) and Center for Biomedical Imaging ([CIBM](#)) of the EPFL and Universities and Hospitals of Lausanne and Geneva.
- The EU Marie Curie Action under grant FP7-PEOPLE-2011-IOF #299500 to Jonas Richiardi.
- The EU Marie Curie Action under grant Horizon2020-2015-GF #654038 to Jessica Schrouff.
- The Laboratory of Behavioral and Cognitive Neuroscience ([LBCN](#)), Stanford University.

Part I

Description of PRoNTo tools

Chapter 2

Data & Design

Contents

2.1	Introduction	27
2.2	The PRT directory	28
2.3	Groups	29
2.4	Subjects/Samples	29
2.5	Modalities/runs	29
2.5.1	Select by samples	30
2.5.2	Select by subjects	32
2.6	Masks	35
2.7	Review data and design	35
2.7.1	HRF correction	36
2.8	Load, Save and Quit	37
2.9	Data & Design output	37
2.10	Batch interface	38
2.11	PRT structure	39
2.11.1	Introduction	39
2.11.2	Changes	40

2.1 Introduction

The first step in a statistical analysis of neuroimaging data, whether it is in a pattern recognition or general linear model (GLM) framework, usually entails providing to the analysis software all the information regarding the data and experimental design. PRoNTo is no exception. After preprocessing the data (if required), the analysis in PRoNTo starts with the ‘Data & Design’ module. It is important to note that PRoNTo does not perform any standard pre-processing steps (except detrending), and if they are not performed with another software, the pattern recognition analysis might be affected by misalignment and noise in the data.

PRoNTo provides two types of interfaces for entering the data and design information, a PRoNTo-specific graphical user interface (GUI) and the `matlabbatch` system that is also currently used by SPM. These two interfaces are also available for the other modules, as discussed in chapter 1.

The information that needs to be entered is almost exactly the same for both the GUI and batch (the small differences are explained later in this chapter, in the `matlabbatch` section) and, more importantly, the output is exactly the same. Therefore it is up to the user to decide which system is best suited for his/her analyses. For instance, the GUI can be used as a first approach to the toolbox and by users not familiar with SPM, whilst the batch can be used by more advanced or SPM users, who know how to take advantage of the batch system to optimise their analyses.

This chapter focuses specifically on the ‘Data & Design’ module. It presents the graphical user interface (GUI) that is used to enter the various data and design information, how everything is organized and used as well as what its main outputs are. It also mentions a few issues that need to be taken into consideration when entering the information and how they affect subsequent steps. Finally, the chapter finishes by mentioning the corresponding ‘Data & Design’ `matlabbatch` module, and particular issues that do not apply to the GUI.

In the ‘Data & Design’ module the user can enter the image/sample files, information related to experimental conditions (TR, durations and onsets of events), as well as other parameters, covariates and regression values. PRoNTo supports multi-modality datasets and therefore it allows the user to enter more than one data modality, such NIfTI images (e.g. MRI, fMRI, PET), SPM MEEG objects and standard .mat files. This module is therefore essential for the rest of the framework and stores all the information that is needed from the data to be used by the rest of the software modules, such as feature set preparation, model specification and estimation.

To start with, the graphical user interface (GUI) to specify the data and design is presented in Figure 2.1. This GUI can be launched by typing ‘prt’ in the MATLAB window and then clicking the first button on the left, named ‘Data & Design’, in the ‘Main steps’ panel.

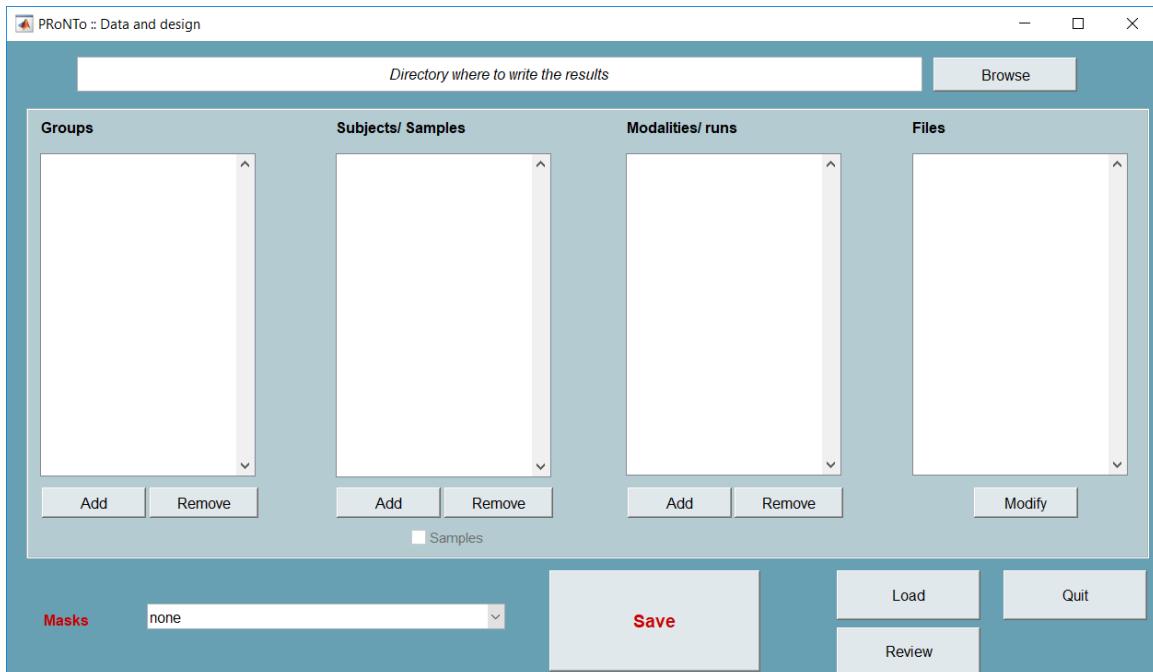


Figure 2.1: Data and design graphical user interface. This interface allows the user to enter all the information relative to the data, including the experimental design and masks. After introducing all the fields, PRoNTo creates the PRT structure, which is saved in the specified directory, as ‘PRT.mat’ file.

2.2 The PRT directory

The first thing the user should specify is the directory in which to save the PRT structure. This can be done by browsing existing directories (previously created by the user) from the top of the Data & Design interface (Figure 2.1). It is recommended to have different directories for different datasets (note that a dataset can include different modalities in case of multimodal analysis) because PRoNTo overwrites an existing PRT in the selected directory. The later modules in PRoNTo will then add more fields to this structure with further information, such as the models, features and kernels used in subsequent analyses. The file created is called ‘PRT.mat’.

2.3 Groups

The group panel allows one to add or remove a group of subjects. The minimum number of groups is one, but there is no maximum number. When ‘Add’ is clicked, the user should provide a name to the group. Any alphanumeric string is sufficient and there should be no spaces in the string (this applies to all names throughout the toolbox). The name of the group can be later modified by right clicking on the name. When ‘Remove’ is clicked, all the information relative to this group (including all subjects and corresponding data) is deleted. PRoNTo does not restore the deleted information and it can only be re-entered again by clicking ‘Add’.

2.4 Subjects/Samples

The following panel after ‘Groups’ is ‘Subjects/Samples’. In neuroimaging datasets, it is common to have a few subjects with a lot of images/samples per subject, such as the time-series in fMRI. However, the opposite is also common: lots of subjects with one image per subject, such as those encountered in PET or MRI studies. Therefore, for each group, PRoNTo provides two ways of entering the rest of the information, i.e. subjects, modalities and design, which are referred to as the ‘select by subject’ or ‘select by samples’ option, respectively. The former is chosen by clicking ‘Add’ under the ‘Subjects/Samples’ panel and filling in the fields for each added subject at a time. The latter is done by clicking the tick box ‘Samples’ under the ‘Subjects/Samples’ panel. The subjects panel is then de-activated and the user can enter the modalities and files straight away. Both options are presented side by side in figure 2.2.

Select by subjects: The ‘Subjects/Samples’ panel allows the user to add/remove subjects. This panel works exactly like the groups panel, but the subject name is automatically generated. This name can be later modified by right clicking on it. For each subject one can then specify the modalities in the next panel.

Select by samples: The ‘Select by samples’ option allows the users to skip the subject step. To identify that this option has been selected, PRoNTo writes ‘Samples’ in the subjects panel, in the same way the subject names were automatically generated when you are in the ‘Select by subjects’ option. It is important to remember that when the ‘samples’ box is clicked all the information in the subjects panel is automatically deleted. Unselecting the ‘samples’ box also deletes all the information!

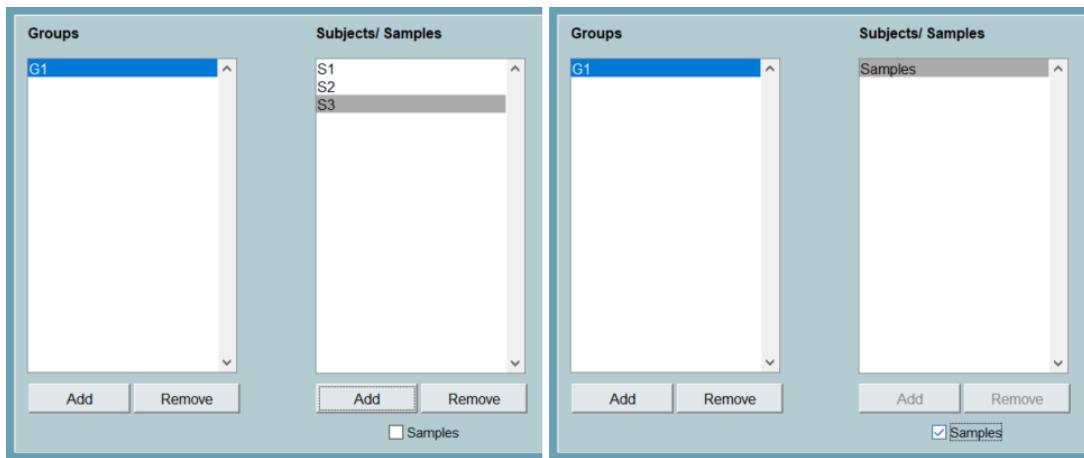


Figure 2.2: The ‘Subjects/Samples’ panel under the ‘Select by subjects’ and the ‘Select by samples’ options.

2.5 Modalities/runs

The modalities panel works like the group and subjects panel, but allows one to add and remove modalities. When a modality is added, a name needs to be provided (unless the modality has already been defined for a previous subject or through the masks menu, see below). **It is important to note that a different modality can be a different type of data, such as fMRI and PET, or a different session of the same type of**

data, e.g. different runs/sessions of the same fMRI experiment. This way the different sessions can be integrated later into the same model and analysis.

From PRoNTo v3.0 your data can have more than one formats, so there is a new field, called ‘Data format’, which offers three input formats to choose from: NIfTI (as before), MEEG (electrophysiological data in the SPM format) or .mat (any type of matrix).

Clicking ‘Add’ to add a new modality, and depending on whether the ‘Samples’ box was ticked or not, one of the two windows of figure 2.3 will appear.

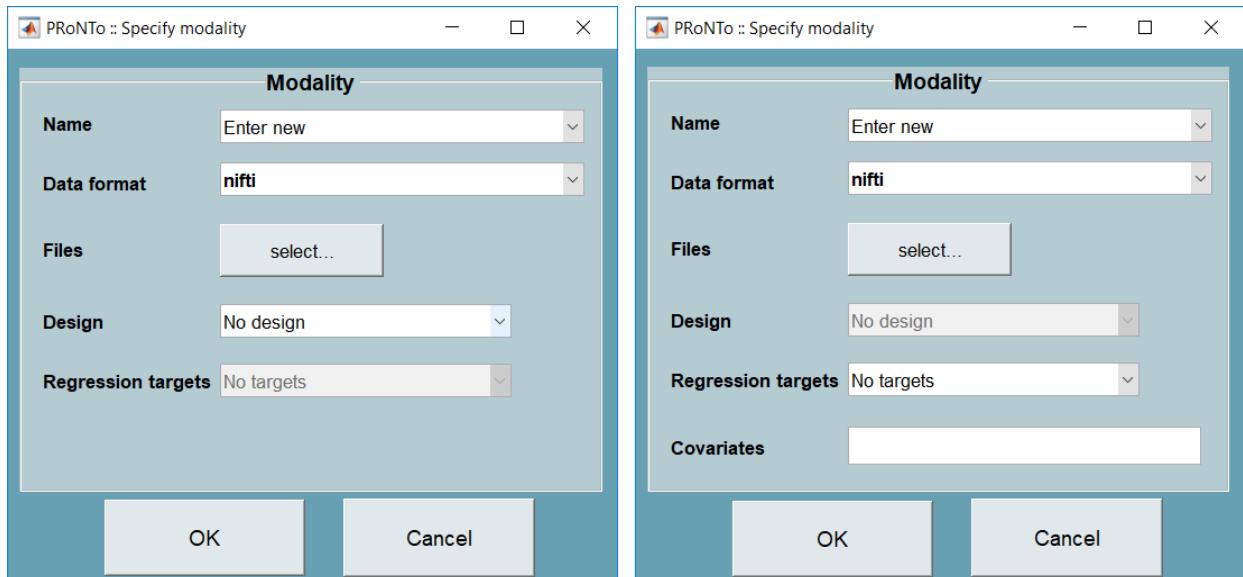


Figure 2.3: ‘Specify modality’ window under the ‘Select by subjects’ (left) and the ‘Select by samples’ (right) options

The first things the user has to define is the name of the modality and the format of the data. Next, independent of the way the user entered the information (by subjects or samples) the ‘Files’ option allows one to choose which image files to use (Figure 2.4). This will open another window that shows all image files available in each directory. These can be selected one by one or all at once, by using the mouse’s right button on the right panel of the window (or shift key).

From that point on, just like in the ‘Subjects/Samples’, the steps to enter the modality information become slightly different if the ‘Samples’ box is ticked or not.

2.5.1 Select by samples

Here the data is assumed to have been acquired without an experimental design (i.e. the data has no temporal information and each subject has only one sample), and therefore the ‘No design’ option is automatically selected and cannot be changed, as shown in figure 2.3.

The empty field below can be used to enter ‘Regression targets’. This option allows the users to introduce a real number per subject to be used later for regression if that is the case. A file can also be selected, with all regression target values stored in a variable called ‘rt_subj’ and of size $\#images \times n$ (where n is the number of regression targets).

Finally, the user can introduce ‘Covariates’, i.e. one or more variables that covary with the data (subjects) but of no interest to the subsequent analyses. The covariates will be regressed out from the data if the operation ‘Regress covariates’ is selected in the ‘Specify Model’ module. This option is functional from v2.1 of PRoNTo.

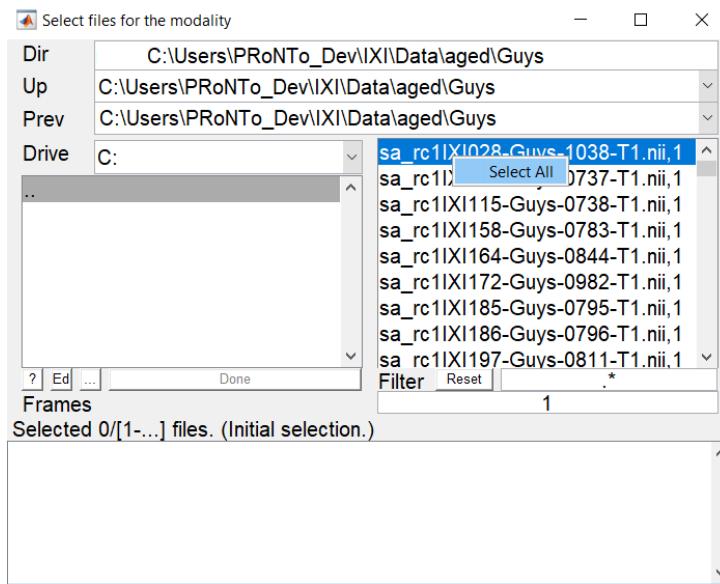


Figure 2.4: This window is called when one clicks ‘Files’ and is used to select the samples/images for each subject/modality.

It requires the input of a matrix, with one row per image/subject and one column per covariate. This matrix can either be entered as a MATLAB command in the editable box, or as the full path to a .mat containing the matrix (variable named ‘R’). This last option is recommended to input a matrix (i.e. more than one covariate). The order of covariates in the rows should match the order the imaging files are selected. Please see below (2.5.1) for important considerations on covariates.

If users want to modify covariates and/or targets, simply right-click on the modality name to amend. From v2.1, we only display the entered value when reviewing single subjects while ‘Entered’ is displayed to indicate that values have been filled (Figure 2.5).

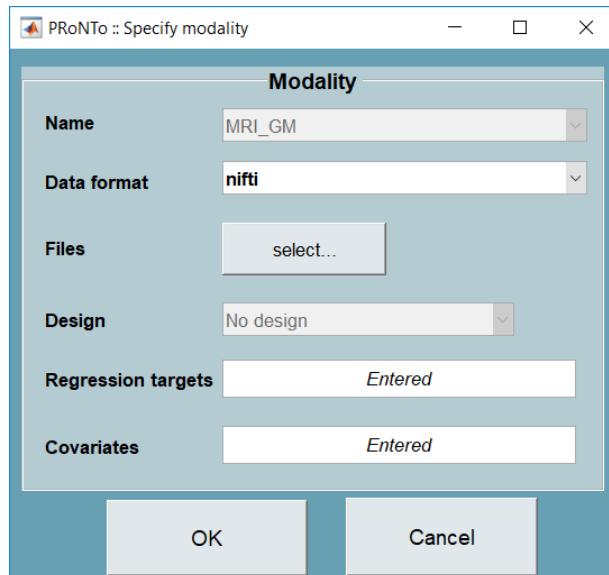


Figure 2.5: When modifying a previously specified modality/run, input covariates and regression targets will appear as ‘Entered’. Actual values are displayed when reviewing single subjects (i.e. after saving and reloading the PRT).

Covariates

The user can also input ‘covariates’ or ‘confounds’ in PRoNTo, i.e. one or more variables that covary with the data (subjects) but are of no interest to the subsequent analyses. PRoNTo requires the input of a matrix, with one row per image/subject and one column per covariate/confound. Before entering covariates in PRoNTo, we recommend users to firstly check that their covariates do not correlate with the targets as this could lead to biases (positive or negative) in the obtained results [16]. Furthermore, since the approach to remove confounds in PRoNTo is based on a linear regression the covariate matrix can be seen as a design matrix and therefore should respect the usual requirements of no-collinearity ([21]).

In addition, if users have categorical covariates, such as genders, scanner centres, handness et al., one could use one-hot encoding for the covariates. One-hot encoding represents each value of a variable (for example ‘center1’, ‘center2’) as a boolean ‘True/False’ (i.e. 1/0) variable. In this case, a confound with 2 values is represented by 2 columns, a confound with 3 values by 3 columns and so on. This process ensures that all values of the categorical variable are treated with the same importance. When accounting for categorical confounds we should be careful with the representations that imply order of values when there is no relative other, for example representing ‘center1’ by 1 and ‘center2’ by 2. In this last example, we would assume that values in ‘center2’ are twice as big as values in ‘center1’ although there is no reason to encode an ordinal relationship between ‘center1’ and ‘center2’. Here is an example code to perform one-hot encoding for an illustration covariate variable ‘cov’ for 12 samples/images and 4 possible values:

```

cov = [4 8 1 5 4 8 1 5 1 5 4 8];
vals = unique(cov);
newcova = zeros(length(cov),length(vals));
for i = 1:length(vals)
    newcova(:,i) = cov== vals(i);
end

```

If multiple categorical variables need to be encoded, this process should be repeated and all obtained matrices concatenated to obtain a $\#samples/images \times \#variables$ matrix.

Important note on one-hot encoding: While one-hot encoding is a useful way of recoding categorical covariates, great care has to be put in order to avoid the dummy variable trap, which essentially is an accidental perfect multi-collinearity when the system of equations formed by the regression ends up not have a unique solution. For further information regarding this, the user is referred to [27], as well as various informal sources (for example <https://www.algosome.com/articles/dummy-variable-trap-regression.html>).

Important note on covariates: Regressing out covariates is still a matter of debate [16] and should be considered with care. The procedure to remove confounds in PRoNTo is based on a linear regression and wouldn’t account for more complex relationships between confounds and variables of interest or between multiple confounds (e.g. all older patients recorded at one site). Balanced and careful experimental designs should be preferred.

2.5.2 Select by subjects

When entering the data by subjects, the modality window allows one to specify the experimental design (Figure 2.3). Here there are three options. The first option is simply ‘No design’, which means that for this modality there are no experimental conditions (this option is normally used when there is only one image per subject e.g. structural MRI or beta images from GLM analysis). The last option is to load an SPM.mat with a previously specified design. This option can be chosen if the user has created an SPM structure containing all the experimental information using the SPM software. In this case, the user does not need to specify anything else, only the files (samples/images) for this subject/modality. The design information is extracted directly from the SPM structure and saved in PRT.mat. Finally, there is also ‘Specify design’ option, which allows one to introduce all the conditions (durations and onsets), TR and other parameters corresponding to the experimental paradigm used for this subject and modality (this option is normally used with time series data, e.g. fMRI). After the design of the first subject has been specified, a new option will appear in the menu that allows to ‘Replicate design of subject 1’, for the same modality and group. This facilitates design specification for groups of subjects with controlled (i.e. non-random) event onset and duration.

Design: The design field has different options according to the data format selected. This is because the events should already be saved in the MEEG data (input: epoched signal). For NIfTI images, the options are the same as v2.1. And for .mat, the options are the same as for NIfTI.

For NIfTI and .mat data formats, to create a new design one selects the option ‘Specify design’ as explained in the previous paragraph (Figure 2.3). Then there are 2 options, to either specify the design manually (using the option ‘Specify’ in the ‘Conditions’ field), or to first load a pre-defined design from a .mat file (using the option ‘From a .mat file’ in the ‘Conditions’ field) and modify it according to your needs.

In case you select the ‘Specify’ option, this will then open another window where you’ll be prompted to choose how many conditions you have. If you select to load a pre-defined design from a .mat file then you will be first prompted to load this .mat file, where PRoNTo will load the first variable of that .mat only. In that case the main design window (described in the next paragraph) will be automatically filled with the events name, onsets (in seconds) and duration (in seconds).

After choosing how many conditions you have or after selecting the .mat file with the pre-defined design, another window will open (Figure 2.6), where in this window one can then write (or modify) the names, onsets, and durations of each condition. From v3.0, for both neuroimaging and MEEG, there are 2 new columns where regression targets and covariates can be entered per trial in each category. These need to have same number of elements as the number of trial onsets.

The units in which this information is read is specified below. There are two options ‘Scans’ or ‘Seconds’. If the unit ‘Scans’ is selected, it is good to bear in mind that PRoNTo follows the convention, adopted in SPM, that the first scan is scan 0. In the durations field, one can introduce as many values as the number of onsets or just simply one value, which assumes the events all have the same duration. In this window there is also the option of introducing the Interscan Interval (TR), which corresponds to the intersample time (i.e. 1/sampling rate) and is always read in seconds.

One issue to have in mind when specifying the design is the following: if there are more samples than experimental events, these extra samples will not be used in later analyses. They are not deleted and the corresponding indexes can be found in the PRT structure:

PRT.group(g).subject(s).modality(m).design.conds(c).discardedscans.

For MEEG data formats, there is only one available option in the ‘Design’ field, and that is the ‘Events in file’ option. Clicking this option will automatically load the design and lead you to the ‘Specify design window’. From that point on you proceed the same way you would for NIfTI and .mat data.

Modify design: The user can later modify a design by loading a PRT.mat in the Data & Design window. Please note that if feature sets or models have been previously computed, they will be discarded if changes are performed to the dataset. If the user wants to keep those, he/she should change the directory before saving any modification to the design.

After loading a previously saved PRT, any change can be performed: subjects, groups or files can be added or removed. If the design needs to be modified, a right-click (ctrl+left-click in Mac) on the name of the concerned modality proposes to re-open the modality definition window. To review or modify the onsets/durations/block-s/regression targets/covariates, the user can access their definition via the ‘specify design option’. Similar right-clicks (ctrl+left-click in Mac) allow renaming groups or subjects.

All that is needed for each group, subject and modality has been specified and can now be viewed on the main window (Figure 2.7) under each panel. The last panel shows which files have been entered for each modality and can be modified directly (click Modify). When Modify is clicked and no files are then selected all the previous files are deleted! Figure 2.7 shows how the Data & Design interface should look like once all the fields have been specified (using select by subject). The design and files for each modality can also be modified by right clicking on the modality name in the modality panel. This option can be useful to visualise the design (onsets and durations) that has been previously entered and change it if necessary. For instance, one can check the design of the first subject and if changes are needed these can then be replicated for all other subjects as

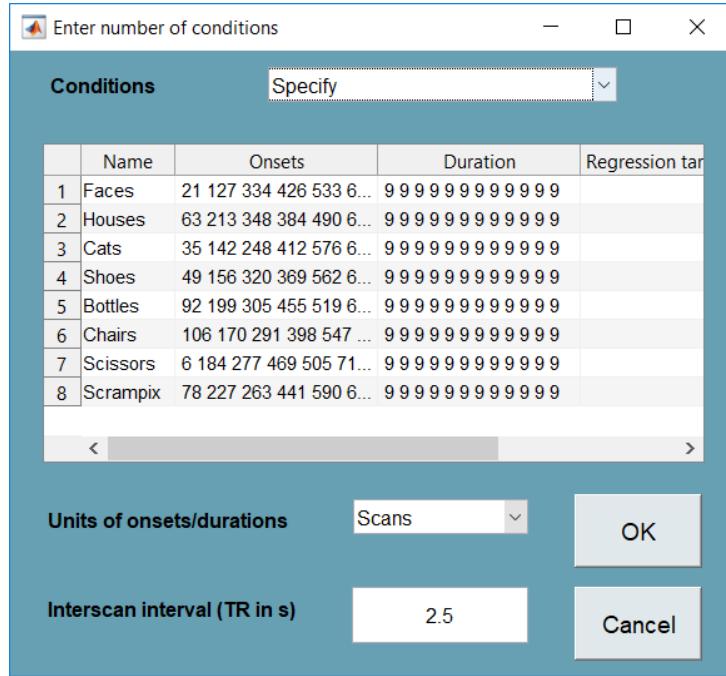


Figure 2.6: Data & Design graphical user interface. The ‘specify conditions’ window is available from the modality interface when the user chooses to enter the data by subjects and clicks ‘specify design’. This window is used to enter the conditions (names, onsets and durations) as well as the units of design, TR and covariates.

explained above.

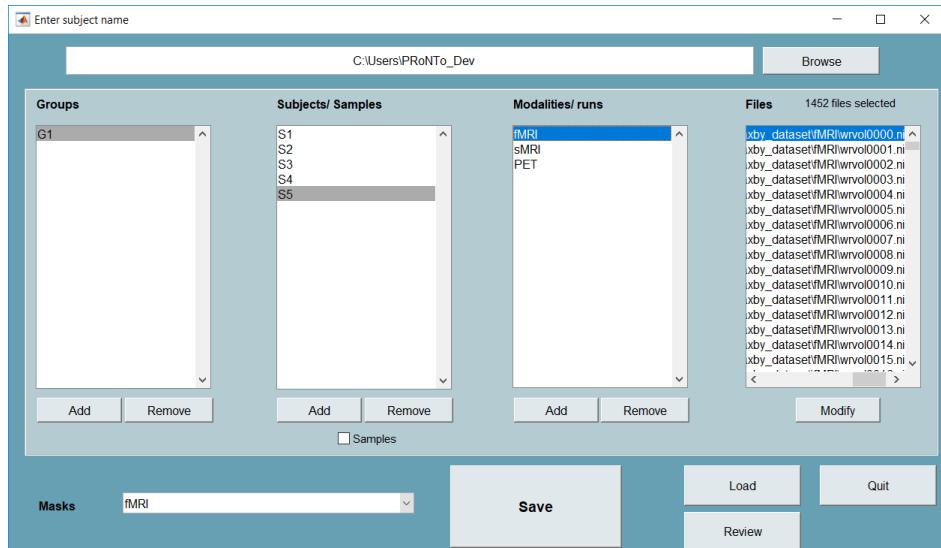


Figure 2.7: Data & Design graphical user interface. After filling in all the fields using the select by subject option (the select by samples case is very similar) the Data & Design interface should look like this example figure.

2.6 Masks

This popdown menu on the bottom of the main Data & Design window is where the user enters a binary image mask for each modality. This applies only to NIfTI data formats, since for .mat and MEEG data formats we assume that the data have already been pre-processed to contain only the relevant features (and therefore there is no need for a mask), as this can be easily performed during pre-processing. Hence the masks are set to ‘none’ for MEEG and .mat types. It is also vital that you ensure that your data (.mat or MEEG) do not contain NaNs.

Regarding the NIfTI data, this mask can be previously created by the user or simply chosen from a list of default masks available in the masks directory of PRoNTo. Every modality based on NIfTI data has to have a mask, which can be the same for all modalities. This is a first-level mask and is used simply to optimise the prepare feature set step by discarding all uninteresting features, such as voxels outside the brain (for the case of MRI data) or features in general.

Later in the analysis one can choose another mask (second-level mask) that is more relevant to the scientific question and that can, for example, restrict the analysis to certain areas of the brain. To specify the mask one needs only to select the modality and then enter an image file. If the modalities have not yet been created, then one can create the modalities here, which will then appear in the modality panel.

Important note: If the first-level mask overlaps with voxels that do not have values in the specified images (i.e. NaN), those areas will still be taken into account for further analysis and the NaNs will be replaced by zeros. This might affect the results if those areas are not the same across images (typically, performance will be lower). We therefore advise the user to check the overlap between the first-level mask and his/her data. This issue would typically arise when using beta images estimated from a SPM GLM analysis. We provide a script to update the mask automatically by removing voxels that contain NaN values from the mask. (see [1](#), Inputs and preprocessing).

2.7 Review data and design

The ‘Data & Design’ module also allows the user to review the information that has been entered (through the GUI, batch or manually). The main aim of the ‘Review’ function is to check if the data and design has been correctly specified. It can also be used to inspect if the design is appropriate for subsequent analysis.

This is done by clicking the ‘Review’ button in the main ‘Data & Design’ window, and as we mentioned, allows the user to review the Data & Design for each modality (Figure [2.8](#)). On the top right is the information relative to the number of groups and modalities that have been entered. The plot on the left displays the number of subjects per group. It is particularly important to check if the design is too imbalanced in terms of subjects. Then on the bottom right panel is the design information for each modality (if the selected modalities have an experimental design). Here, the user can view the number of conditions and can also edit the parameters that control the HRF delay and overlap (as explained above). The user can change the default value of 0 seconds and the effect is immediately seen on the number of samples plotted on the left (number of selected samples and number of discarded samples for each condition). The higher the value of the HRF peak and overlap, the higher the number of discarded samples. One can also read on the main MATLAB window information regarding which group/subjects have had some samples discarded. The information below the HRF parameters corresponds to the interval between successive samples before and after the HRF delay/overlap correction. These values also change according to the changes entered in the boxes above. For further information regarding the HRF correction the user is referred to the following subsection ([2.7.1](#)).

To modify the HRF parameters (delay or overlap), there is no need to load the PRT in Data & Design. Loading it within the Data Review allows the user to keep all previously computed feature sets and models. However, if the HRF parameters are changed, feature sets have to be computed again since they do not correspond to the modified design. Changing the desired parameter (e.g. replacing ‘0’ by ‘6’) and hitting the ‘return’ key updates the PRT directly in terms of samples selected for modelling.

Please note that the review window is frozen, waiting for users’ inputs. Hence other modules won’t continue until this window is closed. This means that users who prefer to use the Batch to execute multiple modules

together and choose Review in Data&Design need to close this window first to let the following modules run.

Furthermore, if you have previously computed feature sets and models, you have to recompute them because they do not correspond to the data anymore (changing the HRF delay and overlap parameters changes the data). The information regarding which samples have been removed or not from the analysis can be found in the PRT structure:

```
PRT.group(g).subject(s).modality(m).design.conds(c).hrfdiscardedscans.
```

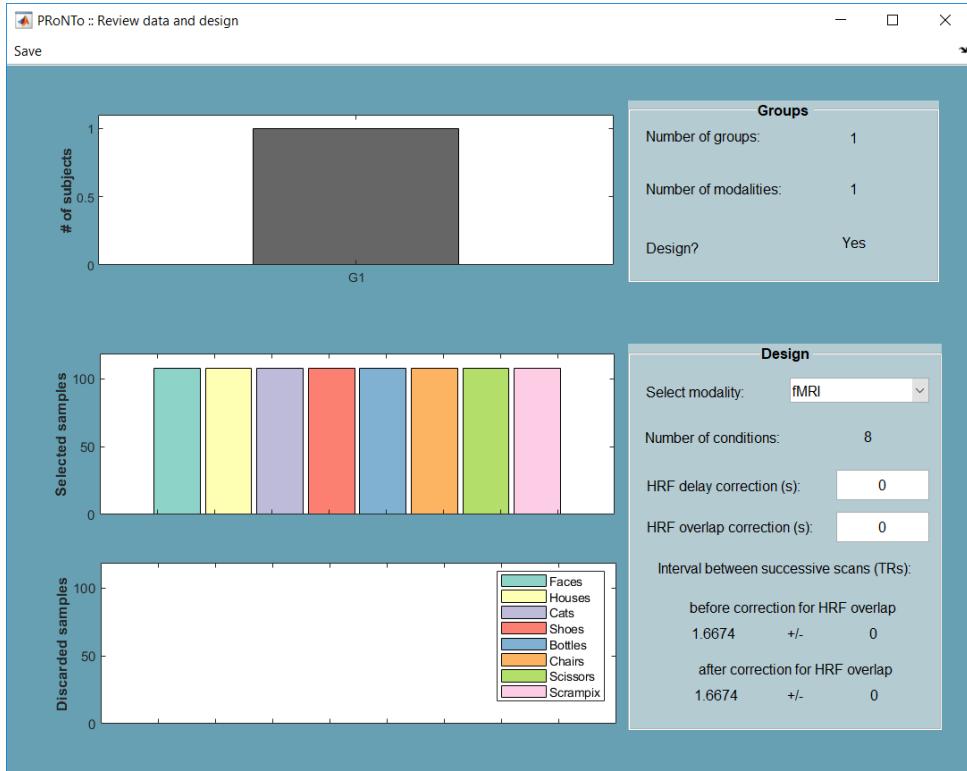


Figure 2.8: Data & Design graphical user interface - ‘Review’ window. This window allows the user to check the data and design, including the number of subjects per group. It also allows the user to change the HRF delay and overlap parameters that control the number of discarded samples (appropriate only for modalities such as fMRI). When there is no experimental design only the top plot and information is shown.

2.7.1 HRF correction

When using raw fMRI data in PRoNTo, there is a very important issue that needs to be carefully addressed when specifying the data and design. As is well known, the hemodynamic response function (HRF) is a delayed and smoothed version of the underlying neuronal response to an experimental event (Figure 2.9). This means that, depending on the TR, the effect of the HRF can be felt over multiple samples, and therefore the acquired samples are not independent and might contain information from both past and present events. This can confound subsequent analyses and needs to be accounted for. For instance, in SPM, the stimulus time-series are convolved with a canonical HRF. Although convenient in the GLM framework, the convolution approach is not appropriate in the pattern recognition context. Therefore, the solution used in PRoNTo is to discard all samples that might have overlapping information from more than one event. This is done as follows: PRoNTo allows the user to input a delay (time it takes for the hemodynamic response to peak after the stimulus) and overlap (width of the response) parameters that determine the shape of the HRF. As can be seen in Figure 2.9, the delay means that the samples corresponding to a given condition are actually shifted in time, and the overlap means that the number of non-overlapping samples, for which the signal corresponds only to a given event or experimental condition, is smaller than the total number of acquired samples for each condition. Given the delay, PRoNTo shifts the onsets of each condition to account for the specified delay. It then uses the overlap to

determine which consecutive samples contain information from only one condition (i.e. the response does not overlap with the response from the previous condition) and discards the ones for which there is overlap (as shown in Figure 2.9, bottom right). The discarded samples are not actually deleted but are not used in further analyses.

When using the GUI, the default value for the HRF parameters is 0 seconds and can only be changed in the ‘Review’ window (as shown below). Therefore, for fMRI, the user should review the data and design and change these parameters to a more appropriate value (e.g. 6 seconds each). In the `matlabbatch`, the default value for these parameters is also 0 seconds but can be changed directly within the batch (no need to open Review window). Again, for fMRI raw signal, these values should be changed (e.g. to 6 seconds).

Importantly, if one wants to avoid discarding samples and having to correct for the shape of the HRF, as explained in the above paragraph, one should use as input the beta (coefficients) images obtained by first running a GLM analysis on the original data. The GLM analysis accounts for the HRF delay and overlap using the convolution approach. This is normally the best approach in case of rapid event-related design experiments, in which there can be a lot of overlap, i.e. the number of discarded samples can be very high.

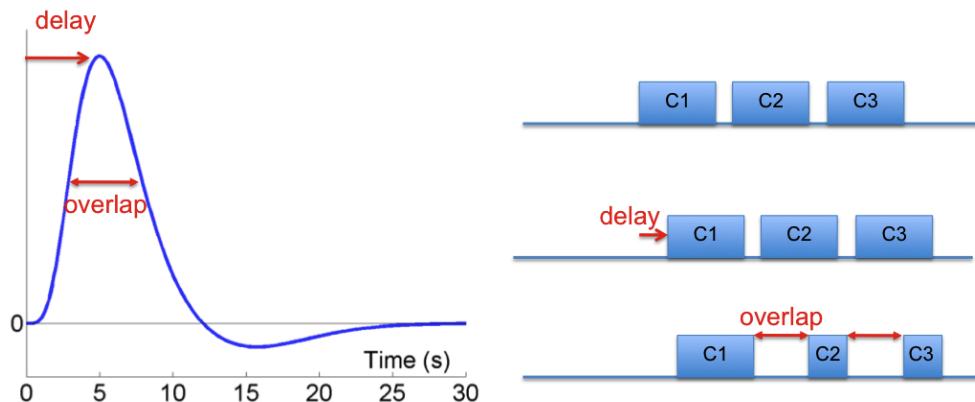


Figure 2.9: HRF correction. On the left is the standard HRF response. On the right is the effect of the delay and overlap on the number of independent samples (C1, C2 and C3 correspond to three different experimental conditions and the blue boxes correspond to various samples acquired during each condition). In fMRI datasets, the nature of the HRF (i.e. being a delayed and dispersed version of the neuronal response to an experimental event) might lead to less independent samples/events than the ones originally acquired. In PRoNTo, this issue is accounted for by discarding overlapping samples.

2.8 Load, Save and Quit

The ‘Save’ button allows the user to create the `PRT.mat` file with the PRT structure containing all the information that has been specified here (Figure 2.7). Incomplete information cannot be saved. At least one group should have all the required fields so that `PRT.mat` can be created. ‘Load’ allows the user to load the data and design information from a previously saved `PRT.mat`. The user can then edit the fields and update PRT by clicking again the ‘Save’ button. It’s very important to click ‘Save’ because all the other steps in the analysis rely on the PRT structure. Without this structure one cannot proceed. However, when the `PRT.mat` contains fields that have been added by the ‘Prepare feature set’ or other modules, if the Save button is clicked, these fields will be deleted. The option ‘Quit’ allows the user to leave the interface without saving the information. This is also the case when the user closes the window without first using the Save button.

2.9 Data & Design output

The output of the ‘Data & Design’ module, as discussed previously, is the PRT structure. This structure contains subfields with all the information that is needed from the data for the subsequent analysis steps and is saved in a ‘`PRT.mat`’ file. For advanced users the fields of this structure can be edited directly and saved,

therefore bypassing the need to use the GUI or `matlabbatch` to create the PRT. However, this structure is the core of PRoNTo and should be carefully created because it affects everything else.

2.10 Batch interface

The ‘Data & Design’ module in the `matlabbatch` is called either by first typing ‘prt’ and clicking the ‘Batch’ button or by typing ‘prt_batch’. The user can then find on top of the batch a PRoNTo menu and under this menu the first module corresponds to the Data & Design module.

The options presented in the ‘Data & Design’ GUI, mentioned above, are all available in the `matlabbatch` interface (Figure 2.10). However, there are a few things in the batch that differ from the GUI. One issue to note here is that, when using the batch one needs to be very careful with the names of the modalities specified for each subject (or using select by samples) and specified for each mask. The number of modalities should be exactly the same for each group and subject and the names should be consistent between groups/subjects and correspond to the names of the modalities under the masks field. In the GUI the names are made automatically consistent. The names of the conditions should also be the same across subjects and will be later used to define classes in the ‘Specify model’ batch module.

Another difference is the HRF delay and overlap correction values. In the batch, the user can directly alter these values (instead of having to use the ‘Review’ window) but the default is 0 seconds and should be changed (e.g. to 6 seconds) for modalities that depend on the HRF, such as fMRI. In earlier versions, HRF delay and overlap values couldn’t be changed separately for each modality. From v2.1, we have corrected this.

As mentioned in the Introduction, the batch job can be saved as a .mat, and loaded again whenever needed, or as a .m that can be edited using the MATLAB editor. This is a powerful tool that can make the specification of the data and design a lot easier and quicker, for example by editing and scripting existing batch files (for further information see the `matlabbatch` chapter below).

All changes in v3.0 in the GUI, have been added to the batch. When choosing a modality, the ‘Data format’ field now allows you to choose between ‘nifti’, ‘MEEG’ or ‘.mat’, whether the user specified the design by scans or by subjects. As the batch is not ‘dynamic’, the different design options specify for which data format they are available. The help of each field was also revised.

Regression targets and covariates per trial: The option to enter regression targets and covariates per trial were also added in the batch (Figure 2.11).

- For the specify design, it has been added within the conditions.
- For the SPM loading and the loading of the events in the MEEG, a new field has been added, where you specify the condition name, and the regression targets and covariates for that condition.
- For loading of a multiple condition file, there is no explicit field because PRoNTo will automatically look for the corresponding variables ‘rt_trial’ and ‘R’, as cell arrays per condition.

Multiple regression targets per subject: Specifying regression targets, when loading by ‘scans’ (i.e. one image per subject per group) has been amended to allow the user to specify multiple regression targets. The multiple regression targets, together with their values, are included in each subject’s structure, inside a sub-structure named `rt_subj`. This structure has as many columns as the regression targets.

- `rt_subj`: [1 x n]
 - `.name`: target name, default ‘Tar 1’.
 - `.tar`: target values.

Besides the appropriate modifications to the code, we also added the script `prt_data_targets` to allow users to specify targets by hand or from file, as well as review via a table.

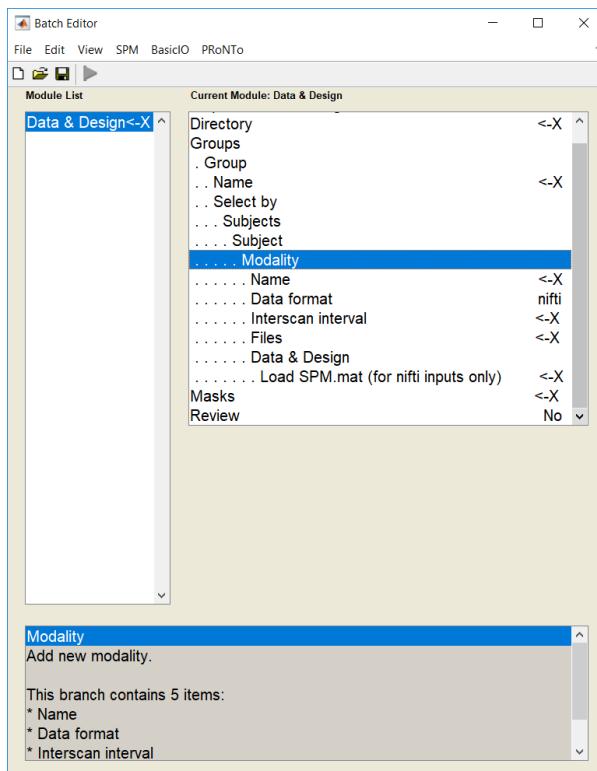


Figure 2.10: Data & Design module in `matlabbatch`. The `matlabbatch` contains two extra options relative to the Data & Design interface. These options allow one to specify the delay and overlap of the HRF response (in the GUI it can only be changed in the ‘Review’ window) separately for different modalities, and which are then used to determine the number of discarded samples.

Masks and modalities: Regarding the masks, the user needs to specify a ‘mask’ for each modality, as before. However they need to choose a file only for the ‘nifti’ input modalities. The need to specify masks for all modalities is due to a coding requirement as the number of masks serves as a check for all the modalities that are defined (i.e. checking if number of modalities is the same across subjects and so on). For MEEG or .mat, the type of modality is displayed but not further action is required. In the code, checks make sure that the right type of design is selected for MEEG (‘Events in MEEG file’) or for .mat (‘no design’). This can easily be modified when extending the ‘.mat’ option to ‘new_design’ later on. There is also a restriction that the number of files for either a MEEG or .mat modality should be 1 (everything should be contained in this specified file, for each subject and modality). Different runs of EEG/MEG experiments can be entered as different modalities and later concatenated in ‘trials’ at the feature set stage.

2.11 PRT structure

2.11.1 Introduction

As already mentioned, the ‘PRT.mat’ is a file that contains a MATLAB structure called ‘PRT’. The ‘PRT.mat’ (and the ‘PRT’ structure within it) is the core file (and structure) of PRoNTo associated with your analysis. This is a fairly deep structure containing a range of subfields which are created and modified across different steps, and which include all information about the data, experimental design, feature sets, models and kernels used in subsequent analyses done with PRoNTo, as well as the results. This structure and its fields can also be edited and saved directly from the MATLAB workspace, therefore bypassing the need to use the GUI or `matlabbatch`.

The 5 main fields of the ‘PRT’ structure are the following:

- **PRT.group:** This field contains all the main information regarding the different groups and subjects

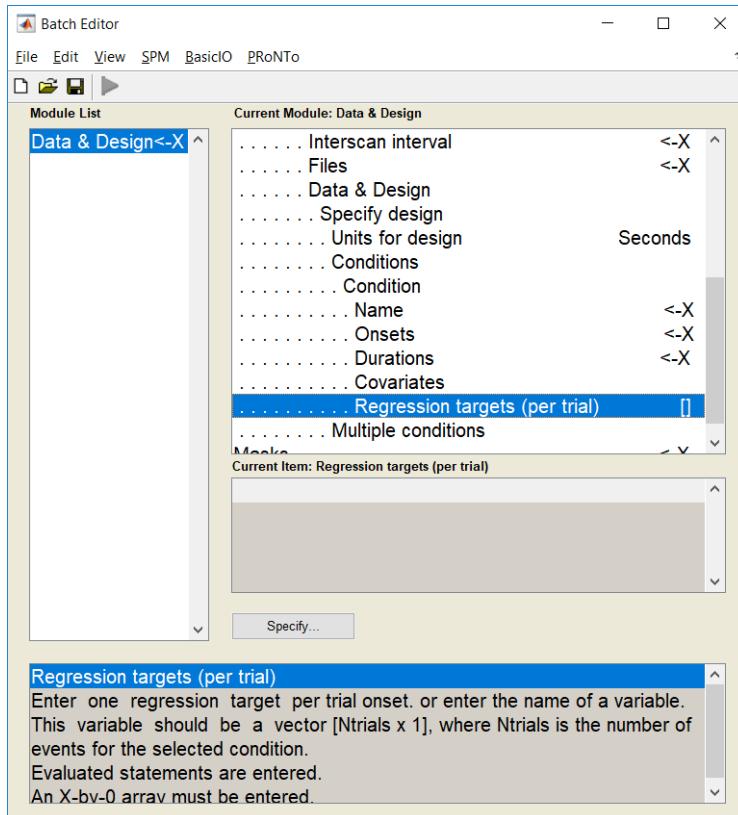


Figure 2.11: ‘Data & Design’ `matlabbatch` module.

included in them, together with their different modalities, regression targets and covariates.

- **PRT.masks:** This field contains the masks for the different modalities.
- **PRT.fs & PRT.fas:** These fields contain the different feature sets with all their supplementary information, such as the ID matrices to link the data to the design matrix or the first-level atlases, as well as the corresponding file array structure.
- **PRT.model:** This field contains the different models created, with all their parameters, the cross-validation schemes and the operations, as well as the output/results when you run these models.

Throughout Part I and Part III you can find both dedicated subsections as well as mentions of the ‘PRT’ structure and its subfields that serve to familiarize the reader with it and give an idea of the subfields created in each step of the process. Finally, Chapter 23 describes a way of creating and visualizing a full-breakdown of the ‘PRT’ structure.

2.11.2 Changes

In this subsection, as well as in the same subsection in the chapters that follow, we will point out the changes in the ‘PRT’ structure from PRoNTo v2.1 to the current version, PRoNTo v3.0.

From v3.0, PRoNTo accepts 2 new data formats, SPM MEEG file formats, and .mat file formats. Therefore in the modality field, a ‘type’ subfield was added, to account for the modality type. The same field was added to the ‘PRT.masks’ as well.

Chapter 3

Prepare feature set

Contents

3.1	Introduction	41
3.2	Feature extraction and pre-processing	42
3.3	Prepare feature set	43
3.3.1	NIfTI and .mat data	44
3.3.2	MEEG data	45
3.4	Batch interface	47
3.5	PRT structure	47

3.1 Introduction

One of the main inputs of a machine learning algorithm consists in a $N_{samples} \times N_{features}$ data matrix, containing the feature values for each sample. This matrix can either be input directly into the machine or be used to compute a ‘similarity matrix’, or kernel, of size $N_{samples} \times N_{samples}$, which is then input into the classification/regression algorithm [see “kernel trick” [12, 1]]. PRoNTo computes a linear kernel (i.e. dot product) between the samples. The ‘Prepare Feature Set’ step computes both the features and linear kernel matrices from one or more modalities, as defined in the previously described ‘Data & Design’ module (see chapter 2). It allows detrending the features in the case of time series (such as fMRI) and scaling each image by a constant factor (input by the user) in the case of quantitative modalities (such as PET). Masks can be specified to select a subset of features for later modeling (e.g. Regions of Interest for MRI or fMRI data or electrodes/channels for M/EEG data).

Multiple runs entered as different modalities (e.g. modality 1 is ‘fMRI_run1’, modality 2 is ‘fMRI_run2’,...) can be concatenated as samples during this step. In this case, the images from different runs should have the same number of features. In addition, from v2.0 one is allowed to build multiple kernels, either from multiple modalities, or based on different anatomically labelled regions as defined by an atlas. In case of multiple modalities, it is required that the selected modalities have the same number of samples.

In ProNTo versions 2.X, modalities could be combined in 2 ways at this stage: either concatenated as samples (e.g. multiple runs of a same experiment), or combined as additional features by computing one kernel per modality and storing them together. This had the limitation that combined modalities (in either way) had to have the exact same number of features. From v3.0 we have decoupled those operations for more flexibility. At the feature set stage, different modalities can only be concatenated as samples. In this case they still need to have the exact same number of features.

3.2 Feature extraction and pre-processing

In order to combine different types of modalities (or data) as different types of features in the predicted models from PRoNTo v3.0 it is necessary to build one feature set per modality. The different features sets can then be combined later on the model specification step, either added/concatenated or using MKL.

In PRoNTo v3.0 the different data formats are handled in different ways: NIfTI images and .mat in a way similar to versions 2.X, while MEEG is handled differently through a new window. This means that now, you have to first load the ‘PRT.mat’ in the ‘Prepare feature set’ GUI in order for PRoNTo to track the different data formats that exist in your PRT. So after clicking on the ‘Prepare Feature Set’ button in the main interface, a second window will appear first, allowing the user to select a saved PRT.mat. If it has more than one different data formats, figure 3.1 will appear prompting you to choose the feature set’s data format that you want to build. Feature sets are built for each specific data format at a time. The buttons available on this interface will reflect the data formats previously specified in the ‘Data & Design’ module. If only one type is present, this step is skipped and PRoNTo launches directly the main ‘Prepare feature set’ window after the user has selected a saved PRT.mat.

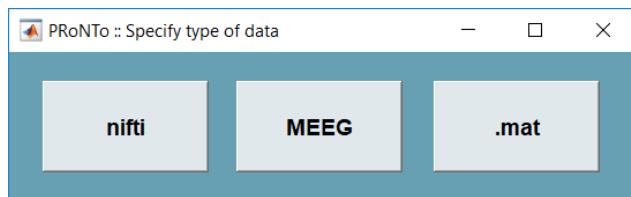


Figure 3.1: ‘Specify type of data’ GUI

In case there are more than one data formats, **if the ‘nifti’ option is selected** to be included in the feature set (further referred to as FS), the toolbox will extract information from the voxels/features contained within the first-level mask that was selected for that modality (mask specified at the ‘Data & Design’ step, see chapter ‘Data & Design’). This is performed by extracting ‘blocks’ of features, not to overload the RAM (Random-Access Memory).

If the ‘MEEG’ or the ‘.mat’ options are selected, as mentioned in the previous chapters, the data are assumed to have already been pre-processed to contain only the relevant features (and therefore there is no need for a mask), as this can be easily performed during pre-processing.

In case of time-series data (e.g. fMRI), the user can specify detrending methods and parameters to apply to the time course of each feature. Methods comprise a polynomial detrending (parameter: order of the polynomial) or a Discrete Cosine Transform high-pass filter (SPM, parameter: frequency cutoff in seconds). An example of a linear detrending (polynomial detrending of order 1) is shown in Fig. 3.2.

After you do that, the main ‘Prepare feature set’ window will appear (Figure 3.3), where you will be prompted to specify the name of the FS and to define the number of modalities which should be included in the FS.

Note: If you have been using PRoNTo versions 2.X you will notice that the checkbox that used to be available and allowed to build one kernel per modality (if multiple modalities were present/had been selected in the FS) is not available anymore since from PRoNTo v3.0 different modalities can be combined at the model level.

To define the number of modalities to include, the user should click in the appropriate edit box, type the number and then ‘enter’. This will launch a third window (Figure 3.4 for NIfTI and .mat, or Figure 3.5 for MEEG), allowing the specification of the different options and parameters for each modality. When the dataset contains only one modality, this window is launched directly and (Figure 3.3) is filled automatically and only expect the feature set name. The two windows are further explained in the sections that follow.

So for each modality, the features are then written in a file array (with a ‘.dat’ extension) on the hard drive (in the same directory as the dataset). Please note that in the case of large datasets, this operation may require many Gb of free space on the hard drive and long computational times. Therefore, if the first condition

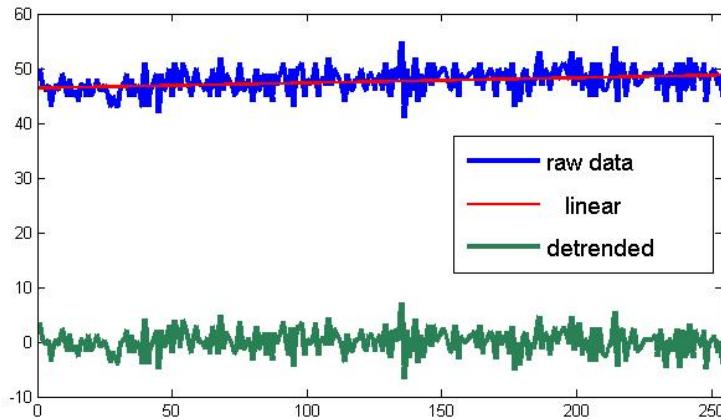


Figure 3.2: Example of detrending: the original signal over time of one feature (in blue) was approximated by a polynomial of order 1 (red line), which was then subtracted from the original signal to give the detrended signal (in green).

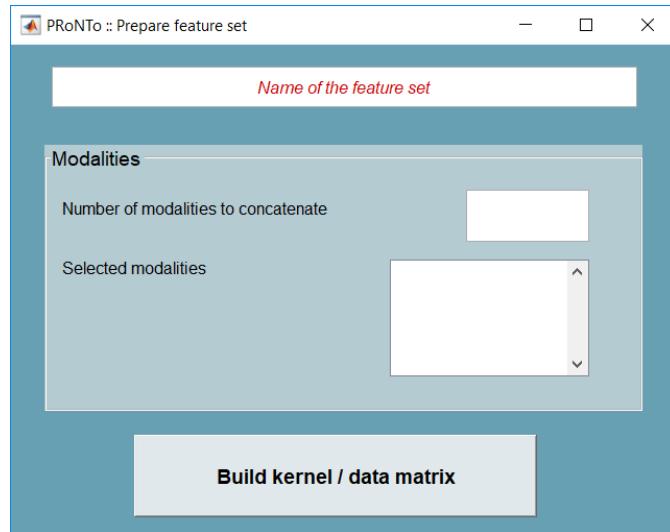


Figure 3.3: ‘Prepare feature set’ GUI

can’t be fulfilled, we recommend the use of external drives for the whole analysis. Regarding the computational expenses, we tried to minimize their effect by extracting/computing the features only once per modality: when preparing other feature sets using the same modality and detrending parameters, the built file array will be accessed for the next steps.

Be careful that using the same modality but different detrending (for NIfTI) or averaging (for MEEG) methods and/or parameters will force the re-computation of the file array for the considered modality. In the same way, changing the dataset (`PRT.mat`) from directory might lead to the re-computation of the feature sets if the file arrays were not moved accordingly.

3.3 Prepare feature set

After defining the name of the FS and also, in case of multiple data formats, the number of modalities to include, the ‘Specify modality’ window is launched to allow the user to specify the different options and parameters for each modality.

3.3.1 NIfTI and .mat data

For NIfTI and .mat data formats, the window looks like figure 3.4.

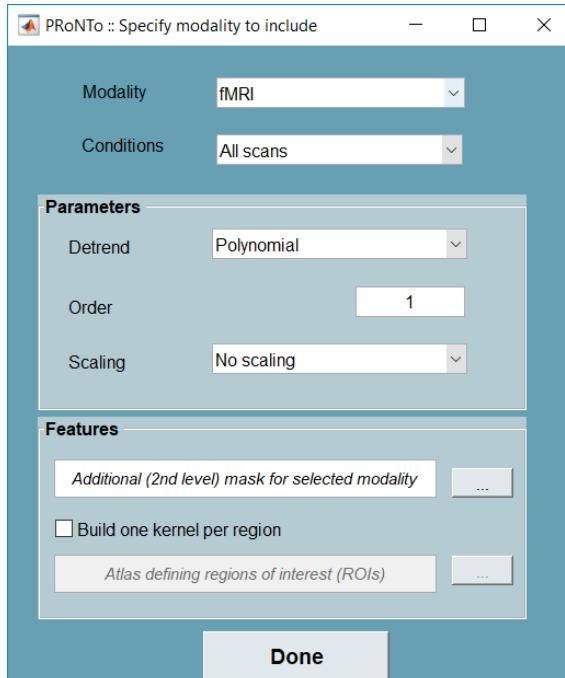


Figure 3.4: ‘Specify modality’ GUI for NIfTI and .mat data.

In this window the user has to first choose which modality to include based on its name (first pull-down menu) and which conditions to use to build the kernel. In ‘all scans’ the kernel matrix will be computed between all scans within the time series of all subjects and in ‘all cond’s’ the kernel matrix is computed only between the scans corresponding to the specified conditions of interest (see ‘Data & Design’). By default, the toolbox will use all scans to compute the kernel. With large datasets however, computational expenses can be reduced by selecting the last option. All other options are facultative.

Parameters

The first panel refers to operations to perform on the features:

- **Detrend:** By default, the parameter is set to ‘No detrending’. However, we recommend to perform a detrending in the case of time series data such as fMRI (and only in that case). When selecting polynomial, the ‘order’ parameter will appear, with a default value of 1. Changing this value will increase the order of the polynomial used to fit the data. If ‘Discrete Cosine Transform’ is selected, the editable parameter corresponds to the cutoff frequency (in seconds) of the high-pass filter. Please note that, when including more than one run (‘modality’) into a feature set, nothing will prevent the user from using different detrending methods/parameters. We however highly recommend to use a consistent detrending in the same FS. **Note:** Since .mat files are assumed to have been pre-processed prior to using them with PRoNTo, detrending is mainly required only for fMRI data.
- **Scaling:** Allows the specification of constant values to scale each scan. ‘no scaling’ is the default option. However, when dealing with quantitative modalities such as PET, the user should provide a .mat file containing a variable vector called ‘scaling’ of the same size as the number of scans in that modality.

As previously mentioned, the detrending is performed before the features are saved in the file array, while the scaling is performed only when building the kernel.

Features

The second panel allows to select a subset of the saved features to build the kernel. Since the data in a .mat file are assumed to have already been pre-processed to contain only the relevant features (and hence there is no need for a mask), this panel is mainly useful for NIfTI data. Two options are available:

- **Additional (2nd level) mask for selected modality:** This option allows the specification of a ‘second-level’ mask, which would for example define Regions of Interest (ROIs) on which the classification/regression can be performed. In this case, the features used to compute the kernel (and only the kernel) would be the ones contained in both the first and second-level masks. Therefore, using one first-level mask and two second-level masks would create two kernels but only one file array. The user has to type the full name (with path) of the mask or browse to select the mask image. When left empty or untouched, features are selected from the first-level mask specified in the ‘Data & Design’ step. Otherwise, features within both the first and second-level masks will be selected to build the kernel.
- **Build one kernel per region:** Starting from v2.0, PRoNTo allows to build one kernel per region as defined by an anatomically defined atlas, specified by the user¹. One atlas (Anatomical Automatic Labelling, AAL) is provided in *your PRoNTo folder/atlas*. Atlases can be generated easily through SPM, or manually by the user. There are no constraints on how regions are built, as long as all the voxels within each region have a specific integer value. The toolbox will identify the different regions based on the values in the voxels. Each region will then act as a second-level mask and one kernel will be built for each region. The kernels are all saved together and will then all be used at the modelling stage.

These options are performed at the kernel level only. This means that any change in one of these options would lead to the computation of a new kernel but not to the (re)computation of the file arrays. The use of different second-level masks or scaling parameters can therefore be easily envisaged.

From PRoNTo v2.0 one is allowed to build multiple kernels. These kernels can be derived from multiple modalities or from multiple regions of interest as defined by an atlas within each modality. These two options are not mutually exclusive and it is also possible to build multiple kernels within each modality and then combine those modalities as multiple kernels. The number of kernels would hence become *number of modalities* \times *number of regions*. In the same way, it is possible to concatenate multiple runs of an experiment while building one kernel per region.

3.3.2 MEEG data

This part is probably the most different from all other new functionalities in PRoNTo v3.0 (Figure 3.5). It reflects the different dimensions of the data that features can be extracted from. There are 3 main panels, corresponding to the 3 types of features (Channels, Time Points and Frequencies) an MEEG file can have.

Channels

First of all, the user can play on the channels, selecting them all or only some, as well as average the signal over the selected channels, or build one kernel per channel. More specifically:

- The list of channels is displayed on the left panel, as unselected. To add a channel to the feature set, click on it. It will then appear on the right panel. Alternatively, you can select ‘All’ channels or all ‘Good’ channels. On Windows OS, the ‘bad’ channels appear in red in the list.
- **Average signal over channels:** this option will average the signal over all selected channels. This might be useful to obtain an average trace across a specific region of interest.
- **One kernel per channel:** whether to build one kernel per channel (ticked) or not. This option corresponds to building an atlas with each channel as a ‘ROI’.

¹The atlas corresponds to a mask, except that the value of the features in each defined area correspond to a unique value, e.g. for the case of fMRI data, all voxels in fusiform have the value 3, and all voxels in orbito-frontal have the value 50.

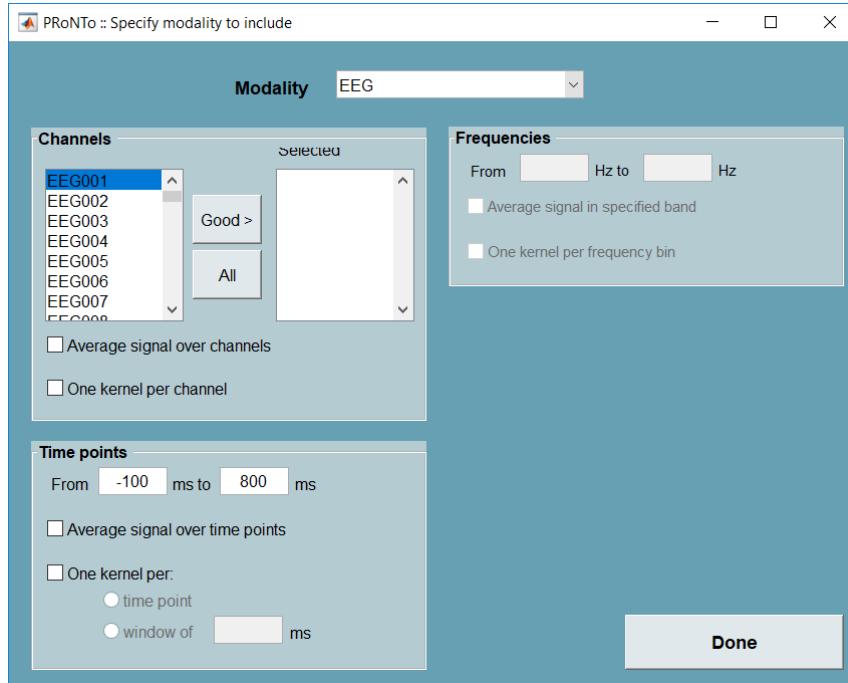


Figure 3.5: ‘Specify modality’ GUI for MEEG data.

Time points

Similarly, the user can select a subset of the whole time window by specifying which time points (milliseconds) to extract features from. The signal can then be averaged over those time points, or the user could build one kernel per time point or one kernel per time window of a specified amount of milliseconds). More specifically:

- **Time window:** The time window is displayed and can be modified to include specific time points. When entering a value in milliseconds, PRoNTo will identify the closest corresponding time point.
- **Average signal over time points:** The signal can be averaged across all selected time points.
- **One kernel per:** One kernel can also be built, either per time point or per sub-windows of XX milliseconds (user-specified).

PRoNTo does not discard kernels when they have time points less than the user’s specified time window, so to avoid big imbalances in the feature numbers across kernels you should ensure you provide the right window length. For example if you want to have 100ms subwindows between -100ms and 1100ms, the end time point should be 1099ms (because of the time point at 0ms). In this example, for balanced kernels, please specify the end of your global time window (here 1100) as the end time point desired minus your sampling interval in ms (here 1ms). So in that case the end of your global time window would be at 1099ms.

As a precautionary measure against a few time points ending up defining a whole kernel, whenever the defined global time window is not properly divisible by the defined time window, PRoNTo will automatically append the extra time points to the last kernel, if these extra time points account for less than 30% of the defined time window, and will only create a new kernel to put the extra time points, if those are more than 30%.

Frequencies

Finally, if the data set contain frequency information (i.e. multiple frequency bins), the user can select which bins to analyze, average the signal within the band of interest or build on kernel per frequency bin. More specifically:

- Similar to time points, the range of frequency bins included in the data can be viewed and specific sub-ranges selected.
- **Average signal in specified band:** This is where the user can specify the frequency band to be averaged, as specified above. This is useful to test multiple frequency bands without having to manually perform the averages as a pre-processing step.
- **One kernel per frequency bin:** This button will make PRoNTo build a kernel for each frequency bin.

Note 1: Features that are not present in the 1st file are greyed out and the corresponding panel cannot be accessed (e.g. frequencies in the present case).

Note 2: It is not possible to average over one feature and select multiple kernels for that same feature (which is logical). It is however perfectly acceptable to build multiple kernels on more than one feature (e.g. one kernel per channel and per sub-time window of 50ms).

The `PRT.mat` structure saves all information linked to the file arrays in a `fas` field (standing for ‘File Array Structure’), which size corresponds to the number of selected modality in all feature sets. The selected options and the link to the kernel (saved on the disk as a `.mat`) are stored in a `fs` field (standing for ‘Feature Set’), which size corresponds to the number of feature sets defined by the user.

Important note: When working with Graphical User Interfaces (GUIs), some messages might appear in MATLAB workspace. These can display information about the operations currently performed or explain why the toolbox does not do as the user expected (e.g. when a file could not be loaded or if information was input in a wrong format). Therefore we strongly encourage the user to have a look at MATLAB prompt when using GUIs.

Finally, we want to remind the reader once more that in the main window, from PRoNTo v3.0 while it is still possible to concatenate multiple modalities (i.e. runs) in samples, it is NOT possible to build one kernel per modality anymore, as this functionality has been transferred to the model step.

3.4 Batch interface

The ‘Feature set / Kernel’ `matlabbatch` module (Figure 3.6) allows the input/selection of all parameters and options aforementioned. Just note that the batch is based on the names of the modalities and/or conditions. Therefore, for the batch to work properly, names should be consistent across all steps, starting from data and design to the model specification and running.

Important note: Defining all important steps in one batch and running that batch will overwrite the `PRT.mat` previously created and thus delete the links between the `PRT.mat` and the computed kernel(s) and feature set(s). The file arrays would then be recomputed each time the batch is launched. For large datasets, we therefore recommend splitting the batch in two parts: a data and design and prepare feature set part and a second part comprising the model specification, run model and compute weights modules. This would indeed allow changing, e.g. model parameters, without recomputing the feature sets and kernels.

3.5 PRT structure

The PRT structure has been affected by these changes for the `fs` and `fas` structures (MEEG example):

In `PRT.fs.modality`, the fields ‘aver’ and ‘multkern’ were added. They represent which features averaging or multiple kernels were performed. This information is important for the construction of the weights later on. The ‘dim_m’ field also keeps the indexes of which channels, frequency bins and time points were used. Again, it is useful for the weights and the display of the weights (to obtain channel labels for example).

Regarding the file array structure, the `hdr` is the object of the first file for the first subject of the modality. It allows to keep the channel labels, time points and frequency bins, as we would for image orientation or so. ‘`idfeat_img`’ is empty as it is not possible/useful to have a 1st level mask.

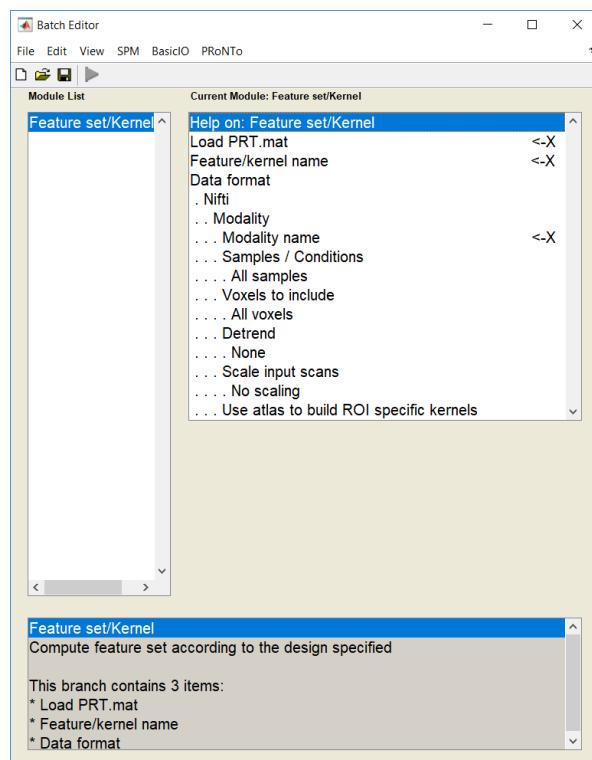


Figure 3.6: `matlabbatch` GUI for feature set building.

Chapter 4

Model Specification and Estimation

Contents

4.1	Introduction	49
4.2	Model specification	49
4.3	Feature set	50
4.4	Model type / pattern recognition algorithm	51
4.4.1	Classification	51
4.4.2	Regression	52
4.4.3	Hyper-parameter optimization	53
4.5	Cross-validation	54
4.6	Model estimation	57
4.7	Batch interface	58
4.8	Model: Specify from	60
4.9	Important changes from PRoNTo v3.0	60

4.1 Introduction

In PRoNTo model specification is the core step of the pattern recognition analysis and entails setting up the combination of the different components. For example, model specification is where you select which data features to use as input (i.e. feature sets or kernels), what type of prediction to perform (e.g. classification or regression), which machine learning algorithm to use (e.g. Support Vector Machines, Gaussian processes, ...), which cross-validation strategy to employ (e.g. ten-fold cross-validation, leave one run out, ...) and which operations to apply to the data/kernel before the algorithm is trained/tested. The framework provided by PRoNTo is highly flexible and supports most types of supervised pattern recognition analysis typically performed in neuroimaging. This chapter provides an overview of each of the components making up a model in PRoNTo. The presentation will focus on the user interface although it is important to note that the batch system provides several advanced options not available in the user interface (described further).

4.2 Model specification

To begin a model specification with the PRoNTo user interface, select ‘Model: Specify new’ from the main PRoNTo window. This will launch the model specification window (Figure 4.1). Next, select the `PRT.mat` containing the data and design information as well as the feature sets and kernels. Note that at least one feature set must be defined in this structure before a model can be created. See chapter 3 for details on constructing feature sets.

Enter a unique name to identify the model, which is used internally in PRoNTo, by the batch system and for display purposes. It is a good idea to select a meaningful but short name (without spaces). **Note:** the `PRT.mat` data structure retains a permanent record of all models created but if a model with the specified name already exists in the `PRT.mat` data structure, it will be automatically overwritten.

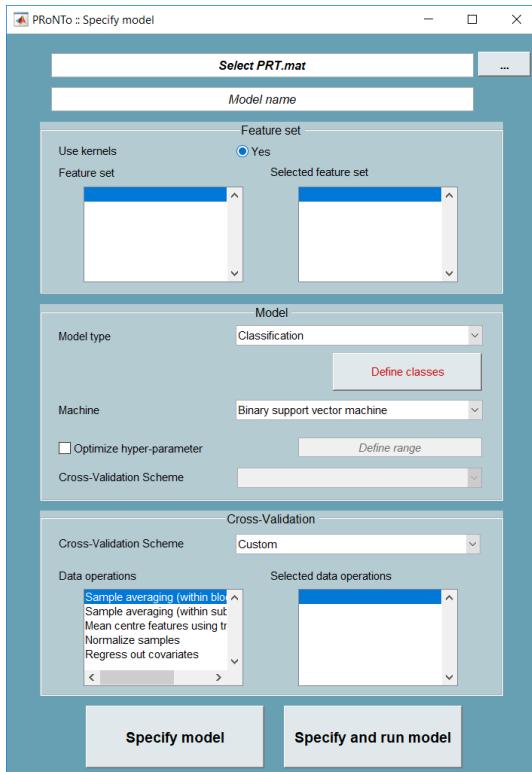


Figure 4.1: Model specification GUI

4.3 Feature set

The drop-down list titled ‘Feature set’ that existed in versions 2.X has now been changed to two lists, the ‘Feature set’ list which includes all the available features sets in that PRT, and the ‘Selected feature set’ which includes the selected feature sets for the particular model that the user wants to build. This change serves the fact that from v3.0 it is now possible to combine feature sets at the modelling level. If a Multiple Kernel Learning (MKL) method is selected, the kernels will be used in the MKL (even if they themselves contain multiple kernels). If a single kernel method is chosen, the kernels will be added, which in case of linear kernels corresponds to the features being concatenated. This feature has been added to both the batch and the GUI. As before, the kernels should have the same number of samples (e.g. number of subjects or number of trials) to be combined, but not necessarily the same number of features.

Select the appropriate feature set(s) from the drop-down list. If more than one run/session are available for each subject, each run could be input as an independent modality in the data and design step and a single-subject classifier might be specified using leave-one-run-out cross-validation.

As already mentioned in the first chapter, PRoNTo interfaces LIBSVM. However, only the SVM algorithm was used so far. In the current release PRoNTo now interfaces LIBLINEAR as well and uses more algorithms from both libraries. In addition, the ‘custom’ machine can now be optimized given hyper-parameters. PRoNTo also allows the optimization of more than one parameter (entered as a cell array of values that is then transformed into a grid).

The non-kernel route has also been enabled, i.e. the ‘Use kernel’ radio button is now functional and can be ticked off to machine learning algorithms that work directly with the features instead of kernel machines. This is recommended for samples with a low number of features. Please note that using these algorithms on typical neuroimaging datasets (i.e. # features >> # samples), will likely lead to time-consuming estimations and poor performance.

4.4 Model type / pattern recognition algorithm

In this part the model type should be chosen (i.e. classification or regression), in addition to the classes/samples to be used and the specific machine learning algorithm. In the current release, three pattern classification algorithms using kernels are supported, binary support vector machines, Gaussian processes (binary and multiclass) and L1 Multi-Kernel Learning. There are also four pattern regression algorithms using kernels, Gaussian process regression, kernel ridge regression¹, relevance vector regression and L1 Multi-Kernel Learning). From PRoNTo v3.0 there is also a new pattern regression algorithm, linear epsilon-insensitive SVM (ϵ -SVM) regression. Finally from PRoNTo v3.0 there are also 5 available non-kernel pattern classification algorithms: Binary SVM using L1- and L2- regularization, Multiclass SVM, as well as Logistic Regression using L1- and L2- regularization.

Note: Be aware that since from PRoNTo v3.0 the option for ‘building one kernel per modality’ has been removed from the feature step, and the option to combine kernels has been transferred to this step, for both classification and regression models. The ‘L1 Multi-Kernel Learning’ machine becomes available as an option in the drop-down list, only if more than one feature set have been selected.

Note: If you select more than one feature set, but the selected classification/regression technique is a single kernel method (e.g. SVM or KRR), the kernels will first be summed before entering the classification/regression modelling. This corresponds to concatenating the features before building the kernel. For regions of interest in a single modality, the summed kernel is hence equivalent to a whole brain model. However, if the ‘Normalize samples’ operation is included this equivalence will not hold.

The PRoNTo user interface provides a mechanism for flexible definition of which parts of the data or experimental design should be used for each classification or regression model. Note that this will not necessarily be the whole experiment; for example, in a complex fMRI experiment there may be several groups, each containing multiple subjects, each in turn can have multiple experimental conditions (e.g. corresponding to different subprocesses of a cognitive task). In such cases, it is usually desirable to ask several different questions using the data, such as discriminating between groups for a given experimental condition (‘between group comparison’), discriminating between experimental conditions for a fixed group (‘between-task comparison’) or training independent pattern recognition models for different subsets of subjects. All of these can be easily defined via the user interface by clicking the ‘Define classes’ button (for classification) or ‘Select subjects/sean’ (for regression).

4.4.1 Classification

The class selection panel is displayed in figure 4.2. First, define the number of classes, noting that some classification algorithms (e.g. support vector machines) are limited to binary classification, while other classification algorithms (e.g. Gaussian processes) support more than two classes. Enter a name for each class - again, it is a good idea to make these names informative but short. Notice that immediately after the number of classes has been specified, the group-, subject- and condition selection panels are greyed out. To enable them, simply select one of the classes from the drop-down menu.

For each class, select the subjects and conditions (if any) that collectively define that class. It is possible to select multiple experimental conditions in the same class, but this complicates model interpretability and potentially also model performance. If a condition or subject is erroneously selected, click on it in the ‘selected subject(s)’ or ‘selected condition(s)’ panel and it will be removed from the list.

When specifying the classes, some of the restrictions we previously imposed were lifted. For example, it is not needed that all the subjects have all the conditions. This is especially the case when the user wants to pool multiple conditions together. The code will simply ‘skip’ that condition for subjects who do not have it. It is also now possible to subsample the classes through a new button ‘Subsample according to smallest class’ in figure 4.2, such that they match (as close as possible) the number of trials/examples in the class with the lowest number of samples. If multiples classes are pooled together, this subsampling takes care to include as many trials/samples from each of the pooled categories as they are represented in the pooling.

¹Kernel ridge regression is equivalent to a *maximum a posteriori* approach to Gaussian process regression with fixed prior variance and no explicit noise term.

The performance of classification models is evaluated based on measures such as total accuracy, class accuracies and positive predictive values (representing the sensitivity and specificity).

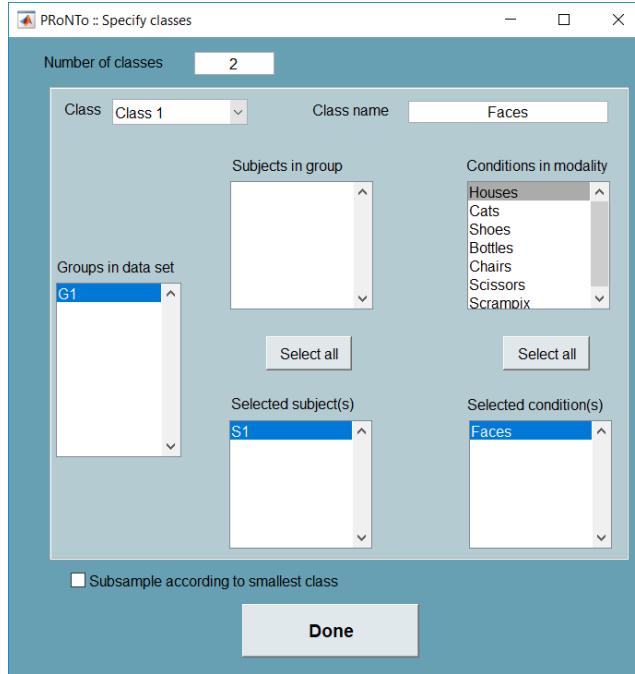


Figure 4.2: Subject / condition selection panel for classification models

4.4.2 Regression

Regression is a generic term for all methods attempting to fit a model to observed data in order to quantify the relationship between two groups of variables. Traditionally in neuroimaging massively univariate strategies (e.g. GLM) have been largely used, where, for example, MRI data for each voxel are independently fitted with the same model. Statistics test are used to make inferences on the presence of an effect at each voxel/feature (e.g. *t-test*). Pattern regression, on the other hand, takes into account several input variables (voxels/features) simultaneously, thus modelling the property of interest considering existing relations among the voxels/features.

Although most studies exploring predictive analyses in neuroimaging have been related to classification, regression analysis has aroused interest in neuroscience community for its ability to decode continuous characteristics from neuroimaging data. This approach has potential to be used when the examples (patterns) can be associated to a range of real values. The objective is to predict a continuous value instead of predicting a class to which the example belongs. These values usually refer to demographic, clinical or behavioural data (as age, blood pressure or scores resulting from a test, for example). Different metrics can be used to compute the agreement between the predicted values and the actual ones, such as Pearson's correlation coefficient (*r*) and Mean Squared Error (MSE).

The specification of which subjects and scans to include in regression models is similar to that for classification, see Figure 4.3. In the current release, multiple regression targets can be specified at the 'Data & Design' level to perform between-subject regression, however only one regression target can be selected at the model level as version 3.0 does not enable multi-output prediction. Moreover regression targets can be also input along the conditions of an experimental design to perform within-subject regression.

The interface for selecting regression targets has also been modified to match that of the classes, as regression targets can be entered within subject.

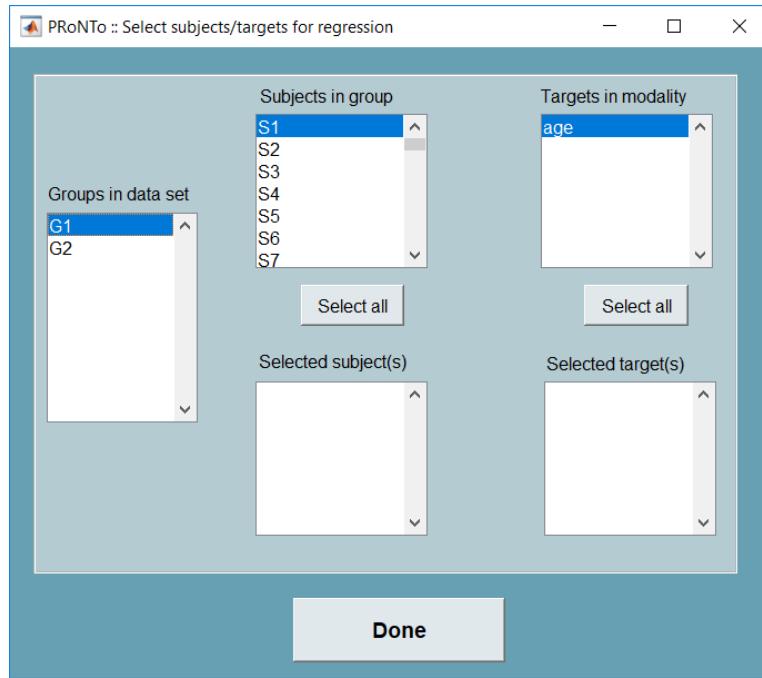


Figure 4.3: Subject / condition selection panel for regression models

4.4.3 Hyper-parameter optimization

From PRoNTo v2.0, it is possible to optimize hyper-parameters of the machine learning models. For example, the soft-margin (a.k.a. C) hyper-parameter in SVM can be optimized, using a nested cross-validation scheme. In this case, there are two loops in the cross-validation scheme. The inner loop is used for parameter optimization and the outer loop is used for assessing the model's performance. More specifically, the data is divided into training and testing sets according to the cross-validation scheme selected (outer loop). For each fold of the outer loop, the training set is further divided into training and testing sets according to the cross-validation scheme selected (inner/nested loop). The inner loop is used to train and test the model with each value of the hyper-parameter specified by the user. The parameter leading to the highest performance in the inner/nested loop (according to balanced accuracy for classification and Mean Squared Error for regression) is then used in the outer loop. For each fold of the outer loop, the model is trained using the 'optimal' value of the hyper-parameter and tested on the data that was left out (and which was not used for parameter optimization). This nested CV procedure can lead to different values of the hyper-parameter to be selected in each fold. These are stored in the outputs of the model and can be reviewed in the 'Display Results' panel.

Optimizing the hyper-parameter might lead to improved results compared to fixed values. This will usually depend on the number of features in the data: for example, for whole brain models based on SVM classifiers, with many more features than images/trials, it is reasonable to assume that changing the hyper-parameter will not affect the model performance significantly due to the high dimensionality of the data with respect to the number of examples. However, when using regions of interest (in a second-level mask or in a MKL model) or .mat files, the ratio between the number of features and the number of examples might be much smaller. In this case, different values of the hyper-parameter might lead to different performance and optimizing the hyper-parameter is desirable.

Performing a nested cross-validation can be computationally expensive. For computational efficiency, PRoNTo allows to specify different cross-validation schemes for the 'outer' and the 'nested' CV².

²For example, the outer CV could have more folds, to use as much data as possible in each fold for prediction, while the nested CV would not need as many folds to select the 'optimal' value of the hyper-parameter (e.g. k-folds CV).

From PRoNTo v3.0, the optimization of more than one parameter (entered as a cell array of values that is then transformed into a grid) is also allowed. For example, the soft-margin parameter can be optimized for SVM and for MKL (classification and regression). In the same way, it is possible to optimize the λ ridge parameter for KRR. If no value is provided, those parameters will take the values 0.01, 0.1, 1, 10, 100 and 1000, i.e. $10^{[-2:3]}$.

4.5 Cross-validation

In the final part of the specify model input form, select the type of cross-validation to employ. Cross-validation (CV) is a crucial part of the pattern recognition modelling and is used to assess the generalisation ability of the model and to ensure the model has not overfit the data. Typically this is done by partitioning the data into one or more partitions: a ‘training set’, used to train the model (e.g. fit parameters) and a ‘testing set’ used to assess performance on unseen data. By repeatedly repartitioning the data in this way, it is possible to derive an approximately unbiased estimate of the true generalisation error of the model.

The cross-validation schemes in neuroimaging applications are leave-one-subject out (LOSO; exclude one subject for testing, train with the remaining), leave-one-run-out (LORO; leave one fMRI run out for testing, train with the remainder) and leave-one-block-out (LOBO; leave out a single block or event and train with the remainder). LOSO is suitable for multi-subject designs, while LORO and LOBO are suitable for single subject designs, where the former is better suited to designs having multiple experimental runs and the latter is appropriate if there is only a single run. From the previous release (PRoNTo v2.1), PRoNTo supports each of these, and also supports leave-one-subject-per-group-out (LOSGO) for classification, which is appropriate if the subjects in each group are paired or for repeated measures experimental designs. For all the aforementioned cross-validation schemes, PRoNTo also includes its k-folds counterparts which have more recently been shown to provide better estimates.

A new cross-validation option was added which is related to the new option ‘Subsample according to the smallest class’. So from PRoNTo v3.0, PRoNTo also supports ‘Leave-One-Block-per-Class-Out’ (and its k-folds counterpart). It gets complex with pooling and subsampling, as the code is selecting the trials condition by condition. We added a random shuffling of the train/test division in the class containing pooled categories to make sure that one category is not left aside entirely in one fold, another in another fold, etc. An example of CV matrix after pooling the first 2 classes together and subsampling is shown in figure 4.4. Please note that LOSGO is not appropriate for classes defined from a design. Versions 1.1 and later allows k-fold cross-validation for each of the available schemes. This means that the user specifies the number of folds (‘k’) and that the data is partitioned according to that number. For example, specifying $k = 4$ will use 25% of the data to test the model, and 75% to train it. Note: $k = 1$ splits the data in half, training the model on the first half and testing on the second, i.e. there is no circular partitioning. We recommend using k-folds CV when possible as Leave-One-Out has been shown to lead to over-optimistic model performance estimation.

From PRoNTo v2.0, a GUI allows the user to fully specify a customized cross-validation scheme. First, a ‘basis’ needs to be specified (Figure 4.5). Three options are available:

- Load a .mat containing a previously computed CV matrix (needs to contain the variable ‘CV’).
- Select a basis from the pop-down list (contains the same options as for the outer CV).
- Specify the number of folds.

When an option has been selected, a new window will appear (Figure 4.6). The top panel of this window is a table that can be edited. Each row refers to a sample selected in the definition of the classes (or to perform regression on). Each column represents a fold. For each column, the different samples can have a value of 2 (test set), 1 (train set) or 0 (unused in this fold). Setting a whole fold to 0 takes it out of the CV matrix. Note: it is possible to change the value of multiple samples by changing the value of the last sample to modify, then shift-select the first one to modify. This also works across folds. The bottom panel displays the structure of the data selected for further classification or regression, along with a preview of the built CV matrix.

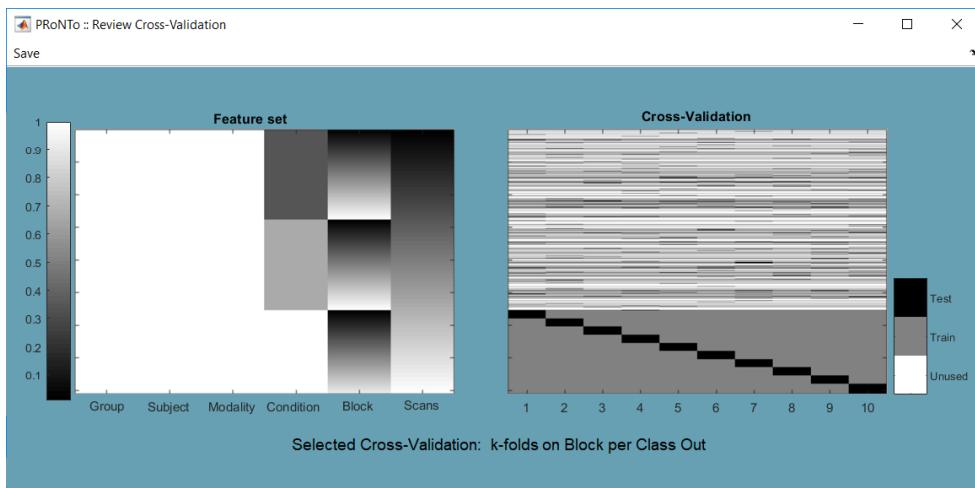


Figure 4.4: Example of CV matrix after pooling the first 2 classes together and subsampling

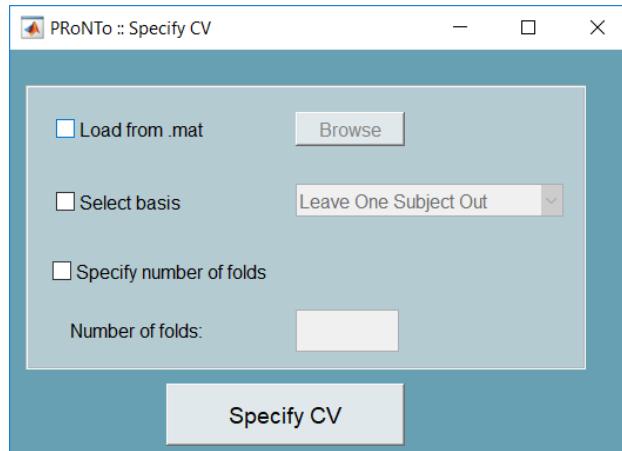


Figure 4.5: Specify CV window to build custom cross-validation

The resulting CV matrix can be saved in a .mat, alongside the PRT (name: *model name_ CV.mat*). This matrix can be loaded as a custom CV in the batch, if exactly the same samples were selected for modelling.

Note: The ‘custom’ CV option is not available as a nested/inner cross-validation scheme.

Information concerning the cross-validation structure is stored internally in matrix format, and can be visualised by clicking ‘Review Kernel and CV’ from the main ProNT0 window (see 4.4 and 4.7 for two examples). In the left panel, this figure indicates which group, subject, modality and condition each scan in the feature set belongs to. On the right, each cross-validation fold (partition) is displayed as a separate column and each scan is colour coded according to whether it is in the training or testing set (or if it is unused).

It should be emphasised that the type of cross-validation selected should be appropriate for the experimental design. For example, it does not make sense to select a leave-one-subject-out cross-validation approach for single subject designs. It is also important to ensure that the training and testing sets are completely independent to avoid the cross-validation statistics becoming biased. This is particularly important for fMRI, where successive scans in time are highly correlated. For example, if a leave-one-block-out approach is employed and the blocks are too close together then the independence of the training and testing set will be violated, and the cross-validation statistics will be biased (technically this is governed by the autocorrelation length of the fMRI timeseries and the temporal blurring induced by the haemodynamic response function). This can be avoided if care is taken to ensure that overlapping scans are discarded from the design (see chapter 2), but it is a very

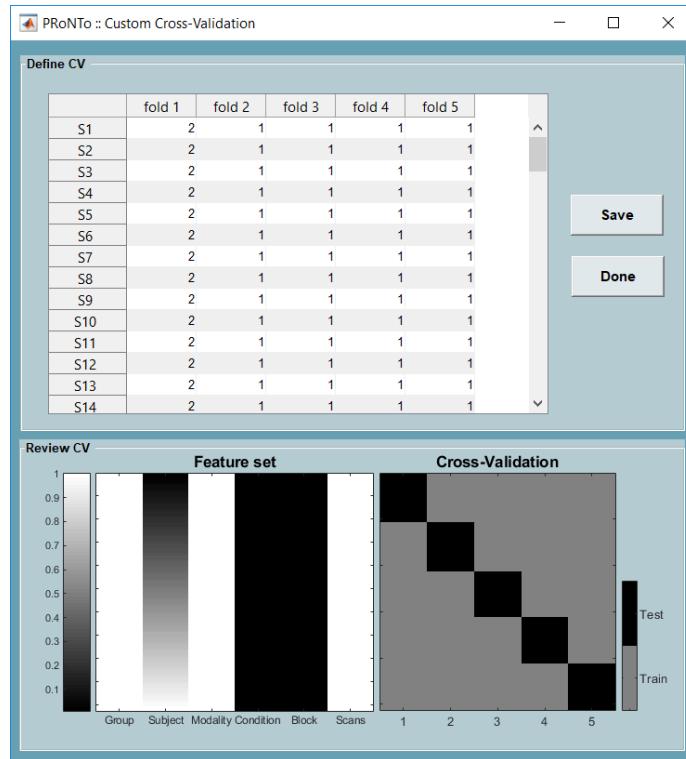


Figure 4.6: Custom Cross-Validation Window. For each fold, the user can specify which samples (e.g. images/-trials) are part of the training and test set, or are unused

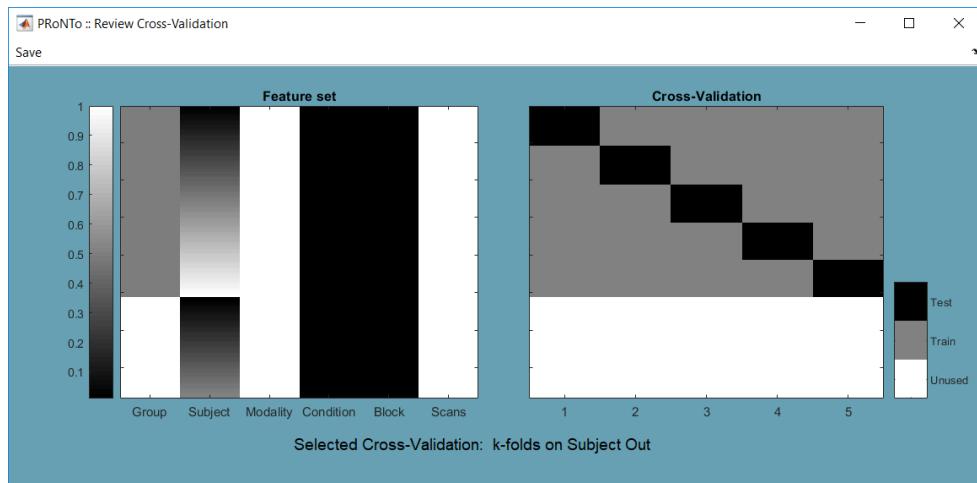


Figure 4.7: Review cross-validation window

important issue, and the user should still be careful to ensure that cross-validation folds are sufficiently far apart in time (especially for LOBO cross-validation).

During the model specification, it is also possible to select one or more operations to apply to the data/kernel. Each of these operations is described below:

1. **Sample averaging (within blocks):** Constructs samples by computing the average of all scans within each block or event for each subject and condition.
2. **Sample averaging (within subjects):** Constructs samples by computing the average of all scans within

all blocks for each subject and condition.

3. **Mean centre features using training data:** Subtract the feature-wise mean from each sample vector.
4. **Normalize samples:** Scales each sample vector (i.e. each example) to lie on the unit hypersphere by dividing it by its Euclidean norm.
5. **Regress out covariates:** Regresses out the covariates entered for each subject/sample using training data only. If this operation is selected, PRoNTo will always remove confounds first before other data operations are performed.
6. In case of non-kernel algorithms we have two extra data operations.
 - **Normalize features:** Also called Min-Max scaling, is to rescale the features so that they lie in a fixed range, with min=0 and max=1.
 - **Z-score features:** Also called standardization, is to rescale the features so that they will have the properties of a standard normal distribution with $\mu=0$ and $\sigma=1$, where μ is the mean (average) and σ is the standard deviation from the mean.

A crucial point to note is that all operations are embedded within the cross-validation structure such that the parameters of the operation (e.g. mean and standard deviation of the features) are estimated using the training data only. This prevents a very common mistake in pattern recognition from occurring, whereby parameters are computed using the whole data set prior to cross-validation. Observing a complete split between training and testing sets during all phases of analysis ensures that accuracy measures are an appropriate reflection of the true generalisation ability of the machine and are not biased because of improper applications of preprocessing operations to the entire dataset.

Other points to note include:

1. The order of operations can make a difference. For example, subtracting the mean then dividing each data vector by its norm is not the same as performing the operations the other way around.
2. Data operations (1) and (2) have no effect if no design is specified or for events with a length of one TR.

At a minimum, we recommend that features should be mean centered during cross-validation. In addition, for multiple kernel learning, we advise the user to normalize each kernel after mean centering. This will compensate for the fact that different kernels might be computed from examples/samples with different number of features (e.g. different regions contain different numbers of features).

The different operations selected for a specific model can then be reviewed using the ‘Review Kernel and CV’ (starting from v2.0). The selected operations will be listed below the kernels (‘Show kernel’).

4.6 Model estimation

Using the GUI, it is possible to either ‘Specify’ the model, or ‘Specify and Run’ the model. The first option saves all the parameters of the model in the PRT structure. This information can be found in `PRT.model(m).input`, where m is the index of the model. The second saves all the parameters of the model and then runs the model. In this case, the inputs, which include the cross-validation matrix, the target values or labels, and the machine (e.g. binary SVM, Gaussian Process, etc.), are fed to the estimation routines, which will then add to the PRT an output field (`PRT.model(m).output`) containing the estimated parameters, statistics, and other information from the learning process.

In some cases (e.g. multiple models to run or models with nested CV and/or slower machines), it would be desirable to estimate models later on (e.g. just before lunch break or at the end of the day). The ‘Model: Run’ option allows to select multiple models and run them one at a time automatically (Figure 4.8).

From PRoNTo v3.0 running permutation tests to check the statistical significance of the results has been transferred from the ‘Display results’ module, to the ‘Model: Run’ module. So from this version, one has to

specify the permutation testing in this step of the process.

The first thing that needs to be done using this window is to specify which PRT we would like to work with. PRoNTo will then read the available models from this structure and display the list of models on the left panel. These models can be selected (the selected models will show on the right panel) by clicking each model individually or by clicking the ‘Select all’ button in the middle of the panels. Then you also have to check the options ‘Perform permutation test’ if you want to run permutation tests, specify the number of repetitions, and also check ‘Save permutation parameters’ to perform further statistical tests if needed³. Finally, to estimate the model(s), one needs only to click the bottom button ‘Run model(s)’.

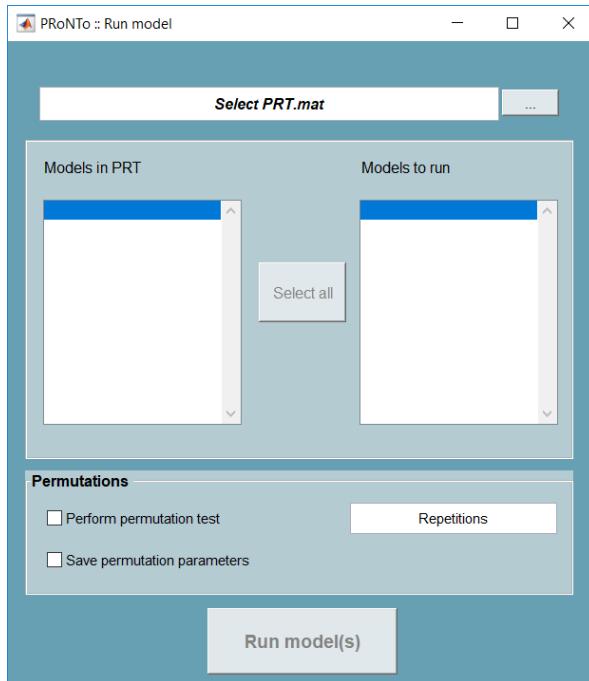


Figure 4.8: Choose models to be estimated

It is useful to have a look at what is displayed in the MATLAB command space while the model is being estimated. Information such as the number of folds can help double-check that everything is going as expected. Furthermore, if some options were specified (e.g. using a feature set with multiple kernels) that are not available at the modelling step (e.g. to be modelled with SVM), warnings will be displayed, as in Figure 4.9.

4.7 Batch interface

The batch module includes all the functionality provided in the user interface and allows complex analyses to be scripted in advance. As noted, the batch module also provides functionality not available in the user interface. The most important difference is that the batch module allows customised MATLAB functions to be used as prediction machines. This functionality allows PRoNTo to be easily extended to allow many types of classification and regression algorithms not provided under the current framework. This can be achieved by selecting ‘Custom machine’ under the ‘Model Type’ heading. This allows a function name to be specified (i.e. any *.m function in the MATLAB search path). The behaviour of this custom machine can then be controlled by a free-format argument string. See the developer documentation and the examples in the `machines/` subdirectory of the PRoNTo distribution for more information. Another important difference between the batch and user interfaces is that mean centering data vectors across scans is enabled by default in the batch. Also, flexible CV is available in the batch only in the form of ‘load a .mat’. This .mat must contain a variable called ‘CV’,

³For further information regarding permutation testing the user is referred to 5.4.3.

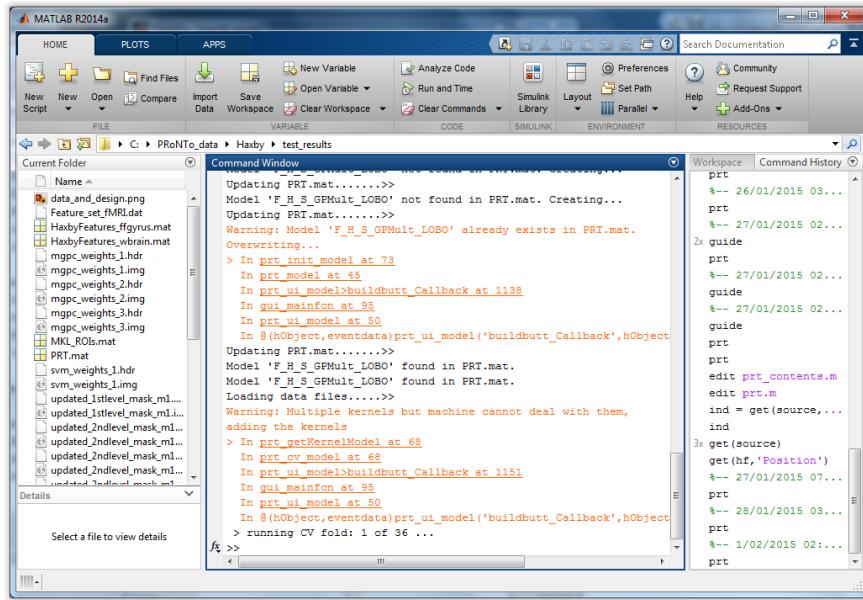


Figure 4.9: MATLAB workspace displaying warnings when kernels are added

specifying the CV matrix for the selected trials/images. An example of the batch window for model specification is provided in figure 4.10.

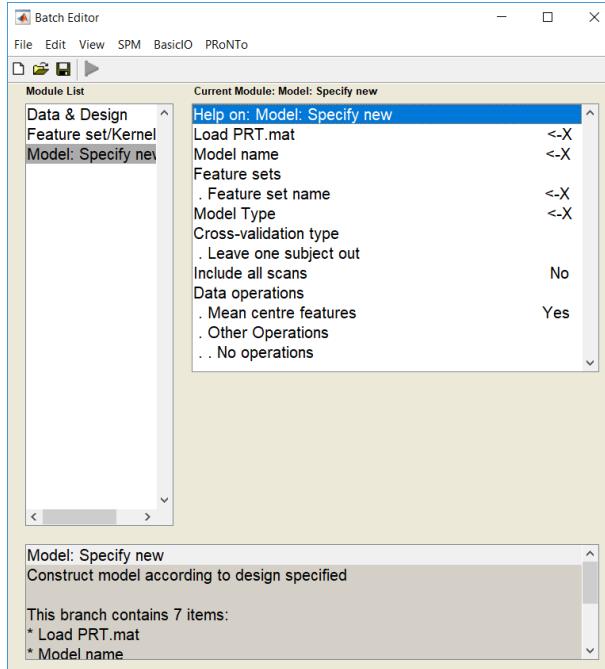


Figure 4.10: Batch interface to specify a model

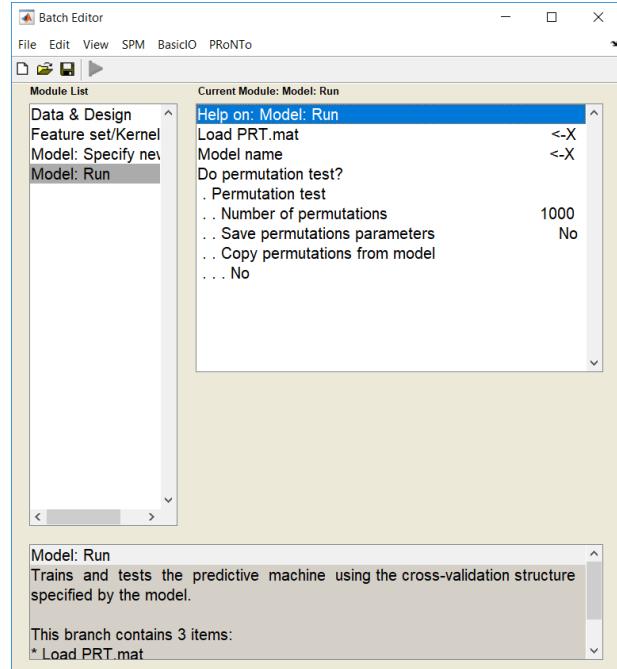


Figure 4.11: Batch interface to run a model

As displayed in Figure 4.10, the batch does not allow to specify and run the model directly. Instead, the user has to add a 'Model: Run' module. Just as in the GUI, it is in the 'Model: Run' **batch** module where you specify whether you want to perform permutations along model estimation. Furthermore, it is possible to save the predictions and the balanced accuracy (for classification) for each permutation, to perform further statistical tests if needed. The 'Model: Run' **batch** module is presented in Figure 4.11. **Note:** Dependencies on the model name are available when performing a 'Model specification' followed by a 'Model: Run' module.

4.8 Model: Specify from

The main ‘Specify model’ interface has been modified to add a new module both in GUI and in Batch, which is called ‘Model: Specify from’. This will launch a window in GUI or in Batch (Figures 4.12 and 4.13, respectively) that is very similar to (and heavily based on) the ‘Model: Specify new’ window, but with a couple of changes: there is an extra popup menu that allows to select which model to copy from. Some fields have also been disabled to ensure comparability of the models: only the feature sets, the machine (with its hyper-parameter optimization) and the operations can be modified. Classes/Regression selection and outer CV options cannot be modified. This new module is intended to facilitate new analysis when the goal is comparing different feature sets or different algorithms keeping the other model specifications fixed.

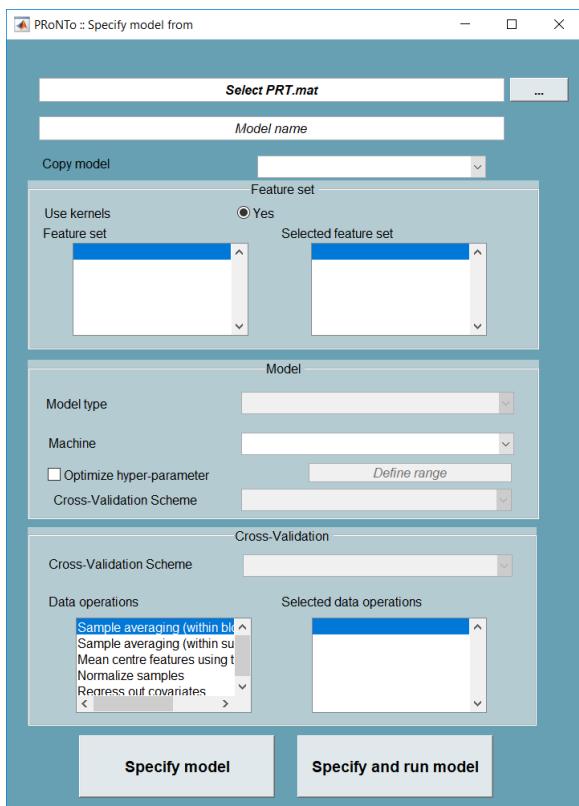


Figure 4.12: ‘Model: Specify from’ in GUI.

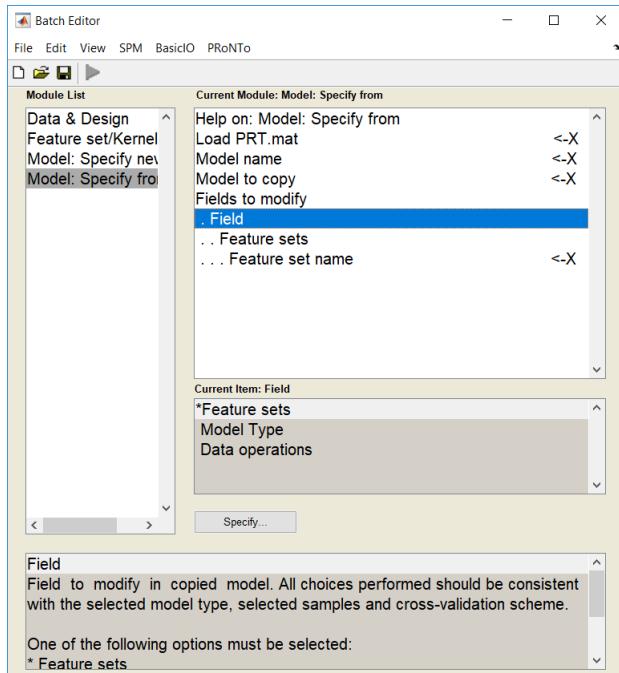


Figure 4.13: ‘Model: Specify from’ in `matlabbatch`.

4.9 Important changes from PRoNTo v3.0

As described in ‘What’s new’ there are two important modifications in the PRoNTo code that will affect the model estimation and performance measures.

The first one is how the parameters are optimized: This change only applies when multiple values of the hyper-parameter lead to the same maximum value of model performance. In previous versions, PRoNTo was choosing the median value. From v3.0, PRoNTo identifies the largest stable region (if the ‘Image Processing’ toolbox from MATLAB is present) and chooses the center of gravity of this region in the hyper-parameter grid as the best hyper-parameter. This change should have limited impact.

The second one is how the stats are computed: Until PRoNTo v2.1 the predictions were concatenated across fold and the overall performance was computed based on the concatenation of predictions. This way to estimate

model performance can however lead to over-optimistic model performance estimations. So from PRoNTo v3.0 the performance measures are computed for each fold and then averaged across folds. This is also reflected in the ‘Display results’ window where some plots have been replaced at the model level (e.g. replacing the overall confusion matrix by a ‘balanced accuracy distribution’ across folds in the plot). In general, the results will reflect more the average but also the deviation of the results across folds.

The reader is referred to [13] for further information regarding the justification for the two aforementioned changes and more general for model performance estimation.

Chapter 5

Display Model Performance

Contents

5.1	Introduction	63
5.2	Launching results display	63
5.3	The main results display window	64
5.4	Measuring model performance	65
5.4.1	Classification	65
5.4.2	Regression	65
5.4.3	Permutation testing	67
5.5	Visualizing the model performance	67
5.5.1	Classification	68
5.5.2	Regression	71
5.5.3	Influence of the hyper-parameter on performance	73

5.1 Introduction

Once a machine (e.g. a classifier or a regression function) has been specified, its parameters have been estimated over training data, and its performance has been evaluated over a testing set using cross-validation (or nested-cross validation), it is necessary to examine the outcome of the whole procedure in detail. The results windows enables the user to see the model's performance evaluated by different metrics.

Examining model output and parameters is helpful in diagnosing the potentially bad performance of a particular model. For example, if the machine cannot perform above chance, it could be due to lack of predictive information in the data, inappropriate experimental design, large amount of noise in the data, insufficient amount of data, wrong choice of features, or the wrong choice of machine. It is important to recognise that any of these factors could cause the modelling to fail.

Model performance can be reviewed using the ‘Display Results’ GUI. Alternatively, all computed statistics are saved within the PRT structure, in the `PRT.model(m).output.stats` field, with `m`, being the index of the model to review.

5.2 Launching results display

Make sure all previous steps have been performed (Data & Design, Chapter 2; Prepare feature set, Chapter 3; and Model: Specify/Run, Chapter 4)

In the ‘Review Options’ panel of the main PRoNTo window, press ‘Display Results’. At the ‘Select PRT.mat’ window, navigate to where your ‘PRT.mat’ file is stored (using the left column), and select it. The main results window will open and look as represented in Figure 5.1. In the ‘Model’ panel in the top-right corner, the list of

models that have been successfully estimated will appear. Note: there will be a ‘beep’ if one or more models were specified but not estimated (‘Model; Run’) and their name will appear in the command window.

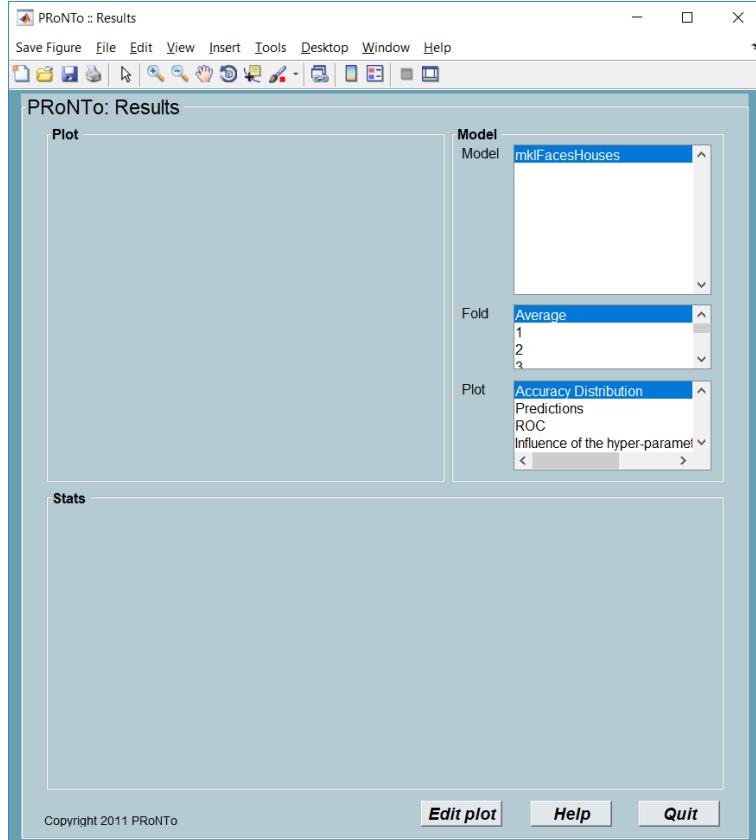


Figure 5.1: Initial state of the results display main window.

5.3 The main results display window

The window is divided into three panels; going clockwise from top left to bottom left, they are:

Plot : This panel displays different result plots that can be visualized in PRoNTo for a specific model. With the exception of the confusion matrix plot, these cannot be interacted with.

Model : This panel allows the user to select the model to visualize, whether to visualize a particular fold or all folds at once, and which plot to produce.

Stats : The **stats** panel allows the user to visualize a variety of performance metrics (based on the selected fold), including accuracy for classification models and MSE for regression models. In addition, p-values for these metrics based on permutation tests can also be visualized.

To populate the ‘Plot’ panel, first click on a model in the ‘Model’ selector, then on ‘All folds / Average’ (or a particular fold) in the ‘Fold’ selector, and finally on a plot in the ‘Plot’ selector. The next section details the plots available. The window also comprises ‘Edit plot’, ‘Help’ and ‘Quit’ buttons. The ‘Edit plot’ button exports the displayed plot in an extra window, such that it can be edited and easily saved. The ‘Help’ button provides information on each panel of the window (not as detailed as in this manual) and the ‘Quit’ button closes the results window. In addition, the GUI menu comports a ‘Save figure’ option (on the top left) that acts as a ‘printscreen’ of this window (with white background), which can be saved for records/publications.

5.4 Measuring model performance

One of the main questions to ask is ‘how good is a predictive model?’. In regression, goodness-of-fit is often assessed via mean squared error and coefficient of determination (R^2). In classification, a common practice is to compute prediction accuracy, both for each class and for all test data. Once a specific performance metric has been obtained, it is also possible to obtain a p-value for the metric, reflecting how certain we are that the result is not due to chance.

The statistics table gives a summary of the model’s performance. Model performance is estimated differently for classification and for regression. In PRoNTo, classification performance is assessed using total accuracy (TA), balanced accuracy (BA), class accuracies (CA) and class predictive values. For regression, the agreement between the real and the predicted targets is computed based on the coefficient of determination (R^2), the mean squared error (MSE), the normalized mean squared error, and the correlation between the targets and the predictions.

5.4.1 Classification

The accuracy acc is the total number of correctly classified test samples divided by the total number of test samples N , irrespective of class. The accuracy is exactly equivalent to

$$acc = 1 - \frac{1}{N} \sum_n l_{01}(y_n, f(\mathbf{x}_n)), \quad (5.1)$$

where $l_{01}(y_n, f(\mathbf{x}_n))$ is a 0-1 loss function that counts each classification error as costing 1 and each classification success as costing 0:

$$l_{01}(y_n, f(\mathbf{x}_n)) = \begin{cases} 0 & y_n = f(\mathbf{x}_n) \\ 1 & y_n \neq f(\mathbf{x}_n) \end{cases} \quad (5.2)$$

Balanced accuracy takes the number of samples in each class into account, and gives equal weight to the accuracies obtained on test samples of each class. In other words, the class-specific accuracy is computed by restricting the sum of equation 5.1 to be taken over C disjoint subsets of the whole testing data, where each subset contains only test samples from one class. This produces a set of class-specific accuracies $\{acc_1, \dots, acc_C\}$, from which the balanced accuracy can be computed as

$$acc^{bal} = \frac{1}{C} \sum acc_c. \quad (5.3)$$

Balanced accuracy is the measure of choice when there is class imbalance (one class, called the *majority class*, has much more data than others).

The stats panel also gives the class accuracies $\{acc_1, \dots, acc_C\}$, useful to check whether the model favours some classes over others. If class 1 represents control subjects, and class 2 represents patients, then class 1 accuracy is equivalent to specificity, and class 2 accuracy is equivalent to sensitivity. In the same way, the figure displays class positive predictive value, which represents the number of false positives for each class. An example of classification stats is displayed in Figure 5.2. Area Under the Curve (AUC) will be described later in the chapter.

5.4.2 Regression

As previously mentioned, model performance for regression is assessed by the correlation between the predictions and the targets (linear correlation), the coefficient of determination (R^2), the mean square error (MSE), and the normalized MSE. An example of such stats window is displayed in Figure 5.3.

The mean-squared error is calculated as:

$$MSE = \frac{1}{N} \sum_n (y_n - f(\mathbf{x}_n))^2 \quad (5.4)$$

Where y_n is the real target for the examples/sample n and $f(\mathbf{x}_n)$ is the predicted target for the examples/sample n. This is the standard measure when assessing goodness-of-fit for regression models.

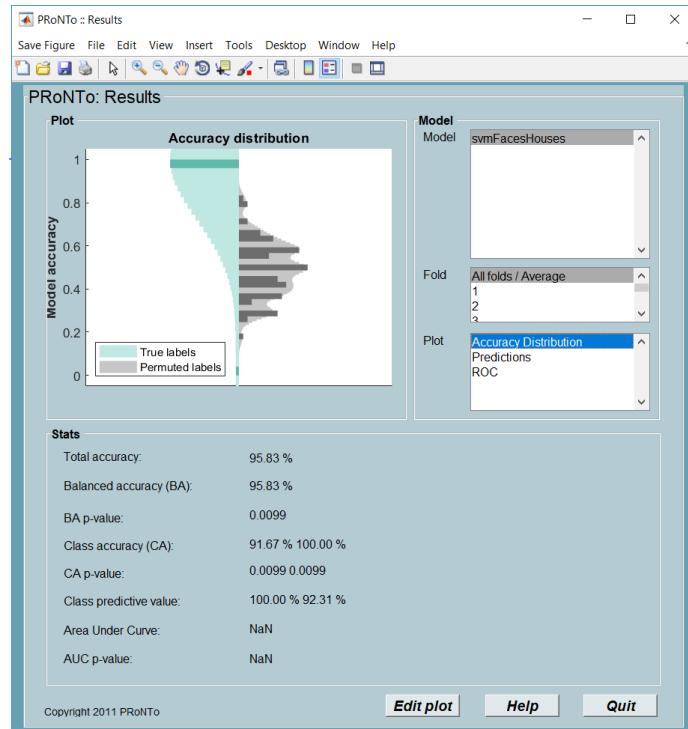


Figure 5.2: Example statistics for all folds of a two-class problem modelled by an L1-MKL.

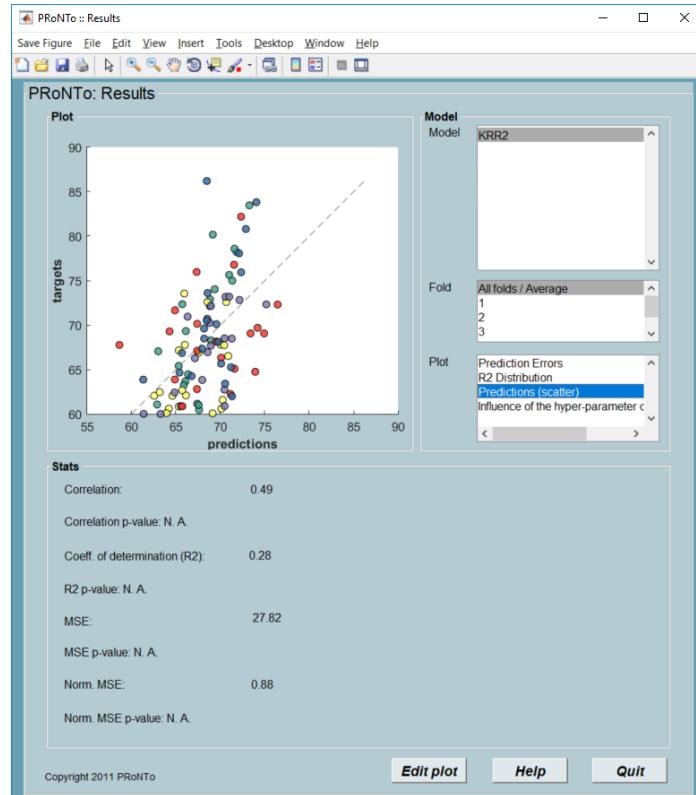


Figure 5.3: Example statistics for all folds of a regression problem modelled with a KRR.

Since the magnitude of the MSE depends on the scale of y , we also calculate the Normalised Mean Square Error, ‘Norm. MSE’:

$$\text{Norm. } \text{MSE} = \frac{\text{MSE}}{\text{var}(y)} \quad (5.5)$$

where we divide the MSE by the variance of the targets over the data. This gives a scale invariant measure of prediction accuracy. In addition, the correlation coefficient of the targets and predictions are determined:

$$\text{CORR} = \frac{\sum_n (y_n - \mu_y)(f(\mathbf{x}_n) - \mu_f)}{\{\sum_n (y_n - \mu_y)^2 \sum_n (f(\mathbf{x}_n) - \mu_f)^2\}^{\frac{1}{2}}} \quad (5.6)$$

where μ_y and μ_f are the sample means of the targets and predictions respectively. The resulting measure $-1 < \text{CORR} < 1$ provides a measure of the strength of the linear dependence between the targets and the predicted targets, with values close to zero indicating no relationship, values close to 1 indicating a positive relationship, and values close to -1 indicating a negative relationship. Values of CORR less than zero would imply that the model has performed poorly, as this would mean that targets with large values tend to be given smaller predicted values than targets with small values. However, it should be remembered that a large positive value of CORR does not *necessarily* mean that the model is giving accurate predictions, since a global scaling and shifting of the predictions gives the same value for CORR . We would therefore recommend examining both CORR and MSE , as well as the scatterplots, to verify that the model is performing well. For completeness, the statistics table also include the ‘coefficient of determination’ R^2 , which is given by

$$R^2 = \text{CORR}^2 \quad (5.7)$$

5.4.3 Permutation testing

Much of statistical theory and machine learning theory rests on the assumption that the data is IID (independently and identically distributed). However, in functional neuroimaging this assumption is often not met, due to e.g. within-run correlations and haemodynamic effects. Therefore, classical estimates of confidence intervals (such as the binomial confidence interval) may not be appropriate. Permutation testing is a non-parametric procedure that allows to obtain p-values for the performance metrics in this case. Because it requires retraining the model a number of times, which can be costly in computation time, this is not done by default.

If a user wants to run permutation test, this can be done in the ‘Model: Run’ module. In order to run permutation test for a specific model the user needs to select the ‘Perform permutation test’ button in GUI, or select the ‘Permutation test’ option in the ‘Do permutation test?’ field in `matlabbatch`. You also have to fill in the `repetitions` field with the desired number of repetitions R . Having done this, when you run the ‘Model: Run’ module, PRoNTo will estimate the model for the specified number of repetitions with permuted labels/targets, and produce a p-value for performance statistics (see Figure 5.2).

The smallest increment in p-value is equal to $1/R$ (e.g. 20 permutations gives you increments of 0.05), with a minimum value of $1/R$ (i.e. running 10 permutations will never lead to statistically significant result at the commonly used threshold of $p < 0.05$). Usually, we would recommend computing several hundreds to a thousand permutations.

For both classification and regression models, the p-value associated with each performance measure can be estimated using permutations. Until they have been estimated, ‘N.A.’ will be displayed (standing for ‘not available’).

Important note: This step is essential to assess model performance! It is not methodologically sound to simply assume that the chance level is close to 50% and that any balanced accuracy higher than that threshold is significant. Please report p-values as computed from permutations along with model performance.

5.5 Visualizing the model performance

Looking at a model output’s graphically can often yield insights into its performance. In PRoNTo, plots are different for classification and regression. As previously described, from PRoNTo v3.0 there is a change in the

way we estimate a model's performance. As mentioned in 1.4.1, model performance was typically concatenated across folds (in the confusion matrix or ROC curve). The concatenation can however lead to over-optimistic model performance estimations and further reflects a model that was not estimated (instead, we estimated multiple models). We have hence modified the code to compute the average of performance across folds. This is also, primarily, reflected in the 'Display results' window where some plots have been replaced at the model level (e.g. replacing the overall confusion matrix by a the new 'balanced accuracy distribution' performance measure across folds in a violin plot). In general, the results will reflect more the average but also the deviation of the results across folds. We encourage the users to report both values.

5.5.1 Classification

Accuracy Distribution

From PRoNTo v3.0, a new performance measure that is being displayed in the 'Plot' panel of the 'Display results' is the 'balanced accuracy distribution' of the model, together with the mean. The balanced accuracy here is defined as the average accuracy obtained on either class. So based on a typical confusion matrix notation, the balanced accuracy is given by equation 5.3. The difference between the balanced accuracy and the conventional accuracy is that the latter doesn't take into account each class individually.

So the benefits of balanced accuracy over the conventional accuracy is that some times the conventional accuracy can be high only because the classifier takes advantage of an imbalanced testing set, where in that case the balanced accuracy would drop closer to chance, which would be much more representative of the model's performance. If on the other hand the classifier performs equally well on either class, then the balanced accuracy reduces to the conventional accuracy (i.e. the number of correct predictions divided by number of predictions). Figure 5.4 shows an example of such a plot for a binary classification (SPM EEG dataset, Faces versus Scrambled).

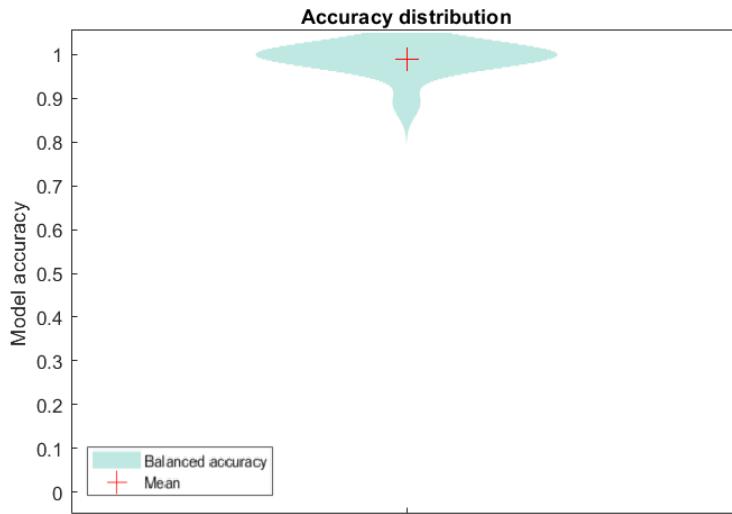


Figure 5.4: Example of a balanced accuracy distribution for a binary problem modelled by L1-MKL.

Histogram plot

The histogram plot is a smoothed density version of the predictions plot, showing how function values are distributed. A good classifier would have minimal overlap between the densities. The error rate of the classifier is proportional to the area of the overlap. The ROC curve can be thought of as the result of sweeping a decision threshold over the range of functional values, and recording the joint sensitivity/specificity values for each decision threshold setting. A typical linear SVM would have a decision threshold at 0. Figure 5.5 shows

an example of such a plot for a binary classification (SPM EEG dataset, Faces versus Scrambled, LOBO CV). From PRoNTo v3.0 with the change in the way we estimate a model's performance, this performance measure is available only for within each fold.

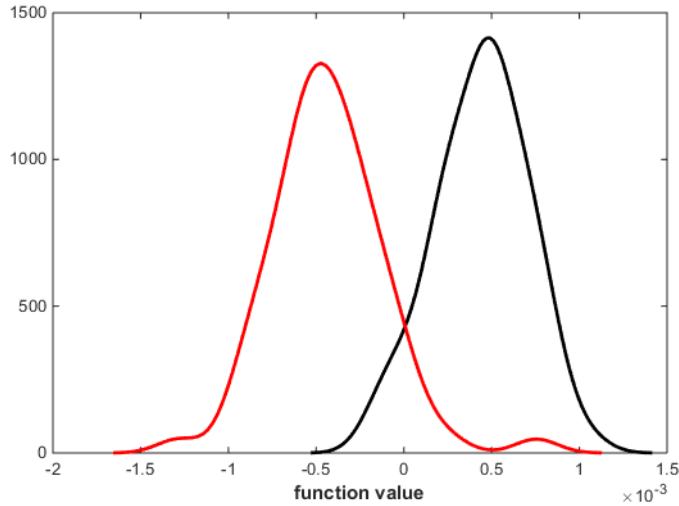


Figure 5.5: Example function values histogram curve for a binary problem modelled by L1-MKL.

Confusion matrix plot

The confusion matrix shows counts of predicted class labels $y_n = f(\mathbf{x}_n)$ (in rows) versus true class labels y_n (in columns). An ideal confusion matrix is diagonal: all predicted class labels correspond to the truth. Off-diagonal elements represent errors. It is important to check that none of the classes is “sacrificed” to gain accuracy in other classes - in other words, if all classes are equally important to classify, no class should have more off-diagonal than on-diagonal entries. Many summary statistics, including class accuracy, total accuracy, sensitivity, and specificity, can be computed from the confusion matrix. Figure 5.6 shows an example of a confusion matrix (Haxby dataset, Faces versus Houses versus Scissors, LOBO CV). From PRoNTo v3.0 with the change in the way we estimate a model's performance, this performance measure is available only for within each fold.

Predictions plot

A prediction plot displays, for a particular fold (y-axis), the output value of the machine's decision function for each test sample (x-axis, e.g., the function value corresponds to the distance of the test example to the boundary in case of an SVM or to the posterior probability of the test example belong to a specific class in case of GP). The decision boundary/threshold is displayed by a vertical line at the centre of the plot. A well-performing classifier will yield very different function values for samples of different classes, i.e. samples from different classes will fall on different sides of the decision threshold. The inspection, in each fold, of the overlap of function values between classes, can help to identify which of the test blocks/subjects/conditions is atypical with respect to the training set. This plot is available for binary classification. On the plot, each class is represented by a different marker and color, and indicated in the legend. Figure 5.7 shows an example predictions plot.

Receiver Operating Characteristic (ROC) plot

ROC curve is a commonly used measure to evaluate a binary classifier performance. It shows the trade off between the true positive rate (TPR) (sensitivity, the number of correctly classified positive instances divided by the total number of positive instances) and false positive rate (FPR) (1-specificity, the number of incorrectly classified negative instances divided by the total number of negative instances) across a number of thresholds. An ideal classifier would have an ROC passing through the top-left corner, depicting all true positives and no false positives. The area under curve (AUC) is a summary measure of classifier performance, where higher is better (1

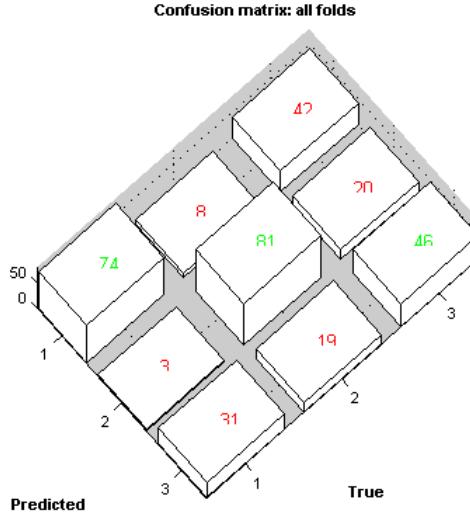


Figure 5.6: Example confusion matrix for all folds of a three-class problem modelled by GP.

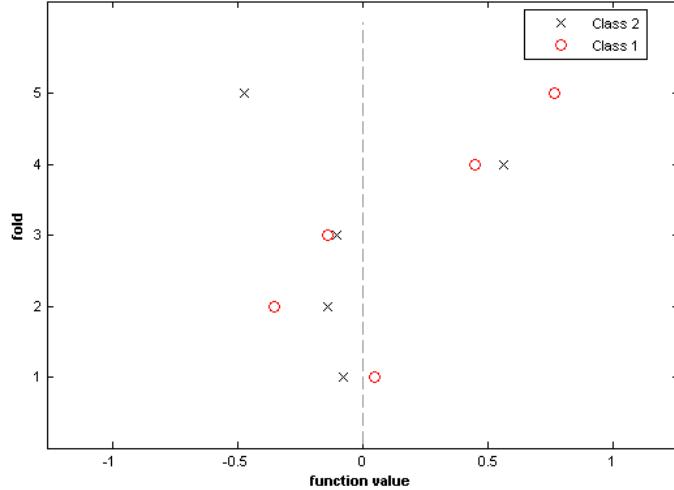


Figure 5.7: Example predictions plot for a two-class problem modelled by an SVM.

represents perfect performance, 0.5 represents random performance). Please note that there might be discrepancies between AUC and balanced accuracy as AUC measures the relative ranking of positive and negative test samples in terms of decision values while balanced accuracy reflects whether each decision value is correct or incorrect [7]. For example: assume a testing set comprising 4 images, 2 of faces (our positive class) and 2 of houses (our negative class) and an SVM binary classifier. The function values are $[-0.5087 \text{---} -0.0371 \text{---} 2.2863 \text{---} 1.5753]$, which corresponds to all scans being classified as ‘houses’ (since the decision threshold for SVM is 0). However, we notice that the decision values for the positive scans (first 2 values) are higher than for the negative scans, which leads to an AUC of 1. It is hence important to report both measures when assessing model performance and/or comparing models.

In PRoNTo, the ROC and AUC are estimated within each fold if a specific fold is chosen. Users may notice that sometimes AUC is ‘NaN’ (Not a Number) within fold. AUC is computed by using TPR and FPR, which are both proportions. However, if at least one of the two proportions doesn’t exist, AUC cannot be computed.

For example, if users choose leave-one-subject-out as the CV scheme with one image per subject, there is only one test target in each fold. Hence, there will be no positive instance or negative instance from which we can compute TPR or FPR. AUC is therefore NaN. Another case leading to a NaN value is when there is only one class in the test targets within one fold (e.g. leave-one-block-out with one class per block as in Haxby dataset), even if there are multiple test targets. In all of those cases where AUC is NaN, we still display it, such that users are aware of this limitation. Figure 5.8 shows an example of ROC plot (Haxby dataset, Faces versus Houses, 4-folds nested CV, LOBO outer CV, display All folds).

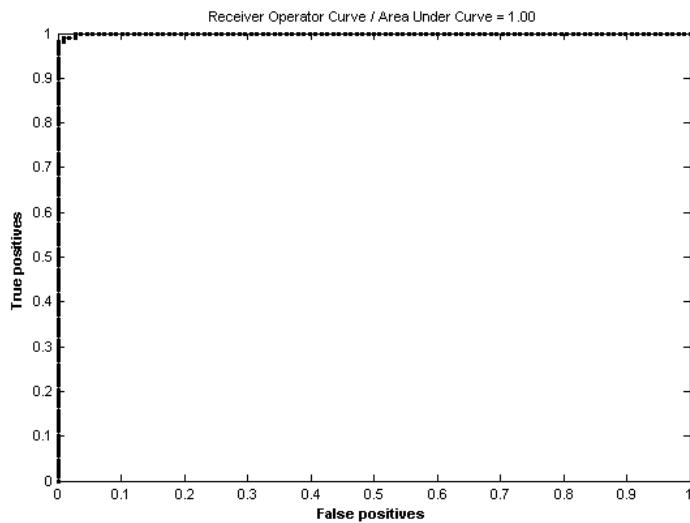


Figure 5.8: Example ROC curve for a two-class problem.

5.5.2 Regression

All plots that follow are from the regression tutorial in Chapter 14. That tutorial uses fMRI data from different subjects and the target is to predict their age. For more detailed information the reader is advised to read the tutorial.

Prediction Errors

As for classification, the plots have been modified according to reflect the fact that we are now estimating model performance within each fold and then averaging the performances. A ‘Prediction Errors’ plot was added to reflect, for each point, the difference between the target and the predictions. A perfect regression model would lead to all points being located on the ‘0’ vertical axis. An example is shown in figure 5.9.

R2 Distribution

Similarly to the ‘Accuracy distribution’ plot, a ‘R2 Distribution’ plot displays the distribution of R2 across the different folds in a violin plot. The plot is symmetric and displays the mean of the R2 distribution if no permutations were performed while it will contain the ‘true label’ R2 distribution on its left and the ‘permuted label’ R2 distribution on its right if permutations were estimated (Figure 5.10). The same thing applies to classification problems (example in figure 5.2). **Note:** Since we are plotting the distribution of the R2s of across the different folds, this performance measure is available only when you choose ‘All folds / Average’.

Predictions (scatter)

This plot represents the predicted values (x-axis) against the real values or targets (y-axis). A perfect correspondence between targets and predictions would be represented by a diagonal on this plot. Figure 5.11 displays such a plot, where each different color represents a different fold.

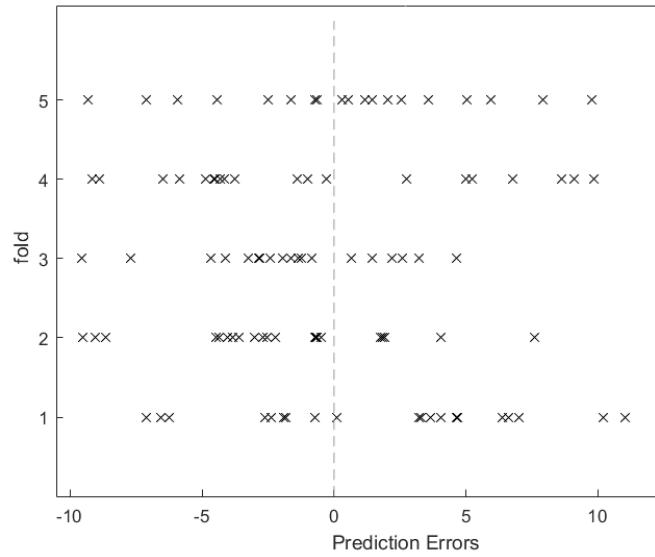


Figure 5.9: Prediction Errors using a KRR model.

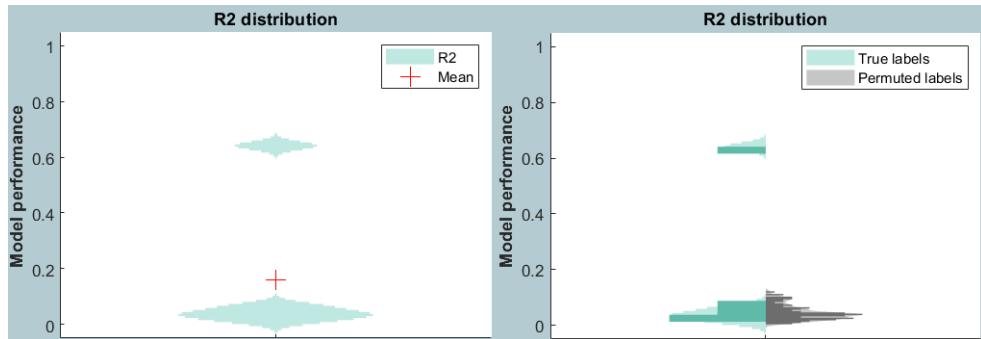


Figure 5.10: R2 distribution plot without (left) and with (right) permutations.

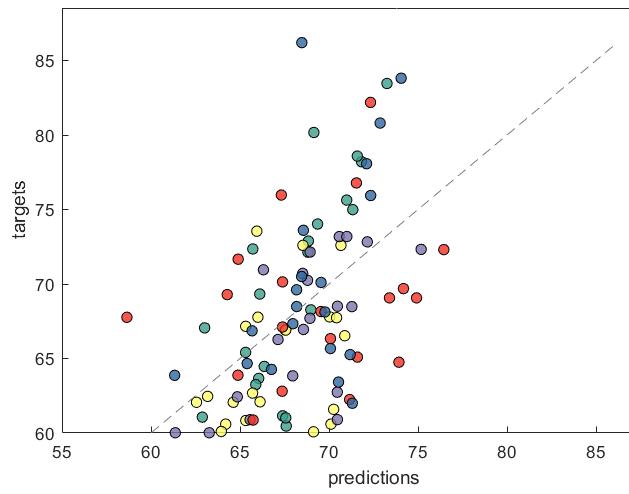


Figure 5.11: Scatter plot of target values and predictions modelled by KRR.

Predictions (bar)

This plot displays, for each image/sample, the target and the prediction in bar plots. An example plot is displayed in Figure 5.12 for the same KRR model. From PRoNTo v3.0, this plot exists only for each fold.

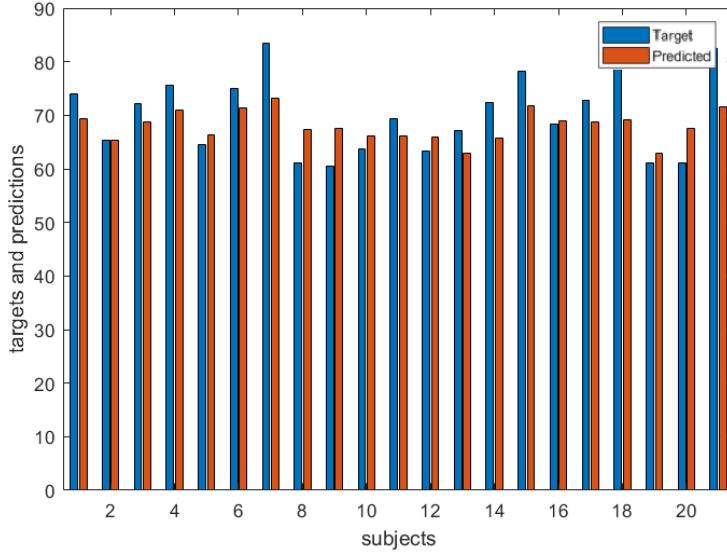


Figure 5.12: Example of bar prediction plot for a KRR model with 25 subjects.

Predictions (line)

This plot displays, for each fold, the target and the prediction, each in line plots. An example plot is displayed in Figure 5.13 for the same KRR model. From PRoNTo v3.0, this plot exists only for each fold.

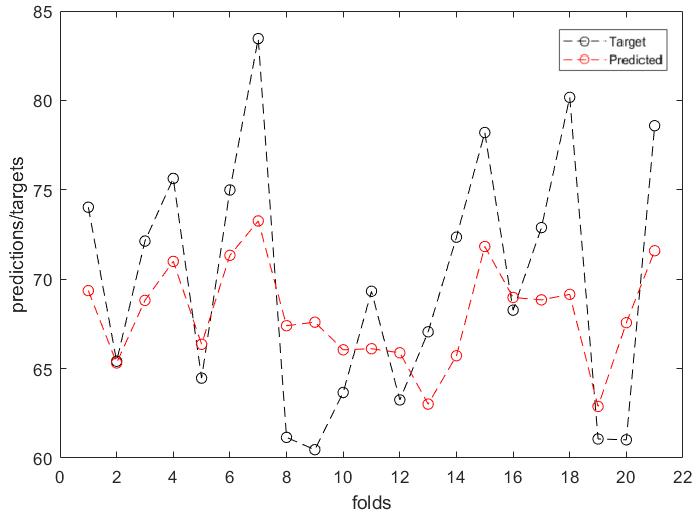


Figure 5.13: Example of line prediction plot for a KRR model with 25 subjects.

5.5.3 Influence of the hyper-parameter on performance

This plot will be present in the list if hyper-parameter optimization was performed. When displaying the average across folds, for each value of the hyper-parameter, it displays the average model performance (balanced accuracy for classification and MSE for regression, line on the plot) across nested folds, with an error bar representing the standard deviation of model performance. The frequency of selection of a hyper-parameter value (i.e. the number of times this value was returned as ‘optimal’ to the outer CV fold) is represented with a gray bar plot on the right-side y-axis. An example of such a plot is displayed in Figure 5.14 for the optimization of the soft-margin parameter in L1-MKL (Haxby dataset, Faces versus Houses, 4-folds nested CV, LOBO outer CV).

When selecting a specific fold, this plot displays the model performance for each value of the hyper-parameter, and represents the optimal value (i.e. the one leading to highest performance) in red.

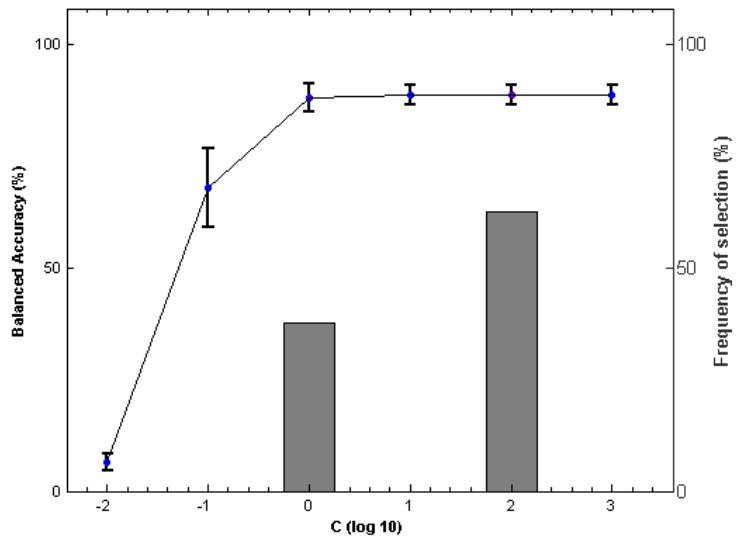


Figure 5.14: Example performance curve depending on the hyper-parameter value with frequency of selection of each hyper-parameter.

Chapter 6

Computing Feature and Region Contributions

Contents

6.1	Introduction	75
6.2	Feature weights	76
6.3	Atlas-based weights	77
6.4	Batch interface	77

6.1 Introduction

The previous modules allow the user to specify one or more models. These include the machine to be used, the cross-validation scheme and the classification/regression problem. The estimation of those models led to predictions on unseen/test data (in each fold), from which measures of performance of the model can be derived and displayed.

In addition, as PRoNTo uses linear models it provides the option of recovering the model weights in the original feature space, and transforming the weights vector into an image, or map. These maps contain at each feature the corresponding weight of the linear model (that together define the predictive function), and which related to how much this particular feature contributed to the classification/regression task in question. The weights can later be displayed using the ‘Display weights’ modules (described below).

Furthermore, the MKL machine estimates contribution of each kernel to the predictive function. This means that there will be one value per region of interest as defined by an atlas and/or per modality (depending on how multiple kernels were built). Therefore, it is possible to build maps at the kernel level, in addition to the maps at the feature level. Kernels (which can correspond to different regions, different channels or different modalities) can then be ranked according to their contribution to the model. Since L1-MKL is a sparse algorithm, only a subset of the kernels will have a non-null contribution to the model, this can facilitate model interpretation.

If the user wants to create images of the weights, using the GUI, the user first needs to click the ‘Compute weights’ button on the main PRoNTo window. This will launch the window shown in Fig 6.1. To estimate the weights and create the weight maps the user needs to select a PRT.mat file. The window is then divided in two panels: a ‘Feature weights’ panel and a ‘Atlas-based weights’ panel which are further explained in the sections below.

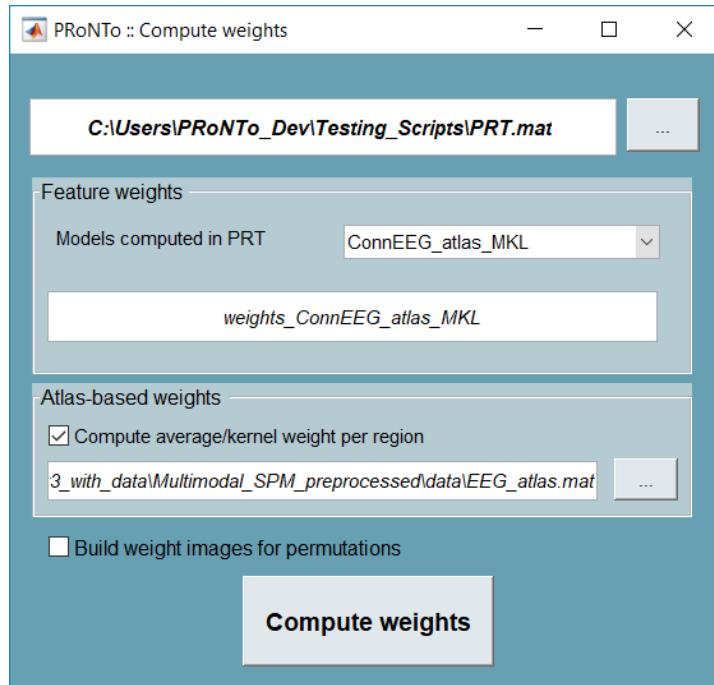


Figure 6.1: Weights computation in GUI.

6.2 Feature weights

For kernel methods the weights can be estimated as a linear combination of the training examples. For non-kernel algorithms the weights are directly computed and saved in the PRT structure. For the kernel algorithms PRoNTo saves the coefficients of the training examples. These coefficients are then multiplied by the training examples to obtain the model weights. The vector of model weights has the same dimensions of the original feature space. In case of NIfTI and MEEG data the model weights can be converted to an image with the same dimensions of the input images. This computation is done for each fold. If for example we have standard MRI data which have 3 dimensions, then the weights can be converted to a 3D image. The resulting 3D images for all folds are then assembled into a single 4D NIfTI file with dimensions [3D x (number of folds + 1)], where 1 corresponds to an extra 3D image with the averaged weights over all folds. The NIfTI file is saved in the same directory as PRT.mat. In the case of multi-class classification, one image will be built for each class, the index of the class being saved in the image name (e.g. *image_name_c1.img*). In the case of multiple modalities being considered as multiple kernels (i.e. not concatenated in samples), one image will be built for each modality, the modality name being appended to the image name. In case of data of lower dimensions, for example MEEG data in sensor space (2D), as well as data in a .mat file format, the procedure remains the same.

In the ‘Feature weights’ panel, PRoNTo will show the list of available models, and the user can choose one model for which to estimate the weights. If the selected model is the MKL modelling of ROI-based kernels, the ‘Atlas-based weights’ panel will be automatically updated, and the name of the selected atlas at the feature set step will appear. Otherwise, those fields will stay blank.

In this panel, it is also possible to define the name of the created image file, which is saved in the same directory as PRT.mat. Alternatively, if left empty, PRoNTo will name the file according to the model name, class (if multi-class machine) and/or modality (if MKL on modalities).

6.3 Atlas-based weights

The ‘Atlas-based weights’ panel allows the user to estimate the contribution to the model, of each anatomically defined region of interest (ROI), as defined by an atlas (same dimensions as for the feature weight images). If the model refers to an MKL machine estimated on kernels per region, the ‘Atlas name’ field will be filled automatically and the contribution of each region is derived from the contribution of each kernel (i.e. they correspond to the kernel weights). If this is not the case (single kernel machine or feature set), an atlas should be loaded (using the browse option). The weights will then be summarized for each region a posteriori.

Two options are available when estimating such contributions:

- Contributions of kernels estimated through MKL: In this case, the contributions of each region to the model are derived from the contributions of each kernel. This option is available for MKL modelling of feature sets containing multiple kernels based on ROIs defined by an atlas.
- Summarizing the weights according to ROIs: If, for example we have MRI data and a whole brain feature set was used, or if the kernels were added to perform single kernel modelling (e.g. SVM, GP, KRR, RVR), it is possible to select an atlas and summarize a posteriori the weights in each anatomically defined ROI. The contribution of each region is then simply the sum of the absolute values of the weights within that region, divided by the number of voxels (or features in general) in that region (see [23] for details). If summarizing weights by using an atlas is desired, this atlas should be used as a second level mask in the ‘feature set’ step to ensure a perfect overlap between the atlas and the model features. Otherwise, any voxels used as features (or features used in general) in the model but not present in the atlas will be summarized in one extra region referred to as ‘other’ that will not correspond to any anatomical region.

In both cases, the contribution of each region is divided by the total contribution of all regions. The derived values can then be seen as percentages of contribution of each region to the decision function. The contributions can be ranked, leading to a list of regions sorted by descending contribution to the model. This list can also be computed if multiple modalities were built and used in an MKL model. In this case, each modality has a contribution to the model, that can be normalized and a sorted list can be derived.

Furthermore, for MKL models - which are sparse in the number of kernels contributing to the model - a bar graph can be built, representing the number of kernels with a non-null contribution to the model. The same graph bar will depict the contribution of each region to the decision function in the case of summarized weights. In this case, the bar graph will not be sparse.

Finally, from PRoNTo v3.0, there is also a new button below the ‘Atlas-based weights’ panel, named ‘Build weight images for permutations’ which when checked, enables to build weight images for each permutation independently.

6.4 Batch interface

The `matlabbatch` module to compute the weights has the same options as the GUI. One main difference being that instead of listing the available models in a given PRT, it will ask for the name (string) of the model to be used. As for estimating models, the name of the model should be exactly the name given in ‘Model: Specify new/from’.

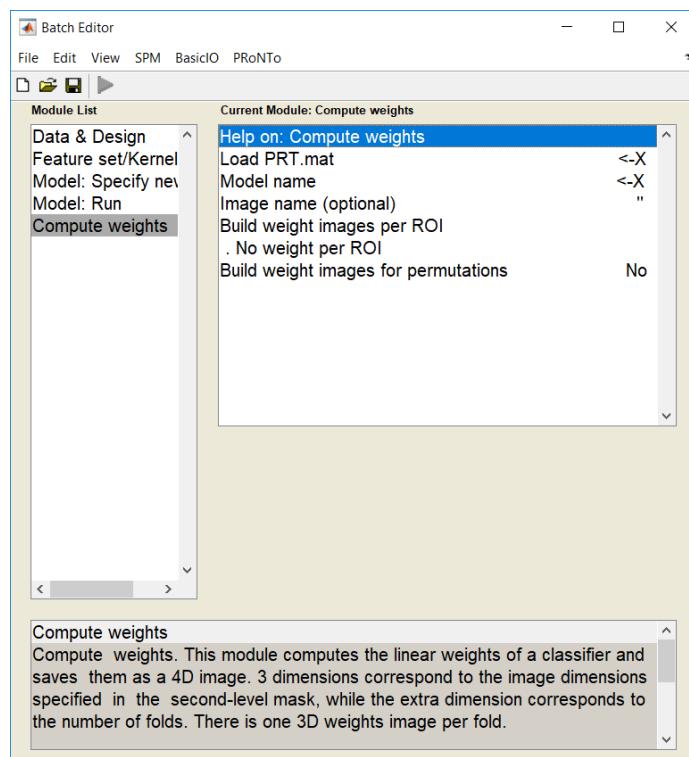


Figure 6.2: Weights computation in `matlabbatch`.

Chapter 7

Display weights

Contents

7.1	Introduction	79
7.2	Displaying weights	80
7.2.1	Select image to display	81
7.2.2	Weights map	83
7.2.3	Anatomical image	83
7.2.4	Additional plots	83

7.1 Introduction

In PRoNTo it is possible to visualize the weights for a specific model. There is a huge debate in the neuroimaging field on how to interpret weights of linear machine learning models ([9, 24, 25, 29]). Nevertheless, the model parameters or weights can provide some insights on which features are driven the predictions.

Some brain areas are probably more informative about class membership/regression targets than others. For example, in a visual task, we would expect discriminative information in the occipital lobe. This can be seen as *information mapping*, and it can be helpful to evaluate a specific model - if the discriminative weights of a machine are concentrated in the eyes, for example, it is important to correct the mask used in the analysis to exclude them. The *weight map* is a spatial representation of the decision function, i.e. every feature with non zero weight contributes to the decision function. Pattern recognition models (classifiers or regression functions) are multivariate, i.e. they take into account correlations in the data. Since the discrimination or prediction is based on the whole pattern, rather than on individual regions or features, all features (with non zero weights) contribute to the classification or regression and no conclusions should be drawn about a particular subset of features in isolation.

It is also possible (starting from ProNTo v2.0) to derive weights at the region level (as anatomically defined by an atlas, from MKL or from summarizing the weights). The ‘Display weights’ window allows to display maps of voxels/features and of region contribution. Furthermore, the region contributions can be ranked in descending order, yielding a sorted list of regions according to their contribution to the classification/regression model. We hope this will help the interpretation of model’s weights.

Moreover, as mentioned in chapter 1, starting from PRoNTo v3.0, PRoNTo supports MEEG (SPM) data as well as any .mat file, for example MEEG data in sensor space (2D, or even 1D), or any kind of .mat data, therefore the models’ weights can have different dimensions (corresponding to the data used).

Important note: The implemented version of MKL ([20]) is sparse in the kernel combination. This means that only a few kernels (which might correspond to different modalities, regions or channels, for example) will contribute to the model. However, this selection of kernels might depend on the dataset, and small variations in the dataset (as induced by cross-validation) might lead to different subsets of kernels begin selected. Therefore,

care should be taken when reporting selected kernels and each fold should be looked at separately. We also provide a quantification of the variability across folds of the ranking of the kernels ('Expected Ranking', see further) to provide some insights on this issue.

7.2 Displaying weights

To launch the 'Display weights' window, make sure that weight maps have been computed for at least one model (Compute Weights, Chapter 6) and press the 'Display weights' option. At the 'Select PRT.mat' window, navigate to where your 'PRT.mat' file is stored (using the left column), and select it in the right column. The display window then opens (Figure 7.1).

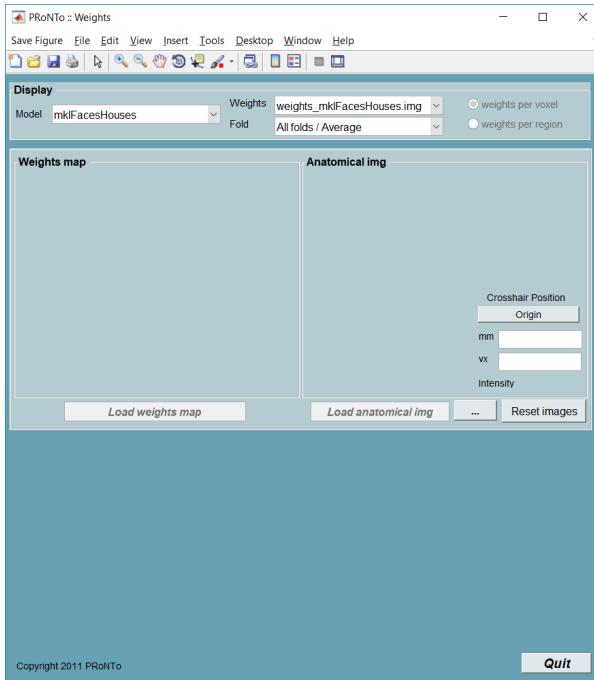


Figure 7.1: Display weights main window after selection of PRT.mat.

The window is divided into four panels. Going from top left to bottom left, they are:

Display : This panel allows the user to choose which model and weights to display. As well as whether to display the voxel/feature weights or the region contributions, whenever this is available. Finally the user can also display the average weights across all folds, or the weights of a specific fold.

Weights map : This displays different weights (according to the type and dimension of data) of the selected weight map and allows to navigate it.

Anatomical img : The user can also load an anatomical image, which if loaded, it will display three projections, and the cross-hair will be synchronised with the weight map. Note: This option only makes sense for weights for NIfTI brain data, e.g. MRI or fMRI.

Additional plots : The blank area at the bottom of the window will display additional information about the model weights, such as a sorted list of the kernels (e.g. regions or channels) according to their contribution (if weights per region/channel were computed, in the form of a table) and a bar plot of the relative kernel contribution. If MKL modelling was performed based on multiple modalities, the same table and bar plot will display the relative contributions of each modality to the decision function.

7.2.1 Select image to display

The ‘Display’ panel shows the models for which weights have been computed and weight images were found in the same folder as the PRT. For each model, the list of images available is displayed in the ‘Weights’ pop-down list. Typically, one image will be created for a binary classifier or regression with only one modality or multiple modalities concatenated as samples (e.g. multiple runs). On the other hand, multiclass classification models will return one image per class (with the index of the class appended to the name of the image). In the same way, multiple modalities used in multiple kernels will lead to the building of one weight image per modality. For each image, the weight map can be displayed for each fold or for their average. Starting from PRoNTo v2.0, in case of NIfTI brain data, it is possible to display the contributions of each voxel (‘weights per voxel’) or of each region (‘weights per region’, if previously computed).

Note: the weight images (per voxel and per region) are automatically detected in the list of files in the PRT folder according to the name specified in the ‘Compute weights’ step (Chapter 6). Modifying the image name afterwards or moving the images might lead to warning messages and the images will not be listed in the GUI.

To display a weight image, select a model, an weight image and a fold. If we have MRI data, and if only weights per voxel were estimated, the window will look similar to Figure 7.2. Notice that the ‘weights per region’ radio button is currently deactivated.

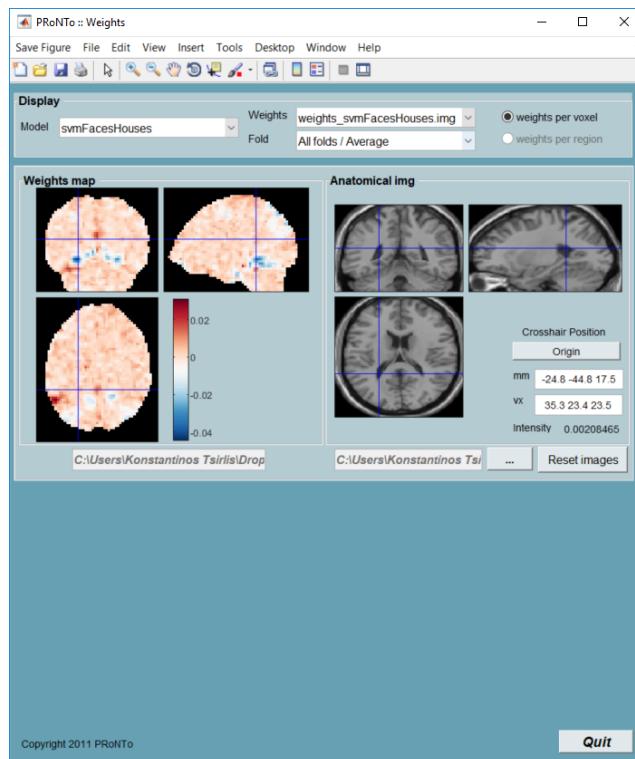


Figure 7.2: Displaying weight image of a two-class SVM model.

As we mentioned before, from the current release (v3.0), PRoNTo also supports MEEG (SPM) data, as well as data in simple .mat format. Therefore, weights maps can be built and displayed for all modality types, i.e. for MEEG data, both directly from their original MEEG format, and also from the NIfTI images generated from them inside SPM, and also from any generic .mat files that were analyzed. More specifically:

- **For 1D data:** For MEEG and .mat, weights are displayed as bar graphs (with the color of the bar representing the amplitude of the weight) (e.g. vector with brain connectivity features used for .mat). Example in figure 7.3.

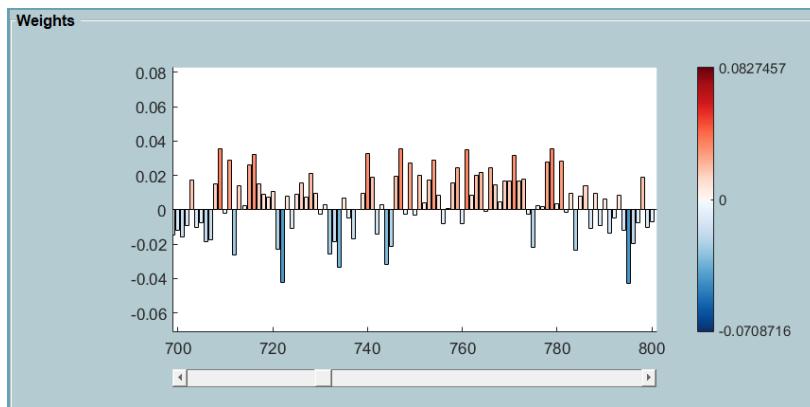


Figure 7.3: ‘Display weights’ for 1D MEEG or .mat data.

- **For 2D data:** The display shows the matrix, with y-axis as the 1st dimension and x-axis as the 2nd dimension (after squeezing out dimensions of size 1), with the color of a (x,y) pair displaying the magnitude of its weight. Example in figure 7.4. Finally, 2D NIfTI images (like MEEG data) are displayed as before, except a small change in the color map. Example in figure 7.5.

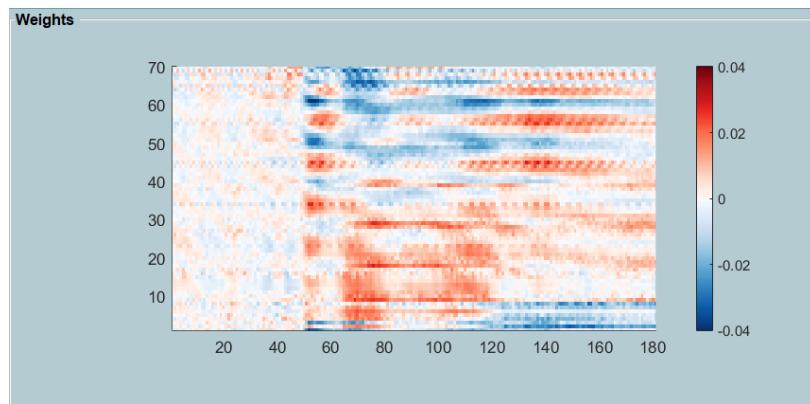


Figure 7.4: ‘Display weights’ for 2D MEEG or .mat data.

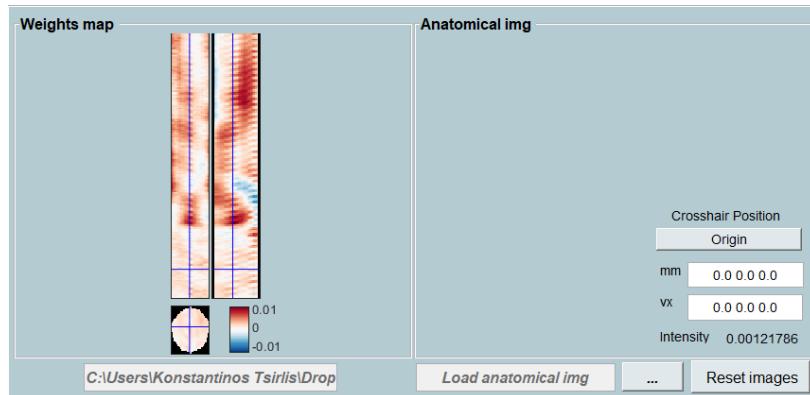


Figure 7.5: ‘Display weights’ for 2D NIfTI images derived from 2D MEEG data (using SPM).

- **For 3D data:** Displaying weights in 3D data in PRoNTo v3.0 is only available using NIfTI brain images, as has been from versions 2.X. Examples in figure 7.2.

7.2.2 Weights map

All the different types of weight maps are displayed with a colorbar. 2D and 3D NIfTI images are also displayed with a cross-hair. The colorbar indicates the relative importance of the voxel/feature to the decision function. This value is also indicated in the ‘intensity’ field of the ‘Anatomical img’ panel. Note that all voxels/features with weight different from zero contribute to the decision function, since the analysis is multivariate. **Contrary to common practice in Statistical Parametric Mapping, which is a mass-univariate approach, it is not recommended to threshold the weight map and report only on the peaks of the decision function’s weight map, unless they have perfectly null contribution (as might happen with sparse models such as L1-MKL modelling).**

7.2.3 Anatomical image

By clicking on the [...] button next to the ‘Load anatomical img’ field, a dialogue opens that allows you to select an anatomical images ‘.img’ file that was co-registered with the data images. In this panel, the cross-hair position is displayed in voxels and in mm. It can also be reset to the origin of the image. For each position, the corresponding voxel weight is displayed in the ‘intensity’ field.

The main reason behind displaying the anatomical image next to the weights map was to help users identify which parts of the brain correspond to which weights in the weights map, therefore it only makes sense for NIfTI brain data (e.g. MRI, fMRI). This correspondence is only used as an approximation and you should not rely on it.

Note: One should only use the co-registered anatomical images as a crude heuristic to help them navigate through the weights maps only in the case of MRI (3D) NIfTI images. There is no real correspondence between the anatomical image and other types of data (for example 2D MEEG data).

7.2.4 Additional plots

Additional information will be displayed in two main cases:

- **Multiple Kernel Learning modelling:** MKL modelling based on modalities, regions or channels, will provide weights at two levels: the kernel level and the voxel/feature level. The kernel contributions, which sum to 1, can then be ranked in descending order.
- **Summarizing weights per region:** Starting from PRoNTo v2.0, weights can be summarized within regions of interest as defined by an atlas (user-specified). For each region, a normalized contribution can be defined, and those contributions can then be ranked in descending order. If the atlas was not selected as second-level mask at the feature set step, users might see one extra region in the ROI label column in a table labelled as ‘other’ in case there are voxels/features that do not overlap with the selected atlas (see Chapter 6).

In both cases, the kernel’s or region’s contributions will be displayed in a table as well as in a bar plot, for each fold and for their average (according to the selected fold in the ‘Display’ panel). An example is displayed in Figure 7.6, for weight summarization per region after MKL modelling.

In the case where kernels were built both at the modality and at the region level (i.e. multiple modalities with each multiple regions as defined by an atlas), two tables will be displayed (one for regions, one for modalities). The table for modalities will sum the contributions of each region within that modality.

Note: Everything mentioned above also applies to all the other types of data formats.

Sorted table of region/modality contributions

The displayed table comprises one row per region and 5 columns (for an example on ROIs, as displayed in Figure 7.6):

- **Index of the ROI:** The first column displays the ranking of the region of interest in the selected fold, according to its contribution.

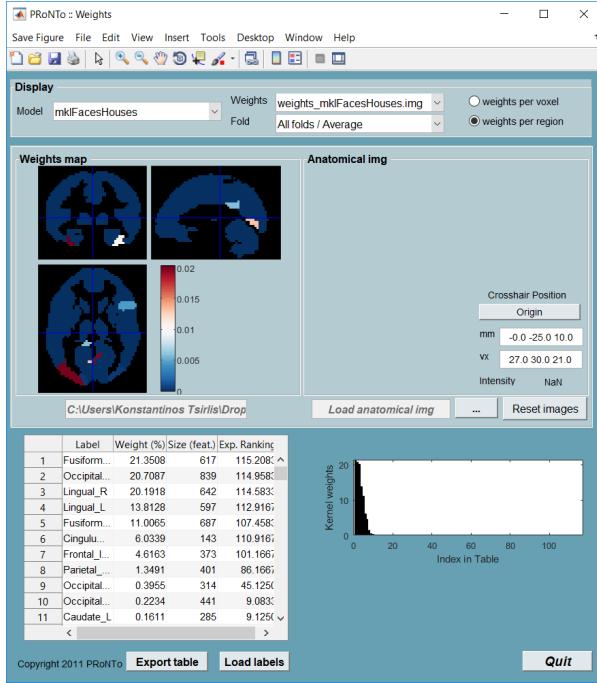


Figure 7.6: Displaying ROI contributions for a two-class MKL model.

- **ROI Label:** When using the atlas provided in *your PRoNTo folder/atlas*, the labels of each region will be loaded automatically from a .mat, stored alongside the atlas. If using another atlas, the labels can be loaded through the ‘Load Labels’ button. In this case, the user should select a .mat comprising a cell array of size (number of regions,1), with the label for each region in the corresponding cell (in characters). The cell array should be saved under the variable name ‘ROI_ names’. Otherwise, generic names will be used (e.g. ROI_ 1).
- **ROI weight:** The (normalized) contribution of each region is displayed in the third column (in %). The rows of the table are sorted in descending order according to this value.
- **ROI size:** This column displays the size of the ROI in voxels/features. This gives indications on the overlap between the atlas and the data.
- **Expected Ranking:** This measure reflects how stable the ranking of the region is across folds. It is computed from the ranking in each fold (see [23] for details), and is therefore the same, whether the user is displaying fold 1, or the average of all folds. If the Expected Ranking (ER) is close to the ranking in the selected fold, then it reflects that this region has a similar ranking across folds. On the contrary, if the ER is quite different from the ranking shown for the selected fold, this means that the ranking might be variable across folds. This variability can come from the fact that the region did not have the same contribution across the different folds. It might also happen that it is not selected at all in some folds (as can happen with L1-MKL since it will not select kernels with correlated information).

When selecting a specific region label in the table, the weight map will only display colored voxel or region weights (according to which plot was selected) for this region, the rest of the image being in grey scale. This allows e.g. to look closely at the voxel weights within a region that highly contributes to the selected model and fold (Figure 7.7). Finally, the table can be exported as a text file using the ‘Export Table’ button.

Bar plot of kernel

The bar plot displays the third column of the table, i.e. the contribution of each kernel to the decision function. The x-axis represents the index of the ROI/channel/modality in the table (i.e. first column of the table), in the selected fold, while the y-axis displays the contribution of each ROI/channel/modality. The bar graph provides insights on how sparse or dense the kernel contributions are for the model.

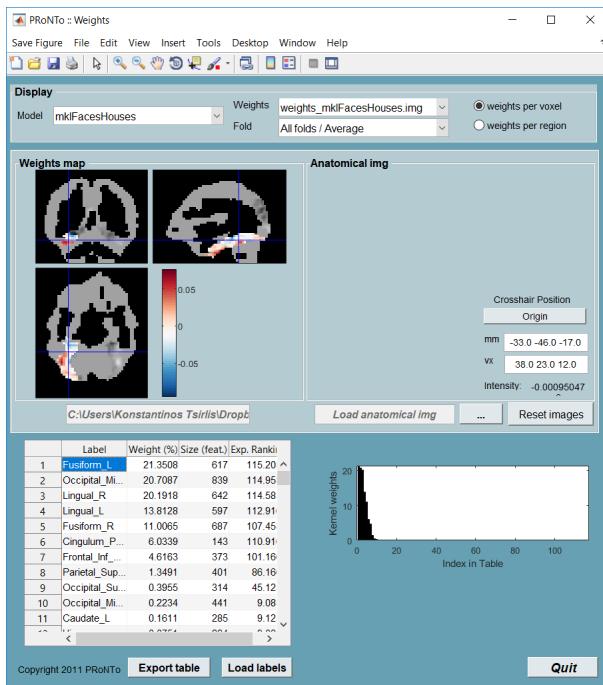


Figure 7.7: Displaying fusiform weights for binary MKL model.

Chapter 8

List of input files

In PRoNTo, different files need to be input by the user depending on the type of data that will be analyzed. These typically need to include a specific variable or satisfy certain conditions. Here is a summary of input files and their requirements:

Data & Design

- **.mat inputs:** For .mat modalities, PRoNTo expects one file per sample, as for nifti. This means that, no matter the dimensionality of the data, one .mat file needs to be saved per subject for ‘Select by Scans’ or per trial ‘Select by Subject’. The first variable of this file will then be extracted as the data for that sample. The data array can be of any dimensionality but needs to be consistent across samples and should not contain missing values.

A script is provided (in Appendix and in PRoNTo/utils) to convert a matrix that contains all the data for the different samples into separate .mat files. The script reads each row of the matrix and saves it as a .mat file in a separate subdirectory.

- **Regression target file - across subjects:** A regression target file needs to have 2 variables:

- ‘names’: a cell array with the names of the regression targets, the first name corresponding to the first column of the matrix, and so on. Example: ‘age’,‘score’.
- ‘rt_subj’: a matrix of size # of subject x # of regression targets. For 3 subjects and the 2 regression targets mentioned above, we would have:

```
>> rt_subj = [25, 6; 26, 8; 30, 7]
```

```
rt_subj =  
25 6  
26 8  
30 7
```

- **Covariate file - across subjects:** The covariate file at the subject level should contain a matrix R, of size # subjects x # covariates. It cannot contain missing or NaN values and cannot be a cell array. For categorical variables, the values need to be one-hot encoded if no ordinal ranking is expected. For the 3 subjects above, a one-hot encoding of gender would be (2 males and 1 female, using the representation [0 1] for female and [1 0] for male):

```
>> R = [1, 0; 1, 0; 0, 1]
```

```
R =  
1 0  
1 0  
0 1
```

The same number of covariates should be provided across subjects.

- **Conditions file - within subject:** To specify the design of a subject using a .mat file, 5 variables can be provided (this option is normally used with data types that have temporal information with an experimental design happening during the data acquisition, e.g. fMRI, MEEG):
 - names: cell array of conditions names, e.g. ‘A’;‘B’
 - onsets: cell array of vectors on onsets for each condition, e.g. [1,4,8],[2,5,7]
 - durations: cell array of vectors or single values for each condition, e.g. 1,1. In the case of a single value, this value will be repeated for all trials.
 - (optional) rt_trial: cell array of vectors containing one target per trial (including bad trials for MEEG), e.g. [1.2, 1.3, 1.1],[2.1, 2.5, 2.3].
 - (optional) R: cell array of matrices or vectors containing covariates per trial (including bad trials for MEEG). Each matrix should have the size # trials in condition x # covariates. The same number of covariates should be provided across conditions and subjects. E.g. [1,0;0,1;1,0],[0,1;1,0;1,0].

Feature set

- **Atlas for .mat:** When loading an atlas for a .mat modality, PRoNTo will extract the first variable of that file. This first variable (which can have any name) should contain one array. This array should have the exact same dimensions as the dimensions of the data from the first (and other) subjects. It should include values between 1 and the number of ROIs. 0 and NaN values will be excluded from the feature set (as the atlas is also forced as 2nd level mask).

To include labels of ROIs with the atlas, a separate file called ‘Labels_atlasname.mat’ needs to be found along the atlas ‘atlasname.mat’. This file should contain the variable ‘ROI_names’, a cell array of ROI labels. The structure of this file is the same as for nifti ROI labels. Alternatively, ROI labels can be loaded at the ‘Display weights’ step.

A script that builds an atlas and its labels automatically from a list of networks for connectivity matrices is provided in appendix and along the PRoNTo distribution (in /utils). If we assume that 2D connectivity matrices are provided as input, estimated from a certain number of time series (e.g. 200). Among those time series, some are thought to pertain to a same ‘network’ (e.g. ‘Default Mode’ or ‘Visual’). The script will build a ROI for each ‘network’ within and between interaction. For example, ‘Visual-Visual’ will be ROI1, ‘Visual-Default Mode’ will be ROI2 and ‘Default Mode-Default Mode’ will be ROI3.

- **Channel selection file for MEEG in batch:** The ‘channel selection’ module in the batch comes from SPM directly. They accept a file containing a cell array called ‘label’, with each cell being a channel name to include.

Model

- **Custom cross-validation:** Custom cross-validation allows to load a matrix. The name of the variable in the .mat file should be ‘CV’ and the matrix should be of size #samples x #folds. The values in the matrix need to be either 0 (sample not selected in fold), 1 (sample used for training) or 2 (sample used for testing).

Display weights

- **Labels for ROIs:** For nifti and .mat, ROI labels can be loaded when displaying the weights (MEEG comes with its own labelling). The loaded .mat file should contain a variable ‘ROI_names’, a cell array of ROI labels. The number of labels has to match the number of regions as defined in the atlas.

Part II

Batch interfaces

Chapter 9

Data & Design

Specify the data and design for each group (minimum one group).

9.1 Directory

Select a directory where the PRT.mat file containing the specified design and data matrix will be written.

9.2 Groups

Add data and design for one group. Click 'new' or 'repeat' to add another group.

9.2.1 Group

Specify data and design for the group.

Name

Name of the group. Example: 'Controls'.

Select by

Depending on the type of data at hand, you may have many samples per subject, such as a fMRI time series, or you may have many subjects with only one sample per subject , such as PET images. If you have many samples per subject select the option 'subjects'. If you have one sample for many subjects select the option 'samples'.

Subjects Add subjects/samples.

Subject Add new modality for this subject.

Modality Add new modality.

NAME Name of modality. Example: 'BOLD'. The names should be consistent accross subjects/groups and the same names specified in the masks.

DATA FORMAT Data format for this modality. The different input files should be in either nifti, SPM MEEG object or .mat format

INTERSCAN INTERVAL Specify interscan interval (TR). The units should be seconds.

FILES Select files for this modality. They must all have the same image dimensions, orientation, voxel size etc. Only one file allowed for MEEG input format.

DATA & DESIGN Specify data and design.

Load SPM.mat (for nifti inputs only) Load design from SPM.mat (if you have previously specified the experimental design with SPM). This option is available for nifti inputs only. Not available for MEEG or .mat input formats.

Specify design Specify design: samples (data), onsets and durations.

Units for design The onsets of events or blocks can be specified in either scans or seconds.

Conditions Specify conditions. This option is not available for MEEG. You are allowed to combine both event- and epoch-related responses in the same model and/or regressor. Any number of condition (event or epoch) types can be specified. Epoch and event-related responses are modeled in exactly the same way by specifying their onsets [in terms of onset times] and their durations. Events are specified with a duration of 0. If you enter a single number for the durations it will be assumed that all trials conform to this duration. For factorial designs, one can later associate these experimental conditions with the appropriate levels of experimental factors.

Condition Specify condition: name, onsets and duration.

Name Name of condition (alphanumeric strings only).

Onsets Specify a vector of onset times for this condition type.

Durations Specify the event durations. Epoch and event-related responses are modeled in exactly the same way but by specifying their different durations. Events are specified with a duration of 0. If you enter a single number for the durations it will be assumed that all trials conform to this duration. If you have multiple different durations, then the number must match the number of onset times.

Covariates Select a .mat file containing your covariates (i.e. any other data/information you would like to include in your design). This file should contain a variable 'R' with a matrix of covariates. Its size should be #samples x #covariates.

Regression targets (per trial) Enter one regression target per trial onset. or enter the name of a variable. This variable should be a vector [Ntrials x 1], where Ntrials is the number of events for the selected condition.

Multiple conditions Select the *.mat file containing details of your multiple experimental conditions.

If you have multiple conditions then entering the details a condition at a time is very inefficient. This option can be used to load all the required information in one go. You will first need to create a *.mat file containing the relevant information.

This *.mat file must include the following cell arrays (each 1 x n): names, onsets and durations. eg. names=cell(1,5), onsets=cell(1,5), durations=cell(1,5), then names2='SSent-DSpeak', onsets2=[3 5 19 222], durations2=[0 0 0 0], contain the required details of the second condition. These cell arrays may be made available by your stimulus delivery program, eg. COGENT. The duration vectors can contain a single entry if the durations are identical for all events.

Time and Parametric effects can also be included. For time modulation include a cell array (1 x n) called tmod. It should have a have a single number in each cell. Unused cells may contain either a 0 or be left empty. The number specifies the order of time modulation from 0 = No Time Modulation to 6 = 6th Order Time Modulation. eg. tmod3 = 1, modulates the 3rd condition by a linear time effect.

For parametric modulation include a structure array, which is up to 1 x n in size, called pmod. n must be less than or equal to the number of cells in the names/onsets/durations cell arrays. The structure array pmod must have the fields: name, param and poly. Each of these fields is in turn a cell array to allow the inclusion of one or more parametric effects per column of the design. The field name must be a cell array containing strings. The field param is a cell array containing a vector of parameters. Remember each parameter must be the same length as its corresponding onsets vector. The field poly is a cell array (for consistency) with each cell containing a single number specifying the order of the polynomial expansion from 1 to 6.

Note that each condition is assigned its corresponding entry in the structure array (condition 1 parametric modulators are in pmod(1), condition 2 parametric modulators are in pmod(2), etc. Within a condition multiple parametric modulators are accessed via each fields cell arrays. So for condition 1, parametric modulator 1 would be defined in pmod(1).name1, pmod(1).param1, and pmod(1).poly1. A second parametric modulator for condition 1 would be defined as pmod(1).name2, pmod(1).param2 and pmod(1).poly2. If there was also a parametric modulator for condition 2, then remember the first modulator for that condition is in cell array 1: pmod(2).name1, pmod(2).param1, and pmod(2).poly1. If some, but not all conditions are parametrically modulated, then the non-modulated indices in the pmod structure can be left blank. For example, if conditions 1 and 3 but not condition 2 are modulated, then specify pmod(1) and pmod(3). Similarly, if conditions 1 and 2 are modulated but there are 3 conditions overall, it is only necessary for pmod to be a 1 x 2 structure array.

EXAMPLE:

Make an empty pmod structure:

```
pmod = struct('name','','param','','poly',);
```

Specify one parametric regressor for the first condition:

```

pmod(1).name1 = 'regressor1';
pmod(1).param1 = [1 2 4 5 6];
pmod(1).poly1 = 1;
Specify 2 parametric regressors for the second condition:
pmod(2).name1 = 'regressor2-1';
pmod(2).param1 = [1 3 5 7];
pmod(2).poly1 = 1;
pmod(2).name2 = 'regressor2-2';
pmod(2).param2 = [2 4 6 8 10];
pmod(2).poly2 = 1;

```

The parametric modulator should be mean corrected if appropriate. Unused structure entries should have all fields left empty.

No design Do not specify design. This option can be used for modalities (e.g. structural scans) that do not have an experimental design. Possibility to specify one regression target and/or covariate for the considered subject.

No conditions No conditions to specify for this subject.

Covariates Select a .mat file containing your covariates (i.e. any other data/information you would like to include in your design). This file should contain a variable 'R' with a matrix of covariates. Its size should be #samples x #covariates.

Regression targets (per trial) Enter one regression target per trial onset. or enter the name of a variable. This variable should be a vector [Ntrials x 1], where Ntrials is the number of events for the selected condition.

Events in MEEG file (for MEEG inputs only) Events already in MEEG file. This option should be used for MEEG object inputs. Possibility to add covariates or regression targets

Events in file Specify condition: name, onsets and duration.

Add regression targets/covariates Events already in MEEG file. This option should be used to add covariates or regression targets for MEEG object inputs.

No regression targets/covariates No regression targets/covariates for this subject/modality.

Conditions Specify conditions for which to add regression targets and/or covariates.

Condition Specify condition: name, regression targets and covariates.

Name Name of condition (alphanumeric strings only).

Covariates Select a .mat file containing your covariates (i.e. any other data/information you would like to include in your design). This file should contain a variable 'R' with a matrix of covariates. Its size should be #samples x #covariates.

Regression targets (per trial) Enter one regression target per trial onset. or enter the name of a variable. This variable should be a vector [Ntrials x 1], where Ntrials is the number of events for the selected condition.

Samples Depending on the type of data at hand, you may have many samples per subject, such as a fMRI time series, or you may have many subjects with only one or a small number of samples per subject, such as PET images. Select this option if you have many subjects per modality to spatially normalise, but there is only one sample for each subject. This is a faster option with less information to specify than the 'select by subjects' option. Both options create the same 'PRT.mat' but 'select by samples' is optimised for modalities with no design.

Modality Specify modality, such as name and data.

Name Name of modality. Example: 'BOLD'. The names should be consistent across subjects/groups and the same names specified in the masks.

Data format Data format for this modality. The different input files should be in either nifti, SPM MEEG object or .mat format

Files Select files for this modality. They must all have the same image dimensions, orientation, voxel size etc. Only one file allowed for MEEG and for .mat input formats.

Regression targets (subject) Add regression targets per subject. Only for nifti or .mat formats.

No TARGETS No regression targets.

FROM FILE Select .mat file containing regression targets. It should contain the values in a matrix rt_subj of size number of subjects times number of regression targets. Additionally, if the variable names is a cell of size number of targets times 1 and contains strings, these strings will be associated with the targets as their names and referred as such later on.

SPECIFY Specify each regression target.

Target Subject regression targets. Specify name and values for each.

Name Name of regression target. Example: 'Age'. The names should be consistent across subjects/groups.

Values Enter one regression target per subject. This vector should have the following dimensions: [Nsubjects x 1], where Nsubjects is the number of subjects in group.

Covariates Select a .mat file containing your covariates (i.e. any other data/information you would like to include in your design). This file should contain a variable 'R' with a matrix of covariates. One covariate per image is expected.

9.3 Masks

Select first-level (pre-processing) mask for each modality format. The name of the modalities should be the same as the ones entered for subjects/scans.

9.3.1 Modality

Specify name of modality and file format for each mask. The name should be consistent with the names chosen for the modalities (subjects/scans).

Name

Name of modality. Example: 'BOLD'. The names should be consistent across subjects/groups and the same names specified in the masks.

Data format

Data format input. Either nifti, MEEG or .mat

Nifti Specify name of mask file for nifti modality.

File Select one first-level mask (image) for each modality. This mask is used to optimise the prepare data step. In 'specify model' there is an option to enter a second-level mask, which might be used to select only a few areas of the brain for subsequent analyses.

HRF overlap If using fMRI data please specify the width of the hemodynamic response function (HRF). This will be used to calculate the overlap between events. Leave as 0 for other modalities (other than fMRI).

HRF delay If using fMRI data please specify the delay of the hemodynamic response function (HRF). This will be used to calculate the overlap between events. Leave as 0 for other modalities (other than fMRI).

MEEG No mask for MEEG object files

.mat No mask for .mat files

9.4 Review

Choose 'Yes' if you would like to review your data and design in a separate window. This window needs to be closed before proceeding further.

Chapter 10

Feature set/Kernel

Compute feature set according to the design specified

10.1 Load PRT.mat

Select data/design structure file (PRT.mat).

10.2 Feature/kernel name

Target name for kernel matrix. This should containonly alphanumerical characters or underscores (.).

10.3 Data format

Data format for selected modalities. The different input files should be in either nifti, SPM MEEG object or .mat format

10.3.1 Nifti

Add modalities in nifti format

Modality

Specify modality, such as name and data.

Modality name Name of modality. Example: 'BOLD'. Must match design specification

Samples / Conditions Which task conditions do you want to include in the kernel matrix? Select conditions: select specific conditions from the design. All conditions: include all conditions extracted from the design. All samples: include all samples for each subject. This may be used for modalities with only one sample per subject (e.g. PET), if you want to include all samples from an fMRI timeseries (assumes you have not already detrended the timeseries and extracted task components)

All samples No design specified. This option can be used for modalities (e.g. structural) that do not have an experimental design or for an fMRI designwhere you want to include all samples in the timeseries

All Conditions Include all conditions in this kernel matrix

Voxels to include Specify which voxels from the current modality you would like to include

All voxels Use all voxels in the design mask for this modality

Specify mask file Select a mask for the selected modality.

Detrend Type of temporal detrending to apply

None Do not detrend the data

Polynomial detrend Perform a voxel-wise polynomial detrend on the data (1 is linear detrend)

Order Enter the order for polynomial detrend (1 is linear detrend)

Discrete cosine transform Use a discrete cosine basis set to detrend the data.

Cutoff of high-pass filter (second) The default high-pass filter cutoff is 128 seconds (same as SPM)

Scale input scans Do you want to scale the input scans to have a fixed mean (i.e. grand mean scaling)?

No scaling Do not scale the input scans

Specify from *.mat Specify a mat file containing the scaling parameters for each modality.

Use atlas to build ROI specific kernels Select an atlas file to build one kernel per ROI. The AAL atlas (named 'aal_79x91x69.img') is available in the 'atlas' subdirectory of PRoNTo

10.3.2 MEEG

Add modalities in SPM MEEG format

Modality

Specify modality, such as name and data.

Modality name Name of modality. Example: 'BOLD'. Must match design specification

Channels

Channel selection Channel selection.

All

Select channels by type Select channels by type.

Custom channel Enter a single channel name.

Regular expression Enter a regular expression for matching multiple channel labels.

Channel file

Average Average across selected.

Multiple kernels Build one kernel per selected feature in this dimension.

This will result in e.g. one kernel per channel, per time point, or per frequency bin. Choosing multiple kernels on one dimension does not exclude the computation of multiple kernels on other dimensions.

Time points

Time window Start and stop of the time window (ms).

Average Average across selected.

Multiple kernels Build one kernel per selected dimension.

No Do not build multiple kernels

One kernel per time point Build one kernel per time point

One kernel per time window Build one kernel per time window

TIME WINDOW (ms) Length of time window to consider (in ms). E.g. 10

Frequencies

Frequency window Start and stop of the frequency window (Hz).

Average Average across selected.

Multiple kernels Build one kernel per selected feature in this dimension.

This will result in e.g. one kernel per channel, per time point, or per frequency bin. Choosing multiple kernels on one dimension does not exclude the computation of multiple kernels on other dimensions.

10.3.3 .mat

Add modalities in .mat format

Modality

Specify modality, such as name and data.

Modality name Name of modality. Example: 'BOLD'. Must match design specification

Samples / Conditions Which task conditions do you want to include in the kernel matrix? Select conditions: select specific conditions from the design. All conditions: include all conditions extracted from the design. All samples: include all samples for each subject. This may be used for modalities with only one sample per subject (e.g. PET), if you want to include all samples from an fMRI timeseries (assumes you have not already detrended the timeseries and extracted task components)

All samples No design specified. This option can be used for modalities (e.g. structural) that do not have an experimental design or for an fMRI design where you want to include all samples in the timeseries

All Conditions Include all conditions in this kernel matrix

Scale input scans Do you want to scale the input scans to have a fixed mean (i.e. grand mean scaling)?

No scaling Do not scale the input scans

Specify from *.mat Specify a mat file containing the scaling parameters for each modality.

Features to include Specify which features from the current modality you would like to include

All features Use all features in the matrix for this modality

Specify mask file Select a mask for the selected modality.

Use atlas to build ROI specific kernels Select an atlas file to build one kernel per ROI. The atlas should have the same dimensions as the input .mat data.

Chapter 11

Model: Specify new

Construct model according to design specified

11.1 Load PRT.mat

Select data/design structure file (PRT.mat).

11.2 Model name

Name for model

11.3 Feature sets

Feature set(s) to include in this model.

11.3.1 Feature set name

Add one feature set to this model. Click 'new' or 'repeat' to add another feature set.

Name

Enter the name of a feature set to include in this model. This can be kernel or a feature matrix.

11.4 Model Type

Select which kind of predictive model is to be used.

11.4.1 Classification

Specify classes and machine for classification.

Classes

Specify which elements belong to this class. Click 'new' or 'repeat' to add another class.

Class Specify which groups, modalities, subjects and conditions should be included in this class

Name Name for this class, e.g. 'controls'

Groups Add one group to this class. Click 'new' or 'repeat' to add another group.

Group Specify data and design for the group.

GROUP NAME Name of the group to include. Must exist in PRT.mat

SUBJECTS Subject numbers to be included in this class. Note that individual numbers (e.g. 1), or a range of numbers (e.g. 3:5) can be entered

CONDITIONS / SAMPLES Which task conditions do you want to include? Select conditions: select specific conditions from the design. All conditions: include all conditions extracted from the design. All samples: include all samples for each subject. This may be used for modalities with only one sample per subject (e.g. PET), if you want to include all samples from an fMRI timeseries (assumes you have not already detrended the timeseries and extracted task components)Target: to specify which regression target to use. This may be used when multiple regression targets were specified while having only one sample per subject.

Specify Conditions Specify the name of conditions or of the target to be included. Multiple conditions can be combined.

Condition Specify condition to use.

Name Name of condition to include.

All Conditions Include all conditions in this model

All samples No design specified. This option can be used for modalities (e.g. structural) that do not have an experimental design or for an fMRI design where you want to include all samples in the timeseries

Target Specify target to use.

Name Name of target to include.

Subsample examples based on class definition

Whether to subsample the example, or not. If Yes, the code will match the number of examples in each class as close as possible. This operation takes the duration of the examples into account (i.e. will not cut an event).

Machine Type

Select whether a kernel or non-kernel method is to be used.

Kernel machine Choose a kernel prediction machine for this model

SVM Classification Binary support vector machine.

SVM string argument String argument for LIBSVM interfacing.

Machine optimization and parameters Choose whether to optimize machine or not

No OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10.[-2:5] or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Gaussian Process Classification Gaussian Process Classification

String arguments String arguments for GPML machine binary classification machine.

Multiclass GPC Multiclass GPC

String arguments String arguments for GPML multiclass classification machine.

L1 Multi-Kernel Learning Multi-Kernel Learning. Choose only if multiple kernels were built during the feature set construction (either multiple modalities or ROIs).

It is strongly advised to "normalize" the kernels (in "operations").

Machine optimization and parameters Choose whether to optimize machine or not

No OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[}-2:5]$ or 1:100:1000 or 0.01 0.1
1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Custom machine Choose another prediction machine

Function Choose a function that will perform prediction.

Custom machine string argument String argument for custom machine.

Custom machine optimization and parameters Choose whether to optimize machine or not

No OPTIMIZATION Enter parameter fixed value if needed.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Hyper-parameter range for prediction machine.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Non-kernel machine Choose a non-kernel prediction machine for this model

Binary L2-SVM Non-kernel L2-regularized L2-Loss support vector machine, can be used for multiclass problem.

String arguments String arguments for LIBLINEAR interfacing.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[}-2:5]$ or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Multiclass SVM Multiclass support vector classification by Crammer and Singer, can also be used for binary classification.

String arguments String arguments for LIBLINEAR interface.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[}-2:5]$ or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Binary L1-SVM Non-kernel L1-regularized L2-Loss support vector machine.

String arguments String arguments for LIBLINEAR interface.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[}-2:5]$ or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

L2-Logistic Regression Non-kernel L2-regularized Logistic Regression from LIBLINEAR.

String arguments String arguments for LIBLINEAR interface.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[}-2:5]$ or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

L1-Logistic Regression Non-kernel L1-regularized Logistic Regression from LIBLINEAR.

String arguments String arguments for LIBLINEAR interface.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10.̂[-2:5] or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Custom machine Choose another prediction machine

Function Choose a function that will perform prediction.

Custom machine string argument String argument for custom machine.

Custom machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Enter parameter fixed value if needed.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Hyper-parameter range for prediction machine.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

11.4.2 Regression

Add group data and machine for regression.

Groups

Add one group to this regression model. Click 'new' or 'repeat' to add another group.

Group Specify data and design for the group.

Group name Name of the group to include. Must exist in PRT.mat

Subjects Subject numbers to be included in this class. Note that individual numbers (e.g. 1), or a range of numbers (e.g. 3:5) can be entered

Conditions / Samples Which task conditions do you want to include? Select conditions: select specific conditions from the design. All conditions: include all conditions extracted from the design. All samples: include all samples for each subject. This may be used for modalities with only one sample per subject (e.g. PET), if you want to include all samples from an fMRI timeseries (assumes you have not already detrended the timeseries and extracted task components)Target: to specify which regression target to use. This may be used when multiple regression targets were specified while having only one sample per subject.

Specify Conditions Specify the name of conditions or of the target to be included. Multiple conditions can be combined.

CONDITION Specify condition to use.

Name Name of condition to include.

All Conditions Include all conditions in this model

All samples No design specified. This option can be used for modalities (e.g. structural) that do not have an experimental design or for an fMRI design where you want to include all samples in the timeseries

Target Specify target to use.

NAME Name of target to include.

Machine Type

Select whether a kernel or non-kernel method is to be used.

Kernel machine Choose a kernel prediction machine for this model

Kernel Ridge Regression Kernel Ridge Regression.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[-2:5]}$ or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

epsilon-SVR Kernel epsilon Support Vector Regression from LIBSVM.

String arguments String arguments for LIBSVM interface.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[-2:5]}$ or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Relevance Vector Regression Relevance Vector Regression. Tipping, Michael E.; Smola, Alex (2001). "Sparse Bayesian Learning and the Relevance Vector Machine". Journal of Machine Learning Research 1: 211-244.

Gaussian Process Regression Gaussian Process Regression

String arguments String arguments for GMPL machine prt_machine_gpr.

Multi-Kernel Regression Multi-Kernel Regression

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10.[-2:5] or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Custom machine Choose another prediction machine

Function Choose a function that will perform prediction.

Custom machine string argument String argument for custom machine.

Custom machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Enter parameter fixed value if needed.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Hyper-parameter range for prediction machine.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Non-kernel machine Choose a non-kernel prediction machine for this model

epsilon-SVR Non-kernel epsilon-Support Vector Regression from LIBLINEAR.

String arguments String arguments for LIBLINEAR interface.

Machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Getting default value.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Value(s) for hyper-parameter. Examples: 10. $\hat{[}-2:5]$ or 1:100:1000 or 0.01 0.1 1 10 100.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

Custom machine Choose another prediction machine

Function Choose a function that will perform prediction.

Custom machine string argument String argument for custom machine.

Custom machine optimization and parameters Choose whether to optimize machine or not

NO OPTIMIZATION Enter parameter fixed value if needed.

OPTIMIZE HYPER-PARAMETER Specify range of values and nested CV.

Regularization hyper-parameter Hyper-parameter range for prediction machine.

Cross-validation type for hyper-parameter optimization Choose the type of cross-validation to be used

Leave one subject out Leave a single subject out each cross-validation iteration

k-folds CV on subjects k-partitioning of subjects at each cross-validation iteration

k Number of folds/partitions for CV. To create a 50

Leave one subject per group out Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

k-folds CV on subjects per group K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k Number of folds/partitions for CV. To create a 50

Leave one block out Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

k-folds CV on blocks k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one block per class out Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

k-folds CV on block per class k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k Number of folds/partitions for CV. To create a 50

Leave one run/session out Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

11.5 Cross-validation type

Choose the type of cross-validation to be used

11.5.1 Leave one subject out

Leave a single subject out each cross-validation iteration

11.5.2 k-folds CV on subjects

k-partitioning of subjects at each cross-validation iteration

k

Number of folds/partitions for CV. To create a 50

11.5.3 Leave one subject per group out

Leave out a single subject from each group at a time. Appropriate for repeated measures or paired samples designs.

11.5.4 k-folds CV on subjects per group

K-partitioning of subjects from each group at a time. Appropriate for repeated measures or paired samples designs.

k

Number of folds/partitions for CV. To create a 50

11.5.5 Leave one block out

Leave out a single block or event from each subject each iteration. Appropriate for single subject designs.

11.5.6 k-folds CV on blocks

k-partitioning on blocks or events from each subject each iteration. Appropriate for single subject designs.

k

Number of folds/partitions for CV. To create a 50

11.5.7 Leave one block per class out

Leave out a single block or event from each class each iteration. Appropriate for single subject designs.

11.5.8 k-folds CV on block per class

k-partitioning on blocks or events from each class each iteration. Appropriate for single subject designs.

k

Number of folds/partitions for CV. To create a 50

11.5.9 Leave one run/session out

Leave out a single run (modality) from each subject each iteration. Appropriate for single subject designs with multiple runs/sessions.

11.5.10 Custom

Load a cross-validation matrix comprising a CV variable

11.6 Include all scans

This option can be used to pass all the scans for each subject to the learning machine, regardless of whether they are directly involved in the classification or regression problem. For example, this can be used to estimate a GLM from the whole timeseries for each subject prior to prediction. This would allow the resulting regression coefficient images to be used as samples.

11.7 Data operations

Specify operations to apply

11.7.1 Mean centre features

Select an operation to apply.

11.7.2 Other Operations

Include other operations?

No operations

No design specified. This option can be used for modalities (e.g. structural scans) that do not have an experimental design or for an fMRI design where you want to include all scans in the timeseries

Select Operations

Add zero or more operations to be applied to the data before the prediction machine is called. These are executed within the cross-validation loop (i.e. they respect training/test independence) and will be executed in the order specified.

Operation Select an operation to apply.

Chapter 12

Model: Run

Trains and tests the predictive machine using the cross-validation structure specified by the model.

12.1 Load PRT.mat

Select PRT.mat (file containing data/design structure).

12.2 Model name

Name of a model. Must match your entry in the
'Specify model' batch module.

12.3 Do permutation test?

Perform a permutation test on accuracy, or not

12.3.1 No permutation test

Do not perform permutation test

12.3.2 Permutation test

Perform a permutation test.

Number of permutations

Enter the number of permutations to perform

Save permutations parameters

Set to Yes to save the parameters obtained from each permutation.

Copy permutations from model

Set to Yes to copy the permutations from another model. This option should be selected to correct for multiple comparisons. The 2 models should contain the exact same samples.

No Do not copy permutation from another model.

Copy from Yes, copy permutations from a previous model.

Mdel name Name of a model. Must match your entry in the 'Specify model' batch module.

Part III

Practical Tutorials

Chapter 13

Block design fMRI dataset

Contents

13.1	GUI analysis	115
13.1.1	Data & Design	116
13.1.2	Prepare feature set	119
13.1.3	Model: Specify new	120
13.1.4	Model: Specify from (optional step)	122
13.1.5	Model: Run	123
13.1.6	Display model (optional step)	123
13.1.7	Display results	124
13.1.8	Compute weights (optional step)	126
13.1.9	Display weights	126
13.2	Batch analysis	126
13.2.1	Data & Design	127
13.2.2	Feature set / Kernel	128
13.2.3	Model: Specify new	130
13.2.4	Model: Specify from (optional step)	132
13.2.5	Model: Run	132
13.2.6	Compute weights (optional step)	132

This chapter will describe the steps necessary to perform a classification using PRoNTo. The dataset¹ used in this chapter can be found in PRoNTo's website <http://www.mlnl.cs.ucl.ac.uk/pronto/prtdata.html> (data set 1) and the whole² dataset is available in <http://data.pymvpa.org/datasets/haxby2001/>.

This fMRI dataset originates from a study on face and object representation in human ventral temporal cortex [10]. In this study, the subject was shown a set of grey scale images of 8 categories (faces, houses, cats, chairs, bottles, scissors, shoes and scrambled pictures), with 12 runs/blocks. Each image was displayed for 500 ms and was followed by a 1500 ms rest interval. This experiment consisted of a block-design of 9 scans of each category followed by 6 scans of inter-stimulus interval. Images were acquired with a TR of 2.5 s. The full-brain fMRI data consisted of 1452 scans/volumes with 40 x 64 x 64 voxels. The dimensionality of each voxel was 3.5 x 3.75 x 3.75 mm.

For simplicity, in this example we will use PRoNTo to predict if the subject is viewing an image of a face or a house based on the fMRI scans. We will classify the whole brain images using Support Vector Machines, using a leave one block out cross-validation scheme.

13.1 GUI analysis

We will first analyse the data using PRoNTo's GUI and then repeat the analysis using the `matlabbatch` system.

¹Pre-processed (realigned and normalised) data from participant 1.

²Not pre-processed.

To start, create a new directory in which to save the results of the analysis, then start up MATLAB and type ‘prt’ or ‘pronto’ in the MATLAB prompt (considering that the PRoNTo and SPM folders have been previously added to the MATLAB path). This will open the main interface of PRoNTo (Figure 13.1).

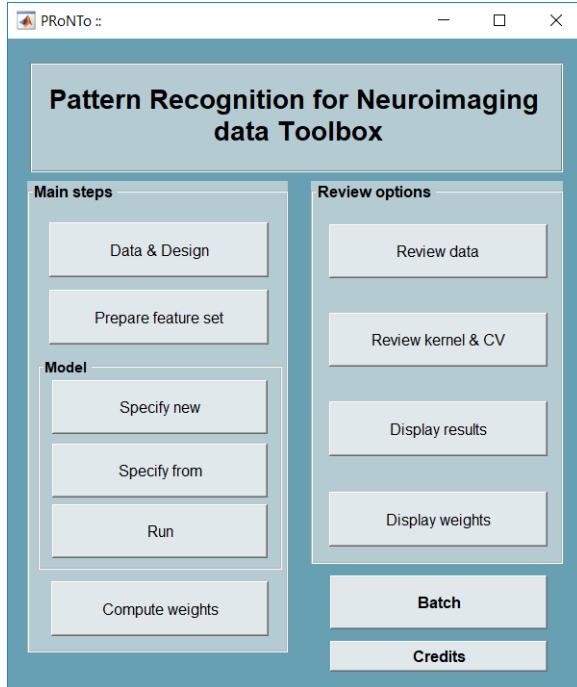


Figure 13.1: Main interface of PRoNTo.

13.1.1 Data & Design

- In PRoNTo’s main window, click on ‘Data & Design’ and a new window will open, ‘Data and design’ (Figure 13.2). Then, browse the directory in which to save the PRT structure (saved as ‘PRT.mat’).

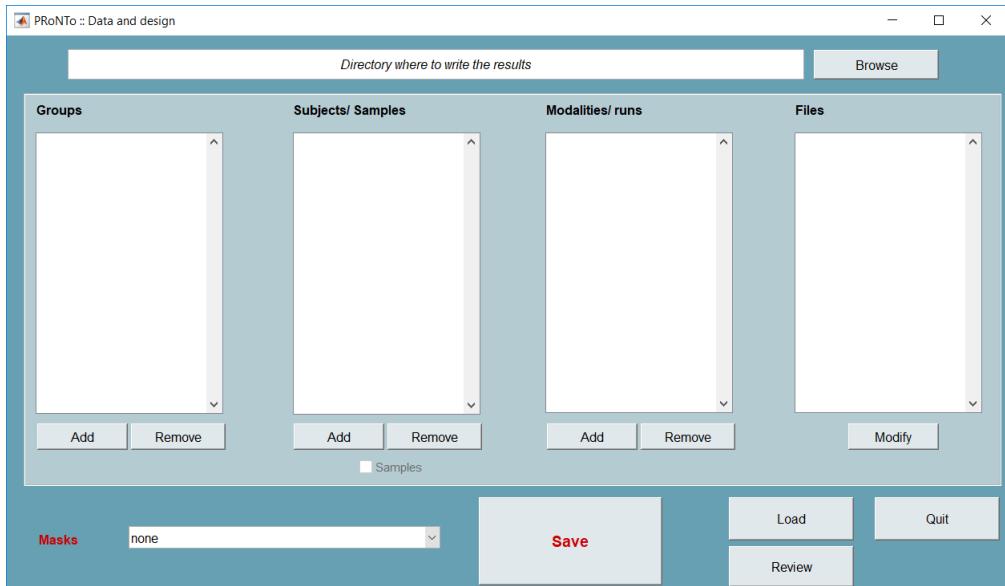


Figure 13.2: ‘Data and design’ GUI.

- In the panel ‘Groups’, click on ‘Add’ and provide a name to the group (we only have one group/subject), with no spaces or special characters, e.g. ‘G1’.

- Add a subject in the ‘Subject/Samples’ option, e.g. ‘S1’, and leave the ‘Samples’ tick box below the panel unchecked. See Chapter 2 of the manual for more information on this option.
- In the ‘Modalities’ panel, click on ‘Add’ and provide a name to the modality, e.g. ‘fMRI’. The ‘Specify modality’ GUI allows one to specify the format of the Data and the design of the experiment. In the ‘Data format’, choose ‘nifti’ (Figure 13.3). In the ‘Design’ field, choose the option ‘Load SPM.mat’ (Figure 13.4). This file is available with the Haxby dataset on PRoNTo’s website³ inside the folder Haxby_dataset/design/. Finally, in this case leave ‘Regression targets’ as it is, in the ‘No targets’ option.

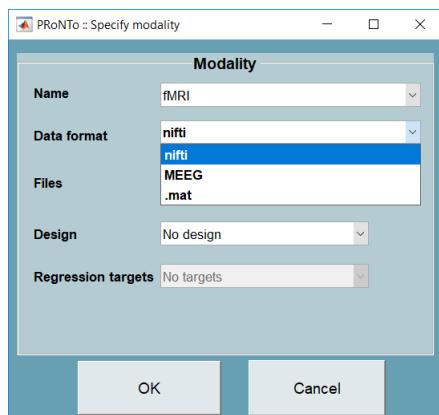


Figure 13.3: ‘Specify modality’ GUI. First we specify the format of our data (here nifti).

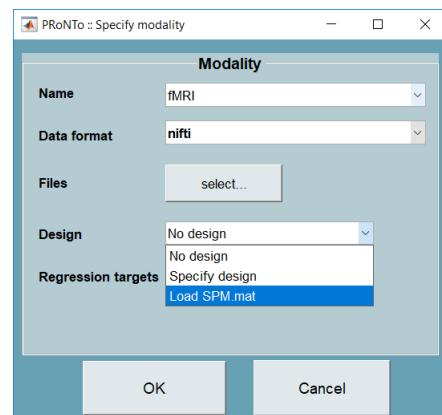


Figure 13.4: ‘Specify modality’ GUI. Here we load a specified design from an ‘SPM.mat’ file.

- In case there is no ‘SPM.mat’ file available to use, create a new design by selecting the option ‘Specify design’. Choose how many conditions you have, which in this case are 8 conditions (corresponding to the 8 categories of images). This will open another window that allows the user to write the names, onsets and durations of each condition (Figure 13.5). If the duration of each event is different, you must specify the duration of all events as shown in the figure above. If however the duration is the same for all events, specifying one value per class will suffice (Figure 13.6). The unit in which the onsets/durations are read in this case is ‘scans’ and the interscan interval (TR) is 2.5 seconds. The design information (names, onsets and durations) can be found inside the ‘Haxby_design.pdf’ file in the Haxby dataset folder.

Enter number of conditions			
Conditions			
	Name	Onsets	Duration
1	Faces	21 127 334 426 533 6...	9 9 9 9 9 9 9 9 9 9 9 9
2	Houses	63 213 348 384 490 6...	9 9 9 9 9 9 9 9 9 9 9 9
3	Cats	35 142 248 412 576 6...	9 9 9 9 9 9 9 9 9 9 9 9
4	Shoes	49 156 320 369 562 6...	9 9 9 9 9 9 9 9 9 9 9 9
5	Bottles	92 199 305 455 519 6...	9 9 9 9 9 9 9 9 9 9 9 9
6	Chairs	106 170 291 398 547 ...	9 9 9 9 9 9 9 9 9 9 9 9
7	Scissors	6 184 277 469 505 71...	9 9 9 9 9 9 9 9 9 9 9 9
8	Scampix	78 227 263 441 590 6...	9 9 9 9 9 9 9 9 9 9 9 9

< >

Units of onsets/durations	Scans	OK
Interscan interval (TR in s)	2.5	Cancel

Figure 13.5: ‘Specify design’ GUI to enter the conditions, the units of design, TR and covariates.

Enter number of conditions			
Conditions			
	Name	Onsets	Duration
1	Faces	21 127 334 426 533 6...	9
2	Houses	63 213 348 384 490 6...	9
3	Cats	35 142 248 412 576 6...	9
4	Shoes	49 156 320 369 562 6...	9
5	Bottles	92 199 305 455 519 6...	9
6	Chairs	106 170 291 398 547 ...	9
7	Scissors	6 184 277 469 505 71...	9
8	Scampix	78 227 263 441 590 6...	9

< >

Units of onsets/durations	Scans	OK
Interscan interval (TR in s)	2.5	Cancel

Figure 13.6: If all events have the same duration, you can specify this with only one number.

³<http://www.mlnl.cs.ucl.ac.uk/pronto/prtdata.html>

- Finally, load all the image files available in the fMRI directory (Haxby_dataset/fMRI/). You can select all the files by using the right mouse button and clicking on the option ‘Select All’ (Figure 13.7). When all the images are selected, click on the ‘Done’ button.
- In the ‘Masks’ field, on the bottom left of the ‘Data and design’ window, select the ‘whole_brain’ mask for the modality specified (Figure 13.8). The mask is available in the masks directory inside the folder Haxby_dataset/masks/. Once you have specified the mask for each modality, you will notice that the color of the word ‘Masks’ changes from red to black. This tells you that you have defined a mask for each modality.

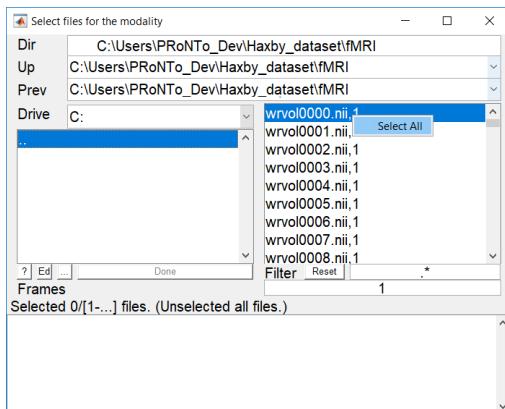


Figure 13.7: ‘Files’ field is used to select the scans/images for the selected subject.

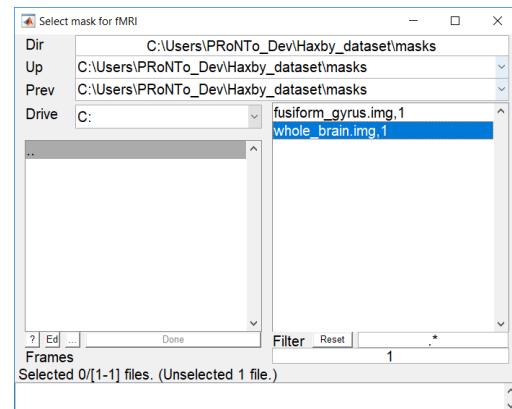


Figure 13.8: This window is called when one clicks ‘Masks’.

- Click on ‘Review’ button to check the data and the design inserted in this modality (Figure 13.9). For more information on what one can do with the Review option please see Chapter 2.

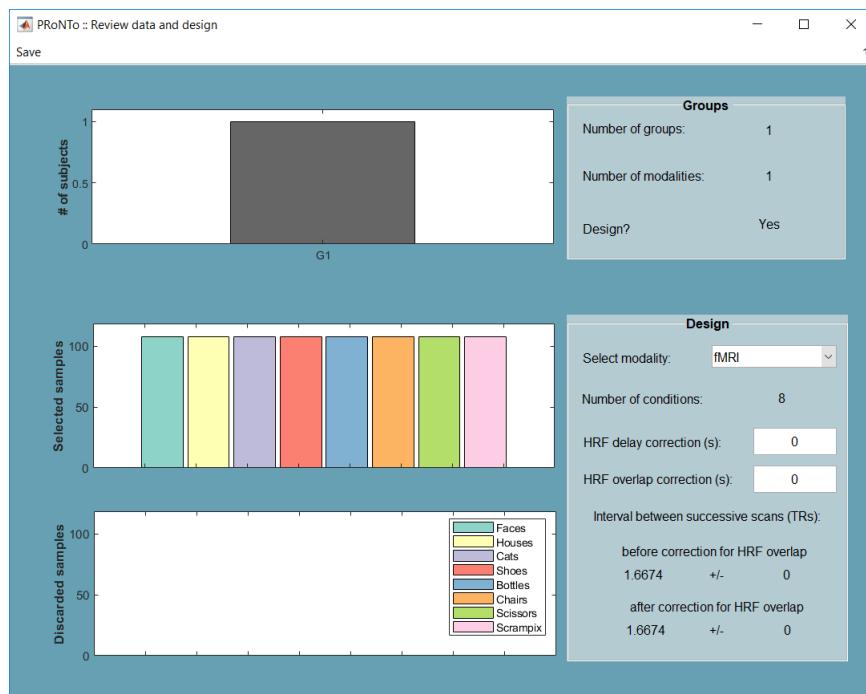


Figure 13.9: ‘Review’ GUI allows the user to check the data and design.

- Click on ‘Save’ button to create ‘PRT.mat’ file with the structure containing the information that has been previously specified. If there is no error, the ‘Save’ button changes color, from red to black; which

also tells you that everything has been properly saved right up until that moment. The ‘Data and design’ window after you click ‘Save’ should look similar to the Figure 13.10. If ‘Save’ turned black and no errors are shown in the MATLAB command window, leave the ‘Data and design’ window by clicking ‘Quit’.

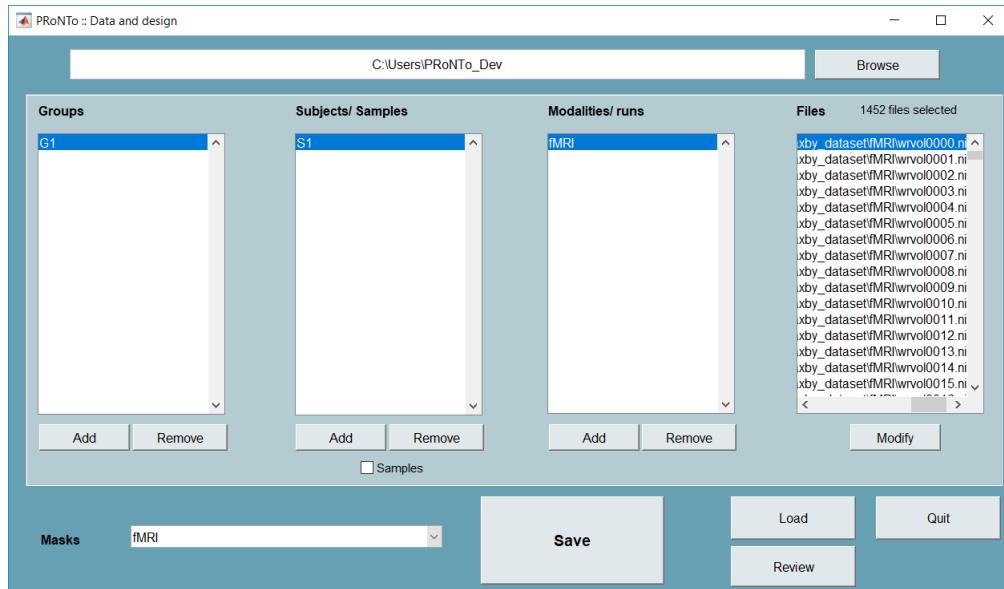


Figure 13.10: ‘Data and design’ GUI final configuration.

13.1.2 Prepare feature set

- Next we have to prepare the feature set, so click on ‘Prepare feature set’ in PRoNTo’s main window. A new window will open prompting you to select a PRT.mat file. Select the ‘PRT.mat’ file previously created in the ‘Data & Design’ step (Figure 13.11).

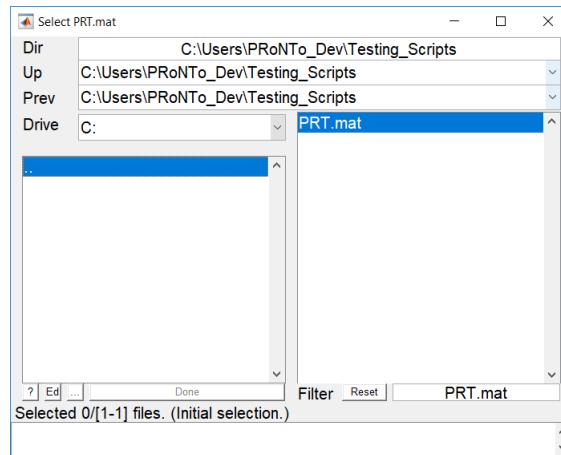


Figure 13.11: ‘Prepare feature set’ GUI.

- Once you click ‘Done’ another window will appear, ‘Specify modality to include’ (Figure 13.12). Here you set the specification of different parameters and options for each modality, which are:
 - ‘Modality’ field: select the modality previously specified in the ‘Data & Design’ step, ‘fMRI’.
 - ‘Conditions’ field: select ‘All scans’.
 - ‘Parameters box’: select the polynomial detrend with order 1 and the ‘No scaling’ option.
 - ‘Features box’: leave the additional mask field as it is and the ‘build one kernel per region’ tick box unchecked. Then, click on the ‘Done’ button.

- * As an optional step, in the ‘Additional mask for selected modality’ field, the user can specify a ‘second-level’ mask, which can be used to select regions of interest (ROIs) on which the classification can be performed. For instance, we can enter the ‘fusiform_gyrus’ mask available with this dataset.
- Once you specify the modality to include and click ‘Done’, yet another window will appear, ‘Prepare feature set’. Here you provide a name for the feature set, e.g. ‘HaxbyFeatures’ and finally you click on ‘Build kernel / data matrix’ to build the kernel. If everything was done correctly, a progress bar will pop up, marking the start of the procedure (Figure 13.13), which can take a few minutes.

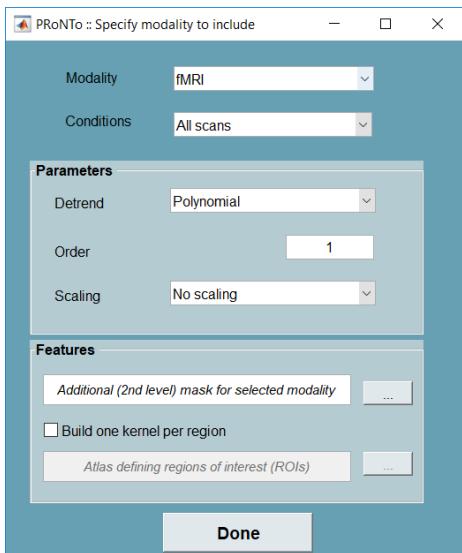


Figure 13.12: ‘Specify modality to include’ GUI.

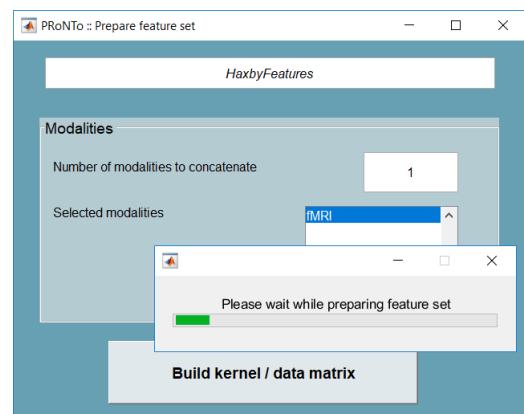


Figure 13.13: Preparing feature set.

13.1.3 Model: Specify new

Next we have to specify a model. In PRoNTo’s main window, click on ‘Specify new’ and a new window will open, ‘Specify model’ (Figure 13.14).

- Select the ‘PRT.mat’ file and provide a name to the model, e.g. ‘svmFaces Houses’.
- Select from the list one of the ‘Feature Set’ previously defined. In this case, there is only one ‘HaxbyFeatures’, but in general from v3.0 you are free to choose more than one feature set that can be combined using Multiple Kernel Learning.
- Leave the option ‘Use kernels’ tick box as it is, i.e. ‘Yes’.
- Select the ‘Classification’ model type and click on ‘Define classes’ button. A new window will open, ‘Specify classes’ (Figure 13.15), to define the number of classes and a name for each class. We will define 2 classes. First click ‘Class 1’ on the tab ‘Class’. For ‘Class 1’ select subject ‘S1’ and the condition ‘Faces’ and, similarly, for ‘Class 2’ select subject ‘S1’ and the condition ‘ Houses’. Leave the ‘Subsample according to smallest class’ as it is for now. Once you have appropriately specified everything, click ‘Done’.
- Select the ‘Binary support vector machine’ option, in the ‘Machine’ field.
- Leave the option ‘Optimize hyper-parameter’ tick box unchecked and ‘Cross-Validation Scheme’ (internal loop) as it is.
- Select the ‘Leave One Block Out’ cross-validation scheme (external loop).
- In the ‘Data operations’ box, select the ‘Sample averaging (within block)’ option, which corresponds to a temporal compression of the data within each block, and ‘Mean centre features using training data’ option. Then, the ‘Specify model’ window should look similar to Figure 13.16.

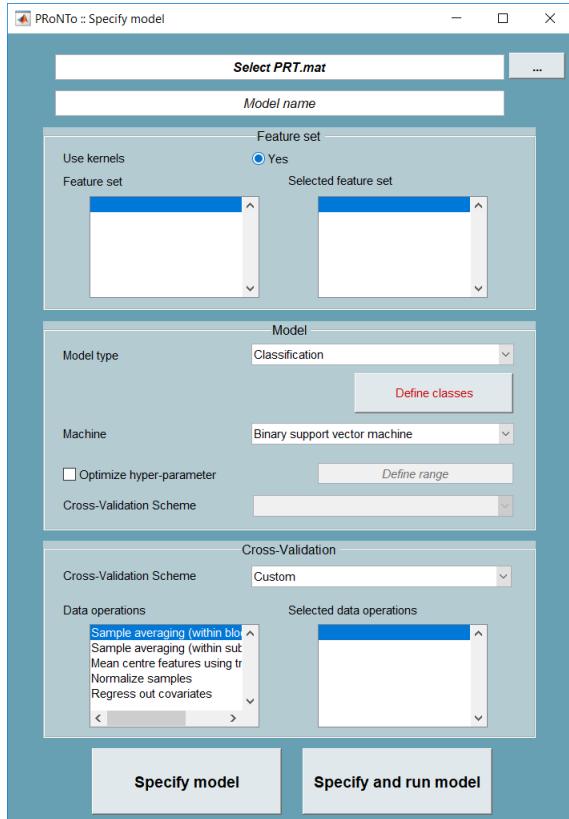


Figure 13.14: ‘Model: Specify new’ GUI.

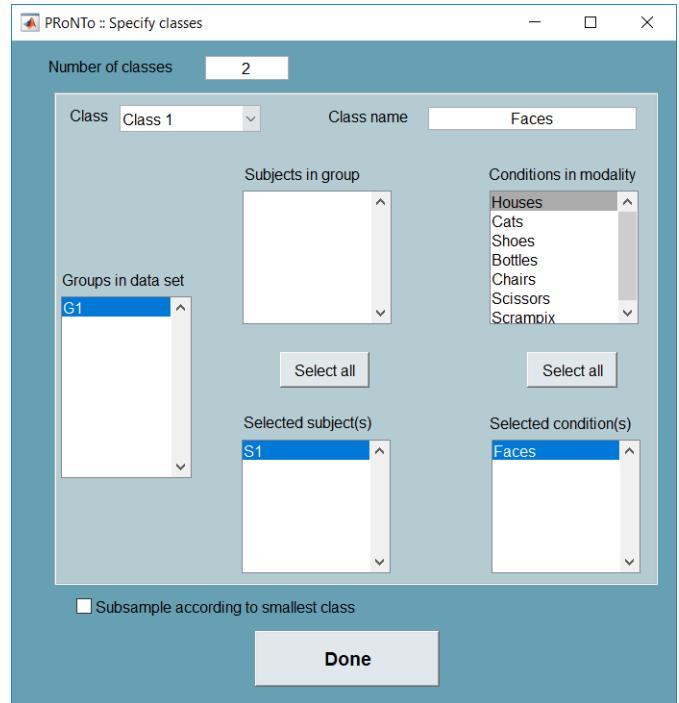


Figure 13.15: ‘Specify classes’ GUI.

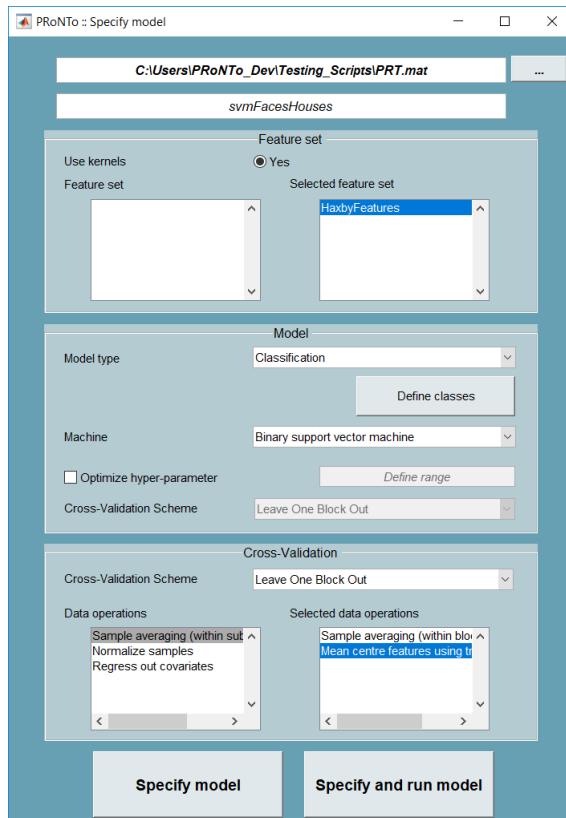


Figure 13.16: ‘Model: Specify new’ GUI final configuration.

- Click on ‘Specify and run model’ and the model will be immediately estimated, therefore there is no need to use the ‘Run model’ module in this case. However if you wish to run permutation tests you need to specify this in the ‘Run model’ module.
- If you do not wish to average the scans within each block (i.e. to do temporal compression), go back to the ‘Specify model’ window, give another name to the model and select the same options mentioned above, except in the data operations part. Here, choose only the ‘Mean centre features using training data’ option. Finish by clicking on the ‘Specify and run model’ button.

13.1.4 Model: Specify from (optional step)

- The ‘Specify from’ window is the same as the ‘Specify new’ window, except that some fields have been disabled to ensure comparability of the models, and there is also an extra popup menu that allows to select which model to copy from (Figure 13.17). For more information on what one can do with the ‘Specify from’ option please see Chapter 4.
- Select the ‘PRT.mat’ file we have been using so far and provide a new name to the model. It’s recommended giving a meaningful name to the model so that there is no confusion later on. We previously used SVMs, so now we can try a different machine for comparison, for example the Gaussian Process (GP) classifier, so we can name it ‘gpFaces Houses’. Alternatively, assuming you had previously created a second feature set called ‘HaxbyFeatures2’ or another more meaningful name according to your preference, you could now try also deselecting ‘HaxbyFeatures’ and selecting for example ‘HaxbyFeatures2’ as your selected feature set.
- Notice that Classes/Regression selection and outer CV options cannot be modified. Only the ‘Machine’ (with its hyper-parameter optimization), the feature sets and the ‘Data operations’ can be modified when using specifications from a previously defined model.
- After you have specified everything, the final configuration would look like the one in Figure 13.18. Now click ‘Specify and run model’ to create a second model comparable to ‘svmFaces Houses’ model.

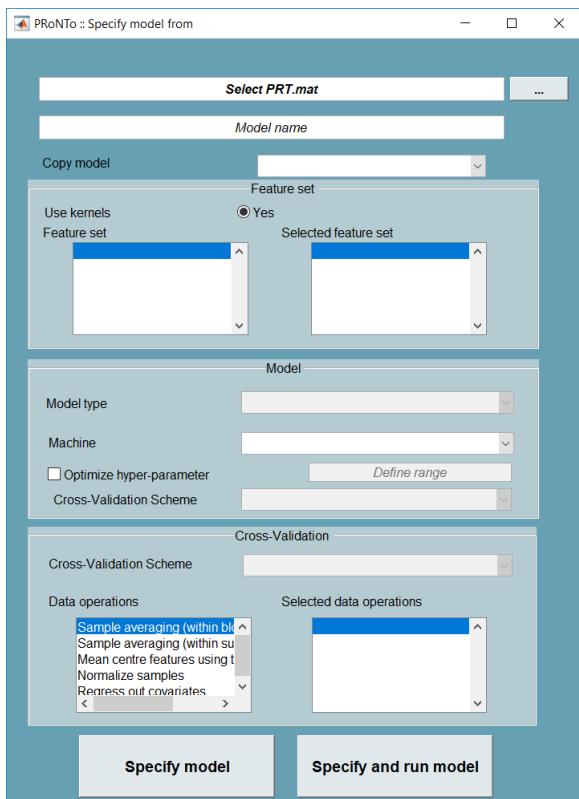


Figure 13.17: ‘Model: Specify from’ GUI.

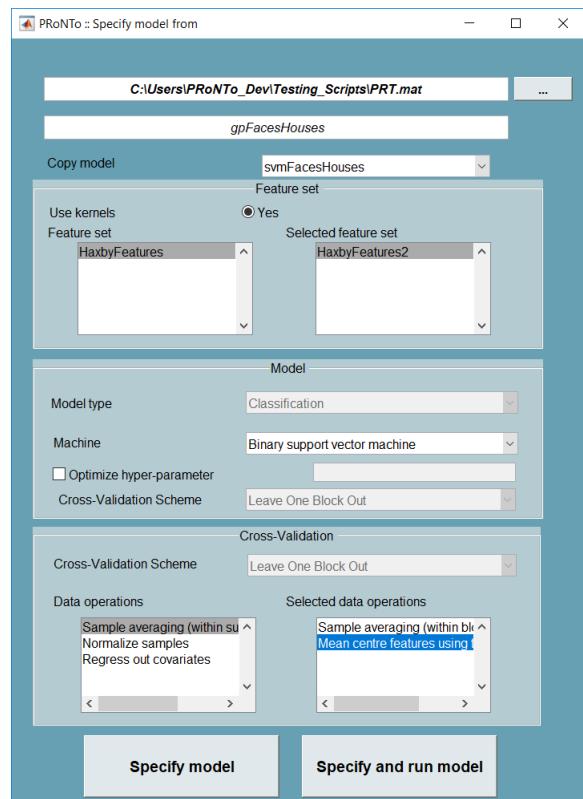


Figure 13.18: ‘Model: Specify from’ GUI final configuration.

For further information the reader should look at the tutorial of Chapter 4.

13.1.5 Model: Run

- From the ‘Run model’ window you can run the model(s) you have specified in the previous section. It is useful when you have specified some model(s) but did not run it/them, and also if you want to run permutations for multiple models (Figure 13.19).

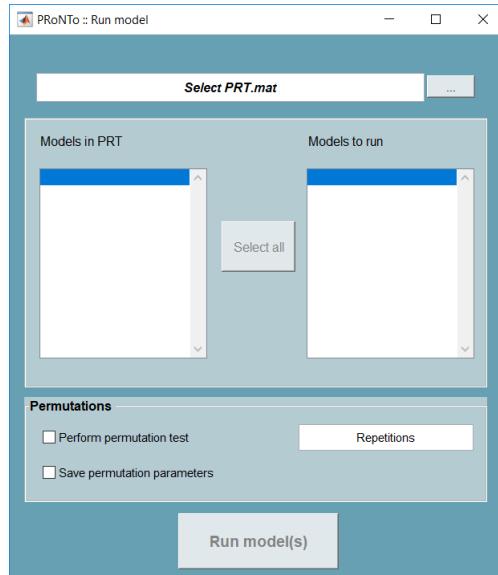


Figure 13.19: ‘Model: Run’ GUI.

- If you want to run permutation tests, check the options ‘Perform permutation test’ and ‘Save permutation parameters’, and also specify the number of repetitions. Keep in mind that in order to check the significance of the results you must run permutation tests. It has been shown that 1000 repetitions approximate well the null distribution, so it’s best if you are as close to that number as possible.

13.1.6 Display model (optional step)

- To review the model specification, in the main PRoNTo GUI, click on ‘Review kernel & CV’ and after you select the ‘PRT.mat’ file to review, a new window will open, ‘Review Model Specification’ (Figure 13.20).

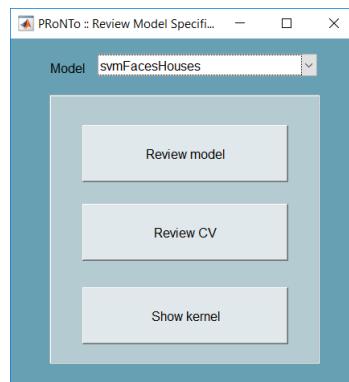


Figure 13.20: Review CV & kernel window.

- Select the model, ‘svmFacesHouses’, from the list at the top and click on ‘Review model’; then, select one class from the list of ‘Class’ to see which groups, subjects and conditions this class comprises (Figure 13.21).

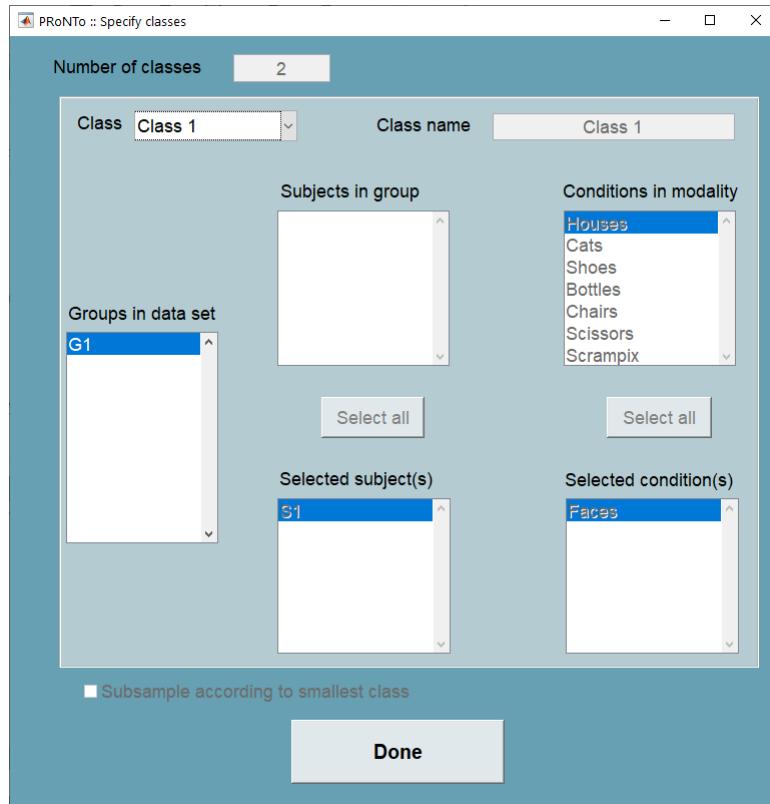


Figure 13.21: Review model specification window for Class 1.

- To review the data and cross-validation matrix click on ‘Review CV’ (Figure 13.22). For more information on what these matrices mean, please consult chapter 4.

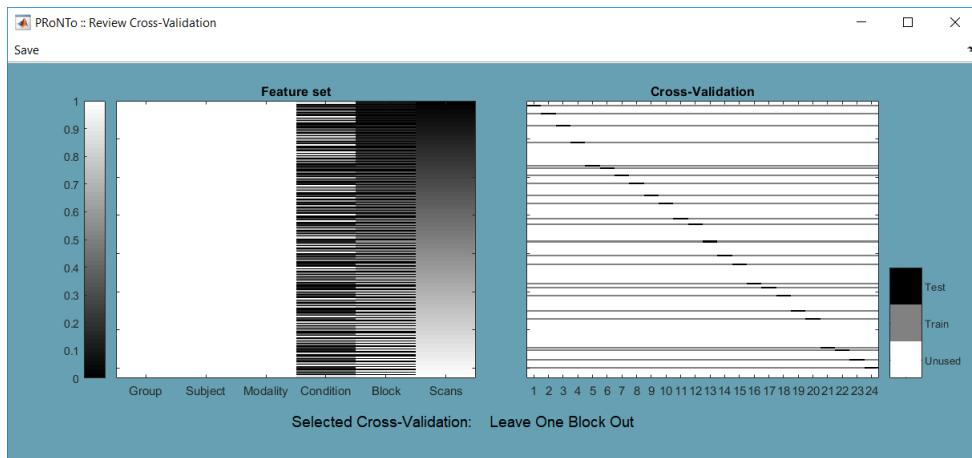


Figure 13.22: Data and cross-validation matrix from ‘Review CV’ option.

- To review the kernel, click on ‘Show kernel’ (Figure 13.23).

13.1.7 Display results

- In PRONT0’s main window, click on ‘Display results’ and select the ‘PRT.mat’ file. This will open the main results window. In the ‘Model’ panel, select the model that you want to view, ‘svmFaces Houses’. The main results window together with the performance stats should be similar to the one in Figure 13.25.

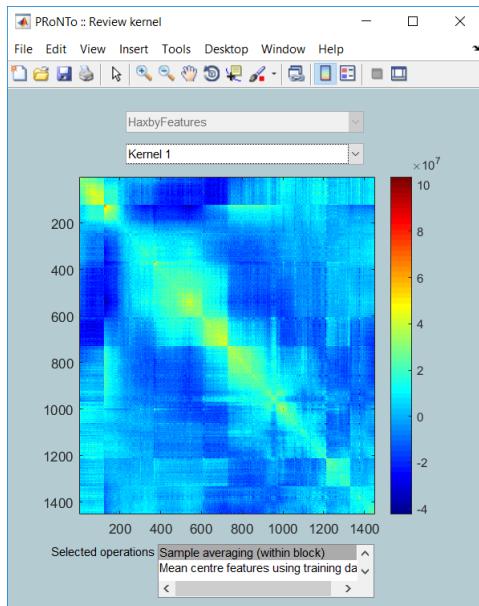


Figure 13.23: Kernel matrix used for classification.

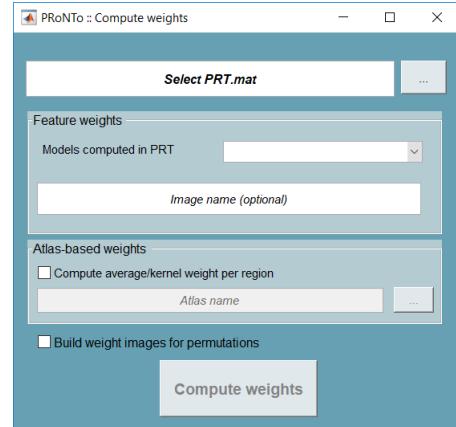


Figure 13.24: ‘Compute weights’ GUI.

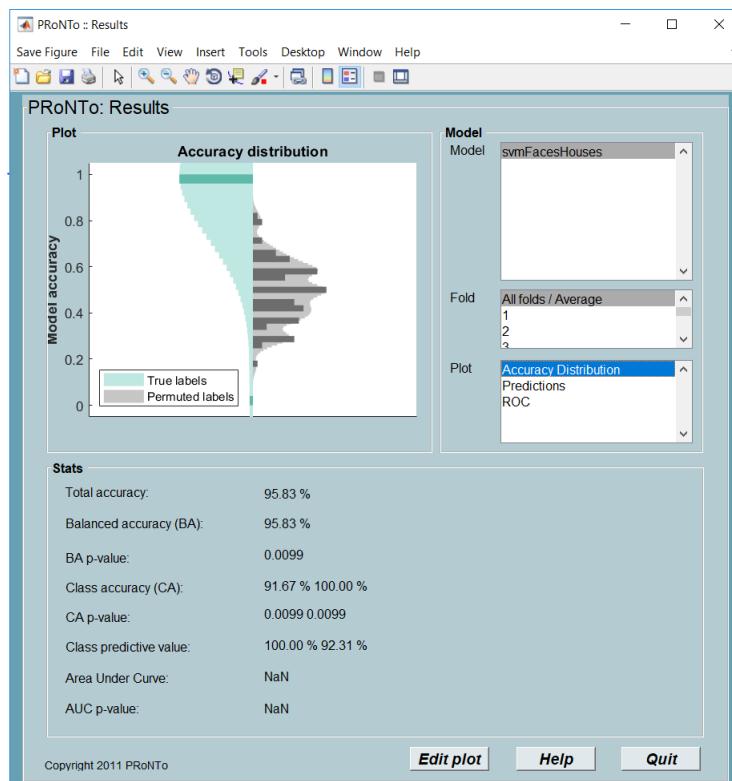


Figure 13.25: ‘Results’ GUI.

- In the ‘Results’ window, one can select specific folds in the ‘Fold’ list and also check their performance with different plots in the ‘Plot’ list.
- Keep in mind that if you have not run permutation tests, the values of ‘BA p-value’ and ‘CA p-value’ would be ‘N.A.’.

A very important change in v3 is that model performance is computed within folds, and then the stats are averaged across all folds. In v2, stats were computed within folds but then predictions were concatenated across folds to produce the model-level stats. This has been advised against as being over-optimistic. For further

information regarding that the reader is referred to chapter 4. This change is reflected in the ‘Display results’ window: the list of plots available across folds is not the same as the list of plots available within folds.

Across fold the user can display:

- ‘Accuracy distribution’: The balanced accuracy in each fold plotted as a violin plot, with its mean displayed in red. If permutations were estimated, the left side of the violin corresponds to the accuracy distribution (with additional histogram), while the right side corresponds to the distribution of average balanced accuracy with permuted labels.
- ‘Predictions’: Same as v2.
- ‘ROC’: In v2, the ROC was built from the concatenated function values. In v3, we now build the average ROC curve and display its standard deviation across folds as shaded area.
- ‘Influence of the hyper-parameters’: Similar to v2, but added scatter plot representing the balanced accuracy within each fold for each value of the hyper-parameter.

Within folds, the plots are the same as in v2. Minor changes were performed to improve their appearance.

13.1.8 Compute weights (optional step)

- In PRoNTo’s main window, click on ‘Compute weights’ and a new window will open, ‘Compute weights’ (Figure 13.24).
- Select the ‘PRT.mat’ file.
- Select the model from the list of ‘Models computed in PRT’, ‘svmFaces Houses’ model.
- If you previously run permutation tests and want to review and display the weights for the permutations, check the option ‘Build weight images for permutations’. Note that this option will create one weight image for each permutation.
- Leave the option ‘Compute average/kernel weight per region’ unchecked.
- Click on ‘Compute weights’ button. Computations will be displayed on the MATLAB command window.

13.1.9 Display weights

- In PRoNTo’s main window, click on ‘Display weights’ and select the ‘PRT.mat’ file. This will open the ‘Model interpretation’ window.
- By clicking on ‘Model’, svmFaces Houses, an image will appear in the ‘Weights map’ box. To show the ‘Anatomical img’ you have to load an anatomical image for reference. A template image can be found in SPM’s canonical folder (‘single_subj_T1’ file). The final window will look similar to the one shown in Figure 13.26.

13.2 Batch analysis

This tutorial will now show how to analyse the same data but using the `matlabbatch` system.

Once again, create a new directory where you wish to save the results. On the main interface of PRoNTo click on the ‘Batch’ button to open the ‘`matlabbatch`’. Alternatively, type ‘prt_batch’ on the MATLAB command window. On the menu bar of the batch, there is a PRoNTo menu with the 6 options shown in the main steps interface (Figure 13.27).

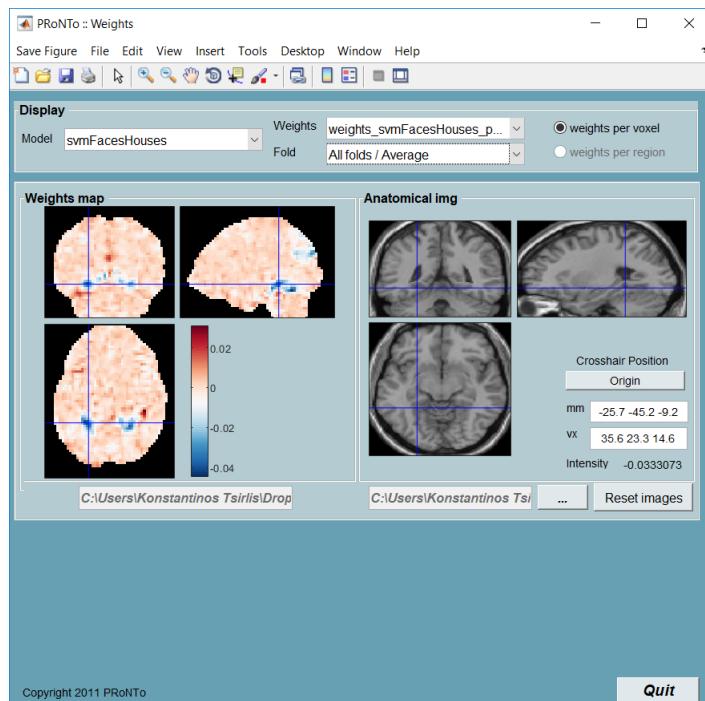


Figure 13.26: ‘Model interpretation’ GUI with results.

13.2.1 Data & Design

- Click on ‘Data & Design’ in the PRoNTo menu. Figure 13.28 is the starting ‘Data & Design’ module menu and Figure 13.29 is the full drop-down list of options. All modules follow the same general structure with some initial options appearing at first, that have a variety of sub-options that appear once you specify them. Figure 13.30 is the final configuration of the matlabbatch ‘Data & design’ module.

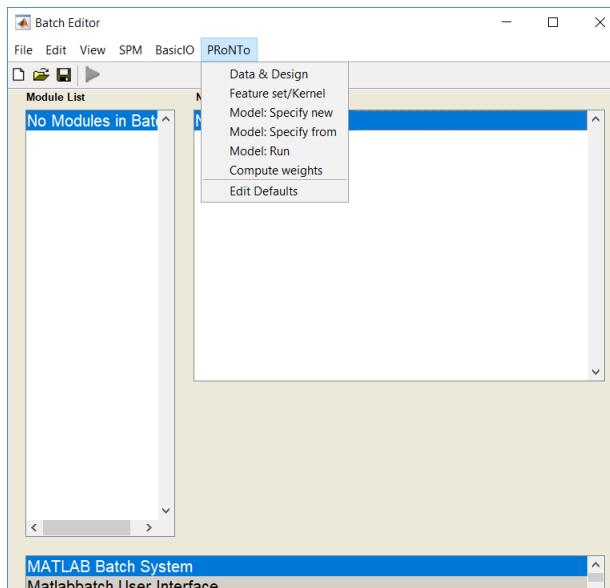


Figure 13.27: Menu PRoNTo in the main matlabbatch window.

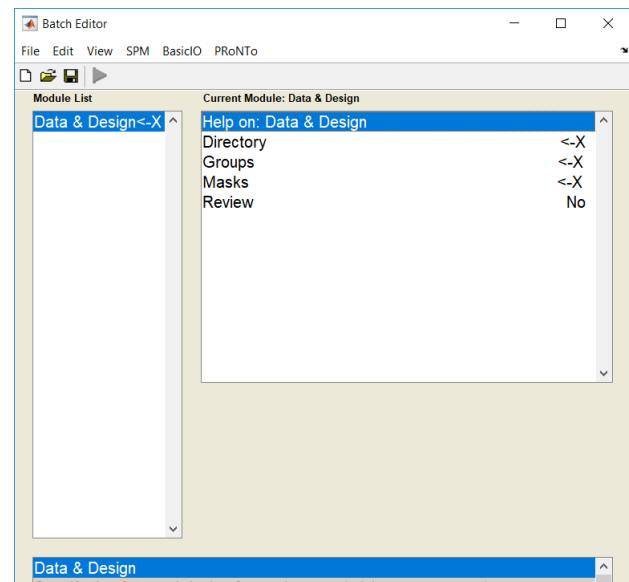


Figure 13.28: ‘Data & design’ module in matlabbatch window.

- In the ‘Directory’ field, select a directory where the ‘PRT.mat’ file will be saved. There are three ways of editing all fields in matlabbatch: (i) by using the right mouse button and clicking on the current option, (ii) clicking on current button in the window or (iii) by double clicking.

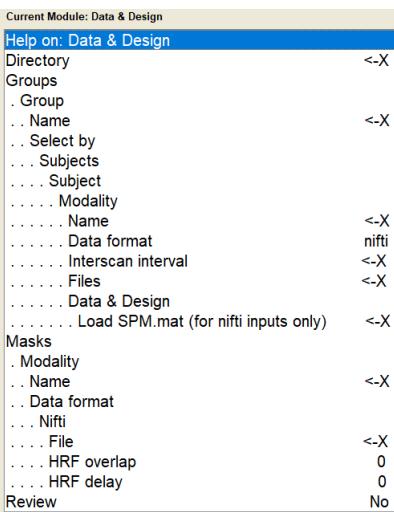


Figure 13.29: Drop-down list of options in the `matlabbatch` ‘Data & design’ module.

Help on: Data & Design	
Directory	<-X
Groups	
. Group	
.. Name	<-X
.. Select by	
... Subjects	
.... Subject	
..... Modality	
..... Name	<-X
..... Data format	nifti
..... Interscan interval	<-X
..... Files	<-X
..... Data & Design	
..... Load SPM.mat (for nifti inputs only)	<-X
Masks	
. Modality	
.. Name	<-X
.. Data format	
... Nifti	
.... File	<-X
.... HRF overlap	0
.... HRF delay	0
Review	No
... Scripts\v3	
G1	
..... Name	fMRI
..... Data format	nifti
..... Interscan interval	2.5
..... Files	1452 files
..... Data & Design	
..... Load SPM.mat (for nifti inputs only)	... \SPM.mat
Masks	
. Modality	
.. Name	
.. Data format	
... Nifti	
.... File	
.... HRF overlap	0
.... HRF delay	0
Review	No

Figure 13.30: Final configuration of the `matlabbatch` ‘Data & design’ module.

- In the ‘Groups’ field:
 - Add one group.
 - In the field ‘Name’, provide a name without spaces to that group, e.g. ‘G1’.
 - In the field ‘Select by’, select the ‘Subjects’ option and add one subject.
 - Add one modality for this subject and provide a name, e.g. ‘fMRI’; choose the appropriate data format (here nifti); define the interscan interval of 2.5 seconds; and in the field ‘Files’, select all the image files available in the fMRI directory of the Haxby dataset.
 - In the ‘Data & Design’ field, since our data are in nifti format, choose the ‘Load SPM.mat’ option. This file is available with the Haxby dataset on PRoNTo’s website⁴ inside the folder Haxby_dataset/design/.
 - In case our data were in MEEG format, we could choose the ‘Events in MEEG file’ option, where we could also further specify regression targets/covariates.
 - * In case there is no ‘SPM.mat’ or ‘MEEG Events’ file available to use, create a new design by selecting the option ‘Specify design’. Choose the units (scans in our case), how many conditions you have, which in this case are 8 conditions (corresponding to the 8 categories of images) and also write the names, onsets, durations and any covariates and/or regression targets of each condition (Figure 13.31). For further information on how to enter covariates and/or regression targets the reader should look at the tutorial of Chapter 14.
- In the ‘Masks’ field, add a new modality and provide the same modality name, ‘fMRI’, choose the appropriate data format and finally select the ‘whole_brain’ mask available in the masks directory of the Haxby dataset. The name of the modality here has to be exactly the same as in ‘Modalities’, otherwise it will not work.
- Leave the ‘HRF overlap’ and the ‘HRF delay’ fields as default.
- In the ‘Review’ field, select ‘Yes’ if you would like to review your data and design in a separate window. Otherwise, leave as it is, i.e. ‘No’. Keep in mind that the procedure pauses while you review the data and that you have to close the ‘Review’ window for the procedure to continue.

13.2.2 Feature set / Kernel

- Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu.

⁴<http://www.mnl.cs.ucl.ac.uk/pronto/prtdata.html>

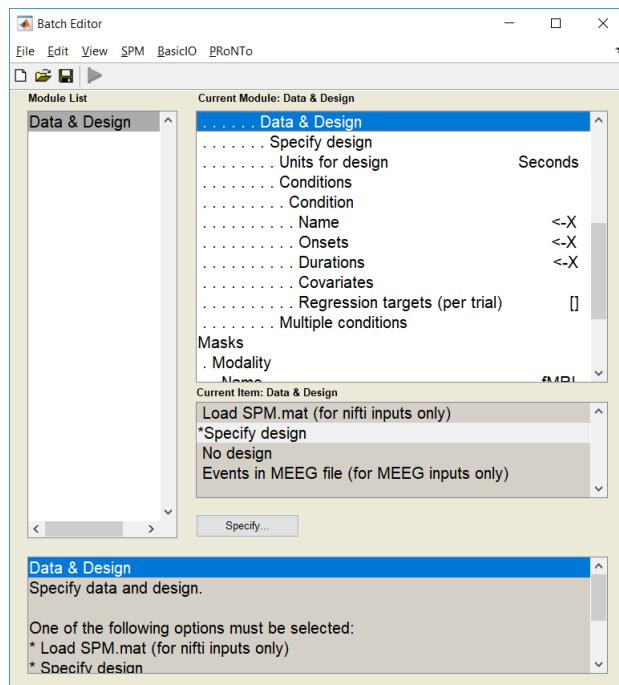


Figure 13.31: ‘Data & design’ module. The ‘Specify design’ option.

- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Data & Design’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved. The window of Figure 13.32 is called to establish a dependency connection with the previous ‘Data & design’ module.

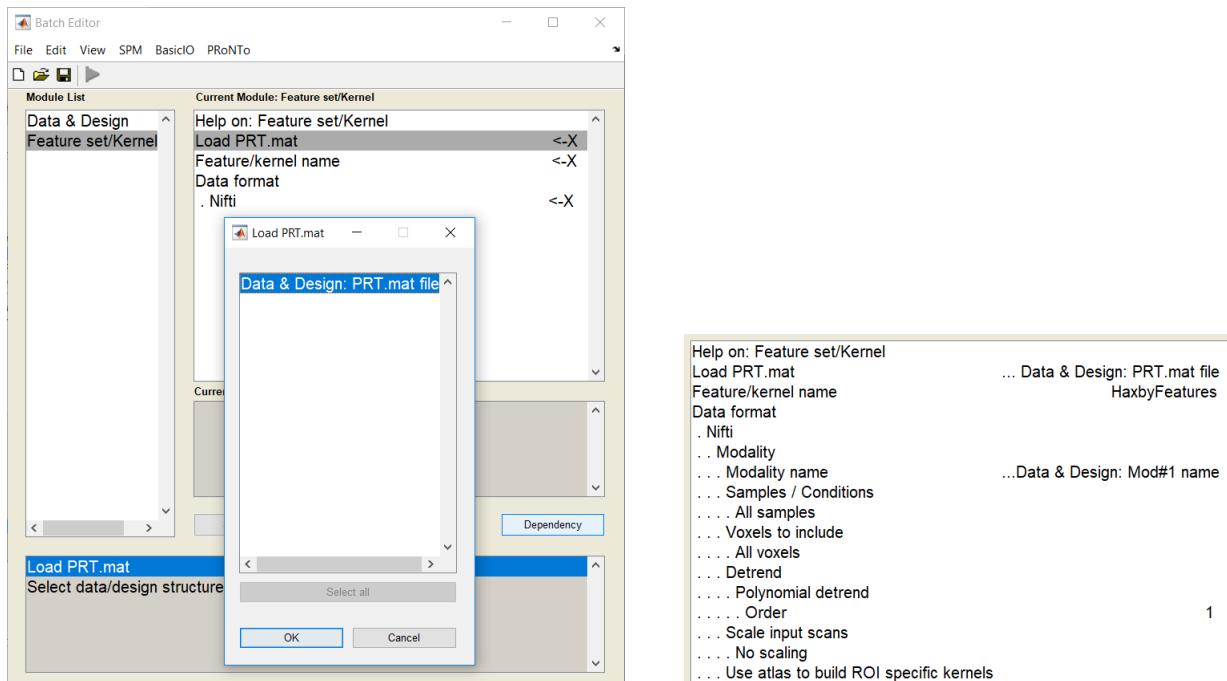


Figure 13.32: ‘Feature set / Kernel’ module in `matlabbatch`.

Figure 13.33: Final configuration of the `matlabbatch` ‘Feature set / Kernel’ module.

- Provide a name to the ‘Feature/kernel’ set, e.g. ‘HaxbyFeatures’.
- Select the ‘nifti’ option for a data format, add one modality and select the modality name with the

‘Dependency’ button⁵(Data & Design:Mod#1 name).

- In the ‘Samples/Conditions’ field , select the ‘All samples’ option.
- In the ‘Voxels to include’ field, select ‘All voxels’ option, this means we are not entering an additional second-level mask.
 - * This is an optional step. In the ‘Voxels to include’ options, the user can specify a ‘second-level’ mask, which would define regions of interest (ROIs) on which the classification can be performed. In this case, select the ‘fusiform_gyrus’ mask.
- In the ‘Detrend’ field, select ‘Polynomial detrend’ option with order 1.
- In the ‘Scale input scans’ field, select ‘No scaling’ option and finally leave ‘Use atlas to build ROI specific kernels’ as default.
- Figure 13.33 is the final configuration of the `matlabbatch` ‘Feature set / Kernel’ module. For the other Data formats please refer to Chapter 17.

13.2.3 Model: Specify new

- Click on ‘Model: Specify new’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Feature set / Kernel’ step, or alternatively either double-click on ‘Load PRT.mat’ option or click on ‘Specify’ button to browse where ‘PRT.mat’ file was saved.
- Provide a name to the model, e.g. ‘svmFaces Houses’.
- In the ‘Feature sets’ field, select the feature set name with the ‘Dependency’ button⁶.
- Select the ‘Classification’ model type:
 - Add 2 new classes.
 - For Class (1) write ‘Faces’ on the name field and add one group. Select the group name from the ‘Data & Design’ module (‘Data & Design:Group#1 name’) with the ‘Dependency’ button⁷. Similarly, for Class (2) write ‘Houses’ on the name field and add the group created in the ‘Data & Design’ module, ‘G1’.
 - In the ‘Subjects’ field, type ‘1’ (only subject 1 is selected).
 - In the ‘Conditions / Scans’ field, select the ‘Specify Conditions’ option and add a new condition. Provide a name for this condition, i.e. for Class (1) ‘Faces’ and for Class (2) ‘Houses’. Note that this name needs to be spelled exactly as specified in the ‘Data & Design’ module: if you simply loaded an ‘SPM.mat’ file for the design, you *must* know the names of the conditions.
- Leave ‘Subsample examples based on class definition’ as it is, i.e. ‘No’.
- In the ‘Machine Type’ field:
 - Select the ‘Kernel machine’ and the ‘SVM Classification’ options.
 - Leave the ‘SVM string argument’ as it is, i.e. ‘-q -s 0 -t 4 -c’.
 - Finally, on the ‘Machine optimization and parameters’ field select ‘No optimization’.
- In the ‘Cross-validation type’ field, select ‘Leave one block out’ option.
- Leave the ‘Include all scans’ field as it is, i.e. ‘No’.
- In the ‘Data operations’ field:
 - Leave the ‘Mean centre features’ field as it is, i.e. ‘Yes’.
 - Leave the ‘Other Operations’ field as it is, i.e. ‘No operations’.

Figure 13.34 is the final configuration of the `matlabbatch` ‘Model: Specify new’ module.

⁵Or type it in manually, ‘fMRI’, but the name needs to be *exactly* the same as the one specified in the ‘Data & Design’ module.

⁶or write it *exactly* as previously defined in the ‘Feature set / Kernel’ module (option ‘Feature set/Kernel: Feature/kernel name’), here ‘HaxbyFeatures’.

⁷Or write it *exactly*, as previously defined in the Data & Design’ module, here ‘G1’

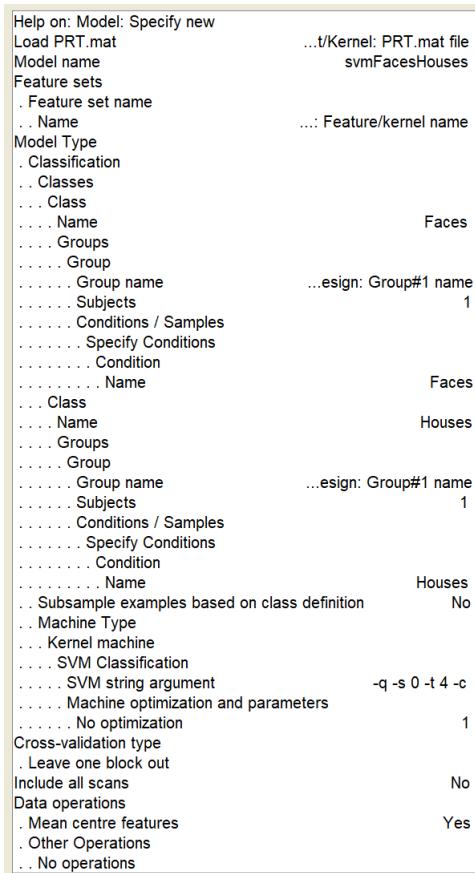


Figure 13.34: Final configuration of the ‘Model: Specify new’ module

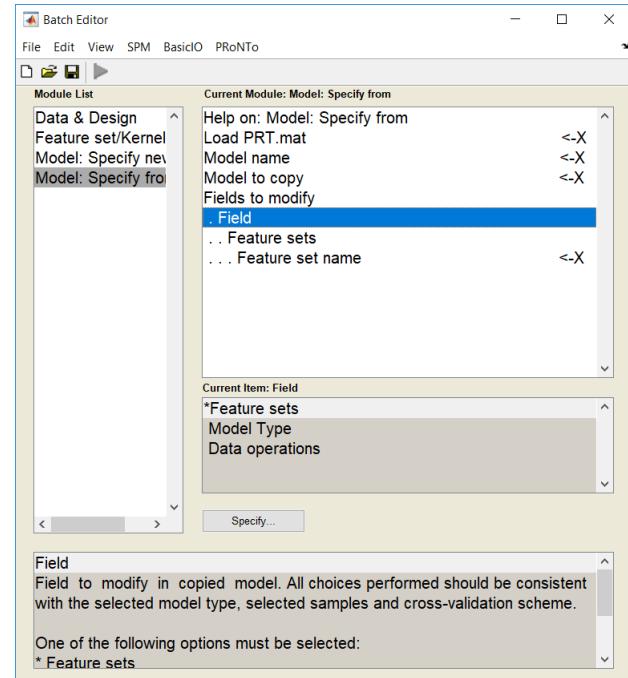


Figure 13.35: ‘Model: Specify from’ module in matlabbatch.

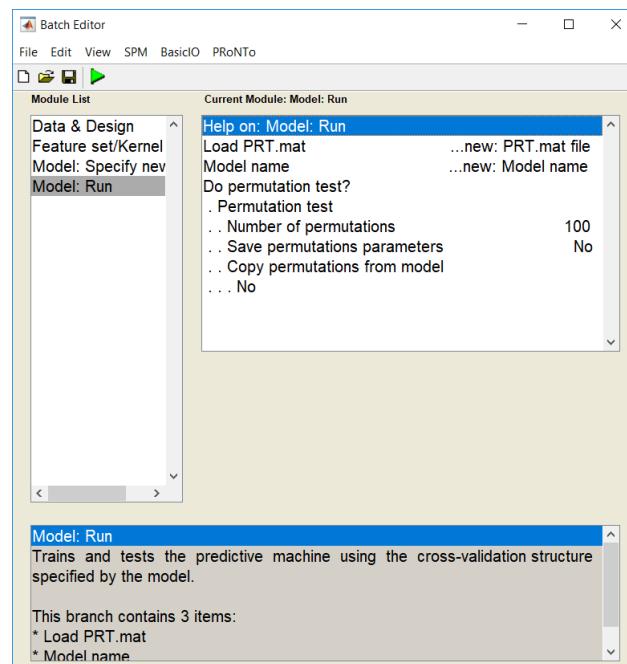


Figure 13.36: Final configuration of the ‘Model: Run’ module in matlabbatch.

13.2.4 Model: Specify from (optional step)

- Click on ‘Model: Specify from’ option on PRoNTo’s `matlabbatch` menu. Figure 13.35 is the full drop-down list of options.
- The ‘Specify model from’ module is similar to the ‘Specify model new’ module, with the appropriate fields disabled to ensure comparability between models. As in the ‘Specify model new’ module first with the ‘PRT.mat’ file selected, click on ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous step, or alternatively either double-click on ‘Load PRT.mat’ option or click on ‘Specify’ button to browse where ‘PRT.mat’ file was saved.
- Choose a ‘Model name’, e.g. ‘gpFaces Houses’, and then specify the name of an existing model from which the previous specifications will be loaded, here ‘svmFaces Houses’.
- There are three different options you can change in your new model. The feature set, the model type and the data operations. For further information regarding these options the user is referred to 4.

13.2.5 Model: Run

- Click on the ‘Model: Run’ option on PRoNTo’s `matlabbatch` menu.
- With the ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Specify model’ step.
- Select the model name from the ‘Model: Specify new’ module with the ‘Dependency’ button, or write it *exactly*, as previously defined in the ‘Model: Specify new’ module, here ‘svmFaces Houses’.
- In the field ‘Do permutation test?’, select ‘Permutation test’ with 1000 repetitions, or as many as you can closer to 1000.
- Leave both ‘Save permutations parameters’ and ‘Copy permutations from model’ as they are, i.e. ‘No’. ‘Copy permutations from model’ should be set to ‘Yes’ if one wants to make sure the same permutations are run for the different models, this will enable applying statistical tests to compare the models. In that case the two models should have exactly the same samples in each fold and use the same cross-validation scheme.

Figure 13.36 is the final configuration of the `matlabbatch` ‘Model: Run’ module.

13.2.6 Compute weights (optional step)

- Click on the ‘Compute weights’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Run model’ step.
- Select the model name from the ‘Model: Specify new’ module with the ‘Dependency’ button.
- It’s optional to define a name for the image.
- Leave the ‘Build weights images for permutations’ field as it is, i.e. ‘No’.

Finally, save the batch (e.g. as `batch_run_all.m`) and click on the ‘Run Batch’ option, in the ‘File menu’. The batch file created can then be opened and edited for further analyses. The results will be the same as those obtained using the GUI (see Section 13.1.7 of this chapter). Please note that in this case the ‘Sample averaging (within block)’ operation was not selected when specifying the model. In order to obtain the same results as before, the model has to use the same data operations.

Figure 13.37 is the final configuration of the `matlabbatch` ‘Compute weights’ module.

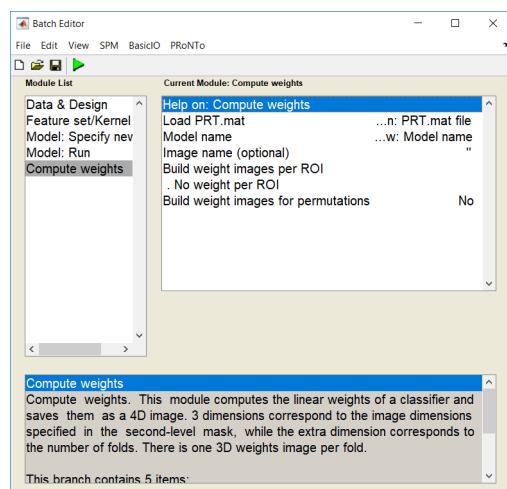


Figure 13.37: Final configuration of the ‘Compute weights’ module in `matlabbatch`.

Chapter 14

Regression dataset

Contents

14.1	GUI analysis	135
14.1.1	Data & Design	135
14.1.2	Prepare feature set	136
14.1.3	Model: Specify new	136
14.1.4	Model: Specify from	137
14.1.5	Display results	137
14.2	Batch analysis	139
14.2.1	Data & Design	139
14.2.2	Feature set / Kernel	139
14.2.3	Model: Specify new (KRR)	140
14.2.4	Model: Run (KRR)	141
14.2.5	Model: Specify and Run (RVR and GPR)	142
14.3	Removing confounds (optional)	142
14.4	Within- and between- subject regression	143

This chapter will describe the steps necessary to perform a regression using PRoNTo. These are similar to the ones in the previous chapter, thus, the reader is advised to complete the tutorial in Chapter 13 before moving on, since the explanation of some steps will be less descriptive. The dataset used in this chapter can be found on PRoNTo’s website <http://www.mlnl.cs.ucl.ac.uk/pronto/prtdata.html> (data set 3).

14.1 GUI analysis

As in Chapter 13, the analysis of the data will start with the PRoNTo’s GUI. Please create a folder in your computer to store the results and type ‘prt’ or ‘pronto’ on the MATLAB command window. This will open the main interface of PRoNTo (see Figure 13.1 in the previous chapter).

14.1.1 Data & Design

- In PRoNTo’s main window, click on ‘Data & Design’. Like in the previous chapter, browse the directory in which to save the PRT structure (saved as ‘PRT.mat’).
- In the panel ‘Groups’, click on ‘Add’ and provide a name to the group, e.g. ‘Aged’.
- Unlike the previous chapter, all the images in the dataset correspond to different subjects; therefore, click on the ‘Samples’ tick box. This will lock the ‘Subjects/Samples’ field, allowing you to skip to the third field.
- In the ‘Modalities’ panel, click on ‘Add’ and provide a name for the modality, e.g. ‘sMRI’.
- In the ‘Data format’ choose the appropriate format for your data (here nifti); For other data formats the reader is referred to the tutorial of Chapter 17.

- You can see that the ‘Design’ section is unavailable. Since we are doing a regression and every subject has only 1 sample/scan, there is no design matrix.
- Load all the image files available in the directory (IXIdata/aged/Guys/). You can select all the files by using the right mouse button and clicking on the option ‘Select All’. When all the images are selected, click on the ‘Done’ button.
- Into the ‘Regression targets’ field click ‘Specify Targets’. A new window will appear where you can either directly write (or paste) the list of target values, where in our case they are available in the ‘Age_old_Guys’ file (IXIdata/aged/), or if you have a .mat file with the regression targets, you can select it directly by selecting the option ‘From .mat file’ and choosing the appropriate .mat file. The two different options are shown in Figures 14.1 and 14.2.

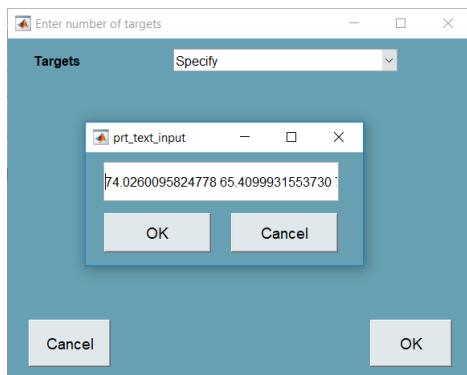


Figure 14.1: ‘Specify’ option in the ‘Specify targets’ GUI.

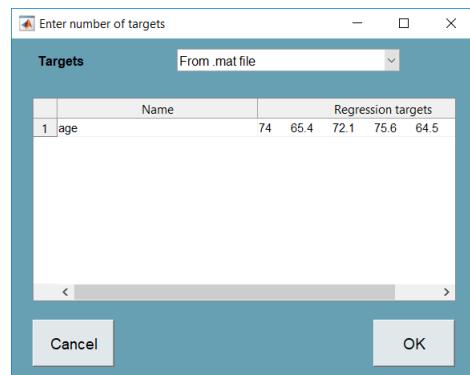


Figure 14.2: ‘From .mat file’ option in the ‘Specify targets’ GUI.

- Leave the ‘Covariates’ section as it is, and press ‘OK’.
- In the ‘Masks’ field, on the bottom left of the ‘Data and design’ window, select the ‘SPM.mask_noeyes’ mask for the specified modality. The mask is available in the path where you have installed PRoNTo (PRoNTo/masks/).
- The ‘Data and design’ window should look similar to Figure 14.3. Click on the ‘Save’ button to create ‘PRT.mat’ file with the structure containing the information that has been previously specified. If no errors are shown in the MATLAB command, leave the ‘Data and design’ window by clicking ‘Quit’.

14.1.2 Prepare feature set

- In PRoNTo’s main window, click on ‘Prepare feature set’ and a new window will open prompting you to select a ‘PRT.mat’ file. Select the ‘PRT.mat’ file previously created in the ‘Data & Design’ step and you are now in the ‘Specify modality to include’ window (see Figure 13.12 in the previous chapter). There is no need to change anything for this example. Just click on the ‘Done’ button.
- Once you click on the ‘Done’ button the window ‘Prepare feature set’ will appear. Provide a name to the feature set, e.g. ‘Scalar_Momentum’; and click on ‘Build Kernel / data matrix’ to build the feature set and kernel.

14.1.3 Model: Specify new

- Next we have to specify a model. In PRoNTo’s main window, click on ‘Specify new’ and a new window will open, ‘Specify model’ (see Figure 13.14 in the previous chapter).
- Select the ‘PRT.mat’ file and provide a name to the model, e.g. ‘KRR’.
- Select from the list one of the ‘Feature Set’ previously defined. In this case, there is only one, ‘Scalar_Momentum’.
- Leave the option ‘Use kernels’ tick box as it is, i.e. ‘Yes’.

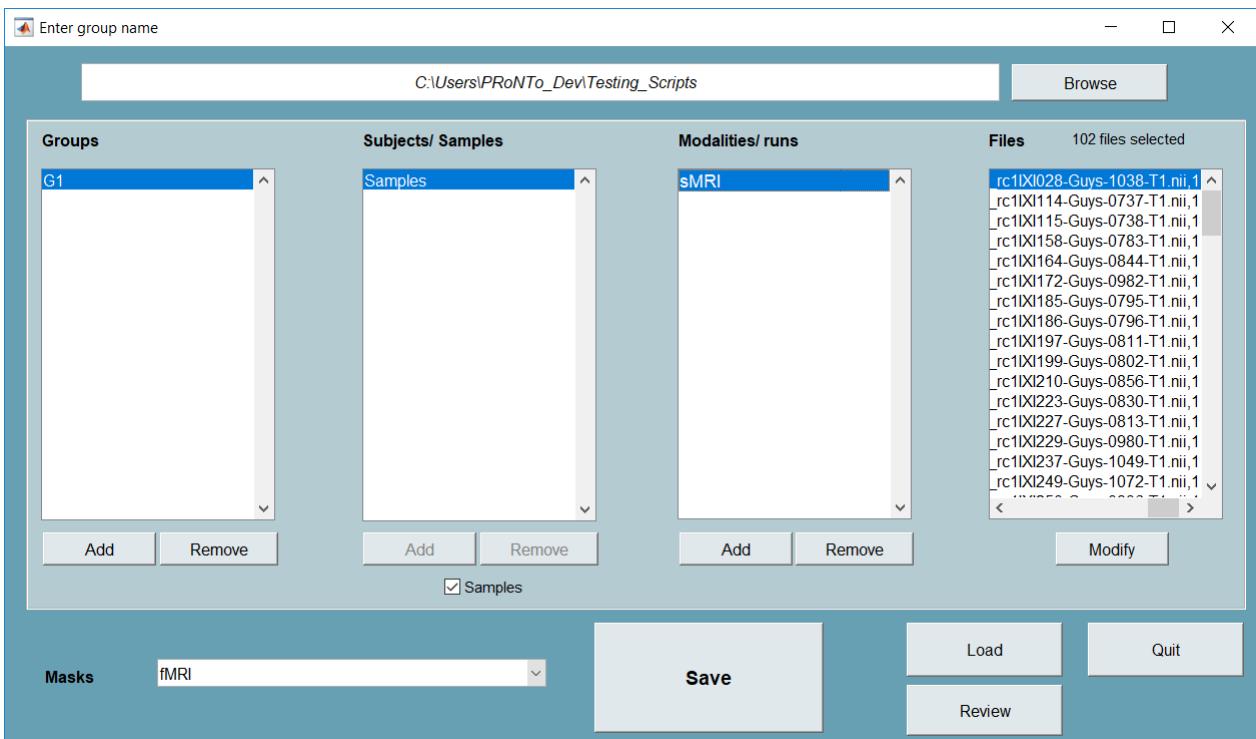


Figure 14.3: ‘Data and design’ GUI final configuration.

- Select the ‘Regression’ model type and click on the ‘Select subjects/scans’ button. This will open a new window, ‘Specify subjects/scans to regress’, click on the ‘Select all’ button to use all the scans for the regression and ‘age’ as our regression target (Figure 14.4). Current models available in PRoNTo do not enable multi-output prediction.
- Select the ‘Kernel Ridge Regression’ option, in the Machine field.
- Leave the option ‘Optimize hyper-parameter’ tick box unchecked and ‘Cross-Validation Scheme’ (internal loop) as it is.
- Select the ‘Leave One Subject Out’ cross-validation scheme (external loop).
- In the ‘Data operations’ box, select only the ‘Mean centre features using training data’ option. The final ‘Specify model new’ window should look similar to the Figure 14.5. Click on the ‘Specify and run model’ button.

14.1.4 Model: Specify from

Use this module to copy the configurations of the previously defined KRR model but select the other options in the ‘Machine’ drop-down list (‘Relevance Vector Regression’ and ‘Gaussian Process Regression’) and give different names to each model. For further information regarding the ‘Model: Specify from’ section the reader is referred to 13.1.4 in the previous chapter and to chapter 4.

14.1.5 Display results

- In PRoNTo’s main window, click on ‘Display results’ and select the ‘PRT.mat’ file. This will open the main results window similar to the Figure 14.6.
- In the ‘Results’ window, one can select the different regression models in the ‘Model’ list on the upper right panel. This will show the results obtained using each one of the regression models.
- Since we didn’t run any permutation testing, the p-values of Correlation, R², MSE and Normal MSE are not available. If one wants to check the statistical significance of the results, one should run his/her

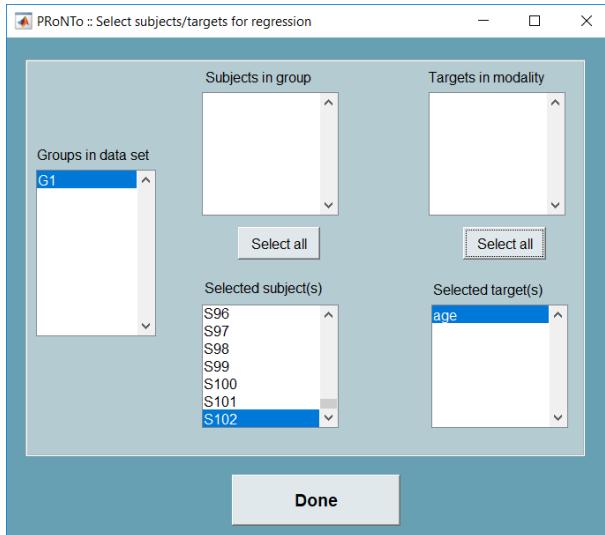


Figure 14.4: ‘Select subjects/scans’ GUI.

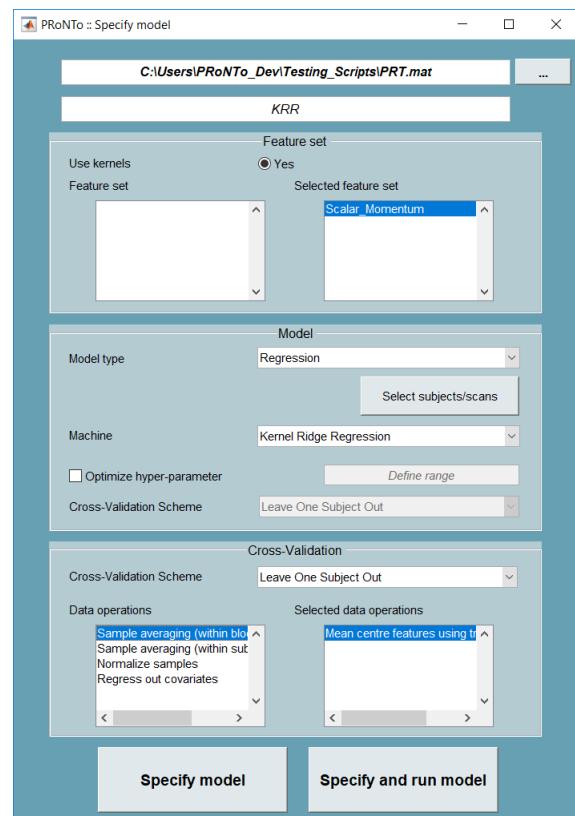


Figure 14.5: ‘Model: Specify new’ GUI.

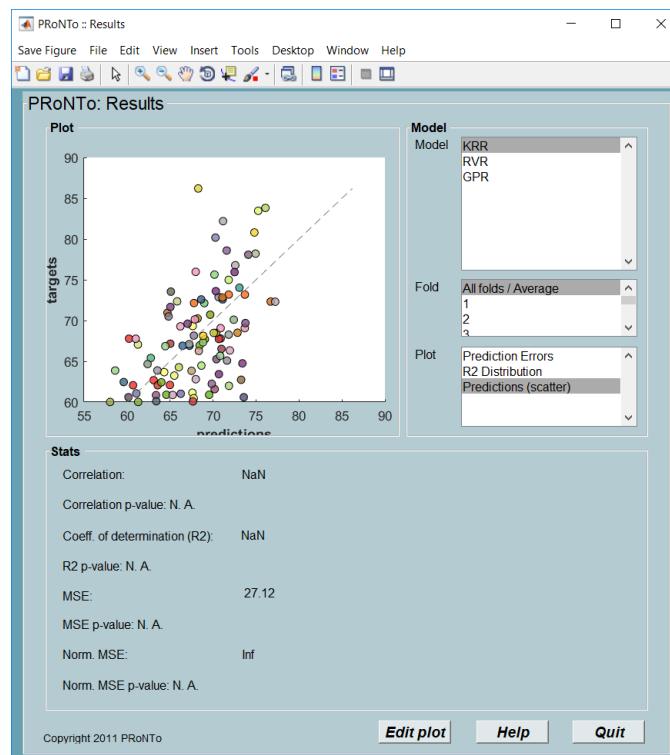


Figure 14.6: ‘Display results’ GUI.

models from the ‘Model: Run’ section of the main window, with permutation testing. For more detailed

information the reader is referred to [13.1.5](#).

14.2 Batch analysis

In this section, the previous experiment will be repeated using the ‘`matlabbatch`’ system. The reader is advised to complete the tutorial in Section [13.2.1](#) before continuing, since the explanation of each step will be less descriptive.

Once again, to analyse the data, create a new directory in which to save the results of the analysis. On the main interface of PRoNTo click on the ‘Batch’ button to open the ‘`matlabbatch`’. Alternatively, type ‘`prt_batch`’ in the MATLAB prompt.

14.2.1 Data & Design

- Click on ‘Data & Design’ in the PRoNTo menu (see Figure [13.29](#) in the previous chapter).
- In the ‘Directory’ field, select a directory where the ‘PRT.mat’ file will be saved.
- In the ‘Groups’ field:
 - Add one group.
 - In the field ‘Name’, provide a name without spaces for this group, e.g. ‘Aged’.
 - In the field ‘Select by’, select the ‘Samples’ option and add a new modality. For more information on the Samples option please consult Chapter [2](#).
 - Provide a name for this modality, e.g. ‘sMRI’ and choose the appropriate data format (here nifti).
 - Select the image files available in the ‘aged/Guy’ directory of the IXI dataset.
 - In the ‘Regression targets (subject)’ option specify the regression targets by selecting the ‘Age_old_Guys’ .mat file (IXIdata/aged/).
 - Leave ‘Covariates’ field as default.
- In the ‘Masks’ field, add a new modality and provide the same modality name, ‘sMRI’; choose the appropriate data format, and select the ‘SPM_mask_noeyes’ mask available in the path where you have installed PRoNTo (PRoNTo/masks/). The name of the modality here has to be exactly the same as in ‘Modalities’, otherwise it will not work.
- Leave the ‘HRF overlap’ and the ‘HRF delay’ fields as default.
- In the ‘Review’ field, select ‘Yes’ if you would like to review your data and design in a separate window. Otherwise, leave as it is, i.e. ‘No’.

Figure [14.7](#) is the final configuration of the `matlabbatch` ‘Data & design’ module.

14.2.2 Feature set / Kernel

- Click on the ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Data & Design’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved.
- Provide a name to the ‘Feature/kernel’ set, e.g. ‘Scalar_Momentum’.
- Select the ‘Nifti’ option for a data format, add one modality and select the modality name with the ‘Dependency’ button¹(Data & Design:Mod#1 name).
 - In the ‘Samples/Conditions’ field, select ‘All samples’ option.

¹Or type it in manually, ‘sMRI’, but the name needs to be *exactly* the same as the one specified in the ‘Data & Design’ module.

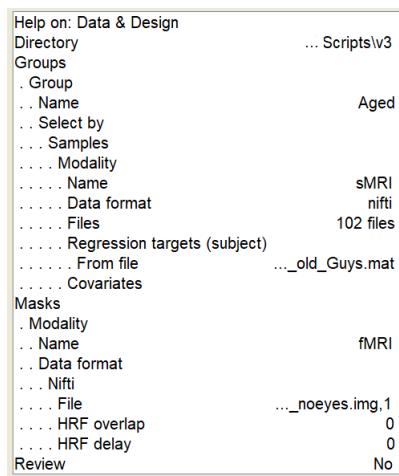


Figure 14.7: Data & design module in matlabbatch.

- In the ‘Voxels to include’ field, select ‘All voxels’ option, this means we are not entering with an additional second-level mask.
- In the ‘Detrend’ field, select the ‘None’ option.
- In the ‘Scale input scans’ field, select the ‘No scaling’ option and finally leave ‘Use atlas to build ROI specific kernels’ as default.
- Figure 14.8 is the final configuration of the matlabbatch ‘Feature set / Kernel’ module. For the other Data formats the reader is referred to chapter 4.

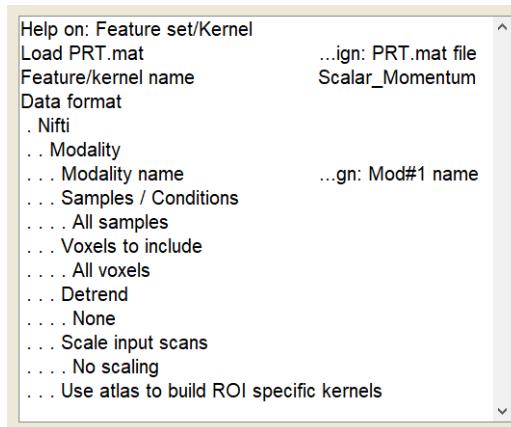


Figure 14.8: Feature set / Kernel module. Selected parameters in the Modality option.

14.2.3 Model: Specify new (KRR)

- Click on the ‘Model: Specify new’ option on PRoNTTo’s matlabbatch menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Feature set / Kernel’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved.
- Provide a name to the model, e.g. ‘KRR’.
- Select the feature set name with the ‘Dependency’ button². From v3.0, multiple feature set is supported so you can put more than one feature set names.

²or write it *exactly* as previously defined in the ‘Feature set / Kernel’ module (option ‘Feature set/Kernel: Feature/kernel name’), here ‘Scalar_Momentum’.

- Select the ‘Regression’ model type:
 - Add a new group and call it ‘Aged’.
 - In the ‘Subjects’ field, type ‘1:102’. This will instruct the program to use all the 102 scans, i.e. from scan 1 to scan 102.
 - In the ‘Conditions / Samples’ choose ‘Target’ and type in ‘age’, as this is the name of our regression target variable in the ‘Age_old_Guys.mat’ file.
- In the ‘Machine’ field:
 - Select ‘Kernel machine’ and then select the ‘Kernel Ridge Regression’ option:
 - In the ‘Machine optimization and parameters’, select ‘No optimization’, and leave ‘No optimization’ as it is, i.e. ‘1’.
- In the ‘Cross-validation type’ field, select ‘Leave One Subject Out’ option.
- Leave the ‘Include all scans’ field as it is, i.e. ‘No’.
- In the ‘Data operations’ field:
 - Leave the ‘Mean centre features’ field as it is, i.e. ‘Yes’.
 - Leave the ‘Other Operations’ field as it is, i.e. ‘No operations’.

The final configuration of the of the `matlabbatch` ‘Model: Specify new’ module should look similar to Figure 14.9.

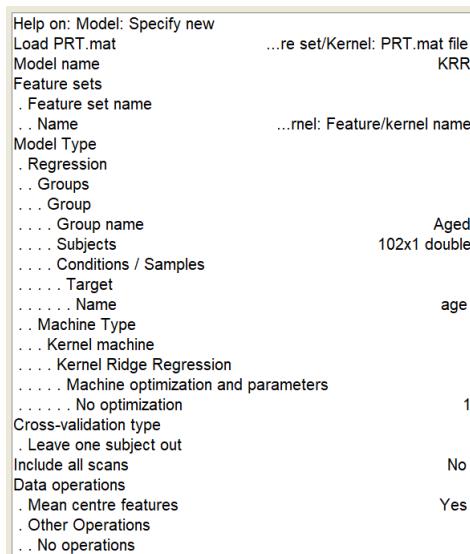


Figure 14.9: ‘Model: Specify new’ module in `matlabbatch`, final configuration.

14.2.4 Model: Run (KRR)

- Click on the ‘Run model’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Specify model’ step.
- Select the model name from the ‘Specify model’ module with the ‘Dependency’ button³.
- In the field ‘Do permutation test?’, leave as it is, i.e. ‘No permutation test’

³or write it *exactly* as previously defined in the ‘Specify model’ module, here ‘KRR’

14.2.5 Model: Specify and Run (RVR and GPR)

The specification of the other models ('Relevance Vector Regression' and 'Gaussian Process Regression') can be done using the 'Specify from' module. The only difference is that in the 'Machine' field of the 'Specify from' module, one has to choose the appropriate machine to use ('Relevance Vector Regression' or 'Gaussian Process Regression'). The parameters used for each machine should be the default ones.

Note that when the 'PRT.mat' file is loaded in each module, the user should select the latest option on the list.

When all the models are defined, the 'Module List' should contain 8 modules:

1. Data & Design.
2. Feature set/Kernel.
3. Mode: Specify new.
4. Mode: Run.
5. Mode: Specify from.
6. Mode: Run.
7. Mode: Specify from.
8. Mode: Run.

Note that modules 3 and 4 correspond to the KRR model; 5 and 6 to the RVR model; 7 and 8 to the GPR model.

When all the modules are added, just click on the 'Run Batch' button. The resulting 'PRT.mat' file will be saved in the specified directory and the results can be viewed using the process described in Section 14.1.5.

14.3 Removing confounds (optional)

In this tutorial until now we only had subjects from a specific testing site. Say now that in your study this isn't true. For example in the IXI study there were 3 testing sites, Hammersmith Hospital using a Philips 3T system, Guys Hospital using a Philips 1.5T system and the Institute of Psychiatry using a GE 1.5T system. What changes? How could we account for the potential differences of the scanner systems? One way to do it is by 'regressing out the covariates'. Please see section 2.5.1 for important considerations on covariates. Chapter 16 is a detailed tutorial about removing confounds in a classification example, so the reader is strongly advised to go through that tutorial first and if one is interested to try a regression example as well, one should come back here afterwards as this is not a complete tutorial regarding removing confounds.

The first main thing that one has to change is in the 'Data & design' module.

- In the 'Files', you need to select all 170 subjects from all the 3 different folders (representing one testing site each).
- In the 'Covariates' field, you need to type the covariates you want to regress out. Covariates can be input to PRoNTo by directly copying values into the Covariate if there is only one confounding factor (e.g. a continuous confound), or by typing the full path of the R matrix containing either one column or multiple columns. As 'Testing site' is categorical, one-hot encoding was used (i.e. one column for each testing site in the 'R' matrix). In this example, you need to select the file Confounds/_one/_hot.mat found in the IXI/Data/aged/ folder.

The second important thing is in the 'Specify model new' module.

- Select the 'Kernel Ridge Regression' option in the Machine field. You can also do the same procedure choosing 'Gaussian Process Regression' if you want to try more than one machines.
- Check the option 'Optimize hyper-parameter' tick box. Type '10.^[-5:5]' on the right field. Then, select 'k-fold CV on Subject-Out' option in the 'Cross-Validation Scheme' field (internal loop), and type '5' in the new window that will open.

- Select the ‘Leave One Subject Out’ cross-validation scheme (external loop).
- Now here comes the important part, where we specify that we want to regress out covariates. In the ‘Data operations’ box, first select the ‘Mean centre features using training data’ option as we are doing so far. But now also select the ‘Regress out covariates (subject level)’ option, which corresponds to the removal of the contribution of some external variables to the data. Click on the ‘Specify and run model’ button.

If you now wish to find the effects of removing the covariates, you have to rerun a new model exactly the same with the previous one, but without regressing out the covariates in the ‘Data operations’ box, and compare the differences in the performance measures.

14.4 Within- and between- subject regression

As mentioned sporadically throughout the whole PRoNTo Manual, until PRoNTo v2.1, the user could only input one regression target per subject. Thus, the option of doing within-subject regression was not available, and between-subject regression could only be done with one regression target.

From PRoNTo v3.0, multiple regression targets can be specified at the Data & Design level (although they can only be modelled independently, i.e. one regression model per target). The new functionalities in this version, enable users to specify both which target and which samples (i.e. which subjects for between-subject regression or which trials for within- subject regression) to use for the regression model.

The user is referred to Part I for further information regarding this topic.

Chapter 15

Multiple Kernel Learning example

Contents

15.1	GUI analysis	146
15.1.1	Data & Design	146
15.1.2	Prepare feature set	146
15.1.3	Model: Specify new	147
15.1.4	Model: Specify from	147
15.1.5	Model: Run	148
15.1.6	Display model (optional step)	148
15.1.7	Display results	149
15.1.8	Compute weights	150
15.1.9	Display weights	150
15.2	Batch analysis	151
15.2.1	Data & Design	151
15.2.2	Feature set / Kernel	151
15.2.3	Model: Specify new	152
15.2.4	Model: Run	154
15.2.5	Compute weights (optional step)	154

This chapter will describe the steps necessary to perform a classification with SimpleMKL <http://asi.insa-rouen.fr/enseignants/~arakoto/code/mklindex.html> [20] using PRoNTo. These are similar to the ones in Chapter 13, thus, the reader is advised to complete the tutorial in Chapter 13 before moving on, since the explanation of some steps will be less detailed.

Many practical learning problems involve multiple and heterogeneous data sources. In this way, Multiple Kernel Learning (MKL) [2] has been proposed to combine different data sources in a single predictive model by learning the relative contribution of each kernel (often corresponding to each data source) to the model. In MKL, the kernel K can be considered as a linear combination of M ‘basis kernels’. For further details, please refer to [2].

One example of a MKL approach based on SVM is the SimpleMKL algorithm [20]. Essentially, the algorithm is based on a gradient descent on the SVM objective value and iteratively determine the combination of kernels by a gradient descent wrapping [20]. For further details, please refer to [20]

We will use the same dataset used in Chapter 13, this fMRI dataset originates from a study on face and object representation in human ventral temporal cortex [10]. The dataset¹ used in this chapter can be found in PRoNTo’s website <http://www.mlnl.cs.ucl.ac.uk/pronto/prtdata.html> (data set 1) and the whole² dataset is available in <http://data.pymvpa.org/datasets/haxby2001/>.

For simplicity, in this example we will use PRoNTo to predict if the subject is viewing an image of a Face or a House based on the fMRI scans. We will classify the whole brain images using SimpleMKL and a leave one block out cross-validation scheme.

¹Pre-processed (realigned and normalised) data from participant 1.

²Not pre-processed.

15.1 GUI analysis

We will first analyse the data using PRoNTo's GUI and then repeat the analysis using the `matlabbatch` system.

To start, create a new directory in which to save the results of the analysis, then start up MATLAB and type ‘prt’ or ‘pronto’ in the MATLAB prompt. This will open the main interface of PRoNTo.

15.1.1 Data & Design

- In PRoNTo’s main window, click on ‘Data & Design’. Like in the previous chapters, browse the directory in which to save the PRT structure (saved as ‘PRT.mat’);
- In the panel ‘Groups’, click on ‘Add’ and provide a name to the group (we only have one group/subject), with no spaces, e.g. ‘G1’;
- Add a subject in the ‘Subject/Samples’ option, e.g. ‘S1’, and leave the ‘Samples’ tick box below the panel unchecked. See Chapter 2 of the manual for more information on this option;
- In the ‘Modalities’ panel, click on ‘Add’ and provide a name to the modality, e.g. ‘fMRI’. In the ‘Data format’, choose ‘nifti’. In the ‘Design’ field, choose the option ‘Load SPM.mat’. This file is available with the Haxby dataset on PRoNTo’s website³ inside the folder Haxby_dataset/design/. Finally, in this case leave ‘Regression targets’ as it is, in the ‘No targets’ option.
 - The reader is referred to Chapter 13 and chapter 2 for the case where there is no ‘SPM.mat’ file available.
- Finally, load all the image files available in the fMRI directory (Haxby_dataset/fMRI/). You can select all the files by using the right mouse button and clicking on the option ‘Select All’. When all the images are selected, click on the ‘Done’ button;
- In the ‘Masks’ field, on the bottom left of the ‘Data and design’ window, select the ‘whole_brain’ mask for the specified modality. The mask is available in the masks directory inside the folder Haxby_dataset/masks/;
- Click on the ‘Review’ button to check the data and the design inserted for this modality. For more information on what one can do with the Review option, please see Chapter 2;
- Click on the ‘Save’ button to create the ‘PRT.mat’ file with the structure containing the information that has been previously specified. If no errors are shown in the MATLAB command, leave the ‘Data and design’ window by clicking ‘Quit’. The ‘Data and design’ window should look similar to the one in Figure 13.10 from the previous chapter.

15.1.2 Prepare feature set

- Next we have to prepare the feature set, so click on ‘Prepare feature set’ in PRoNTo’s main window. A new window will open prompting you to select a PRT.mat file. Select the ‘PRT.mat’ file previously created in the ‘Data & Design’ step, as in Figure 13.11 from Chapter 13. Once you click ‘Done’ another window will appear, ‘Specify modality to include’, as in Figure 13.12 from Chapter 13. Here you set the specification of different parameters and options for each modality, which are:
 - ‘Modality’ field: select the modality previously specified in the ‘Data & Design’ step, ‘fMRI’;
 - ‘Conditions’ field: select ‘All scans’;
 - ‘Parameters’ box: select the polynomial detrend with order 1 and the ‘No scaling’ option;
 - ‘Features’ box: select the ‘Build one kernel per region’ tick box and load the ‘AAL’ atlas (named ‘aal_79x91x69’) available in the PRoNTo directory (PRoNTo/atlas/). Then, click on the ‘Done’ button. The final ‘Specify modality to include’ window should look similar to the one in Figure 15.1;
- Once you specify the modality to include and click ‘Done’, yet another window will appear, ‘Prepare feature set’. Here you provide a name for the feature set, e.g. ‘HaxbyFeatures’ and finally you click on ‘Build kernel / data matrix’ to build the kernel. If everything was done correctly, a progress bar will pop up, marking the start of the procedure, similar to the Figure 13.13 from Chapter 13, which can take a few minutes.

³<http://www.mnl.cs.ucl.ac.uk/pronto/prtdata.html>

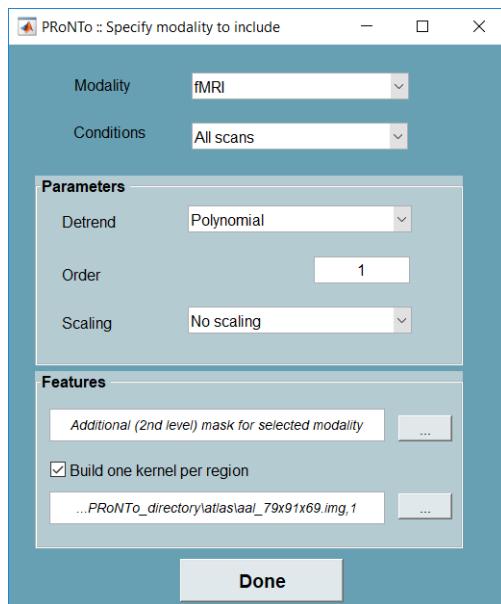


Figure 15.1: ‘Specify modality to include’ GUI final configuration.

15.1.3 Model: Specify new

- In PRoNTo’s main window, click on ‘Specify model’ and a new window called ‘Specify model’ will open (see Figure 13.14 in Chapter 13);
- Select the ‘PRT.mat’ file and provide a name to the model, e.g. ‘mklFacesHouses’;
- Select one of the feature sets previously defined. In this case, there is only one: ‘HaxbyFeatures’;
- Leave the option ‘Use kernels’ tick box as it is, i.e. ‘Yes’;
- Select the ‘Classification’ model type and click on the ‘Define classes’ button. A new window will open, ‘Specify classes’, to define the number of classes and a name for each class. We will define 2 classes. First click ‘Class 1’ on the tab ‘Class’. For ‘Class 1’ select subject ‘S1’ and the condition ‘Faces’ and, similarly, for ‘Class 2’ select subject ‘S1’ and the condition ‘Houses’. Leave the ‘Subsample according to smallest class’ as it is for now. Once you have appropriately specified everything, click ‘Done’.
- Select the ‘L1 Multi-Kernel Learning’ option, in the ‘Machine’ field;
- Select the ‘Optimize hyper-parameter’ tick box, leave the ‘Define Range’ at its default range (i.e. [0.1, 1, 10, 100]) and in the ‘Cross-Validation Scheme’ (internal loop) field, select the option ‘k-fold CV on Block’. A window will appear asking to define the value of k, set it to 4.
- Select the ‘Leave One Block Out’ cross-validation scheme (external loop);
- In the ‘Data operations’ box, select the ‘Mean centre features using training data’ and ‘Normalize samples’ options. Then, the ‘Specify model’ window should look similar to the one in Figure 15.2;
- Click on ‘Specify and run model’ and the model will be immediately estimated, therefore there is no need to use the ‘Run model’ module in this case. If however you wanted to check the statistical significance of your results, you need to use the ‘Run model’ module in order to do permutation testing. It will take a few minutes to complete.

15.1.4 Model: Specify from

- The reader is referred to Chapter 13 and chapter 4 for more details regarding the ‘Model: Specify from’ module.

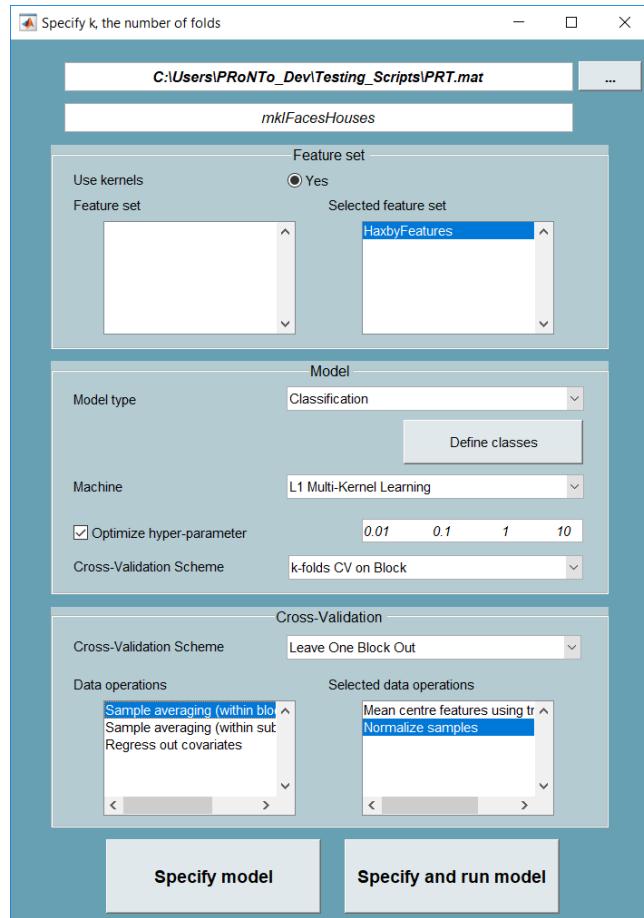


Figure 15.2: ‘Specify model new’ GUI final configuration.

15.1.5 Model: Run

The reasons for using the ‘Model: Run’ module and not just click ‘Specify and run model’ in the ‘Model: Specify new/from’ module are the following:

- One might want to specify one or more models now but run it/them later.
- One might want to specify more than one models first and run them all together at once afterwards.
- One might want to run permutation tests in order to check the statistical significance of the results. The reader is referred to Chapter 13 and section 5.4 for more details.

15.1.6 Display model (optional step)

- To review the model specification, in the main PRoNTo GUI, click on ‘Review kernel & CV’ and after you select the ‘PRT.mat’ file to review, a new window will open, ‘Review Model Specification’ (Figure 13.20) in Chapter 13).
- Select the model, ‘mklFacesHouses’, from the list at the top and click on ‘Review model’; then, select one class from the list of ‘Class’ to see which groups, subjects and conditions this class comprises, similar the one to Figure 13.21) in Chapter 13;
- To review the data and cross-validation matrix click on ‘Review CV’ and a Figure similar to Figure 13.22 in Chapter 13 will appear. For more information on what these matrices mean, please consult chapter 4.
- To review the kernel, click on ‘Show kernel’ (Figure 15.3).

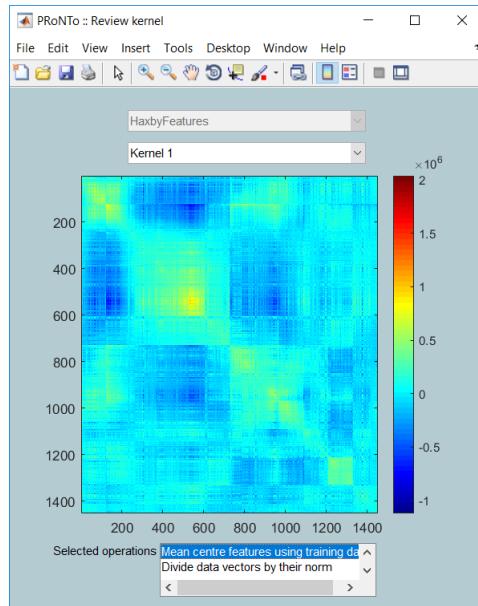


Figure 15.3: Kernel matrix used for classification.

15.1.7 Display results

- In PRoNTo's main window, click on 'Display results' and select the 'PRT.mat' file. This will open the main results window. In the 'Model' panel, select the model that you want to view, 'mklFacesHouses', and the results through performance will be similar to the one in Figure 15.4;

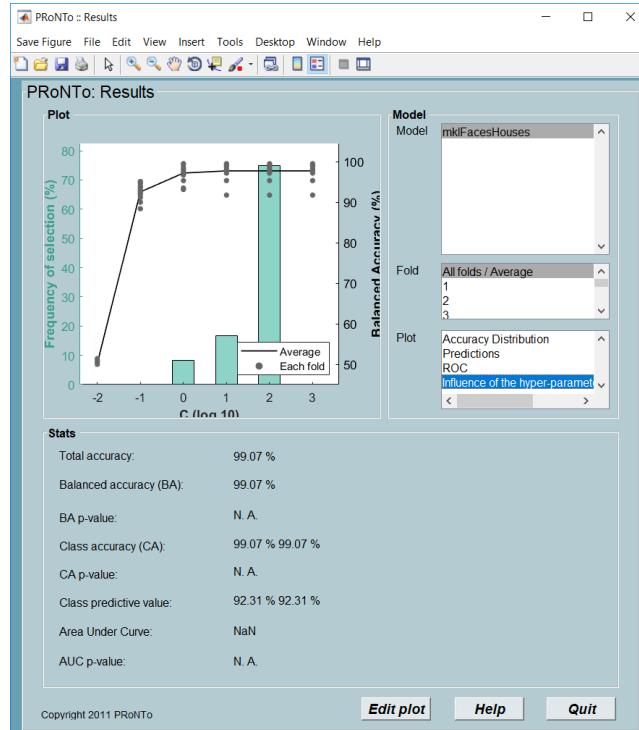


Figure 15.4: 'Display results' GUI.

- In the 'Results' window, one can select the different plots in the 'Plots' list.
- In the 'Results' window, one can select specific folds in the 'Fold' list and also check their performance with

different plots in the ‘Plot’ list. Please note that there is a new plot on the list, this displays information about the hyper-parameter optimization, for more information, please refer to Chapter 5.

- Also keep in mind that if you have not run permutation tests, all p-values would be ‘N.A.’.

15.1.8 Compute weights

- In PRoNTo’s main window, click on ‘Compute weights’ and a new window will open, ‘Compute weights’ (see Figure 13.24 in Chapter 13);
- Select the ‘PRT.mat’ file;
- Select the model from the list to ‘Models computed in PRT’, ‘mklFaces Houses’ model;
- Check the tick box option ‘Compute average/kernel weight per region’;
- Click on the ‘Compute weights’ button. Computations will be displayed on the MATLAB prompt.

15.1.9 Display weights

- In PRoNTo’s main window, click on ‘Display weights’ and select the ‘PRT.mat’ file. This will open the primary ‘Display weights’ window. By clicking on ‘Model’, mklFaces Houses, an image will appear in the ‘Weights map’ box; and to show the ‘Anatomical img’ you have to load an anatomical image for reference. A template image can be found in the SPM’s canonical folder ‘single_subj_T1’. The final result window will look similar to that shown in Figure 15.5.
- Since the machine used in the example was MKL with a kernel calculated for each brain region, it is possible to see the contributions of each region (i.e. the kernel’s weights). The labels for the regions can be found in the same folder where the atlas is located (PRoNTo/atlas). For more information, please refer to Chapter 7.

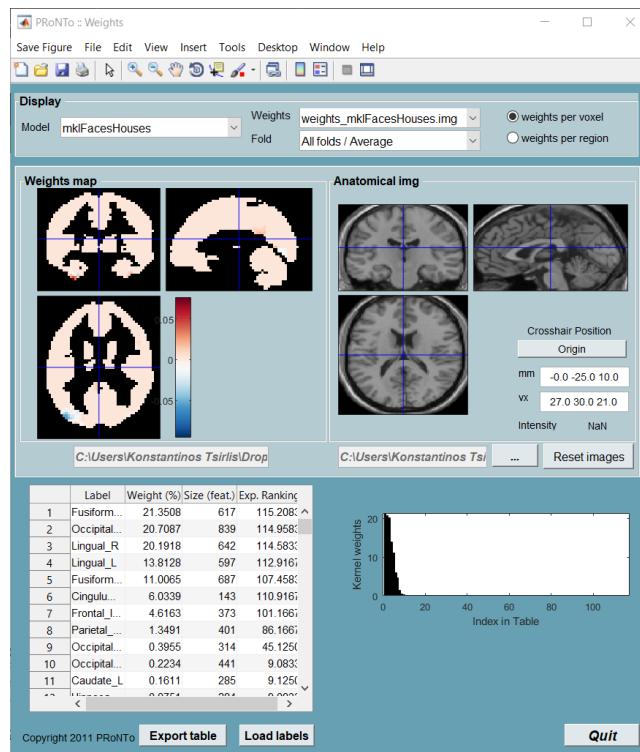


Figure 15.5: ‘Display weights’ GUI.

15.2 Batch analysis

This tutorial will now show how to analyse the same data but using the `matlabbatch` system.

Once again, to analyse the data, create a new directory in which to save the results of the analysis, saved as ‘PRT.mat’. On the main interface of PRoNTo click on the ‘Batch’ button to open the ‘`matlabbatch`’. Alternatively, type ‘prt_batch’ in the MATLAB prompt.

15.2.1 Data & Design

- Click on ‘Data & Design’ in the PRoNTo menu and a window like the one in Figure 13.28 will appear.
- In the ‘Directory’ field, select a directory where the ‘PRT.mat’ file will be saved. There are three ways of editing all fields in `matlabbatch`: (i) by using the right mouse button and clicking on the current option, (ii) clicking on current button in the window or (iii) by double clicking;
- In the ‘Groups’ field:
 - Add one group;
 - In the field ‘Name’, provide a name without spaces to that group, e.g. ‘G1’;
 - In the field ‘Select by’, select the ‘Subjects’ option and add one subject;
 - Add one modality for this subject and provide a name, e.g. ‘fMRI’; choose the appropriate data format (here nifti); define the interscan interval of 2.5 seconds; and in the field ‘Files’, select all the image files available in the fMRI directory of the Haxby dataset;
 - In the ‘Data & Design’ field, since our data are in nifti format, choose the ‘Load SPM.mat’ option. This file is available with the Haxby dataset on PRoNTo’s website⁴ inside the folder Haxby_dataset/design/. The reader is referred to Chapter 13 in case there is no ‘SPM.mat’ file available.
 - The reader is referred to the tutorial of chapter 17 in case the data are not in nifti format.
- In the ‘Masks’ field, add a new modality and provide the same modality name, ‘fMRI’, choose the appropriate data format and finally select the ‘whole_brain’ mask available in the masks directory of the Haxby dataset. The name of the modality here has to be exactly the same as in ‘Modalities’, otherwise it will not work;
- Leave the ‘HRF overlap’ and the ‘HRF delay’ fields as default;
- In the ‘Review’ field, select ‘Yes’ if you would like to review your data and design in a separate window. Otherwise, leave as it is, i.e. ‘No’. Keep in mind that the procedure pauses while you review the data and that you have to close the ‘Review’ window for the procedure to continue.

The final configuration should look similar to the Figure 13.30 in Chapter 13.

15.2.2 Feature set / Kernel

- Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Data & Design’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved. The window of Figure 13.32 in Chapter 13 is called to establish a dependency connection with the previous ‘Data & design’ module.
- Provide a name to the ‘Feature/kernel’ set, e.g. ‘HaxbyFeatures’;
- Select the ‘Nifti’ option for a data format, add one modality and select the modality name with the ‘Dependency’ button⁵(Data & Design:Mod#1 name);
 - In the ‘Samples/Conditions’ field , select the ‘All samples’ option;

⁴<http://www.mlnl.cs.ucl.ac.uk/pronto/prtdata.html>

⁵Or type it in manually, ‘fMRI’, but the name needs to be *exactly* the same as the one specified in the ‘Data & Design’ module.

- In the ‘Voxels to include’ field, select ‘All voxels’ option;
- In the ‘Detrend’ field, select ‘Polynomial detrend’ option with order 1;
- In the ‘Scale input scans’ field, select ‘No scaling’ option.
- In the ‘Use atlas to build ROI specific kernels’, select an atlas file AAL (named ‘aal_79x91x69’) available in the PRoNTo directory (PRoNTo/atlas/).

Figure 15.6 shows the final configuration of the `matlabbatch` ‘Feature set / Kernel’ module.

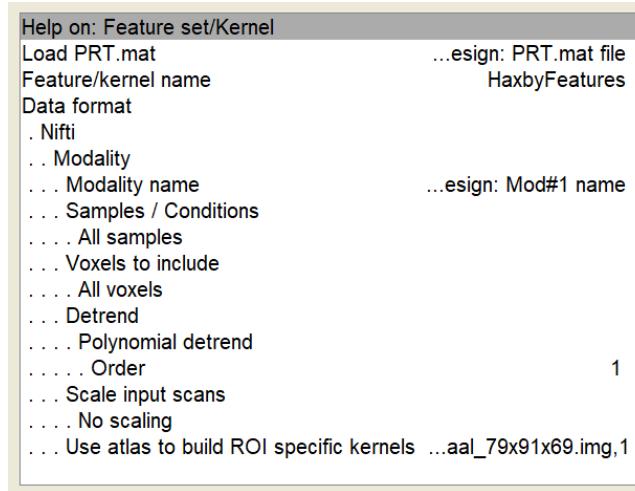


Figure 15.6: Final configuration of the `matlabbatch` ‘Feature set / Kernel’ module.

15.2.3 Model: Specify new

- Click on the ‘Model: Specify new’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Feature set / Kernel’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved;
- Provide a name to the model, e.g. ‘mklFacesHouses’;
- In the ‘Feature sets’ field, select the feature set name with the ‘Dependency’ button⁶;
- Select the ‘Classification’ model type:
 - Add 2 new classes;
 - For Class (1) write ‘Faces’ on the name field and add one group. Select the group name from the ‘Data & Design’ module (‘Data & Design:Group#1 name’) with the ‘Dependency’ button⁷. Similarly, for Class (2) write ‘Houses’ on the name field and add the group created in the ‘Data & Design’ module, ‘G1’;
 - In the ‘Subjects’ field, type ‘1’ (only subject 1 is selected);
 - In the ‘Conditions / Scans’ field, select the ‘Specify Conditions’ option and add a new condition. Provide a name for this condition, i.e. for Class (1) ‘Faces’ and for Class (2) ‘Houses’. Note that this name needs to be spelled exactly as specified in the ‘Data & Design’ module: if you simply loaded an ‘SPM.mat’ file for the design, you *must* know the names of the conditions;
- Leave ‘Subsample examples based on class definition’ as it is, i.e. ‘No’.

⁶or write it *exactly* as previously defined in the ‘Feature set / Kernel’ module (option ‘Feature set/Kernel: Feature/kernel name’), here ‘HaxbyFeatures’.

⁷Or write it *exactly*, as previously defined in the Data & Design’ module, here ‘G1’

- In the ‘Machine’ field:
 - Select the ‘Kernel machine’ and the ‘L1 Multi-Kernel Learning’ option;
 - In the ‘Machine optimization and parameters’ field, select the ‘Optimize hyper-parameter’ option;
 - Here we need a range of values only for the ‘Regularization hyper-parameter’ optimization, which you can leave with the defaults (i.e. [0.01, 0.1, 1, 10, 100, 1000]);
 - In the ‘Cross validation type for hyper-parameter optimization’ (internal loop) field, select the ‘k-folds CV on blocks’ option and on the field ‘k’ input the value 4;
- In the ‘Cross validation type’ (external loop) field, select ‘Leave one block out’ option;
- Leave the ‘Include all scans’ field as it is, i.e. ‘No’;
- In the ‘Data operations’ field:
 - Leave the ‘Mean centre features’ field as it is, i.e. ‘Yes’;
 - Click on the ‘Other Operations’ field and select the option ‘Select Operations’, then add a new operation and select the ‘Normalize samples’ option;

Figure 15.7 shows the final configuration of the `matlabbatch` ‘Model: Specify new’ module.

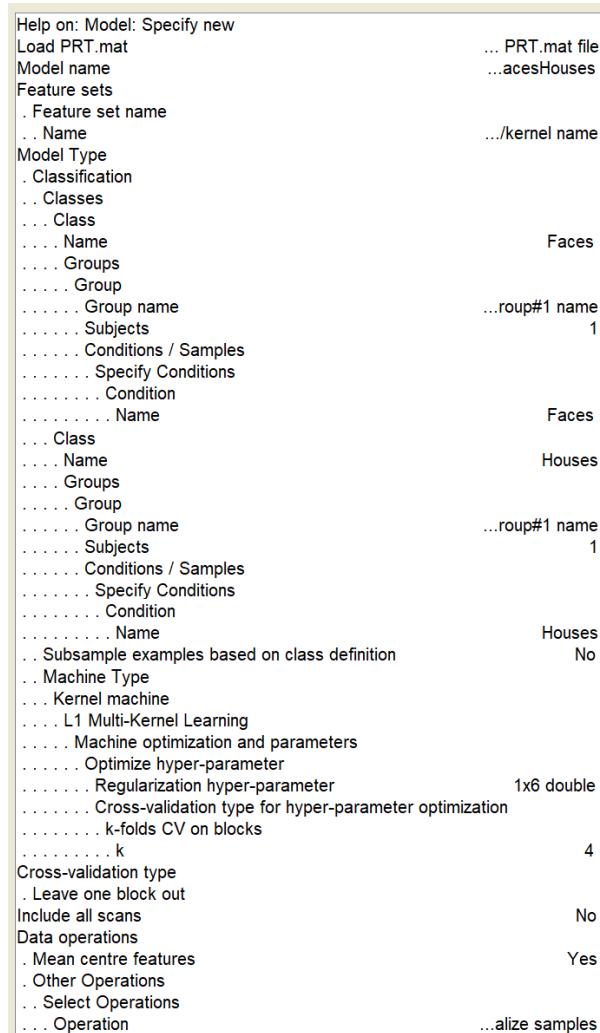


Figure 15.7: Final configuration of the `matlabbatch` ‘Model: Specify new’ module.

15.2.4 Model: Run

- Click on the ‘Model: Run’ option on PRoNTo’s `matlabbatch` menu.
- With the ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Specify model’ step;
- Select the model name from the ‘Model: Specify new’ module with the ‘Dependency’ button, or write it *exactly*, as previously defined in the ‘Model: Specify new’ module, here ‘mklFaces Houses’;
- Finally leave ‘Do permutation test?’ as it is.

Figure 15.8 shows the final configuration of the `matlabbatch` ‘Model: Run’ module.

15.2.5 Compute weights (optional step)

- Click on the ‘Compute weights’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Model: Run’ step;
- Select the model name from the ‘Model: Specify new’ module with the ‘Dependency’ button;
- It’s optional to define a name for the image;
- In the ‘Build weight images per ROI’, load the atlas that was used for building the feature set, which can be found in PRoNTo/atlas/;
- Leave the ‘Build weights images for permutations’ field as it is, i.e. ‘No’;

Finally, save the batch (e.g. as `batch_run.all.m`) and click on the ‘Run Batch’ option, in the ‘File menu’. The batch file created can then be opened and edited for further analyses. The results will be the same as those obtained using the GUI (see Section 15.1.7 of this tutorial).

Figure 15.9 shows the final configuration of the `matlabbatch` ‘Compute weights’ module.

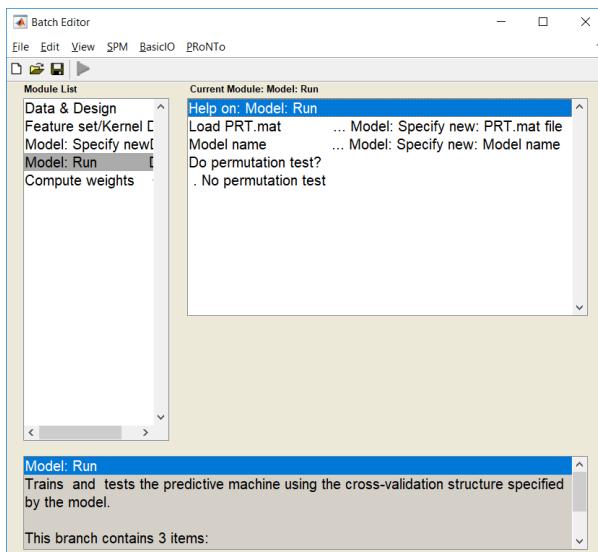


Figure 15.8: Final configuration of the `matlabbatch` ‘Model: Run’ module.

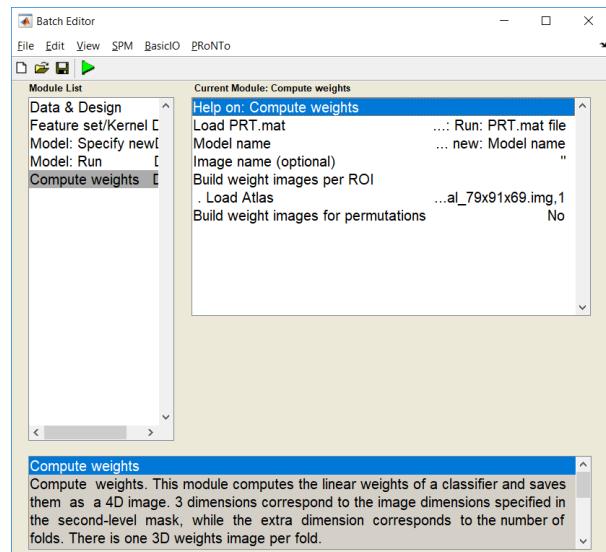


Figure 15.9: Final configuration of the `matlabbatch` ‘Compute weights’ module.

Chapter 16

Removing confounds: a classification example

Contents

16.1	Introduction	155
16.2	GUI analysis	156
16.2.1	Data & Design	156
16.2.2	Prepare feature set	157
16.2.3	Model: Specify new	158
16.2.4	Model: Specify from	159
16.2.5	Model: Run	159
16.2.6	Display results	160
16.3	Batch analysis	160
16.3.1	Data & Design	160
16.3.2	Feature set / Kernel	161
16.3.3	Model: Specify new	162
16.3.4	Model: Run	163
16.3.5	Compute weights	163
16.3.6	Display weights	164
16.4	Effects of removing covariates	164

16.1 Introduction

This chapter describes the steps necessary to remove confounding effects using PRoNTo. Age, gender or scanner/site (in case of a multi-site study) are examples of potential confounds (or covariates) that can be removed from the pattern regression analysis. Please see section 2.5.1 for important considerations on covariates.

The dataset used in this chapter can be found on the OASIS's website <http://www.oasis-brains.org> (data set 1). In this example, 100 subjects were selected from the OASIS dataset with 50 demented patients and 50 nondemented subjects. Age and gender are considered confounds. Gender is a categorical variable, so it was represented using one-hot encoding (i.e. male was represented as [0 1] and female was represented as [1 0]). The preprocessed data can be found on PRoNTo's website.

We will use PRoNTo to classify subjects with dementia from healthy subjects based on structural MRI scans. We will use Multiple Kernel Learning to classify the subjects based on whole brain images, so the reader is advised to complete the tutorial in Chapter 13 and Chapter 15 before moving on.

As in previous Chapters, we will start analyzing the data with PRoNTo's GUI and then repeat the analysis using the `matlabbatch` system. Please create a folder on your computer to store the results and type `prt` in MATLAB.

16.2 GUI analysis

16.2.1 Data & Design

- In PRoNTo's main window, click on 'Data & Design' and a new window will open, 'Data and design'. Like in previous chapters, browse the directory in which to save the PRT structure (saved as 'PRT.mat').
- In the panel Groups, click on 'Add' and provide a name to the group, e.g. 'Dem'. We will first add all the information about the demented subjects.
- All the images in the dataset correspond to different subjects; therefore, click on the Samples tick box. This will lock the Subjects/Samples field, please see Chapter 2.4 of the manual for more information on this option.
- In the 'Modalities' panel, click on 'Add' and provide a name to the modality, e.g. 'MRI_GM', where we used gray matter images.
- In the 'Data format' choose the appropriate format for your data (here nifti); For other data formats the reader is referred to the tutorial of chapter 17.
- Load all the image files belonging to the first group. You can select all the files by using the right mouse button and clicking on the option 'Select All', or clicking on the first subject then Shift-click on the last. Please ensure that the order of the images in the selection is as expected (SPM file selector sorts the files based on their names, which can lead to 'Scan10' being selected before 'Scan2'). When all the images are selected, click on the 'Done' button.
- You can see that the 'Design' section is unavailable. Since each subject has only 1 sample/scan, there is no design.
- Leave the 'Regression targets' as it is, i.e 'No targets'.
- In the 'Covariates' field, type the covariates you want to regress out. Alternatively, you can save all the covariates in a matrix named R and input the full path of this matrix to 'Covariates'. Each covariate should be stored in one (continuous confound) or multiple (one-hot encoded categorical confound) columns, hence if there is only one continuous covariate, R is a column vector; if there are multiple and/or categorical confounds, R will be a matrix containing multiple columns. In this example, we input the full path of a matrix where gender and age are the chosen confounding factors (Figure 16.1). The file containing matrix R (Dem_cov.mat) can be found together with the pre-processed OASIS data on PRoNTo's website.

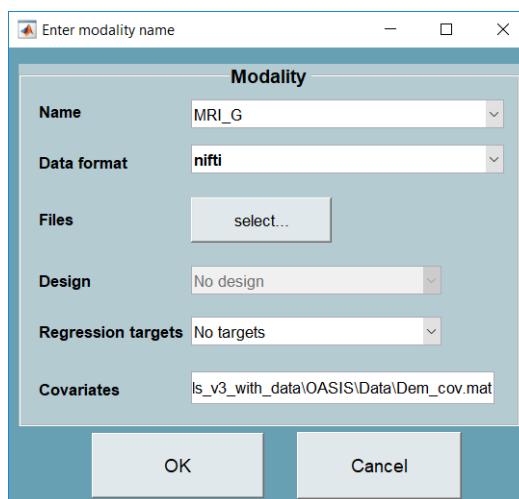


Figure 16.1: 'Specify modality' GUI allows one to enter the covariates to be regressed out.

- In the 'Masks' field, on the bottom left of the 'Data and design' window, select the 'mask_GM' mask (found on OASIS/Data) for the modality specified.

- Finally, repeat the previous steps for a new group, the non-demented group, 'NonDem', including the corresponding images and covariates, and using the same modality and mask as for the 'Dem' group. Click on the 'Save' button to create the 'PRT.mat' file. Notice that the only difference with the tutorials of the previous Chapters is that now we have two groups instead of one (Figure 16.2. The 'Data and design' window should look similar to Figure 16.3.



Figure 16.2: Review data in the 'Data and design' GUI.

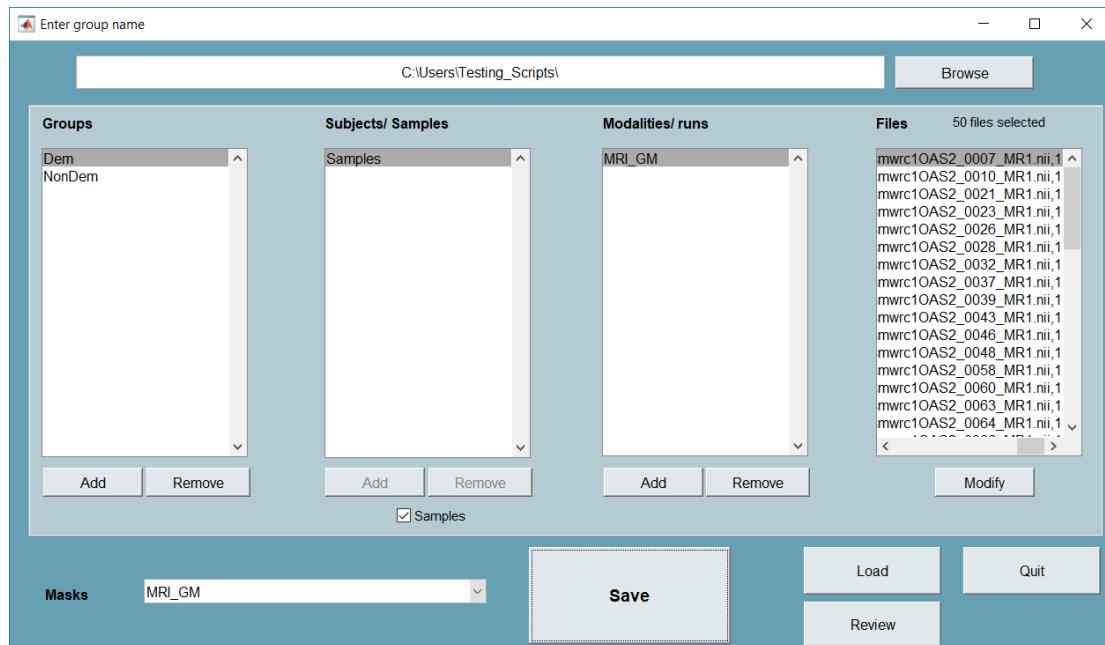


Figure 16.3: 'Data and design' GUI final configuration.

If no errors are shown in the MATLAB command, leave the 'Data and design' window by clicking 'Quit'.

16.2.2 Prepare feature set

- In PRONTo's main window, click on 'Prepare feature set' and a new window will open, 'Prepare feature set'.
- Select the 'PRT.mat' file previously created in the 'Data & Design' step and another window will open, 'Specify modality to include'. Select the 'Build one kernel per region' tick box and load the 'AAL' atlas (named 'aal_79x91x69') available in the PRONTo directory (PRONTo/atlas/). Leave all the other default options and click 'Done'.
- This will bring you back to the 'Prepare feature set' window. Provide a name for the feature set, e.g. 'OASIS_ConfEffects_withCov'.
- Click 'Build kernel/data matrix' to build the feature set and kernel.

16.2.3 Model: Specify new

- In PRoNTo's main window, click on 'Specify new' and a new window will open (see Figure 13.14 in Chapter 13).
- Select the 'PRT.mat' file and provide a name to the model, e.g. 'mkl_OASIS_ConfEffects_withCov'.
- Select one of the 'Feature Set' previously defined. In this case, there is only one: 'OASIS_ConfEffects_withCov'.
- Leave the option 'Use kernels' tick box as it is, i.e. 'Yes'.
- Select the 'Classification' model type and click on the 'Define classes' button. A new window will open, 'Specify classes', to define the number of classes and a name for each class. We will define 2 classes. For 'Class 1' select group 'Dem', and all subjects in this first group and, similarly, for 'Class 2' select group 'NonDem' and all the subjects in this group. The class names can be any names the user prefers (in alphanumeric characters). Here we simply use the same names as the group names. Leave the 'Subsample according to smallest class' as it is for now. Once you have appropriately specified everything, click 'Done'. The final configuration of the 'Specify classes' window should look similar to Figure 16.4

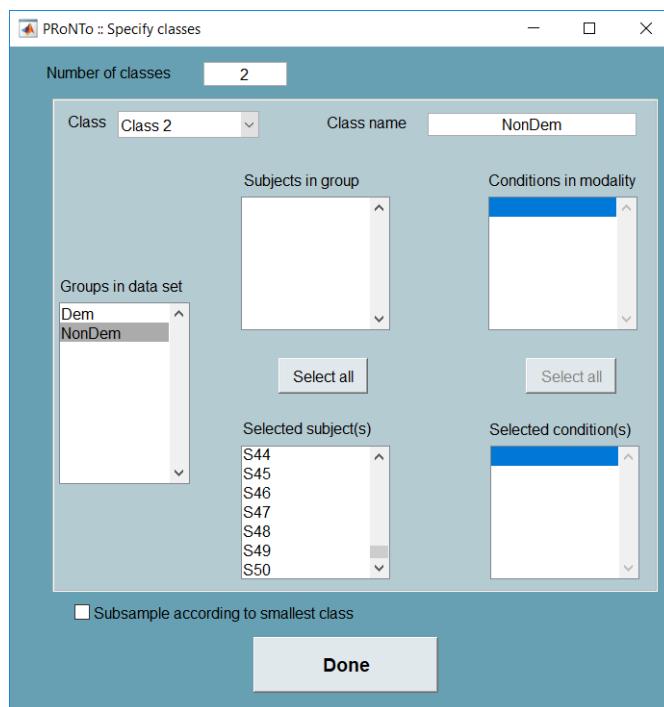


Figure 16.4: 'Specify classes' GUI.

- Select the 'L1 Multi-Kernel Learning' option, in the 'Machine' field.
- Click the 'Optimize hyper-parameter' tick box and leave the range unspecified, i.e. use the default hyper-parameter range. Choose 'k-folds CV on Subject per Class' for the 'Cross-Validation Scheme' (internal loop), and input 5 in the pop up window defining the number of folds.
- Similar to the inner loop, select the 'k-folds CV on Subject per Class' cross-validation scheme for the external loop, where k=10.
- In the 'Data operations' box, select the 'Mean centre features using training data', 'Normalize samples' and 'Regress out covariates (subject level)' option. The 'Regress out covariates (subject level)' option corresponds to the "regressing out" effect of confounds from the data. Note that the order of the operations is important and can affect the results. However, when selected, the 'Regress out covariates' option will be performed first (even if not selected as first). Then, the 'Specify model' window should look similar to (Figure 16.5). Click on the 'Specify and run model' button.

- Remember that if you want to check the statistical significance of the results you should specify your models here and run them using the ‘Model: Run’ module where you will have the option of choosing or not to run permutation tests.

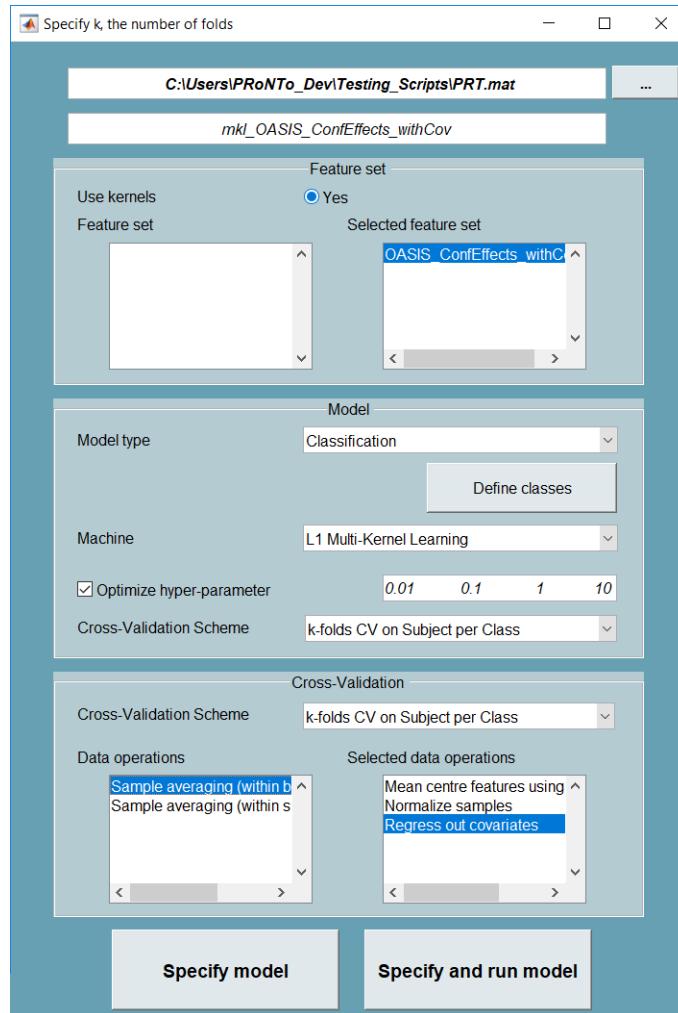


Figure 16.5: ‘Specify model’ GUI final configuration.

16.2.4 Model: Specify from

- The reader is referred to Chapter 13 and chapter 4 for more details regarding the ‘Model: Specify from’ module.

16.2.5 Model: Run

The reasons for using the ‘Model: Run’ module and not just click ‘Specify and run model’ in the ‘Model: Specify new/from’ module are the following:

- One might want to specify one or more models now but run it/them later.
- One might want to specify more than one models first and run them all together at once afterwards.
- One might want to run permutation tests in order to check the statistical significance of the results. The reader is referred to Chapter 13 and section 5.4 for more details.

16.2.6 Display results

- In PRoNTo's main window, click on 'Display results' and select the 'PRT.mat' file. This will open the main results window (Figure 16.6).

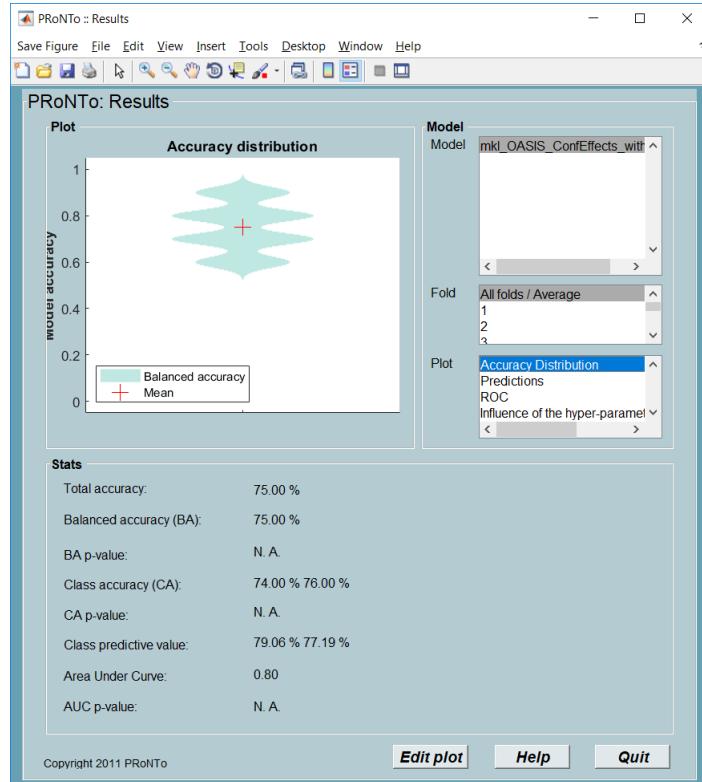


Figure 16.6: 'Results' GUI.

- In the 'Results' window, we have different performance measures, as well as different plots; both the average as well as for each fold. One can select different plots in the 'Plots' list. For further information regarding the 'Display results' module the reader is referred to chapter 5.

16.3 Batch analysis

In this section, the previous experiment will be repeated using the '`matlabbatch`' system. The reader is advised to complete the tutorial in Section 13.2 before continuing, since the explanation of each step will be less detailed.

Once again, to analyse the data, create a new directory to save the results of the analysis. On the main interface of PRoNTo click on the 'Batch' button to open the '`matlabbatch`'. Alternatively, type '`pvt_batch`' in the MATLAB prompt.

16.3.1 Data & Design

- Click on 'Data & Design' in the PRoNTo menu.
- In the 'Directory' field, select a directory where the 'PRT.mat' file will be saved.
- In the 'Groups' field:
 - Add two groups.
 - In the field 'Name', provide a name without spaces to both groups, e.g. 'Dem' and 'NonDem'.
 - In the field 'Select by', select the 'Samples' option and add a new modality. For more information on the Samples option please refer to chapter 2.

- For both groups add one modality and provide the same name, e.g. ‘MRI_GM’.
 - For both groups select ‘nifti’ for our data format.
 - In the field ‘Files’, select the proper images for each of the groups.
 - For both groups leave the ‘Regression targets (subject)’ as it is, i.e. ‘No targets’.
 - Specify as ‘Covariates’ the file with the confounds you want to remove. Remember that this file should contain a variable ‘R’ with a matrix of covariates.
- In the ‘Masks’ field, add a new modality, provide the same modality name, ‘MRI_GM’; choose the appropriate data format and select the ‘mask_GM’ mask available in the Oasis dataset. The name of the modality here has to be exactly the same as in ‘Modalities’.
 - Leave the ‘HRF overlap’ and the ‘HRF delay’ fields as default.
 - In the ‘Review’ field, select ‘Yes’ if you would like to review your data and design in a separate window. Otherwise, leave as it is, i.e. ‘No’. Note: if ‘Yes’ is selected, users will need to close the Review window after Running the batch to allow the execution of following modules.

The final configuration of the `matlabbatch` ‘Data & Design’ module should look similar to Figure 16.7.

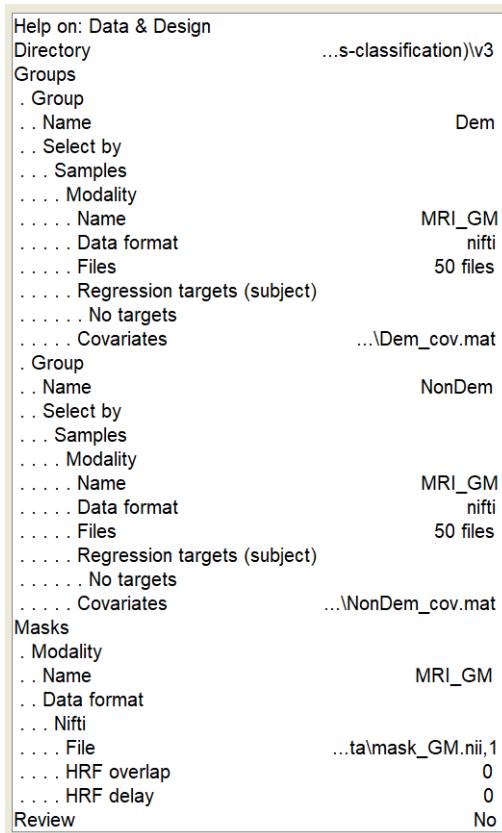


Figure 16.7: ‘Data & Design’ module in `matlabbatch`.

The batch can either be run directly or the following modules can be added to complete multiple steps of the analysis at once. We typically suggest users to save and run the ‘Data and Design’ step in a separate batch to avoid deleting/overwriting the PRT each time the batch is run.

16.3.2 Feature set / Kernel

- Click on the ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu.

- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Data & Design’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved. The window of Figure 13.32 in Chapter 13 is called to establish a dependency connection with the previous ‘Data & design’ module.
- Provide a name for the ‘Feature set/kernel’, e.g. ‘OASIS_ConfEffects_withCov’.
- Choose the appropriate data format, add one modality and select the modality name with the ‘Dependency’ button¹(Data & Design:Mod#1 name).
 - In the ‘Samples/Conditions’ field , select the ‘All samples’ option.
 - In the ‘Voxels to include’ field, select ‘All voxels’ option, this means we are not entering an additional second-level mask.
 - In the ‘Detrend’ field, select ‘None’.
 - In the ‘Scale input scans’ field, select the ‘No scaling’ option.
 - And finally in the ‘Use atlas to build ROI specific kernels’ field, load the AAL atlas as we did in the GUI section. After all these steps, the batch editor should look similar to the one in Figure 16.8.

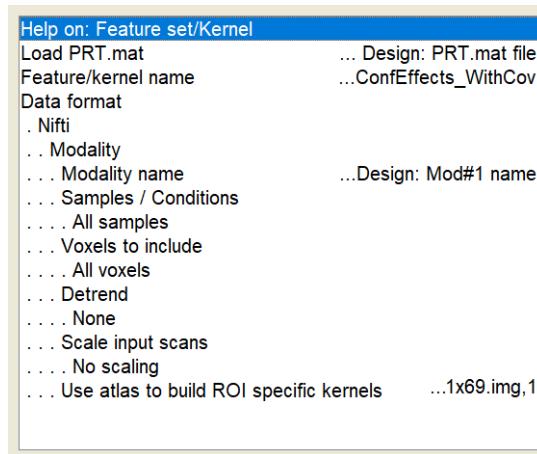


Figure 16.8: Final configuration of the `matlabbatch` ‘Feature set / Kernel’ module.

16.3.3 Model: Specify new

- Click on the ‘Model: Specify new’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Feature set / Kernel’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved.
- Provide a name for the model, e.g. ‘mkl_OASIS_ConfEffects_withCov’.
- In the ‘Feature sets’ field, select the feature set name with the ‘Dependency’ button².
- Select the ‘Classification’ model type:
 - Add 2 new classes.
 - For Class (1), specify a name by writing ‘Dem’ or using Dependency to choose the first group’s name defined in ‘Data & Design’ and add one group. The class name doesn’t have to be the same as the group name, this example uses this only for simplicity. After this step, select the group name from the ‘Data & Design’ module (‘Data & Design:Group#1 name’) with the ‘Dependency’ button³.

¹Or type it in manually, ‘MRI-GM’, but the name needs to be *exactly* the same as the one specified in the ‘Data & Design’ module.

²or write it *exactly* as previously defined in the ‘Feature set / Kernel’ module (option ‘Feature set/Kernel: Feature/kernel name’), here ‘OASIS_ConfEffects_withCov’.

³Or write it *exactly*, as previously defined in the Data & Design’ module, here ‘Dem’

- In the ‘Subjects’ field of Group 1, type ‘1:50’. This will tell the program to use all the 50 subjects, i.e. from subject 1 to scan 50. In Conditions/Samples, select ‘All Samples’.
- Similar to Class (1), write ‘NonDem’ on the name field for Class (2), or use Dependency to choose the second group’s name. Then fill the group name using Dependency, i.e. the second group’s name created in the ‘Data & Design’ module, ‘NonDem’. In the ‘Subjects’ field of Group 2, type ‘1:50’. In Conditions/Scans, select ‘All scans’.
- Finally leave ‘Subsample example based on class definition’ as it is, i.e. ‘No’.
- In the ‘Machine’ field:
 - Select ‘Kernel machine’ and the ‘L1 Multi-Kernel Learning’ option.
 - In the ‘Machine optimization and parameters’ select the ‘Optimize hyper-parameter’ and leave it to the default values (i.e. ‘ $10^{-2:3}$ ’).
 - In the ‘Cross validation type for hyper-parameter optimization’ field, choose ‘k-folds CV on subjects per group’, and input ‘5’ in the field ‘k’.
- In the ‘Cross-validation type’ field, choose ‘k-folds CV on subjects per group’, and input ‘10’ in the field ‘k’.
- Leave the ‘Include all scans’ field as it is, i.e. ‘No’.
- In the ‘Data operations’ field:
 - Leave the ‘Mean centre features’ field as it is, i.e. ‘Yes’.
 - Click on the ‘Other Operations’ button, choose ‘Select operations’, ‘New Operation’ and select the ‘Normalize samples’.
 - Click ‘Select operations’, ‘New Operation’, and select ‘Regress out covariates (subject level)’ option from the list.

After these steps, the batch editor should look similar to the one in Figure 16.9.

16.3.4 Model: Run

- Click on the ‘Model: Run’ option in PRoNTo’s `matlabbatch` menu.
- With the ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file updated in the previous ‘Specify model’ step.
- Select the model name from the ‘Specify model’ module with the ‘Dependency’ button⁴.
- In the field ‘Do permutation test?’, leave as it is, i.e. ‘No permutation test’.

16.3.5 Compute weights

- Click on the ‘Compute weights’ option in PRoNTo’s `matlabbatch` menu.
- With the ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file updated in the previous ‘Run model’ step.
- In the ‘Model name’ field, click on the ‘Dependency’ button to associate the model name from the ‘Specify model’ module.
- In ‘Build weight images per ROI’, users can choose ‘Load atlas’ to build a weight image showing where the value in each region corresponds to the region/kernel’s weight. This example uses MKL, where the AAL atlas was selected during the Feature set module, hence the atlas will be automatically loaded here if ‘Load atlas’ is selected.
- Leave the other fields as default.

We are now ready to run the batch: the arrow on top of the `matlabbatch` should be green and can be clicked. If the arrow is gray, please check for missing inputs in the different modules (displayed with an ‘< --X’).

⁴or write it *exactly* as previously defined in the ‘Specify model’ module, here ‘mkl.OASIS_ConfEffects_withCov’

Help on: Model: Specify new	
Load PRT.mat	...ernel: PRT.mat file
Model name	...nfEffects_withCov
Feature sets	
. Feature set name	
. . Name	...ature/kernel name
Model Type	
. Classification	
. . Classes	
. . . Class	
. . . . Name	..ign: Group#1 name
. . . . Groups	
. Group	
. Group name	..ign: Group#1 name
. Subjects	50x1 double
. Conditions / Samples	
. All samples	
. Class	
. Name	..ign: Group#2 name
. Groups	
. Group	
. Group name	..ign: Group#2 name
. Subjects	50x1 double
. Conditions / Samples	
. All samples	
. . Subsample examples based on class definition	No
. . Machine Type	
. . . Kernel machine	
. . . . L1 Multi-Kernel Learning	
. Machine optimization and parameters	
. Optimize hyper-parameter	
. Regularization hyper-parameter	1x6 double
. Cross-validation type for hyper-parameter optimization	
. k-folds CV on subjects per group	
. k	5
Cross-validation type	
. . k-folds CV on subjects per group	
. . . k	10
Include all scans	No
Data operations	
. . Mean centre features	Yes
. . Other Operations	
. . Select Operations	
. . . Operation	Normalize samples
. . . Operation	...ress out covariates

Figure 16.9: Final configuration of the `matlabbatch` ‘Model: Specify new’ module.

16.3.6 Display weights

- In PRoNTo’s main window, click on ‘Display weights’ and select the ‘PRT.mat’ file. This will open the ‘PRoNTo::Weights’ window.
- In the Display panel, click the model name in the ‘Model’ field, ‘mkl_OASIS_ConfEffects_withCov’. An image will appear in the ‘Weights map’ panel. To show the ‘Anatomical img’ you have to load an anatomical image for reference. A template image can be found in SPM’s canonical folder (‘single_subj_T1’ file). Select ‘weights per region’ to display the kernel/ROI contributions (see Chapter 6 for details on this image). Select ‘weights per voxel’ to display the voxels contributions. In the table below the weight map, the contribution of ROI to the model and its corresponding rank are listed. A bar graph shows the contribution of each ROI to the model (sorted), displaying how ‘sparse’ the model is. The final window will look similar to the one shown in the Figure 16.10.

16.4 Effects of removing covariates

This section shows the differences between results with and without removing covariates.

In order to check the effects of regressing out covariates one has to rerun the whole procedure keeping

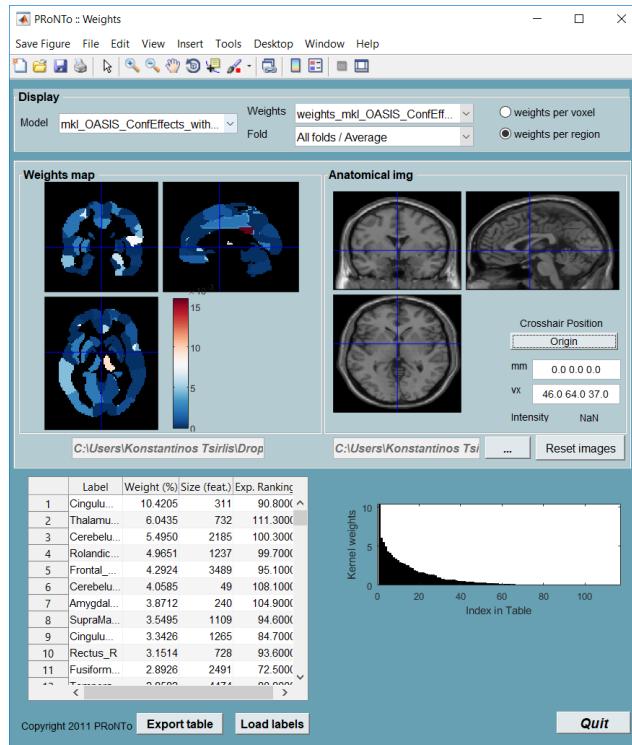


Figure 16.10: ‘Display weights’ module where a weight map per region is shown.

everything the same, except that now we have to not choose ‘Regress out covariates’ in the ‘Data operations’ when we specify our new model. This can be more easily done using the module ‘Model: Specify from’.

Figure 16.11 shows the accuracy of the same MKL model but without removing confounds: the average balanced accuracy decreased from 75% (see 16.6) to 64%.

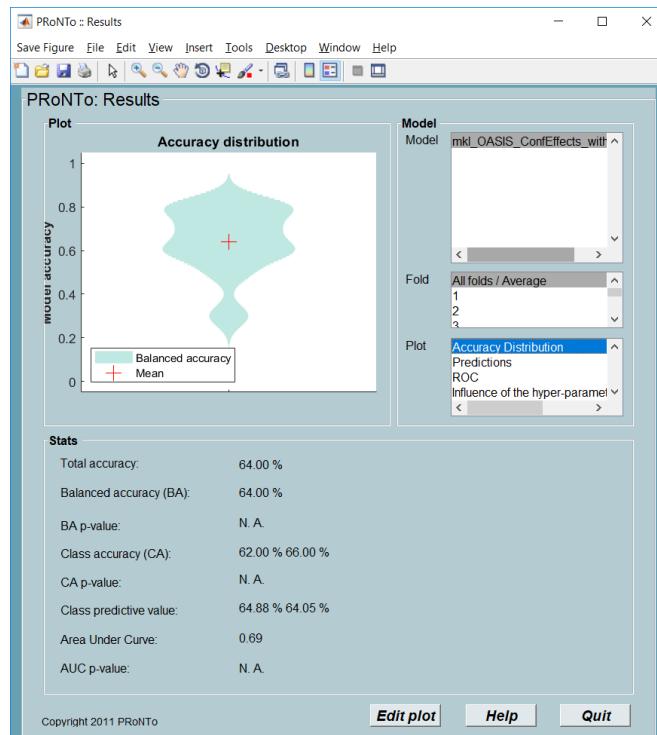


Figure 16.11: ‘Display results’ GUI with the accuracies of MKL without removing covariates.

Chapter 17

Multi-modal face recognition example

Contents

17.1	GUI analysis	168
17.1.1	Data & design	168
17.1.2	Prepare feature set	172
17.1.3	Model: Specify new	175
17.1.4	Model: Specify from	175
17.1.5	Model: Run	177
17.1.6	Display results	178
17.1.7	Compute weights	178
17.1.8	Display weights	178
17.1.9	Using an atlas with .mat	180
17.2	Batch analysis	181
17.2.1	Data & Design	182
17.2.2	Feature set / Kernel	185
17.2.3	Model: Specify new	186
17.2.4	Model: Run	188
17.2.5	Compute weights	190
17.2.6	Display results & weights	190
17.2.7	Using an atlas with .mat	190

The dataset used in this tutorial consists of is a visual experiment where pictures of famous faces (Famous), unfamiliar faces (Unfamiliar) and scrambled faces (Scrambled) are presented to the subjects. In total 16 subject were recorded with fMRI, EEG and MEG. The data is available for research purposes from ftp://ftp.mrc-cbu.cam.ac.uk/personal/rik.henson/wakemandg_hensonrn/ and is fully detailed in the publication: *A multi-subject, multi-modal human neuroimaging dataset, Wakeman, D. G. and R. N. Henson, Scientific Data, vol. 2, 150001, 2015.*

A within-subject version of the dataset can be found, along with pre-processing batches, on the SPM website (<https://www.fil.ion.ucl.ac.uk/spm/data/mmfaces/>).

In this tutorial, the EEG and MEG data was used to derive 6 modalities:

- MEG: average trace for each channel, in the considered time window. There are 102 MEG channels. There is one file with the 3 traces per subject.
- Interpolated EEG: nifti images from spatial interpolation of average trace per condition on the scalp, for EEG. We thus obtain one image per condition and per subject.
- Interpolated MEG: nifti images from spatial interpolation of average trace per condition on the scalp, for MEG.

- Connectivity EEG: for each pair of channels, we computed the correlation between the average traces of a condition. The output is a 70*70 matrix per condition and per subject. Only the upper triangular part of the matrix was extracted.
- Connectivity MEG: for each pair of channels, we computed the correlation between the average traces of a condition. The output is a 102*102 matrix per condition and per subject. Only the upper triangular part of the matrix was extracted.

The EEG, MEG, interpolated EEG and MEG files were obtained from following the batches provided with the corresponding chapter in the SPM manual. The connectivity was built from the obtained EEG and MEG files, based on in-house scripts (in the Appendix).

17.1 GUI analysis

We will first analyse the data using PRoNTo's GUI and then repeat the analysis using the `matlabbatch` system. As this is a much more complex tutorial, readers are strongly advised to go through the other tutorials first to get a better understanding of PRoNTo's functionalities. Also, since this is the fourth tutorial, some parts that have been previously described will be only shortly described here. Therefore, the reader is advised to go through the previous tutorials first.

To start, create a new directory in which to save the results of the analysis, then start up MATLAB and type ‘prt’ or ‘pronto’ in the MATLAB prompt. This will open the main interface of PRoNTo.

17.1.1 Data & design

There are 16 subjects to input. For each subject, there are 6 modalities to specify. The easiest way to do this is to fully specify subject 1, then use the entered modalities and designs in further subjects.

- Choose a directory to save the resulting PRT.mat.
- Add a group (e.g. ‘G1’), one subject (‘S1’), and leave the ‘Samples’ tick box below the panel unchecked. See Chapter 2 of the manual for more information on this option.

EEG/MEG (MEEG data format)

- The first modality to be specified will be the EEG. Provide the name ‘EEG’ and choose its type as ‘MEEG’. The MEEG data formats usually consist of two files, one `.mat` and one `.dat`. In this case we have to go to `/Multimodal_face_dataset/data/S1/EEG` where we’ll find the EEG data of the first subject (‘S1’).
- When the file is selected, PRoNTo will automatically read the design from it. The design can be reviewed by clicking on ‘Events in file’.

In Figure 17.1 we see both the MEEG modality specification in mention, and the MEEG design review. There we see 3 conditions: Unfamiliar, Famous and Scrambled. The units are already set to seconds and the TR is the sampling interval (in seconds) in the file. In this example, the TR of 0.005 corresponds to a sampling rate of 200Hz.

- The same can be done for the MEG modality, where you will only have to change the name and choose the appropriate MEG files, instead of the EEG files that we chose before.

Interpolated EEG/MEG (nifti data format)

- The interpolated EEG and MEG consist of nifti files. This is similar to entering beta images from a GLM analysis in previous versions of PRoNTo. The only difference is that the type now needs to be specified; even though nifti is the default option.
- In the ‘Specify modality’ GUI, provide a name to the modality, e.g. ‘interpEEG’, choose ‘nifti’ and select the appropriate files, which are found in `/Multimodal_face_dataset/data/S1/EEG_interp`. As there are 3 conditions, we have 3 different files, one for each condition.

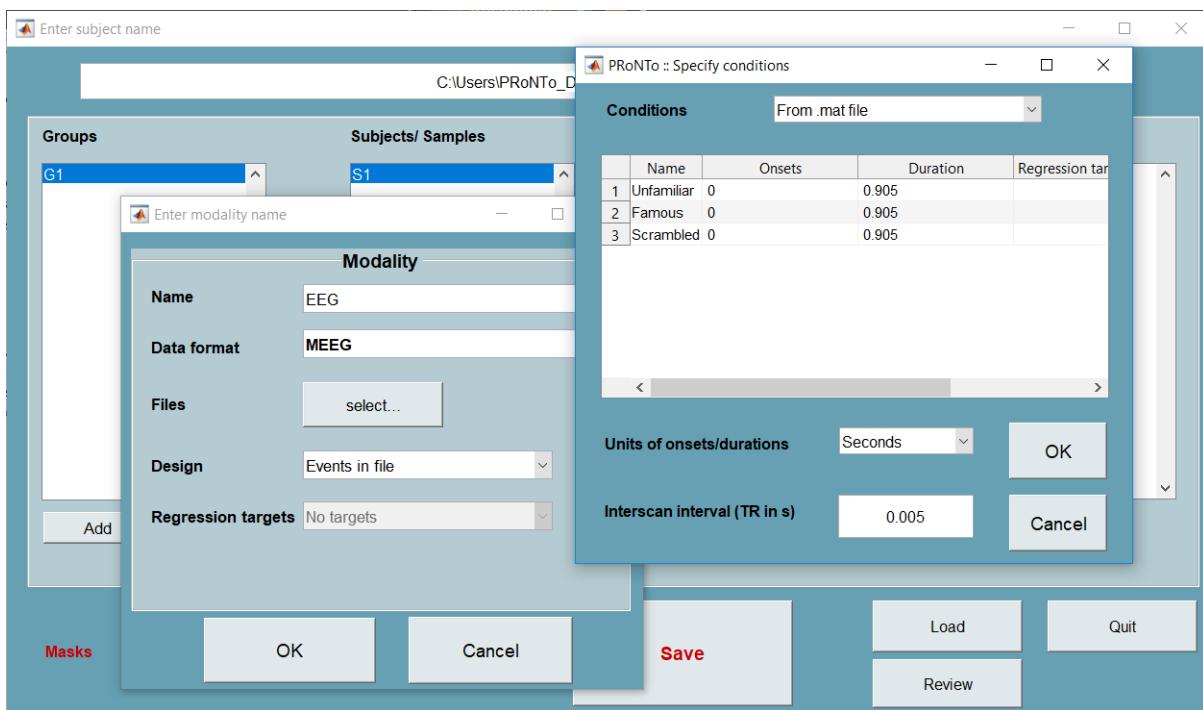


Figure 17.1: ‘Specify modality’ and ‘Specify conditions’ GUIs.

- **Very important note:** the order you choose the conditions (their files) should be the same across all modalities that will be used in a single model. This means that we should use the same order (Unfamiliar, Famous, Scrambled) that was input with the EEG and MEG files.
- Since there is no SPM.mat file for the design, we need to specify the design ourselves.
 - Create a new design by selecting the option ‘Specify design’.
 - If we do not have a .mat file with the design information, we need to specify the experimental design manually. Select the ‘Specify’ option in the ‘Specify conditions’ GUI. First you need to write how many conditions you have, which in this case is 3 (corresponding to the 3 image files). This will open another window that allows the user to write the names, onsets and durations of each condition.
 - As when using beta images, the onsets should correspond to the index of the image in the file selection, with the smallest index (i.e. first selected image) being 0. The duration should be set to 1 for all images, and the units should be set to ‘Scans’, with a TR of 1. Take great care when specifying conditions manually. The initial and final ‘Specify conditions’ window should look similar to the one in Figure 17.2.
- The same can be done for the interpolated MEG modality, where you will only have to change the name (“interpMEG”) and choose the appropriate MEG files, instead of the EEG files that we chose before.

Note: If we have many subjects and many conditions, this procedure can be tiresome and is quite prone to mistakes. Therefore what one could do is manually create a ‘Design.mat’ file and include the information regarding the names, onsets and durations inside that file. The name of the .mat file itself is irrelevant. What is important is to have 3 different cell arrays named ‘names’, ‘onsets’ and ‘durations’. The dimensions of all 3 cell arrays should be 1 x #conditions. So in our case all 3 cell arrays will have 1x3 dimensions for the 3 conditions (Unfamiliar, Famous, Scrambled). You can find one such example design file called ‘Design.mat’ in the */Multimodal_face_dataset/data*.

So for the interpolated MEG modality and from now on, do not input manually all the conditions in the ‘Specify conditions’ GUI. Instead, select the ‘From .mat file’ option and choose the ‘Design.mat’ file located in

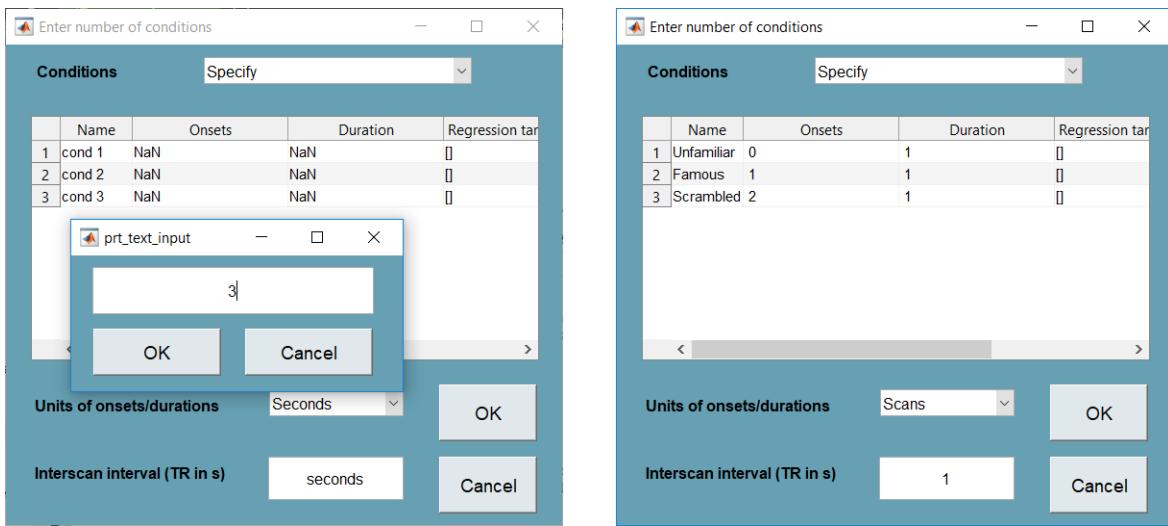


Figure 17.2: ‘Specify conditions’ GUI and final configuration.

the */Multimodal_face_dataset/data* folder. As you can see one still needs to choose the ‘Units of onsets/durations’ and the ‘Interscan Interval’ options. So again set the units to ‘Scans’, and the TR to 1.

Connectivity EEG/MEG matrices (.mat data format)

- Finally we have the connectivity derived EEG and MEG matrices, which can be entered in a similar way as for images. We set the name as ‘ConnEEG’.
- The ‘Data format’ needs to be set to ‘.mat’. The files in this case are ‘.mat’ files, and usually we will have 1 file for each condition, hence 3 for each subject. The order with which we input the files needs to be the same as before (Unfamiliar, Famous, Scrambled).
- The possible design options are then Specify design or No design. Specifying the design is identical to that of (nifti) images. As for images, PRoNTo expects one .mat file per trial/sample. **Important note:** When loading the file, it will read the first variable only (independently of the variables name). You hence need to ensure that the variable of interest is saved as the first one, or in a specific .mat. For ease of use, keep using the last way we mentioned to specify conditions (using the design file called ‘Design.mat’ in the */Multimodal_face_dataset/data*).
- The same can be done for the connectivity MEG modality, where you will only have to change the name (‘ConnMEG’) and choose the appropriate MEG files, instead of the EEG files that we chose before.

At this point we have finished inputting the first subject¹.

- You can now start the second subject. You first create a new subject, e.g. ‘S2’.
- The modalities of the second subject can then be entered by selecting the already specified modalities, where first you will need to select the appropriate data files of that subject, those of the second subject instead of the first one we selected before.
- You will also need to specify the design wherever you specified it manually. So for EEG and MEG, the design is read automatically. For nifti and .mat data formats, a new option will appear in the ‘Design’ menu of the ‘Specify modality’ GUI. The ‘Design’ menu will have the extra option ‘Replicate design from subject 1’, that is very convenient in this application. You can of course re-specify it manually again by locating the design file called ‘Design.mat’ in the */Multimodal_face_dataset/data*, as we did before. You will need to do this for all 16 subjects.

¹For the more experienced users we have included a small MATLAB script inside the */PRoNTo_main_folder/manual* called ‘16_script.m’ that automatically creates the rest of the subjects provided you have followed exactly the same steps as the ones mentioned in the tutorial, you have completely finished the first subject, you have specified the interpolated EEG and MEG masks and you have saved the struct. You need to load the ‘PRT.mat’, and be sure you’re in the same directory of your ‘PRT.mat’.

- Finally we need to specify the masks. While mask files definitely need to be specified for nifti images, for .mat and MEEG data formats, we indeed assume that the data only contains relevant features (therefore there is no need for a mask), as this can be easily performed during pre-processing. Please ensure that your data (.mat or MEEG) do not contain NaNs.

If the user wants to use a mask, it can still be done at the feature set level (for connectivity matrices, we'll see in another tutorial that another option could be to enter the full matrix and specify a 2nd level mask).

After you have specified everything, the final configuration should look like the one in figure 17.3, with 16 subjects, each one with its 6 modalities.

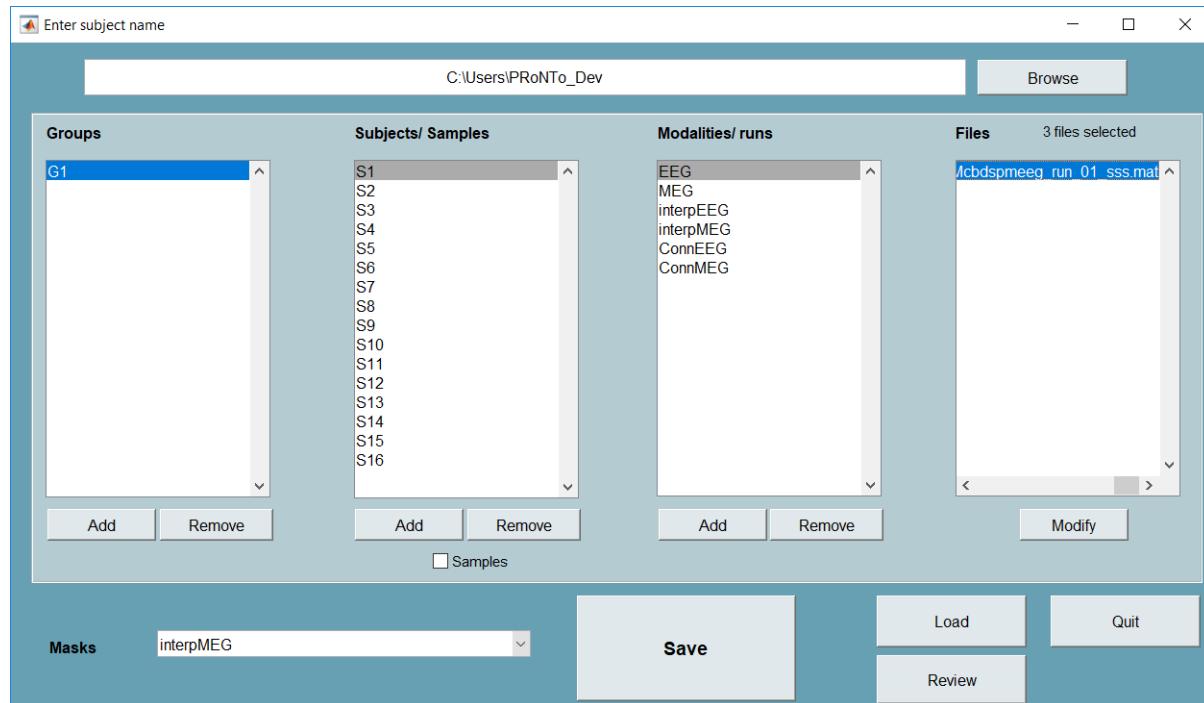


Figure 17.3: ‘Data & Design’ GUI final configuration.

Note regarding NaN values: A simple ‘Check Reg’ in SPM will show that the interpolated EEG (MEG) images have NaNs at different pixels, for different subjects. We hence need to create a common mask to discard those NaNs. This can easily be performed using SPM *ImCalc* batch (select all images from the EEG interpolated modality, select ‘read as matrix’, operation can be: `isnan(sum(X))`) and should be done for EEG and MEG separately.

Understanding the ‘PRT’ structure: At this point we have a ‘PRT.mat’ in the directory you previously chose. It would be quite useful to load this in MATLAB and take a look at its structure, in order to start familiarizing yourself with how PRoNTo structures everything. Once you slowly start being comfortable with this, you will start to understand its great practicality. Maybe for example you want to inspect something you did and you are not sure of, and maybe there is no explicit way to inspect this through the available GUIs. Instead of re-doing everything right from the start, you can inspect ‘PRT.mat’ directly from the MATLAB Workspace.

For example, if you load the ‘PRT.mat’ file that was created and you go inside the ‘PRT.group’ field, you will see a field called ‘gr_name’ which only has one group, ‘G1’, and another field called ‘subject’ whose value is 1 x #subjects. If you now go inside ‘subject’, you will notice two different fields. Now we are inside the subjects of ‘G1’. Here you can see one field with the name ‘subj_name’ where we can see the 16 different subjects we created, and another field called ‘modality’ which includes the 6 different modalities for each subject. Go further inside if you wish, to the ‘modality’ field of ‘S1’. Here you see 6 entries (one for each modality), each of which has 6 fields, ‘mod_name’ which includes the names we input, ‘covar’ which includes possible covariates, ‘rt_subj’ which includes potential regression targets, ‘design’ which includes the designs we input,

'scans' which includes the data files we input for each subject and finally 'type' which is the data format of each data file. That way you can easily check in a direct manner what you have input in every step you did previously.

Learning to explore and manoeuvre around the 'PRT.mat', while not obligatory, will definitely help you a lot and make your life much more practical. It can save you from potential typo or other types of mistakes because it's an easy and fast way to check your inputs. So we strongly advise readers who intend to make heavy use of PRoNTo to start familiarizing themselves with the MATLAB Workspace and the main PRoNTo structures and even write their own MATLAB scripts automating some of the trivial procedures, like the one we did. A MATLAB script is a much safer way to process mundane tasks that require a lot of mouse clicks which can be tiresome and quite prone to human errors.

As in the previous tutorials the design can be reviewed. If properly specified, it will display one group of 16 subjects, with 6 modalities. For nifti modalities, this is where you would specify HRF parameters if you were using a design specified in seconds. This option is (obviously) not available for .mat or MEEG. In the present case, our design is in terms of images/scans. The default values of 0 and it should not be modified. The 'Review data' GUI would look like Figure 17.4.

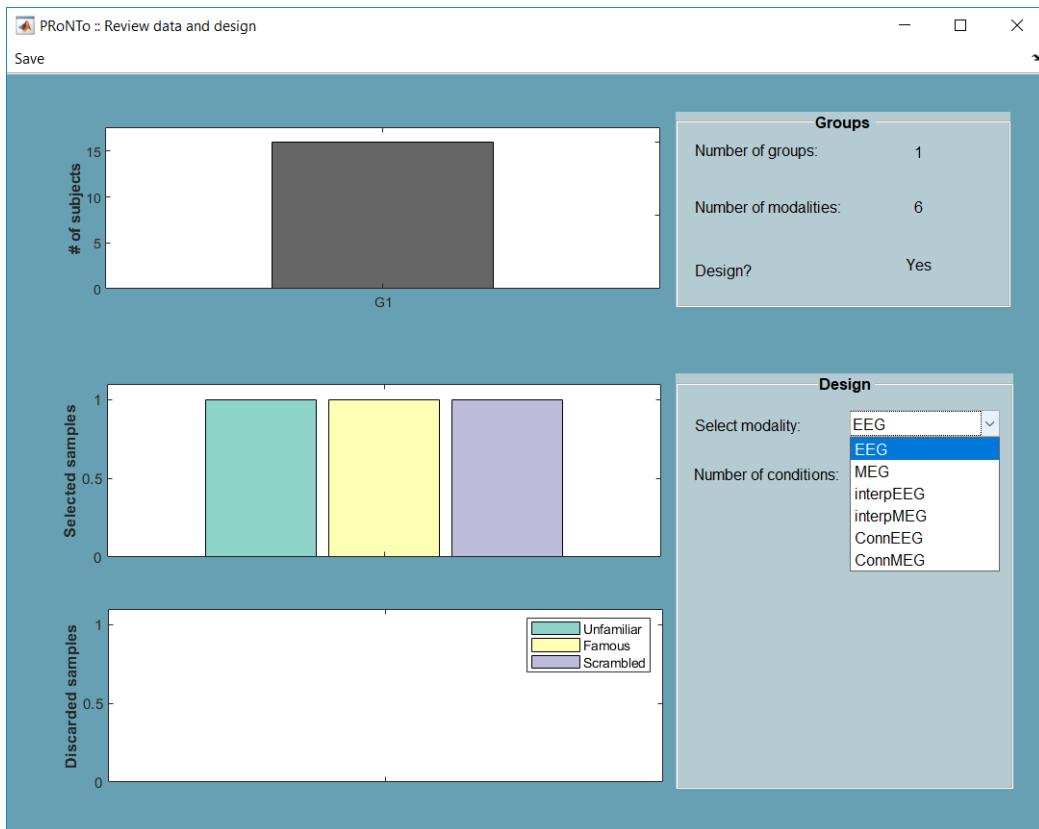


Figure 17.4: 'Review data' GUI.

17.1.2 Prepare feature set

In versions 2.X, modalities could be combined in 2 ways at this stage: either concatenated in samples (e.g. multiple runs of a same experiment), or by computing one kernel per modality and storing them together. This had the limitation that combined modalities (in either way) had to have the exact same number of features. We have now decoupled those operations for more flexibility. At the feature set stage, modalities can only be concatenated in samples. In this case they still need to have the exact same number of features.

As we want to use our modalities as different 'features for the same model and not as different samples, we hence need to build one feature set per modality. The different data types are handled in different ways: images

and .mat in a way similar to v2, MEEG with a new window. This means that, after loading the ‘PRT.mat’ in the ‘Prepare feature set’ GUI, a new window like the one in figure 17.5 will appear if multiple data formats are contained in the ‘PRT.mat’. The window asks which data format the specified feature set will have. Feature sets are built for only one data format at a time. The buttons available on this interface will reflect the data formats present in your PRT. If only one type is present, this step is skipped.

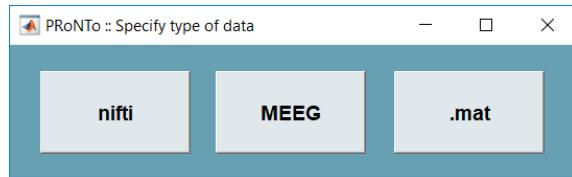


Figure 17.5: ‘Specify type of data’ GUI

Interpolated EEG/MEG (nifti data format)

- We first choose ‘nifti’ and will build the feature set for interpolated EEG. Once the data format has been chosen, the process is similar to that of v2: if there are multiple modalities of that data format in the PRT, the left window of figure 17.6 will appear, asking how many modalities to concatenate. Here, there is only one modality to consider.
- After specifying ‘1’ and ‘Enter/Return’, the appropriate window, like the one in figure 17.7 will then open to specify parameters for this specific modality. If there were multiple modalities to concatenate, the following windows would open as many times as the number entered.

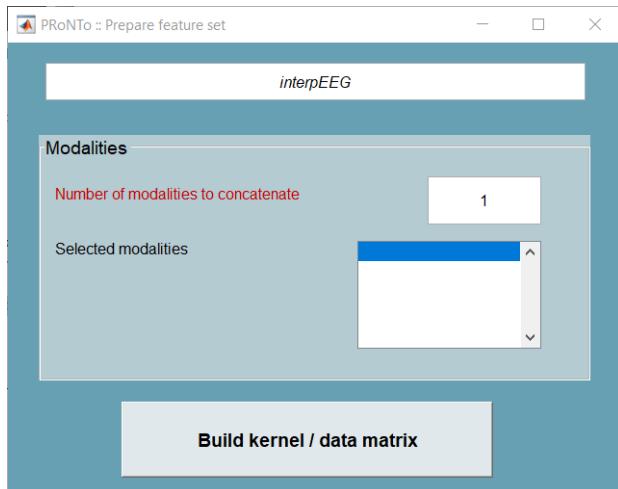


Figure 17.6: ‘Prepare feature set’ GUI.

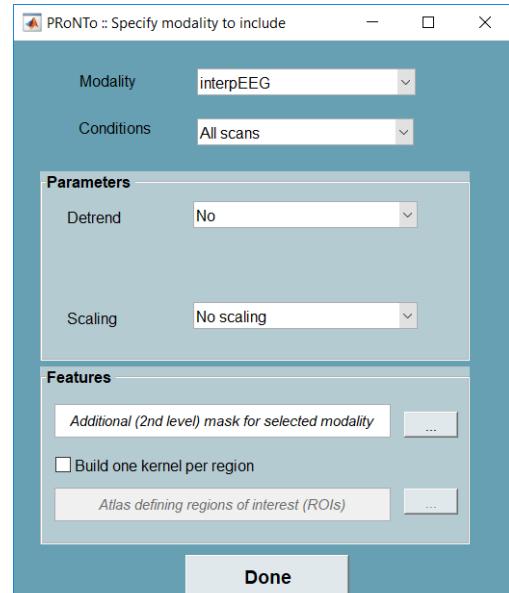


Figure 17.7: ‘Specify modality’ GUI.

- For nifti images, the whole process as we know it from versions 2.X and from the previous tutorials is the same. All options previously available are included in v3, with no addition. We select all default options. We then repeat the same procedure for the interpolated MEG modality (‘interpMEG’) to create a second feature set.

Connectivity EEG/MEG (.mat data format)

- To create the connectivity EEG/MEG feature sets, we repeat the first step but select ‘.mat’. As we have 2 .mat modalities (EEG and MEG), window in Figure 17.6 appears first.

- Again, we enter ‘ConnEEG’ as the feature set name and enter ‘1’ as the number of modalities to include. The next window is the same as for images, like the one in Figure 17.7. The only option not available is the ‘Detrend’. In addition, 2nd level masks and atlases can be entered. An example is provided in another tutorial. In the present case, we use all default options.

EEG/MEG (MEEG data format)

- To build a MEEG feature set, we select ‘MEEG’ in the first window and specify ‘EEG’ as the feature set name and ‘1’ as the number of modalities to include in the second window.
- Then, a new window appears. It allows to ‘play’ along the different dimensions of the file, averaging across dimensions or building multiple kernels. This last option will be equivalent in the code to creating an atlas for the MEEG file. The atlas is not saved but the features selected along each dimension are stored and further used to enable building the weights per kernel.
- In the present case (figure 17.8), we choose all ‘good’ channels (which is equivalent to all channels as this is a multi-subject study), use the whole time window for consistency with the other modalities and do not specify kernels along a dimension as we did not use an atlas for nifti or .mat.
- The ‘Frequencies’ panel is disabled as the signal is in voltage and was not decomposed in time-frequency. Files with TF information enable this panel. Chapter 18 elaborates more on the different things one can do when specifying MEEG modalities.

Important note: For .mat and nifti, we assume that feature 1 in sample 1 is equivalent to feature 1 of sample 2. For MEEG, the assumption is the same: the first time point in the first channel and frequency bin should be equivalent across subjects/runs. This means that if the channel montage was different or epoching was different, preprocessing steps need to be performed to ensure compatibility across files.

After we build the 6 feature sets, we can see in figure 17.9 the 12 new files along the ‘PRT.mat’: 6 binary file arrays (.dat) that collect the information across all subjects and samples in a modality, and 6 linear kernels (.mat) that store the pair-wise similarity between all selected samples in a feature set.

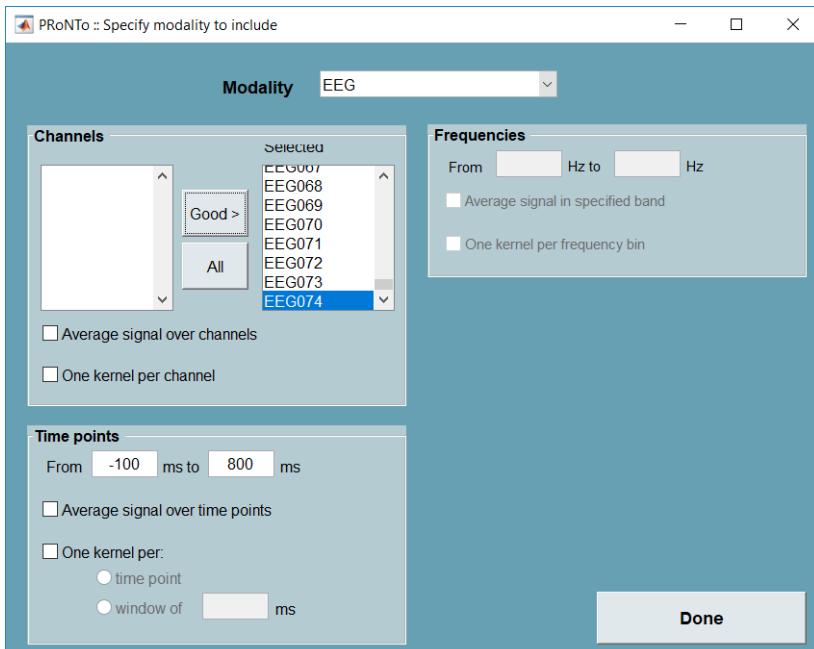


Figure 17.8: ‘Specify modality’ GUI for MEEG.

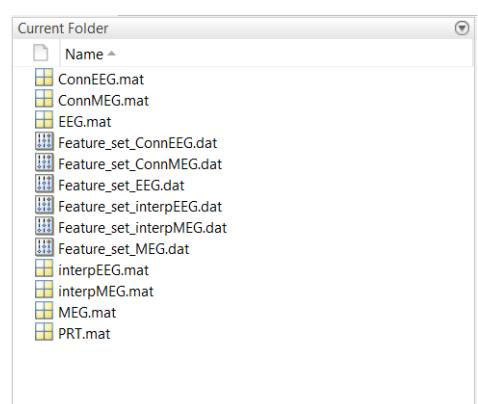


Figure 17.9: The files of the 6 feature sets.

17.1.3 Model: Specify new

Our goal is now to discriminate between the brain signals recorded during visualization of ‘Famous’ faces and ‘Scrambled’ faces. We will first build one model for each modality separately, to see how they perform, then we’ll build a model that combines all the information and look at the contribution of each modality to the predictive model. By the end of this tutorial we are going to have a total of 8 different models, 6 single kernel models (one for each modality), one multiple kernel model, and a single kernel model using a 2nd level atlas only for demonstration purposes.

- In PRoNTo’s main window, click on ‘Specify model’ and a new window called ‘Specify model’ will open (see Figure 13.14 in Chapter 13).
- Select the ‘PRT.mat’ file and provide a meaningful name to the first model, e.g. ‘interpEEG_SVM’.
- Select the ‘interpEEG’ feature set previously defined.
- Leave the option ‘Use kernels’ tick box as it is, i.e. ‘Yes’.
- Select the ‘Classification’ model type and click on the ‘Define classes’ button. A new window will open, ‘Specify classes’, to define the number of classes and a name for each class. We will define 2 classes. First click ‘Class 1’ on the tab ‘Class’. For ‘Class 1’ select all subjects and the condition ‘Famous’ and, similarly, for ‘Class 2’ select all subjects and the condition ‘Scrambled’. A novelty in the class specification is the possibility to randomly subsample the over-represented class (to match as close as possible the under-represented class). However, we use only one condition per class and one image per subject, so our dataset is balanced. So leave the ‘Subsample according to smallest class’ unchecked. Once you have appropriately specified everything, click ‘Done’.
- Select the ‘Binary support vector machine’ option, in the ‘Machine’ field.
- Select the ‘Optimize hyper-parameter’ tick box, in the ‘Define Range’ put ‘[0.1 1 10 100]’ and in the ‘Cross-Validation Scheme’ (internal loop) field, select the option ‘k-fold CV on Subject out’. A window will appear asking to define the value of k, set it to 4.
- Select the same cross-validation scheme for the external loop as well.

Note: Leave One Subject per Class Out or its k-folds variant is not appropriate here as we need to leave out all images from a single subject. Using the ‘per Class Out’ CV will throw an error and ask to select Leave Subject Out instead.

- In the ‘Data operations’ box, select the ‘Mean centre features using training data’ option.
- In the end, the ‘Specify model new’ window should look similar to the one in figure 17.10.
- Permutation testing is set in the ‘Run model’ module and since we are going to run permutations, we are only going to specify the model here, and run it later. So click on ‘Specify model’.

This operation could be repeated across the different modalities. However, this is tiresome and prone to human errors. In addition, if we had used the subsampling option, the different models would very likely select different samples, which would make them harder to compare. Instead, we can now copy most of the parameters from the model we just defined. This is done in the next section using the ‘Specify model from’ module.

17.1.4 Model: Specify from

The main interface has been modified to add a ‘Specify from’ model option. This will launch a window that is very similar to the ‘Specify model’ window, but with a couple of changes: there is an extra popup menu that allows to select which model to copy from. Some fields have also been disabled to ensure comparability of the models: only the feature sets, the machine (with its hyper-parameter optimization) and the operations can be modified. Classes/Regression selection and outer CV options cannot be modified.

- To create the other models based on each modality, we simply need to load our PRT and select the interpEEG_SVM model to copy from (it is by default the 1st model in the PRT).
- We need to specify a model name for this new model, e.g. ‘interpMEG_SVM’.

- In the present case, we only want to change which feature set is used, so we click on ‘interpEEG’ to deselect it and then click on the modality we would like to add, namely ‘interpMEG’.
- In the ‘Data operations’ select the ‘Mean centre features using training data’ option.
- Leave all the other parameters intact.

The ‘Specify model from’ window should look similar to the one in figure 17.11.

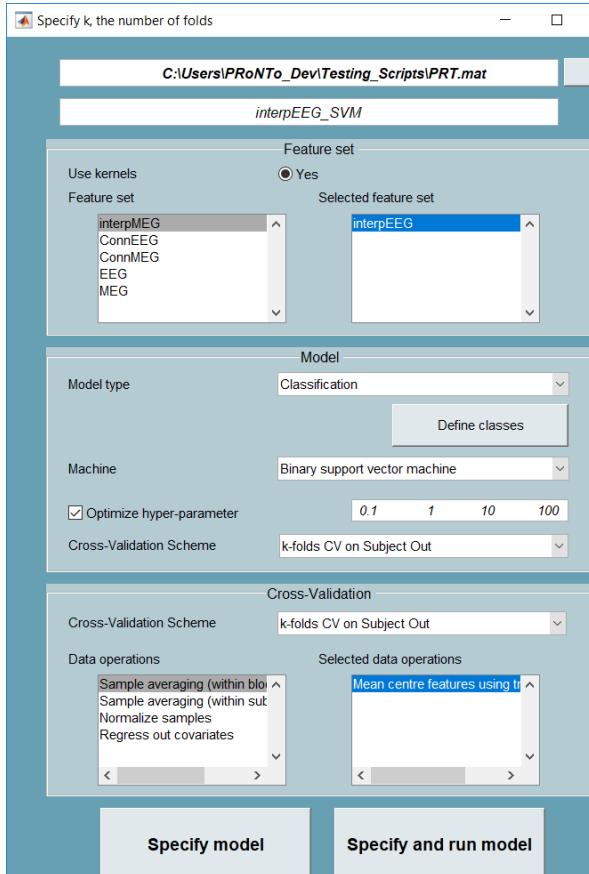


Figure 17.10: ‘Model: Specify new’ GUI final configuration.

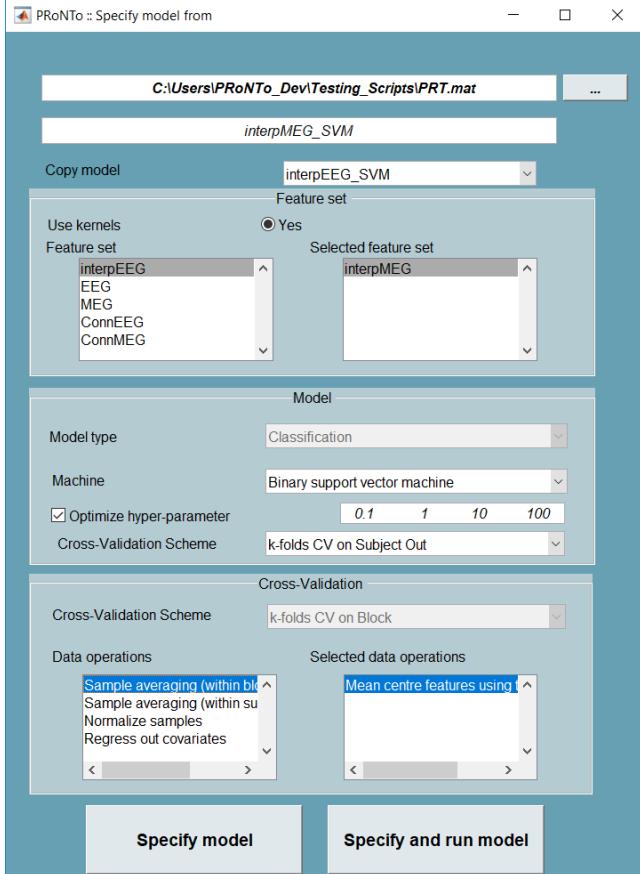


Figure 17.11: ‘Model: Specify from’ GUI final configuration.

- Do exactly the same procedure for the 4 other modalities, where each time you will only select the feature set of that modality with a name exactly the same as the feature set itself, adding a ‘_SVM’ in the end, e.g. ‘EEG_SVM’, ‘MEG_SVM’, ‘ConnEEG_SVM’, ‘ConnMEG_SVM’.

Understanding the ‘PRT’ structure: Now if you load and explore the ‘PRT.mat’ file in the MATLAB Workspace, you are going to see that there are some new structures, which correspond to the feature sets and the models you created. If you open the ‘PRT.model’ you will see it has 3 fields, with 6 entries. The first one, ‘model_name’, you will recognize as the name of the model you wrote. Then we have the ‘input’, which is the different parameters of the model, some of which you input manually and others that were defined automatically. Finally there is the ‘output’, which is where the results of each model will be written once you run the models. At the moment it is empty since you haven’t run the models yet.

If for example you want to do a quick check of the feature set that you input in a model you want to build, one way through the MATLAB Workspace is to get inside the ‘PRT.model’ and go to the input of the first entry, ‘interpEEG_SVM’. So now we ought to be inside ‘PRT.model(1).input’ if you followed the instructions properly. Inside there, you will find a field named ‘fs’. ‘fs’ in general is short for ‘feature set’, and in that particular case corresponds to the feature set of this particular model. So if you followed the instructions properly, the feature set ought to be the ‘interpEEG’. It is in that particular structure that you will find more than one feature sets,

corresponding to the feature sets, in the case of multiple kernel learning.

MKL model

We can also build multimodal models in PRoNTo and use it to investigate the different contribution of each modality for the predictive model. In this example we can use the MKL model to investigate which modality contains most information for the ‘Faces’ versus ‘Scrambled’ comparison, and see if using different features improves performance when compared to the single modalities. So use the ‘Specify model from’ module again, but this time instead of having only one feature set, add all feature sets to the model. You will also need to change the machine to the ‘L1 Multi-Kernel Learning’. Finally you will also need to add the ‘Normalize samples’ operation in the ‘Data operations’ to compensate for the fact that different modalities have different numbers of features. Leave all the other options as they are. The ‘Specify model from’ window for the MKL model should look similar to the one in figure 17.12.

17.1.5 Model: Run

We can use the ‘Run’ module to run the previously specified models. After you select the ‘PRT.mat’ file you will see the 7 models we have created so far, 6 single kernel models and the MKL model. You can either select and run them one by one, or you can select all and run them all together.

In ‘Permutations’ select ‘Perform permutation test’ with 100 repetitions. Be aware that 100 repetitions is a small number when running permutation tests, and 1000 repetitions (or even more) are recommended if you want to be able to obtain smaller p-values (since the minimum p-value possible is equal to 1/number of permutations). The ‘Run’ window should look similar to the one in figure 17.13.

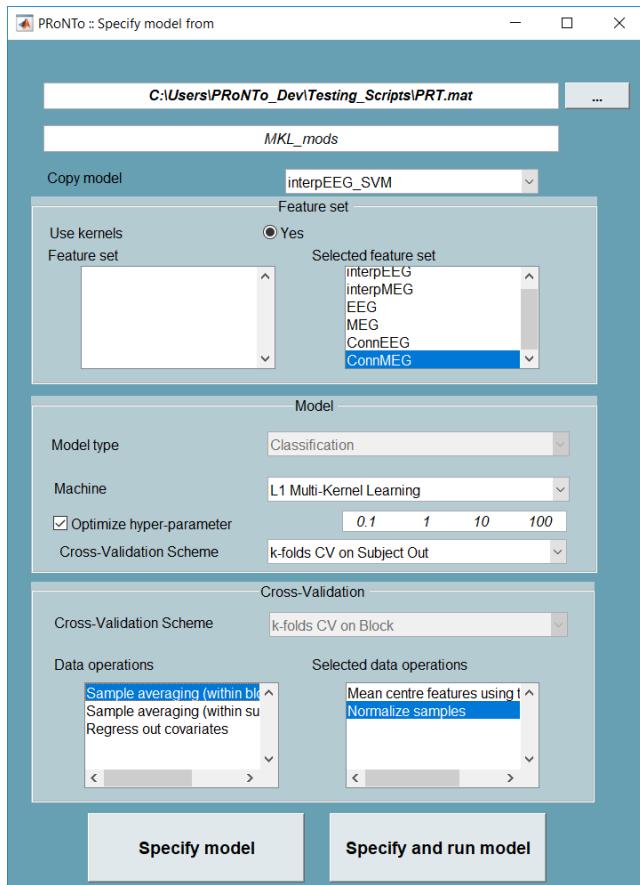


Figure 17.12: ‘Model: Specify from’ GUI final configuration for the MKL model.

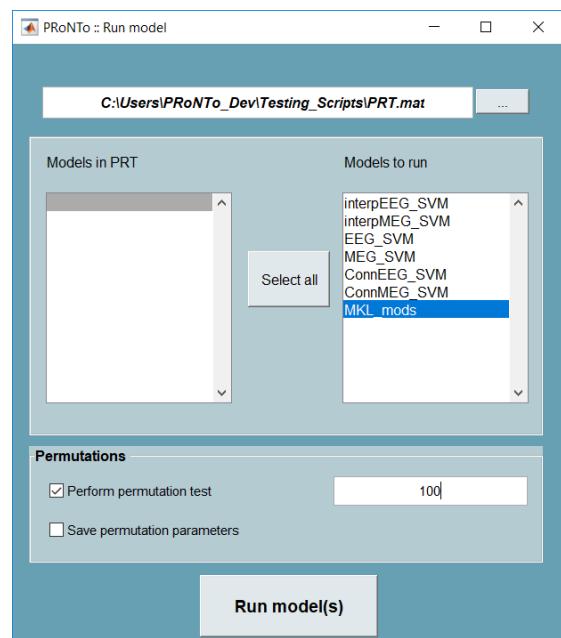


Figure 17.13: ‘Model: Run’ GUI final configuration to run all models together.

17.1.6 Display results

Taking a quick look at the results we see that ‘interpMEG SVM’ and ‘MEG SVM’ are the best performing models, and ‘ConnMEG_SVM’ the worst performing model. We also see that the MKL model that included all modalities had in fact worse performance than the best single-kernel modalities alone.

For more detailed information regarding the interpretation of the results, the different types of ways to measure performance and the stats of the results the reader should take a look at all the previous tutorials, especially chapters 13 and 15.

17.1.7 Compute weights

Accordingly, the weight computation window includes a tick box to allow building the average weights for each permutation. As in the v2.X batch, these are saved in another folder. The window will also allow to select one atlas per feature set included in the model. To this end, simply select multiple images or .mat, in the correct order. If the Computer average/kernel weight per region option is selected, feature sets that were built using an atlas will automatically use this atlas. If this option is selected and no atlas is specified for some feature sets, only the weights will be built, not the weights per region/kernel. Please note that loading an atlas for weight summarization is not available for MEEG data (only nifti and .mat).

- In PRoNTo’s main window, click on ‘Compute weights’ and a new window will open, ‘Compute weights’ (Figure 13.24).
- Select the ‘PRT.mat’ file.
- Select one by one all models from the list of ‘Models computed in PRT’.
- Leave the options ‘Compute average/kernel weight per region’ and ‘Build weight images for permutations’ unchecked.
- Click on ‘Compute weights’ button. Computations will be displayed on the MATLAB command window.

The final ‘Compute weights’ window should look like the one in figure 17.14

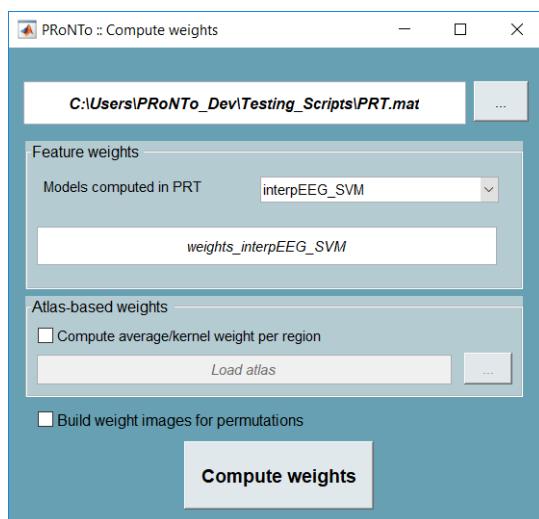


Figure 17.14: ‘Compute weights’ GUI.

17.1.8 Display weights

Weights can be displayed for all modality types. For 1D MEEG or 1D .mat data, weights are displayed as bar graphs, with the color of the bar representing the amplitude of the weight (e.g. the connectivity vector used for .mat). Example in figure 17.15.

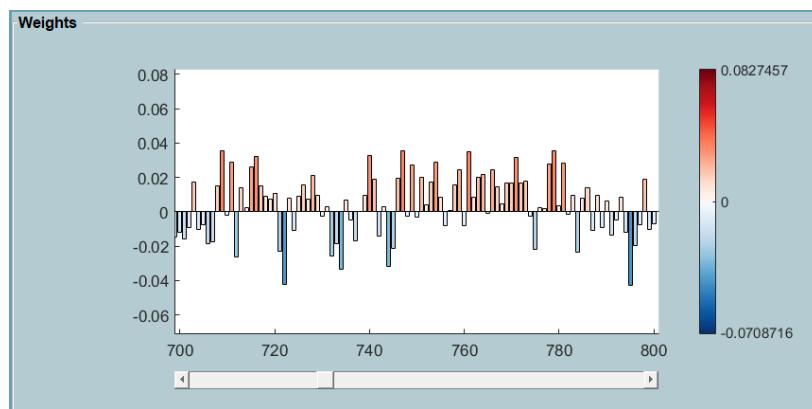


Figure 17.15: ‘Display weights’ for 1D MEEG or .mat data.

For 2D data, the display shows the matrix, with y-axis as the 1st dimension and x-axis as the 2nd dimension (after squeezing out dimensions of size 1), with the color of a (x,y) pair displaying the magnitude of its weight. Example in figure 17.16. Finally, nifti images are displayed as before, except a small change in the color map. Example in figure 17.17.

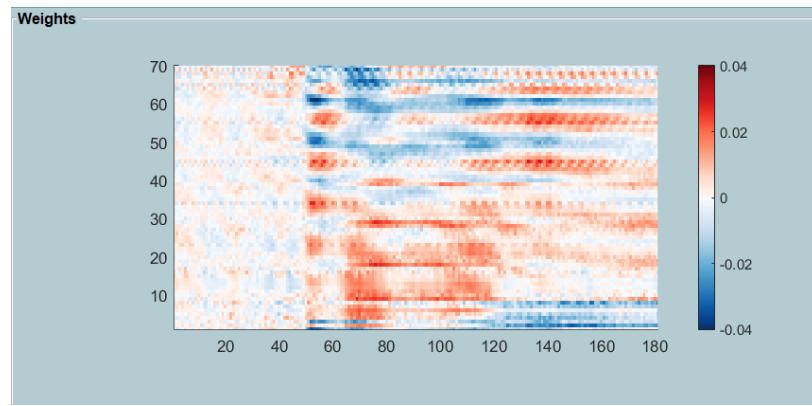


Figure 17.16: ‘Display weights’ for 2D MEEG or .mat data.

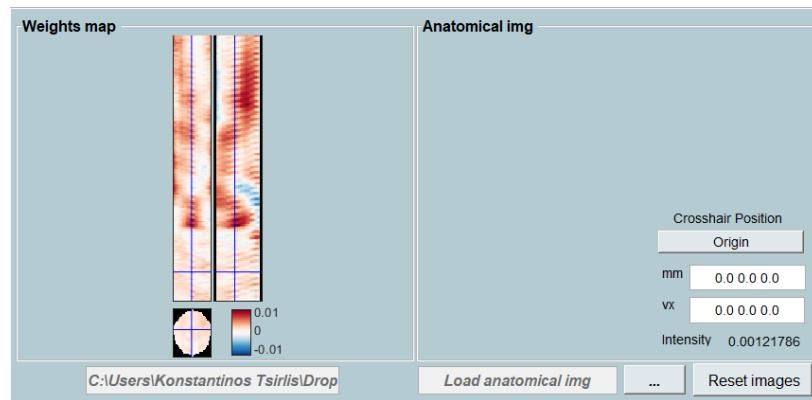


Figure 17.17: ‘Display weights’ for nifti images.

When building the weights from the ‘MKL_mods’ model, we can also look at the feature set contributions. From figure 17.18 we can see that the interpolated EEG has the largest contribution to the MKL model, followed by the EEG and MEG traces. This can be due to the interpolated EEG smoothing some of the anatomical variance in electrode placement, while EEG and MEG keep subject-specific information.

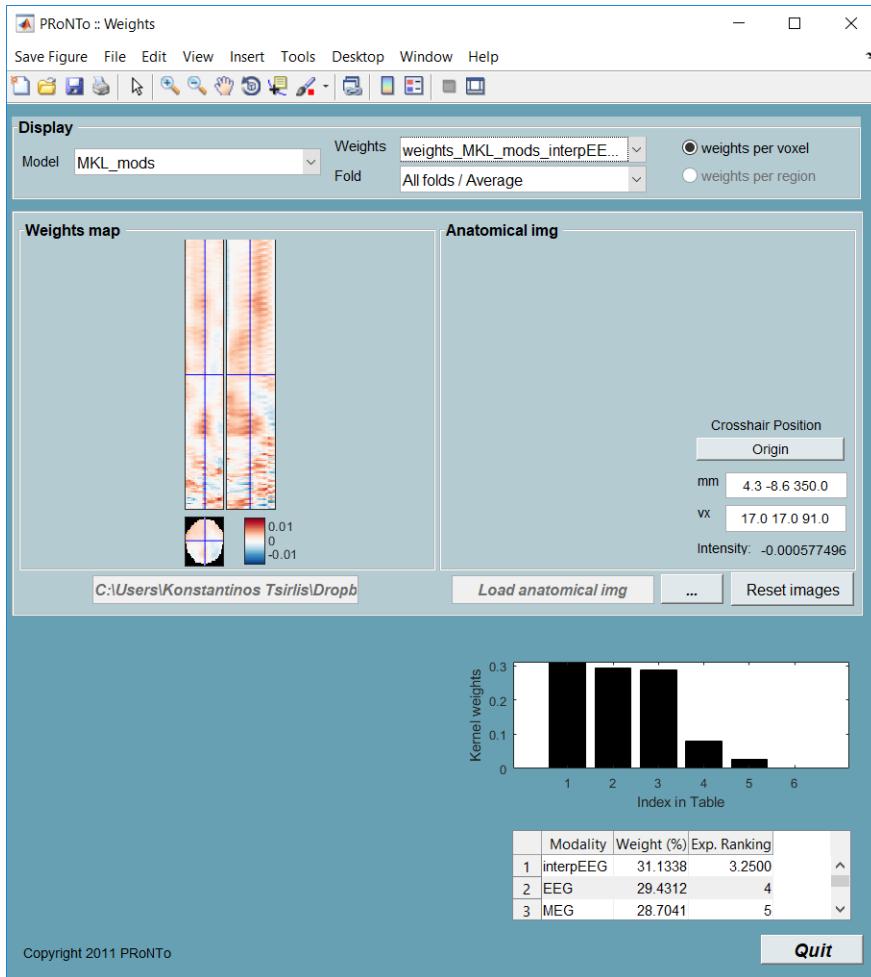


Figure 17.18: ‘Display weights’ for MKL model, with additional weight contribution shown.

17.1.9 Using an atlas with .mat

As for nifti, an atlas can be used with .mat and MEEG files. For MEEG files, the atlas is implicit, resulting from the selection of channels, time points and kernels on which to build the kernels on. However, for .mat, PRoNTo is agnostic of the type of features entered, so the atlas needs to be built by the user. As an example, we built an atlas for the EEG connectivity derived modality. We defined 11 ‘networks’ that the channels could belong to based on their anatomical position (e.g. ‘orbitofrontal’, ‘occipital’, ‘left-parietal’, ‘central-parietal’, ...). The atlas then builds ROIs from pair-wise network interactions (i.e. ‘orbitofrontal-occipital’, ‘orbitofrontal-left-parietal’, ...) leading to 66 ROIs ($11 \times 10 / 2$ between networks interactions + 11 within-network interactions). The atlas looks as in figure 17.19 (2D version but only the upper triangular part is saved).

Going back to the feature set step, we can build another .mat feature set (called ‘ConnEEG_atlas’) including the ConnEEG modality and ticking the ‘Build one kernel per ROI’ box (figure 17.20). We can load the designed atlas and PRoNTo will automatically gather ROI labels if a label file (called ‘Labels_atlasname.mat’) is present in the same folder and contains a ‘ROI_names’ variable.

Model specification can then be performed as in v2.

- Select the ‘ConnEEG_atlas’ feature set.
- Define the classes as we did before.
- Choose the ‘L1 Multi-Kernel Learning’ machine.
- Use 4-folds CV on subjects out for both inner and outer CV.

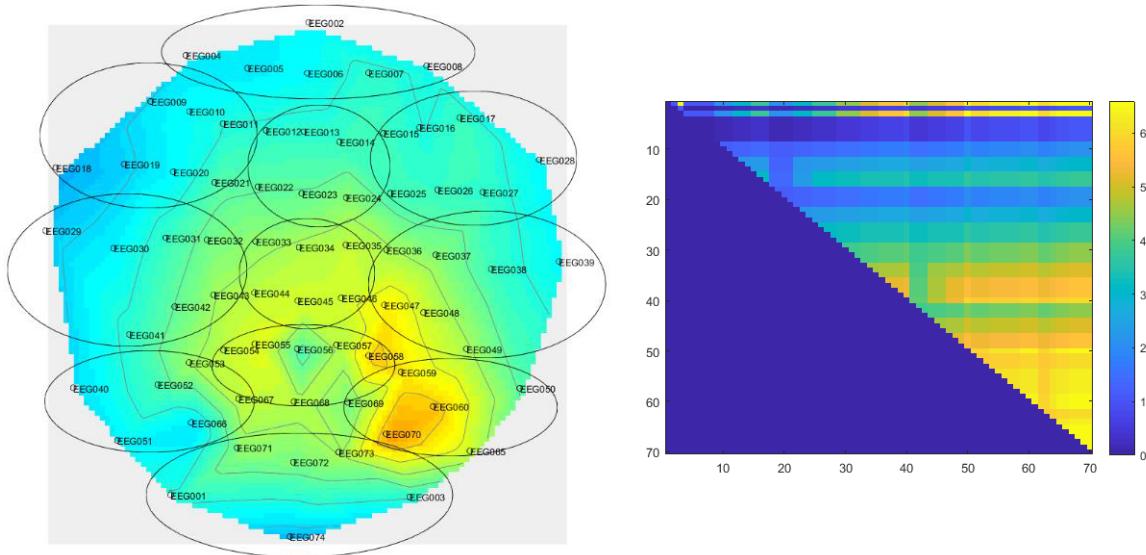


Figure 17.19: Atlas for EEG connectivity modality, defining interactions between and within 11 ‘networks’ (identified based on their anatomical position as displayed on the left).

- And finally, add the mean centering and the normalize samples operations before specifying and running.

The window should look like the one in figure 17.21.

The model does not perform very well (53.13% balanced accuracy) for this modality taken on its own. For information, the same model using SVM (i.e. discarding the atlas information) leads to 43.75% balanced accuracy. Both models display a large variance across folds and would not be considered as significantly discriminating between famous faces and scrambled faces.

We can compute the weights for this model and select the option to build the weights per region. This will build 2 .mat files: one including the weights per feature, one including the weights per ROI. The latter gives the same value (i.e. its kernel contribution) to each feature within a ROI. The ‘Display weights’ window in figure 17.22 shows the table of kernel contributions.

Note: If for any reason the ROI labels haven’t loaded, you can manually load them using the button ‘Load labels’ and locating the file ‘Labels.EEG_atlas.mat’ found in *Multimodal_face_dataset/data*.

The weight file can then be postprocessed for better visualization. This has to be performed outside of PRoNTo as PRoNTo does not know what type of data the .mat represents (i.e. it does not know this is connectivity data).

As an example, in figure 17.23 we have summarized the weights per region in a 11*11 matrix instead of the 70*70 matrix. Furthermore, a schemaball plot (code modified from MATLAB’s file exchange schemaball submission, in appendix) was derived. It depicts the contribution of each between-network interaction to the classification as a colored curve and the within-network contributions as node color and size.

17.2 Batch analysis

This tutorial will now show how to analyse the same data but using the `matlabbatch` system.

Once again, to analyse the data, create a new directory in which to save the results of the analysis, saved as ‘PRT.mat’. On the main interface of PRoNTo click on the ‘Batch’ button to open the ‘`matlabbatch`’. Alternatively, type ‘prt_batch’ in the MATLAB prompt.

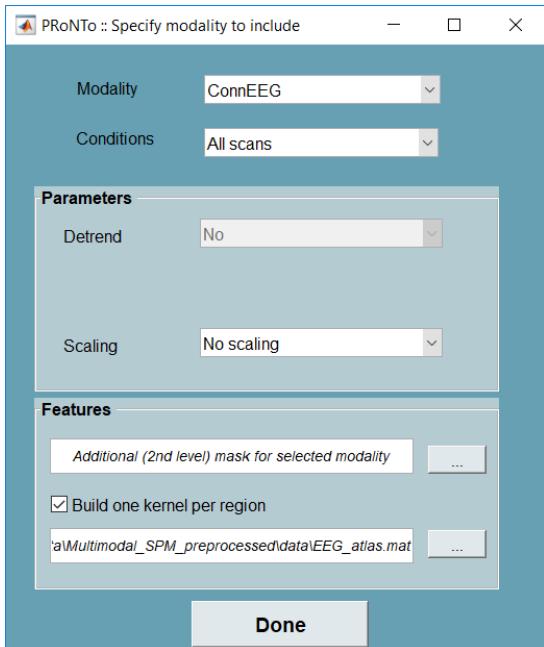


Figure 17.20: ‘Specify modality’ GUI with atlas.

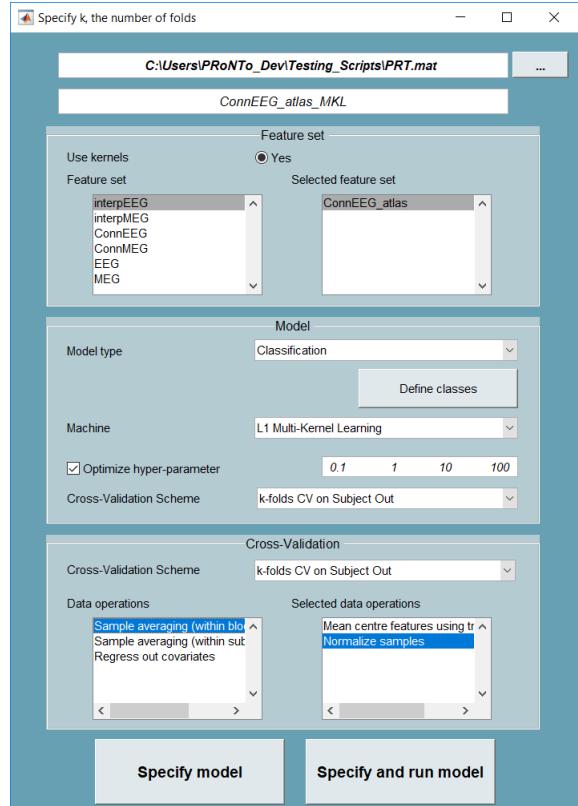


Figure 17.21: ‘Specify model’ GUI final configuration.

17.2.1 Data & Design

When adding a modality, the user now has to specify its Data format as either nifti, MEEG or .mat. As the batch is agnostic of the users choices, all design options are available for each type. Please make sure to use the appropriate option (i.e. loading an SPM.mat is only for nifti, and Events in file is only for MEEG).

For the same reason, one mask has to be entered per modality, independent of its type. Only ‘nifti’ data format requires a file, ‘.mat’ and ‘MEEG’ just need a modality name. This is in contrast with the GUI that only requires masks for ‘nifti’ files (as it derives the masks for the MEEG and .mat automatically).

- Click on ‘Data & Design’ in the PRoNTo ‘matlabbatch’ menu.
- In the ‘Directory’ field, select a directory where the ‘PRT.mat’ file will be saved.
- In the ‘Groups’ field:
 - Add one group and in the field ‘Name’, provide a name without spaces to that group, e.g. ‘G1’.
 - In the field ‘Select by’, select the ‘Subjects’ option and add one subject.

EEG/MEG (MEEG data format)

- * Add one modality for this subject and provide a name, e.g. ‘EEG’; choose the appropriate data format (here MEEG); define the interscan interval of 0.05 seconds; and in the field ‘Files’, select the EEG (MEEG) file of the first subject (‘S1’), found in the *Multimodal_face_dataset/data/S1* directory.
- * In the ‘Data & Design’ field, since our data are in MEEG format, choose the ‘Events in MEEG file’ (for MEEG inputs only option), and select no regression targets/covariates.

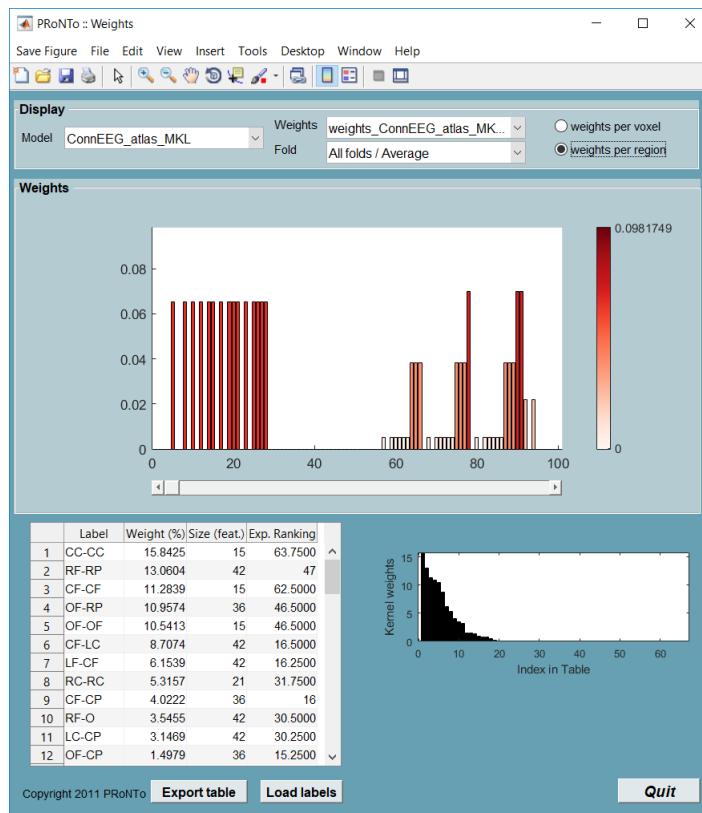


Figure 17.22: Atlas for EEG connectivity modality, defining interactions between and within 11 ‘networks’ (identified based on their anatomical position as displayed on the left).

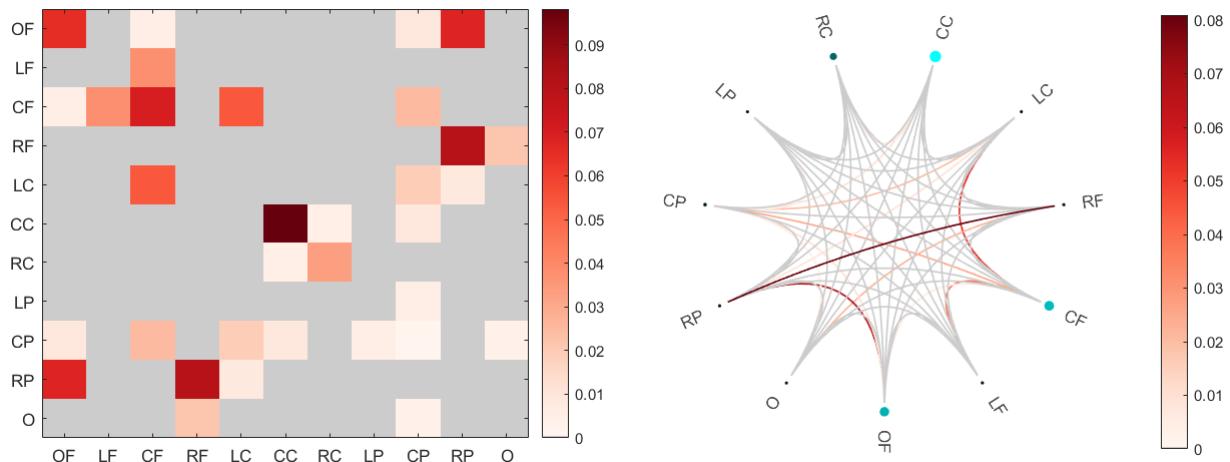


Figure 17.23: Summarized ROI weights across networks, displayed as a matrix (left) and as a schemaball (right). The size and brightness of the network marker displays the within-network contribution while the curves display the between-network contributions. Grey lines or matrix entries represent 0 contribution.

* Add one more modality and repeat the whole procedure for MEG.

Connectivity EEG/MEG (.mat data format)

* Add two more modalities, these would be the ‘ConnEEG’ and ‘ConnMEG’; choose the appropriate data format (here .mat); define the interscan interval at 1; and in the field ‘Files’, select the 3 ConnEEG/ConnMEG (.mat) files of the first subject (‘S1’), found in the *Multimodal_face_dataset/data/S1* directory. Remember that the order (Unfamiliar, Famous, Scrambled) with which you input the

files is of great importance.

- * In the ‘Data & Design’ field, since our data are in .mat format, choose the ‘Specify design’ option, set the ‘Units for design’ to ‘Scans’, and either manually set the names, onsets and durations of each of the 3 conditions, or select the practical option ‘Multiple conditions’ and select the ‘Design.mat’ file found in the *Multimodal_face_dataset/data/S1* directory.
- * Repeat the whole procedure for MEG.

Interpolated EEG/MEG (nifti data format)

- * Add two more modalities, these would be the ‘interpEEG’ and ‘interpMEG’; choose the appropriate data format (here nifti); define the interscan interval at 1; and in the field ‘Files’, select the 3 interpEEG/interpMEG (NIfTI) files of the first subject (‘S1’), found in the *Multimodal_face_dataset/data/S1* directory in the folders *EEG_interp/MEG_interp*. Remember that the order (Unfamiliar, Famous, Scrambled) with which you input the files is of great importance.
- * In the ‘Data & Design’ field, since our data are in .mat format, choose the ‘Specify design’ option, set the ‘Units for design’ to ‘Scans’, and either manually set the names, onsets and durations of each of the 3 conditions, or select the practical option ‘Multiple conditions’ and select the ‘Design.mat’ file found in the *Multimodal_face_dataset/data/* directory.
- * Repeat the whole procedure for MEG.

- In the ‘Masks’ field, add 6 new modalities and provide the appropriate modality names. The name of the modality here has to be exactly the same as in ‘Modalities’, otherwise it will not work.
 - For MEEG and .mat you only have to choose the appropriate data format as for both of these data formats we assume that the users have provided only useful features, as this can be easily performed during pre-processing. Please ensure that your data do not contain NaNs.
 - For nifti select the ‘mask_EEG’ or the ‘mask_MEG’ masks found in the *Multimodal_face_dataset/data/* directory and ensure that ‘HRF overlap’ and the ‘HRF delay’ fields are 0.
- In the ‘Review’ field, select ‘Yes’ if you would like to review your data and design in a separate window. Otherwise, leave as it is, i.e. ‘No’. Keep in mind that the procedure pauses while you review the data and that you have to close the ‘Review’ window for the procedure to continue.

After you have set everything for the first subject, you can go to ‘Subjects’ and select ‘Replicate: Subject (1)’. That way you will have everything set and you will only need to change the files from those of ‘S1’ to the ones of the subject you are setting. Alternatively, you could use the ‘PRT.mat’ you already have from the GUI part of the tutorial, or use the script we provide. The final configuration including the first subject only should look like figure 17.24.

Figure 17.24: ‘Data & Design’ module in `matlabbatch`.

17.2.2 Feature set / Kernel

Interpolated EEG/MEG (nifti data format)

- Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu.
 - With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Data & Design’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved. The window of Figure 13.32 in Chapter 13 is called to establish a dependency connection with the previous ‘Data & design’ module.
 - Provide a name to the ‘Feature/kernel’ set, e.g. ‘interpEEG’.
 - Select the ‘Nifti’ option for a data format, add one modality and select the modality name with the ‘Dependency’ button or type it in manually but the name needs to be *exactly* the same as the one specified in the ‘Data & Design’ module, here ‘interpEEG’.
 - Leave all the other parameters at their default options/values.
 - Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu to add one more ‘Feature set / Kernel’ module and repeat the procedure for the interpolated MEG (‘interpMEG’) data.

EEG/MEG (MEEG data format)

- Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu.
 - With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved.
 - Provide a name to the ‘Feature/kernel’ set, e.g. ‘EEG’.

- Select the ‘MEEG’ option for a data format, add one modality and select the modality name with the ‘Dependency’ button or type it in manually but the name needs to be *exactly* the same as the one specified in the ‘Data & Design’ module, here ‘EEG’.
- Leave all the other parameters at their default options/values.
- Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu to add one more ‘Feature set / Kernel’ module and repeat the procedure for the MEG (‘MEG’) data.

Connectivity EEG/MEG (.mat data format)

- Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved.
- Provide a name to the ‘Feature/kernel’ set, e.g. ‘ConnEEG’.
- Select the ‘.mat’ option for a data format, add one modality and select the modality name with the ‘Dependency’ button or type it in manually but the name needs to be *exactly* the same as the one specified in the ‘Data & Design’ module, here ‘ConnEEG’.
- Leave all the other parameters at their default options/values.
- Click on ‘Feature set / Kernel’ option on PRoNTo’s `matlabbatch` menu to add one more ‘Feature set / Kernel’ module and repeat the procedure for the connectivity MEG (‘ConnMEG’) data.

At this point your `matlabbatch` should look like figure 17.25.

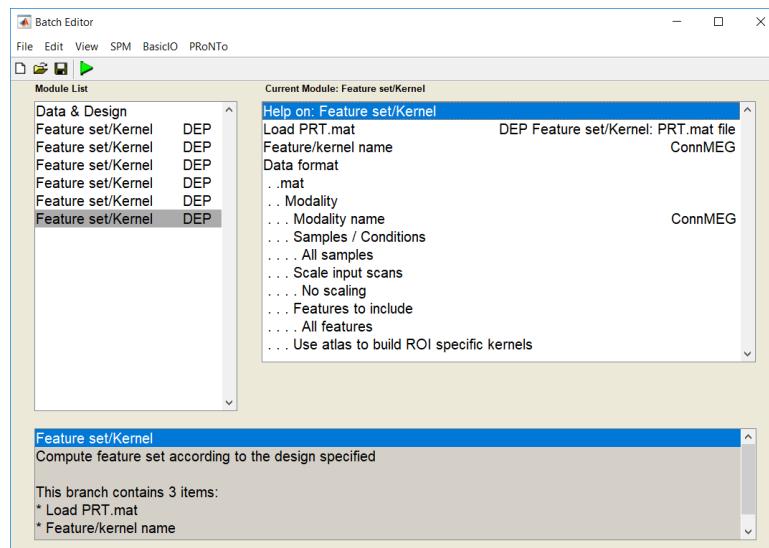


Figure 17.25: ‘Feature set / Kernel’ module in `matlabbatch`.

17.2.3 Model: Specify new

Single kernel models

- Click on the ‘Model: Specify new’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous ‘Feature set / Kernel’ step or click on the ‘Select files’ button to browse where ‘PRT.mat’ file was saved.

- In the ‘Feature sets’ field, select the ‘New: Name’ in the ‘Feature set name’ and either use the ‘Dependency’ button to find the relevant feature set name (here ‘interpEEG_SVM’) or write it *exactly* as previously defined in its relevant ‘Feature set / Kernel’ module.
- Select the ‘Classification’ model type:
 - Add 2 new classes.
 - For Class (1) write ‘Famous’ on the name field and add one group. Select the group name from the ‘Data & Design’ module (‘Data & Design:Group#1 name’) with the ‘Dependency’ button, or write it *exactly*, as previously defined in the Data & Design’ module, here ‘G1’. Similarly, for Class (2) write ‘Scrambled’ on the name field and add the group created in the ‘Data & Design’ module, ‘G1’.
 - In the ‘Subjects’ field, type ‘1:16’ to include all 16 subjects.
 - In the ‘Conditions / Scans’ field, select the ‘Specify Conditions’ option and add a new condition in each class. Provide a name for this condition, i.e. for Class (1) ‘Famous’ and for Class (2) ‘Scrambled’. Note that this name needs to be spelled exactly as specified in the ‘Data & Design’ module.
- Leave ‘Subsample examples based on class definition’ as it is, i.e. ‘No’.
- In the ‘Machine’ field:
 - Select the ‘Kernel machine’ and the ‘SVM Classification’ option.
 - In the ‘Machine optimization and parameters’ field, select the ‘Optimize hyper-parameter’ option.
 - Set the ‘Regularization hyper-parameter’ to ‘0.1, 1, 10, 100’.
 - Set the ‘Cross-validation type for hyper-parameter optimization’ to ‘k-folds CV on subjects’ and the k to ‘4’. This is the internal cross-validation loop.
- In the ‘Cross validation type’ (external loop) field, select ‘k-folds CV on subjects’ option and again set k to ‘4’.
- Leave the ‘Include all scans’ field as it is, i.e. ‘No’.
- Leave all the rest to their default parameters/values.

Figure 17.26 shows the final configuration of the first `matlabbatch` ‘Model: Specify new’ module for the interpolated EEG data.

We now need to do the same procedure for 5 more single kernel models, each one including only 1 feature set, and for 1 multi-kernel model including all feature sets together.

Compared to the GUI, when we are in `matlabbatch` and depending on whether you want to modify the feature sets, the model types and the data operations from one model to the other, it might be more practical either to replicate the ‘Model: Specify new’ module that we have already finished, or to create a new ‘Model: Specify from’ module. In our case that we do not want to modify anything other than changing the feature set, it is in fact faster to just right-click on the ‘Model: Specify new’ and select the ‘Replicate module’ option.

After replicating the finished module 5 times, for each module you will only need to change its ‘Model name’ and the ‘Name’ option of the ‘Feature set name’ field. So for the other modules these would have to be `interpMEG_SVM` and `interpMEG` respectively, ‘EEG_SVM’ and ‘EEG’, and so on.

If one wanted to use the ‘Model: Specify from’ module, then they would mostly have to follow the same procedure as in the ‘Model: Specify new’ module, but they should also input the model they want to copy from in the ‘Model to copy’ field, and also to specify in the ‘Fields to modify’ any possible changes they wanted in either the feature sets, the model types or the data operations.

Help on: Model: Specify new	
Load PRT.mat	...stPRT.mat
Model name	...EEG_SVM
Feature sets	
. Feature set name	
. . Name	interpEEG
Model Type	
. Classification	
. Classes	
. . Class	
. . . Name	Famous
. . Groups	
. . . Group	
. . . . Group name	G1
. . . . Subjects	16x1 double
. . . Conditions / Samples	
. . . . Specify Conditions	
. . . . Condition	
. Name	Famous
. . . . Class	
. Name	Scrambled
. . . . Groups	
. Group	
. Group name	G1
. Subjects	16x1 double
. Conditions / Samples	
. Specify Conditions	
. Condition	
. Name	Scrambled
. . Subsample examples based on class definition	No
. . Machine Type	
. . . Kernel machine	
. . . SVM Classification	
. . . . SVM string argument	-q -s 0 -t 4 -c
. . . . Machine optimization and parameters	
. Optimize hyper-parameter	
. Regularization hyper-parameter	[0.1 1 10 100]
. Cross-validation type for hyper-parameter optimization	
. k-folds CV on subjects	
. k	4
. . Cross-validation type	
. . . k-folds CV on subjects	
. . . . k	4
. . Include all scans	No
. . Data operations	
. . . Mean centre features	
. . . Other Operations	Yes
. . . No operations	

Figure 17.26: Final configuration of the `matlabbatch` ‘Model: Specify new’ module.

Multi-kernel model

- Replicate one ‘Model: Specify new’ module one more time.
- Set a name for the model, ‘MKL_mods’.
- We are going to include all 6 feature sets in this model, so in the ‘Feature set name’ include 6 entries and put the 6 names of the feature sets, ‘interpEEG’, ‘interpMEG’, ‘EEG’, ‘MEG’, ‘ConnEEG’ and ‘ConnMEG’.
- The second thing we will need to change is in the ‘Data operations’ where in the ‘Other Operations’ we are going to select the ‘Select Operations’ option, include a new operation, and choose the ‘Normalize samples’ option from the list.
- Leave all the other fields as they are.

Figure 17.27 shows all 7 `matlabbatch` ‘Model: Specify new’ modules, with the MKL model showing as an example.

17.2.4 Model: Run

- Click on the ‘Model: Run’ option on PRoNTo’s `matlabbatch` menu.
- Select the appropriate ‘PRT.mat’ file, either manually, or if you have all modules together in the ‘Module List’ using the Dependency button to associate it with the module of the previous step.

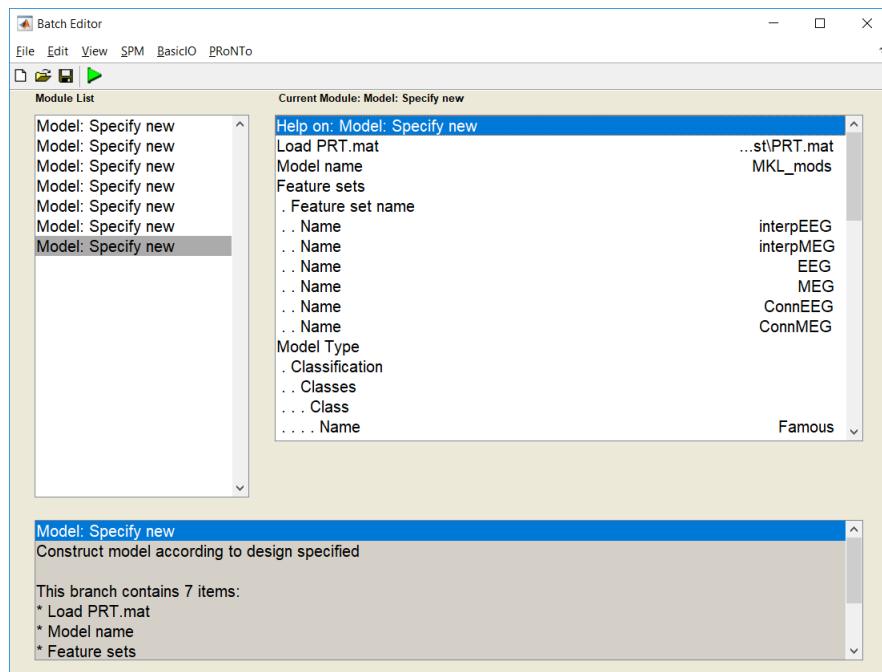


Figure 17.27: All 7 `matlabbatch` ‘Model: Specify new’ modules.

- Select the model name from the first ‘Model: Specify new’ module with the ‘Dependency’ button, or write it *exactly*, as previously defined in the ‘Model: Specify new’ module, here ‘interpEEG_SVM’.
- In the ‘Do permutation test?’ select ‘Permutation test’, with 100 repetitions. Be aware that 100 repetitions is a small number when running permutation tests, and 1000 repetitions (or even more) is a more realistic one if you want your results to have a good statistical power.

Figure 17.28 shows the final configuration of the `matlabbatch` ‘Model: Run’ module.

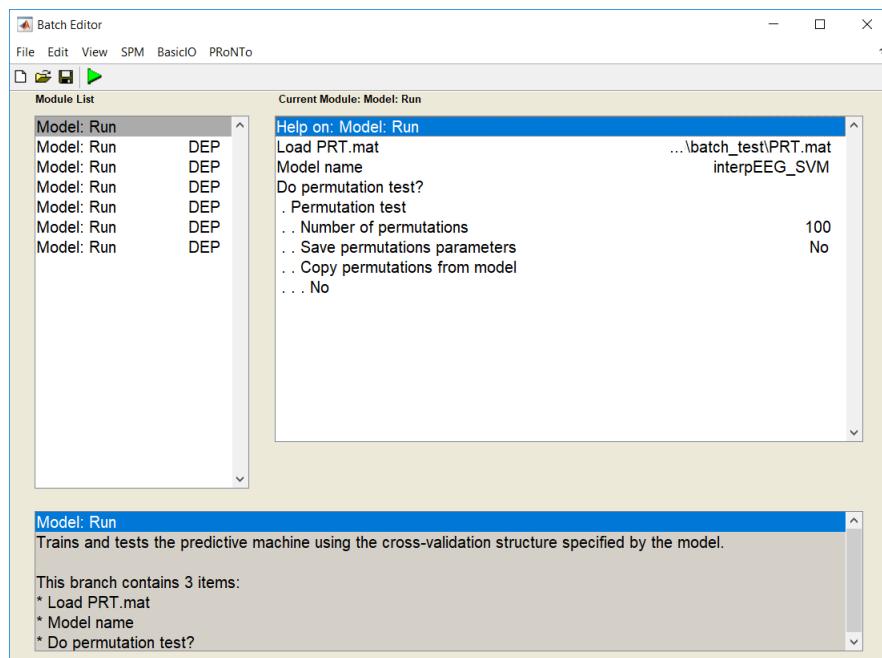


Figure 17.28: The `matlabbatch` ‘Model: Run’ modules.

Follow the same procedure (as in the previous section) by replicating the now finished ‘Model: Run’ module 6

more times, each time changing only the name of the model to be run. Remember that we have 7 models in total, 6 single kernel models and 1 multi-kernel model.

17.2.5 Compute weights

- Click on the ‘Compute weights’ option on PRoNTo’s `matlabbatch` menu.
- With ‘Load PRT.mat’ field selected, click on the ‘Dependency’ button to associate the ‘PRT.mat’ file created in the previous step, or locate the ‘PRT.mat’ manually.
- It’s optional to define an image name.
- Finally, set the ‘Build weights images for permutations’ field to ‘No’.

As we did before, replicate this module 6 times. Each time you will only need to change the name of the model (either set it manually, or associate it properly using the ‘Dependency’ button, if you have all modules in the module list) so that we compute weights for all 7 models. Figure 17.29 shows the final configuration of the `matlabbatch` ‘Compute weights’ module.

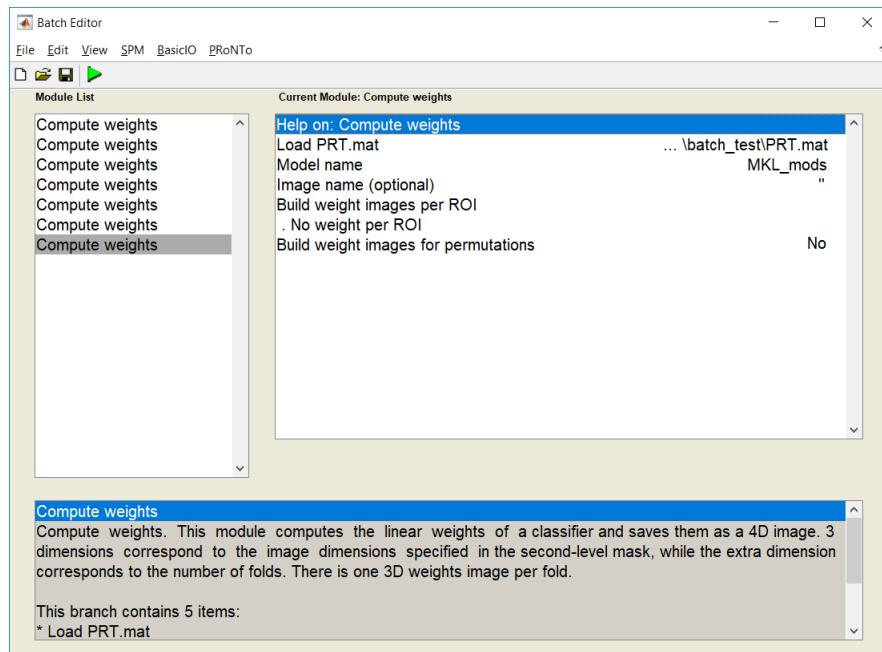


Figure 17.29: Final configuration of the `matlabbatch` ‘Compute weights’ module.

It is advised that you save the batches, either in groups or if possible all together, so that you can open and edit them for further analyses.

17.2.6 Display results & weights

If everything was set properly, the results should be the same as those obtained using GUI.

17.2.7 Using an atlas with .mat

The changes needed to use an atlas with .mat, compared to not using an atlas, are straight forward, and the reader is referred to 17.1.9 where the same procedure is followed.

Chapter 18

Classification of semi-simulated ECoG data

Contents

18.1	GUI analysis	192
18.1.1	Data & design	192
18.1.2	Prepare feature set	192
18.1.3	Model: Specify new	194
18.1.4	Model: Specify from	195
18.1.5	Model: Run	195
18.1.6	Display results	196
18.1.7	Compute weights	196
18.1.8	Display weights	196
18.2	Batch analysis	197
18.2.1	Data & Design	197
18.2.2	Feature set / Kernel	199
18.2.3	Model: Specify new	200
18.2.4	Model: Run	202
18.2.5	Display results	202
18.2.6	Compute & Display weights	203

The data consists of a semi-simulated data set from a single subject in electrocorticography (ECoG, a.k.a. intracranial EEG). It was built from a 5 minutes rest session to which a fake design was added: 2 conditions, ‘A’ and ‘B’, randomly interspersed. The pre-processing discarded noisy and pathological channels, leaving 38 ‘good’ channels. A rectangular signal window was added to events from ‘A’, after extraction of the power in the High Frequency Broadband (HFB). The magnitude of the signal to add was calculated based on the A vs B contrast. The chosen signal (A) to noise (B) ratio varied in the original experiment but is fixed to 3 in the present case. Similarly, the simulated signal was not added to all channels but to roughly half of them. After artefact rejection, 60 events of A and 56 of B remain for classification. This dataset was published in:

J. Schrouff and J. Mouro-Miranda, ”Interpreting weight maps in terms of cognitive or clinical neuroscience: nonsense?,” 2018 International Workshop on Pattern Recognition in Neuroimaging (PRNI), Singapore, 2018, pp. 1-4. doi: 10.1109/PRNI.2018.8423944

And the code to generate this data is available at: https://github.com/JessicaSchrouff/Simulated_ECoG

In this tutorial the GUI analysis is going to be slightly different from the Batch analysis in the interest of variety. The reader is advised to read the previous tutorials in case he/she wants to familiarize himself/herself more with the differences in the way we input values, names, parameters, etc. between the GUI and the Batch mode.

18.1 GUI analysis

18.1.1 Data & design

- This is a single-subject analysis, so enter one group (e.g. ‘G1’) and one subject.
- Add a new modality and give it a new name (e.g. ‘ECoG’).
- Set the type of the modality as ‘MEEG’ and select the .mat file corresponding to the ECoG simulated recording (found in the *SimulatedECoG/Data/* directory).
- Once we select the appropriate .mat file, the experimental design will automatically load from that file. It can be reviewed by clicking the ‘Events in file’ option in the ‘Design’ section (figure 18.1) where a new window will appear, like the one in figure 18.2.

Please note that there is no need to specify a mask file for the MEEG modality type as all the useful information is already contained inside the MEEG file.

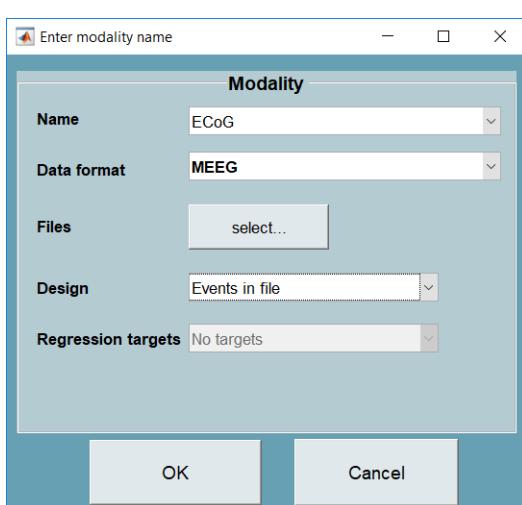


Figure 18.1: ‘Specify modality’ GUI for MEEG.

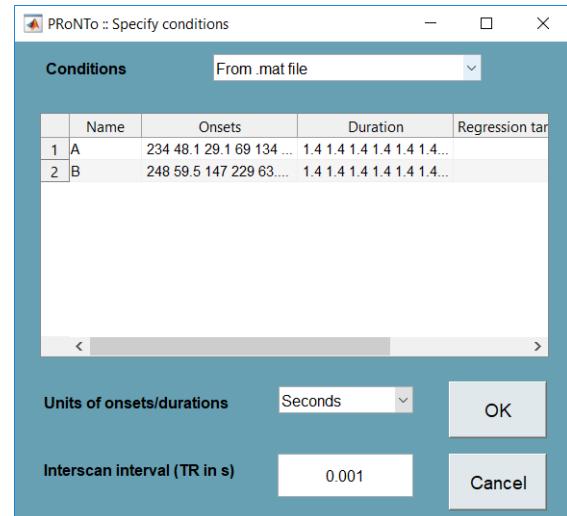


Figure 18.2: ‘Specify conditions’ GUI.

- After clicking ‘OK’ on both windows, you can save the data into a PRT.mat in the directory of your choice, and also review the data, either prior to saving them through the ‘Review’ button in the ‘Data & Design’ window, or after saving them, using the ‘Review data’ button in PRoNTo’s main window.

As we can see in figure 18.3, we have one subject in one group that contains one modality (ECoG). There are 2 conditions (A and B). The ‘selected scans’ reflect the ‘good’ epochs, while the ‘discarded scans’ reflect the epochs marked as ‘bad’ in the SPM file.

18.1.2 Prepare feature set

This step is probably the most different from all other new functionalities in v3. It reflects the different dimensions of the data and allows to ‘play’ on them. Click on ‘Prepare feature set’ in the main window and load the PRT.mat. As there is only one modality of type MEEG, the corresponding MEEG modality window will be launched directly. There are 3 main panels, corresponding to the 3 dimensions an MEEG file can have:

Channels:

- The list of channels is displayed on the left panel, as unselected. To add a channel to the feature set, click on it. It will then appear on the right panel. Alternatively, you can select ‘All’ channels or all ‘Good’ channels. On Windows OS, the ‘bad’ channels appear in red in the list.

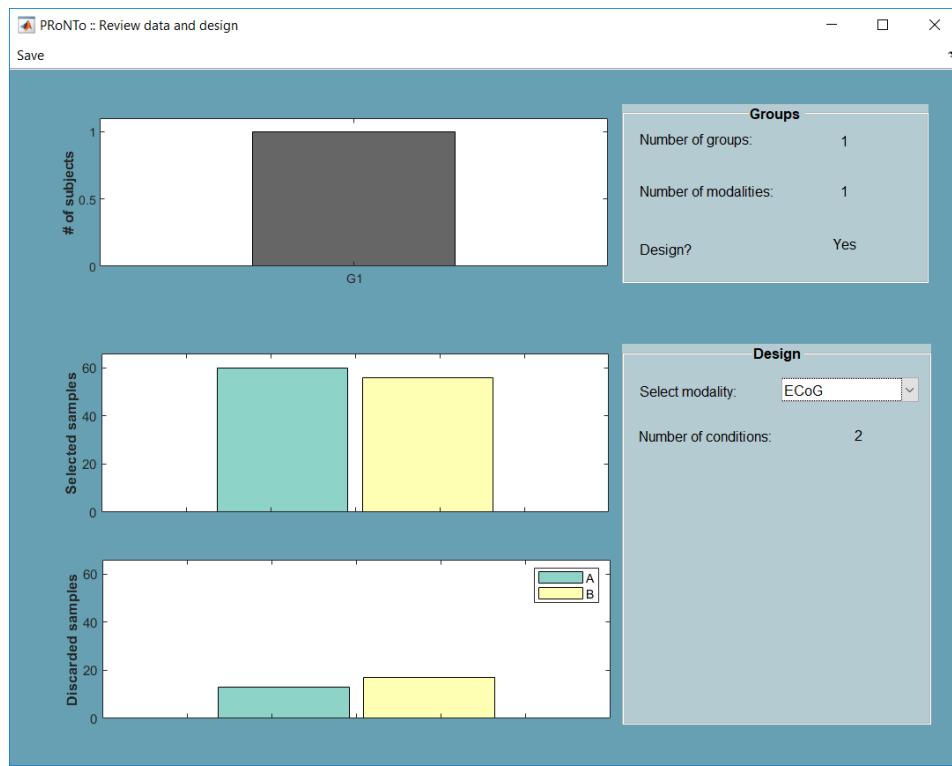


Figure 18.3: ‘Review data’ GUI.

- Average signal over channels: this option will average the signal over all selected channels. This might be useful to obtain an average trace across a specific region of interest.
- One kernel per channel: whether to build one kernel per channel (ticked) or not. This option corresponds to building an atlas with each channel as a ‘ROI’.

Time points:

- Similarly, the time window is displayed and can be amended. When entering a value in ms, PRoNTo will identify the closest corresponding time point.
- The signal can be averaged across all selected time points.
- One kernel can also be built, either per time point or per sub-windows of XX ms (user-specified). PRoNTo does not discard kernels with less than a few time points, so to avoid big imbalances in the feature numbers across kernels you should ensure you provide the right window end. For example if you want to have 100ms subwindows between -100 and 1100ms, the end time point should be 1099ms, otherwise a kernel with only one time point will be created (see batch tutorial below for example).

Frequencies:

- Similar to time points, the range of frequency bins included in the data can be viewed and specific sub-ranges selected.
- The data can be averaged over a frequency band, as specified above. This is useful to test multiple frequency bands without having to manually perform the averages as a pre-processing step.
- A kernel can also be built on each frequency bin.

Note 1: Dimensions that are not present in the 1st file are greyed out and the corresponding panel cannot be accessed (e.g. frequencies in the present case).

Note 2: It is not possible to use ‘average’ and ‘multiple kernels’ on the same dimension simultaneously.

- In the present case, let's first build a simple feature set, using all 'Good' channels and the time points between -100ms and 1100ms (keep in mind that the simulated window was added between 0 and 1000ms). Having explained all the above it will be quite straight forward to do so.
 - So, after you click the 'Prepare feature set' button and the main window where you define the feature set has appeared, you first need to select only the 'Good' channels by clicking the 'Good' button, in the 'Channels' section.
 - Then you need to change the time points in the 'Time points' section from -100ms to 1100ms. The window at this point should look like figure 18.4.

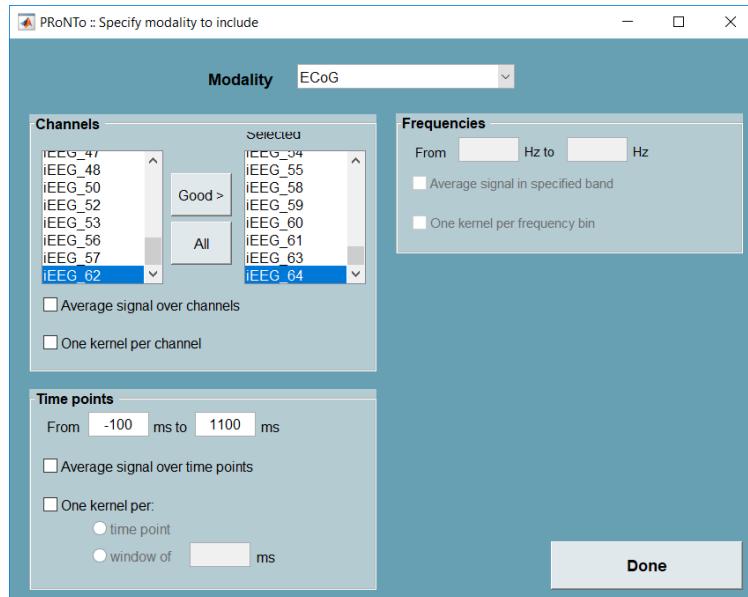


Figure 18.4: 'Specify modality to include' GUI. 'All' feature set.

- Clicking 'Done' sends us to the main feature set window, where we specify the feature set name (e.g. 'All').
- Clicking on Build kernel/data matrix builds the binary file array (as for nifti) with all the features, as well as the kernel(s) containing only the selected features. Hence, the MEEG modality window builds a second-level mask but implicitly (instead of the user specifying a file).
- We will then repeat this operation but building one kernel per channel, for later use with L1 MKL machine, and name it 'MKChans' (for Multiple Kernels on channels). As displayed in the main window, 38 kernels were built, one for each channel containing all the time points between -100 and 1100ms around onset. The final configuration of this feature set should look like figure 18.5.

18.1.3 Model: Specify new

We can now build two models discriminating A from B: one using SVM and the other using MKL on the channels. The model specification is very similar to PRoNTo v2.1 and there is nothing specific to MEEG at this stage.

- For the SVM model, first choose the appropriate PRT.mat file;
- Specify the name of our model (e.g. 'SVM');
- Choose the 'All' feature set and define the classes as 'A' modality A and as 'B' modality B. Although the imbalance is small, we also select the 'Subsample according to smallest class' option to randomly subsample 56 epochs from A to match the number of B epochs. The window should look like figure 18.6.
- Choose the SVM machine and optimize the hyper-parameter C (range: [0.01 0.1 1 10 100 1000]).

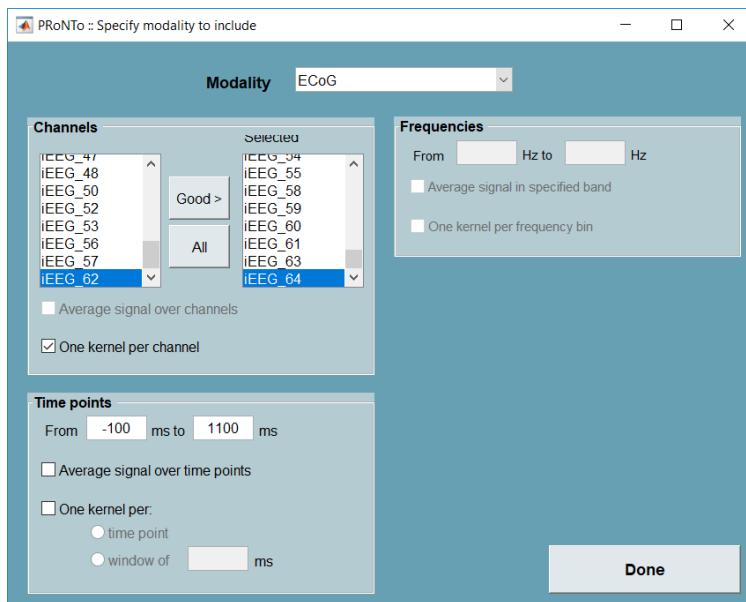


Figure 18.5: ‘Specify modality to include’ GUI. ‘MKChans’ feature set.

- The nested cross-validation scheme (inner loop) is ‘k-folds on Block per Class’ with k=4, meaning that we will leave approximately 25% of A and approximately 25% of B epochs out for testing in each fold.
- Choose the same scheme for the outer CV with k=5.
- Finally, mean centre the kernel and specify the model.

After you have specified everything, the final configuration should look like the one in figure 18.7.

18.1.4 Model: Specify from

- To specify the MKL model, you need to ‘copy’ the SVM model specifications if you want to use the same samples as the random subsample option was selected.
- In this case, click ‘Specify from’ and load the PRT.mat. The SVM model is loaded by default (as the first) and you can only modify the feature set, the machine (with hyper-parameter optimization) and/or the kernel operations.
- Use the ‘MKChans’ feature set and change the machine to ‘L1 Multi-Kernel Learning’.
- Use the same hyper-parameter optimization as before;
- But you need to add the ‘Normalize samples’ operation. Note: as the kernels all include the same number of features (i.e. the number of time points considered), this operation could also be left out.

After you have specified everything, the final configuration should look like the one in figure 18.8.

18.1.5 Model: Run

- It is now time to run our models so open the ‘Run’ module.
- After you select the ‘PRT.mat’ file you will see the 2 models we have created so far. You can either select and run them one by one, or you can select all and run them all together.
- In ‘Permutations’ select the ‘Perform permutation test’ option with 100 repetitions. Be aware that 100 repetitions is a small number when running permutation tests, and 1000 repetitions (or even more) is a more realistic one if you want your results to have a good statistical power. You could also select the ‘Save permutation parameters’ option, in case you wanted to perform further statistical tests.

The ‘Run’ window should look similar to the one in figure 17.13 of Chapter 17;

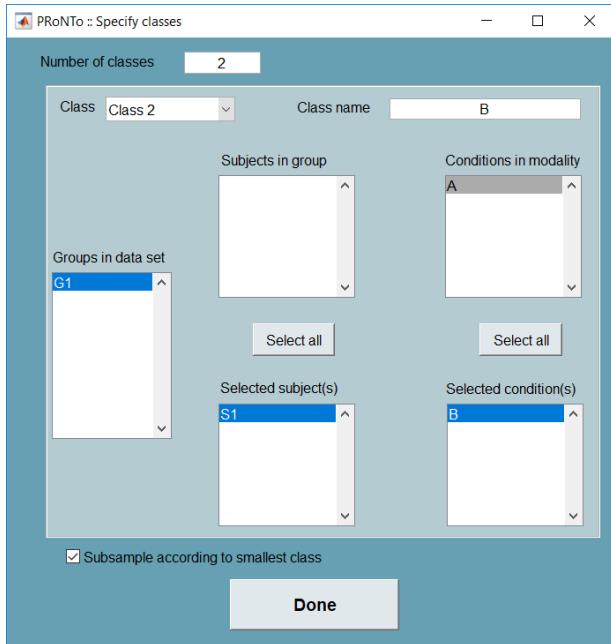


Figure 18.6: ‘Specify classes’ GUI.

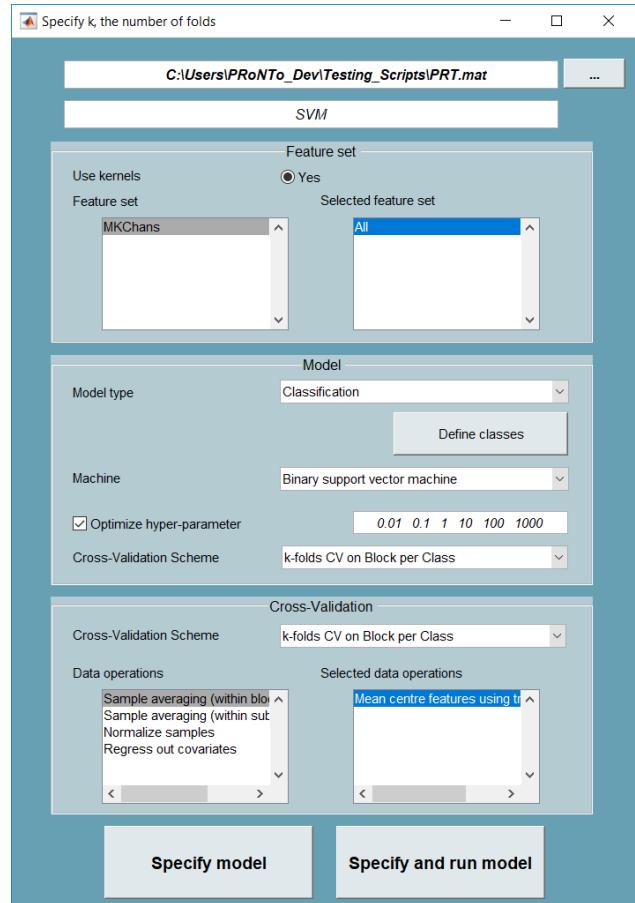


Figure 18.7: ‘Specify model’ GUI, ‘SVM’ model.

18.1.6 Display results

The SVM model leads to 93.86% balanced accuracy, while the MKL model leads to 95.72%. Their AUC is however the same (1.00). It is expected that MKL performs better as only half of the good channels contain signal of interest while the other half only contains noise. The results of the ‘SVM’ model are shown in figure 18.9.

18.1.7 Compute weights

- We can now build the weights for the SVM model. In this case, select the PRT.mat, the SVM model and leave the other options as default. Please note that weight summarization is not available for MEEG data. A new file, in MEEG format, will be built containing the weights per feature. It will have the same dimensions as the input file but contain NaN where the second-level masks did not overlap with the data (i.e. the features not selected at the feature step stage).
- Repeat the operation for the MKL model but this time tick the ‘Compute average/kernel weight per region’. Two images are then built: one with weights per feature, one with weight per kernel. Again, the dimensions are identical to the dimensions of the input file.

18.1.8 Display weights

The weight display for MEEG is very basic, just a color-coded 2D matrix if there are 2 dimensions in the file (3D files are not displayed). We refer the user to SPM (or other software after conversion) for more advanced plotting.

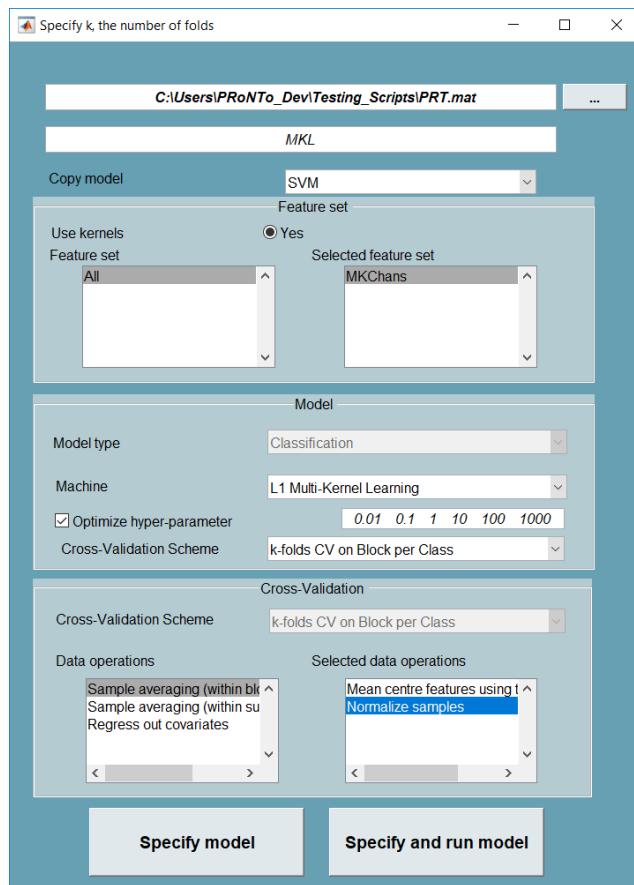


Figure 18.8: ‘Specify model’ GUI, ‘MKL’ model.

The weights for the SVM model are displayed in the following figure. As we can see in figure 18.10, channels are quite consistently either positive or negative. On the other hand, for the MKL model some channels have no contribution, as shown in figure 18.11.

The weights per regions can also be displayed and the table of ‘region’ (here channels) contributions is shown along a histogram depicting the sparsity of the solution. As we can see from figure 18.12, interestingly, the histogram shows that around half of the channels have a contribution to the model (21 channels out of 38 on average across all folds).

18.2 Batch analysis

In the Batch analysis of this tutorial we are not going to fully replicate the GUI analysis, so if one is not yet familiar with the Batch analysis, he/she should first go through the previous tutorials that are much more detailed.

18.2.1 Data & Design

- Select a ‘batch’ subfolder to write the results.
- Add a group, ‘G1’.
- Add a subject, ‘S1’.
- Add a modality ‘ECoG’.
 - Specify the modality type as ‘MEEG’.

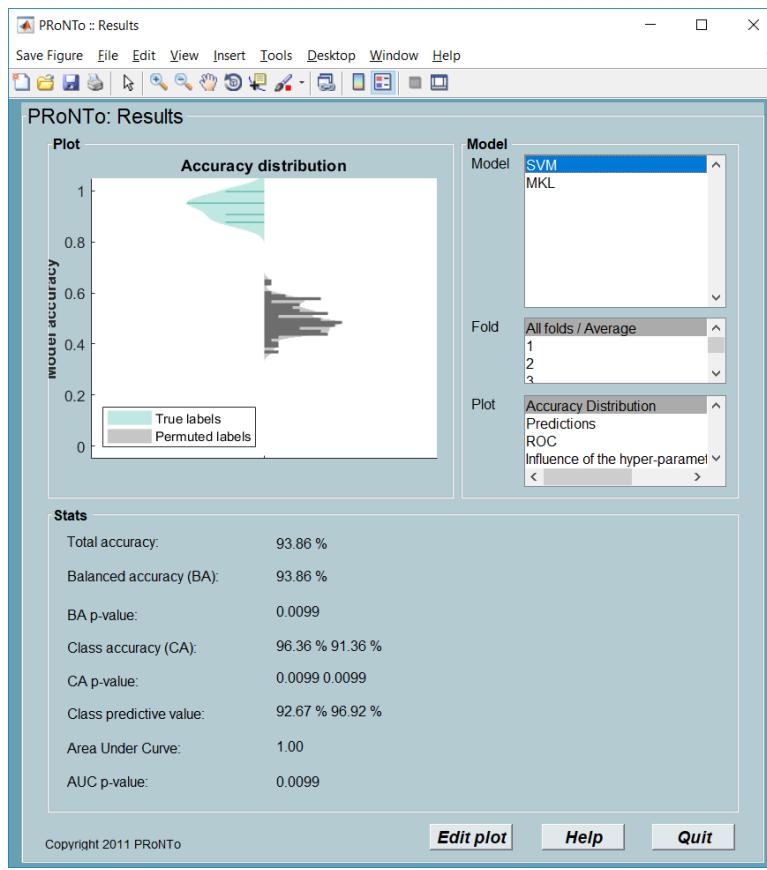


Figure 18.9: ‘Display results’ GUI..

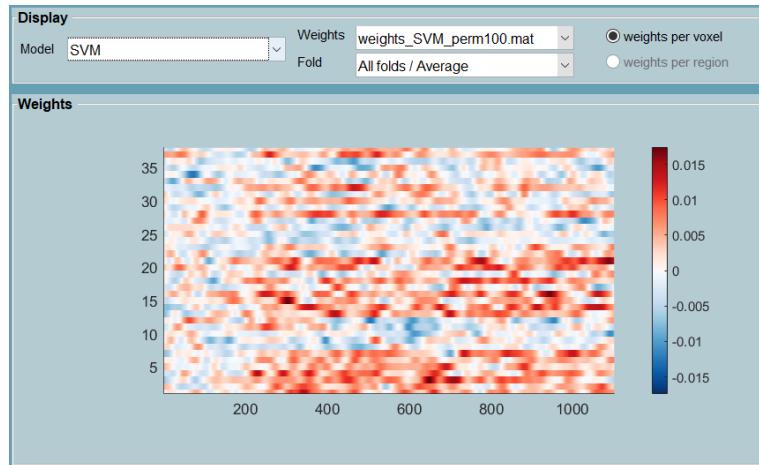


Figure 18.10: ‘Display weights’ GUI. ‘SVM’ model.

- The interscan interval as the sampling interval (here 0.001s).
- Select the file
- Under ‘Data & Design’, select ‘Events in MEEG file’. Note: this option is not selected by default as in the GUI, because the batch is agnostic of the user choices.
- No regression targets/covariates.
- In ‘Masks’ add a modality, specify the name (‘ECOG’), which needs to be exactly the same as the original name of the modality and finally set the data format to ‘MEEG’. All the important information is assumed to be already included in the MEEG file, so there is no need to specify a mask here.

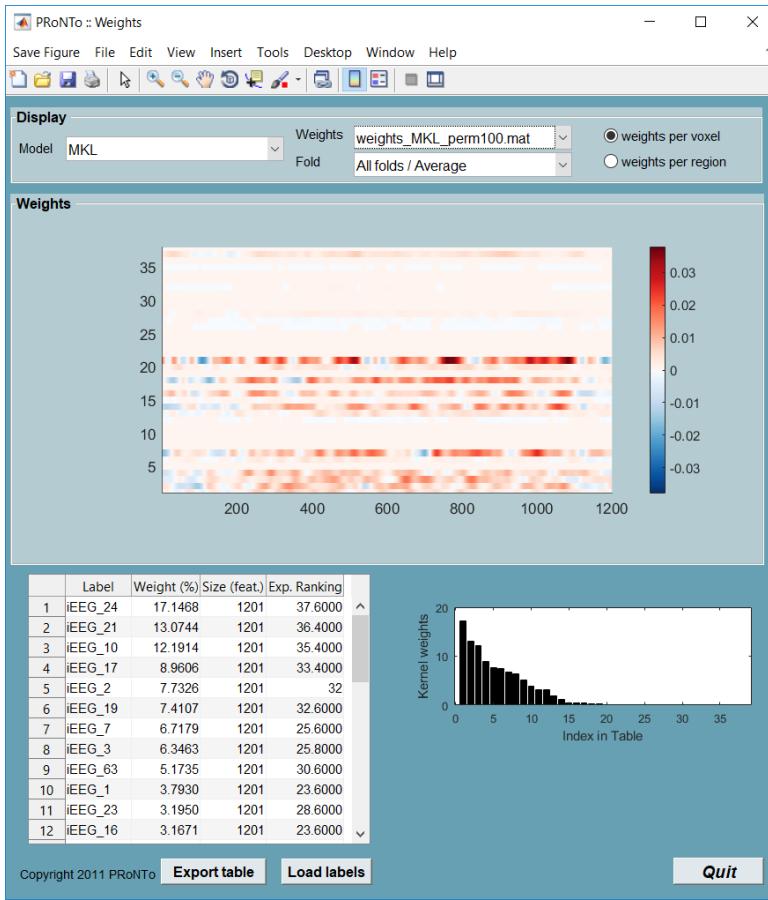


Figure 18.11: ‘Display weights’ GUI (per voxel). ‘MKL’ model.

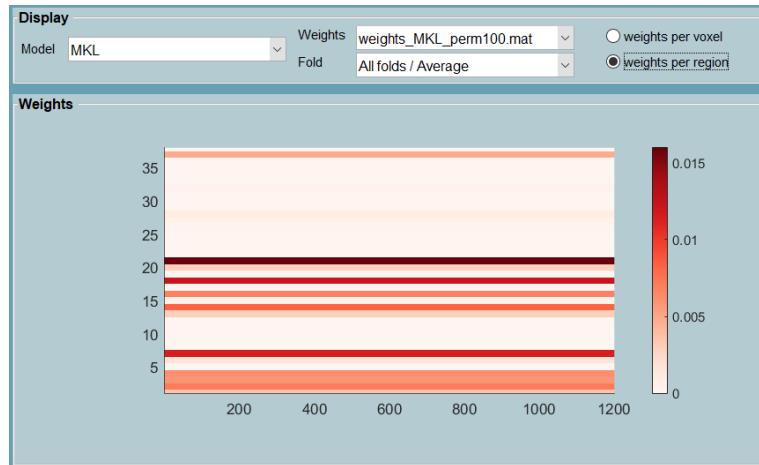


Figure 18.12: ‘Display weights’ GUI (per region). ‘MKL’ model.

After you have specified everything, the final configuration should look like the one in figure 18.13.

18.2.2 Feature set / Kernel

In this step, we will specify a feature set for our ECoG modality. This time however we will specify a model with multiple kernels to be built at the channel level, as well as for subwindows of 100ms in our window of interest, and see how it performs compared to the models we built in the GUI analysis.

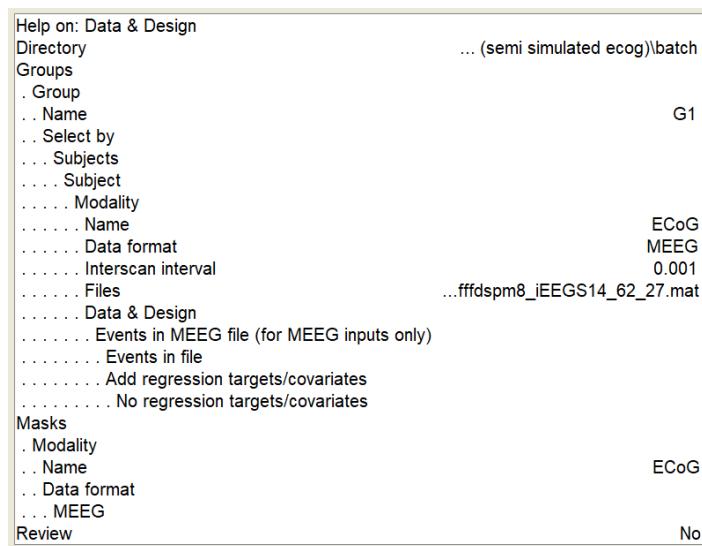


Figure 18.13: ‘Data & Design’ module in `matlabbatch`.

- First load the PRT.mat file, or create a dependency with the previous step (‘Data & Design’).
- Give a name to the feature set, e.g. ‘MKLChanstp’.
- Choose the Data format as ‘MEEG’ and add a new modality.
 - Name the modality ‘ECoG’ as defined in the previous step (‘Data & Design’).
 - **Channels:** delete the ‘All’ selected by default and replace it with the ‘New: Channel file’ option. Choose the file ‘Good_channels.mat’ provided with the data set. This file contains the labels (in a cell array) of all the channels marked as ‘good’ during preprocessing. The channel selector is the same as SPM. Please refer to SPMs help for more details. Finally choose ‘Yes’ in the ‘Multiple kernels’ option under channels.
 - **Time points:** Specify the time window as [-100 1100]. Select ‘Multiple kernels’ under time points and select ‘One kernel per time window’, entering 100ms as the time window to consider for each kernel. Please note: this will create 13 kernels, from -100 to -1, 0 to 99, 100 to 199, from 200 to 299, plus one kernel comprising the last 1100ms time point. For balanced kernels, please specify the end of your global time window (here 1100) as the end time point desired minus your sampling interval in ms (here 1). So in that case the end of your global time window would be at 1099ms.
- Leave all the rest of the options at their defaults.

After you have specified everything, the final configuration should look like the one in figure 18.14.

18.2.3 Model: Specify new

- Specify a new model (named ‘sMKLChanstp’) that accesses your PRT.mat and choose the ‘MKLChanstp’ feature set.
- As in previous versions, define 2 classes, based on conditions A and B (subsampling to match the least represented class), and choose the ‘L1 Multi-Kernel Learning’ machine.
- As this is a within-subject design, we can use the ‘k-folds on block per class’ cross-validation, for hyper-parameter optimization (inner loop) as well as for model performance evaluation (outer loop). Choose (arbitrarily) k=4 for the nested CV (inner loop) and k=5 for the outer CV (outer loop). As the kernels do not include the same number of features, it is important to add the Normalize samples operation. For a more detailed way of inputting all the options and parameters the reader is referred to Chapter 17 in the parts regarding the MEEG data format.

After you have specified everything, the final configuration should look like the one in figure 18.15.

Help on: Feature set/Kernel	DEP Data & Design: PRT.mat file
Load PRT.mat	MKLChanstp
Feature/kernel name	
Data format	
. MEEG	
. . Modality	
. . . Modality name	ECoG
. . . Channels	
. . . . Channel selection	
. . . . Channel file	...th_data\SimulatedECoG\Data\Good_channels.mat
. . . . Average	No
. . . . Multiple kernels	Yes
. . . . Time points	
. . . . Time window	[-100 1100]
. . . . Average	No
. . . . Multiple kernels	
. One kernel per time window	
. Time window (ms)	100
. . . . Frequencies	
. . . . Frequency window	[-Inf Inf]
. . . . Average	No
. . . . Multiple kernels	No

Figure 18.14: ‘Feature set / Kernel’ module in `matlabbatch`.

Help on: Model: Specify new	.../Kernel: PRT.mat file
Load PRT.mat	sMKLChanstp
Model name	
Feature sets	
. Feature set name	
. . Name	MKLChanstp
Model Type	
. Classification	
. . Classes	
. . . Class	
. . . . Name	A
. . . . Groups	
. Group	
. Group name	G1
. Subjects	1
. Conditions / Samples	
. Specify Conditions	
. Condition	
. Name	A
. Class	
. Name	B
. Groups	
. Group	
. Group name	G1
. Subjects	1
. Conditions / Samples	
. Specify Conditions	
. Condition	
. Name	B
. Subsample examples based on class definition	Yes
. . Machine Type	
. . . Kernel machine	
. . . . L1 Multi-Kernel Learning	
. . . . Machine optimization and parameters	
. Optimize hyper-parameter	
. Regularization hyper-parameter	1x6 double
. Cross-validation type for hyper-parameter optimization	
. k-folds CV on block per class	
. k	4
. Cross-validation type	
. k-folds CV on block per class	
. k	5
. Include all scans	No
. Data operations	
. Mean centre features	
. Other Operations	
. Select Operations	
. Operation	Normalize samples

Figure 18.15: ‘Model: Specify new’ module in `matlabbatch`.

18.2.4 Model: Run

The ‘Run model’ is the same as in previous PRoNTo versions. Simply load your PRT, type the name of the model as chosen in the previous step and specify whether you want to do permutation test or not (along with its options). After you have specified everything, the final configuration should look like the one in figure 18.16.

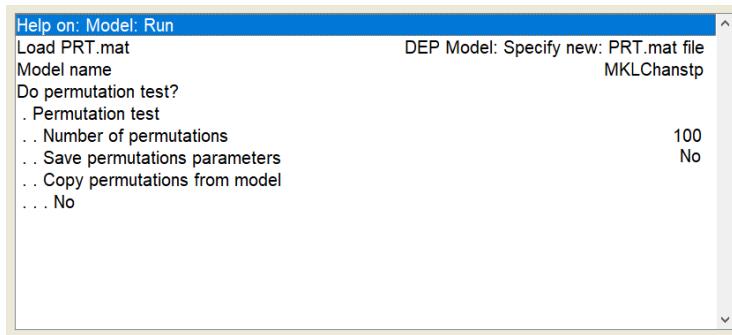


Figure 18.16: ‘Model: Run’ module in `matlabbatch`.

18.2.5 Display results

In the ‘Display Results’ GUI in figure 18.17, we see that this model does not perform as well as the previous channel MKL model that we defined on the GUI analysis.

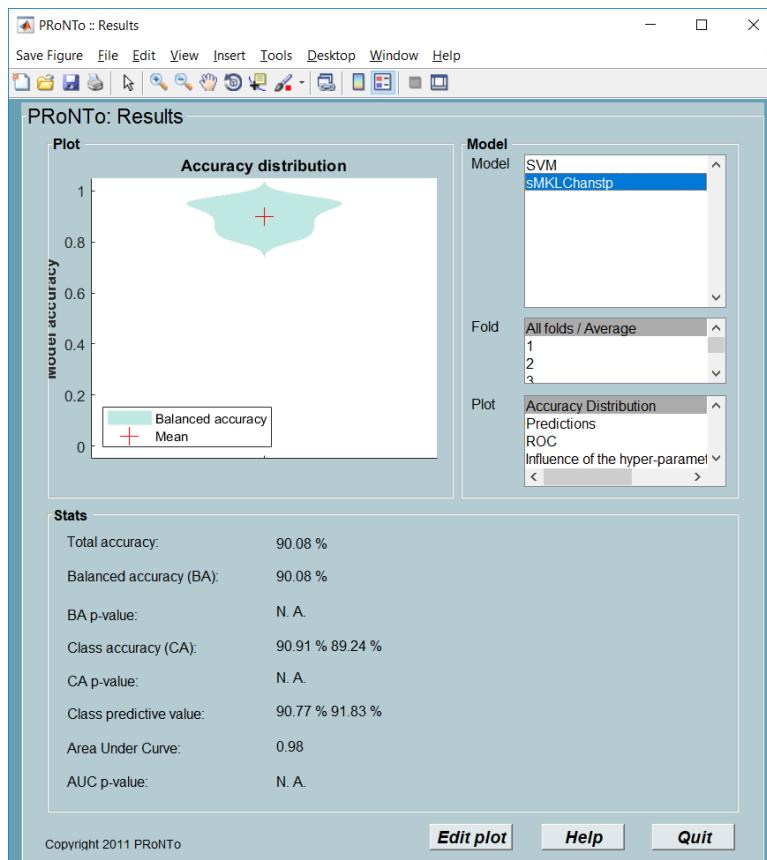


Figure 18.17: ‘Model: Run’ module in `matlabbatch`.

18.2.6 Compute & Display weights

In this last (optional) step, we're interested in the kernel contributions. More specifically, is the discrimination distributed in time or not? In the present case, we know the ground truth as from the simulated signal, the discrimination is uniform in the [0 1000]ms time window. It is hence interesting to look at kernel contributions to see if this information from the simulated signal was properly recovered.

- To this end, select ‘Compute weights’ and load the PRT.mat.
- Specify the model name as before and select the ‘Build weights per region’ option, leaving the ‘Load atlas’ as it is (blank). PRoNTo will automatically access the atlas defined at the feature set step if no atlas is specified.

After you have specified everything, the final configuration should look like the one in figure 18.18.

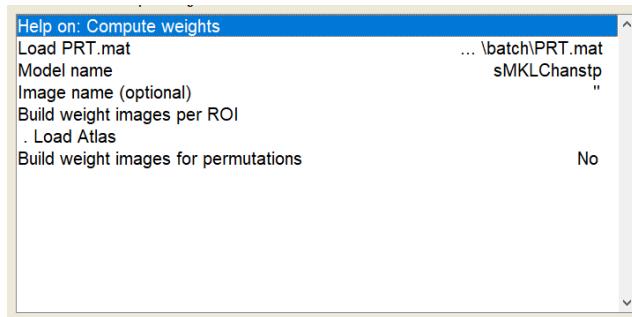


Figure 18.18: ‘Compute weights’ module in `matlabbatch`.

In the ‘Display Weights’ window, click on the model and select ‘Weights per region’. You will see the table of kernel contributions for each channel and time window (referred to by their starting time point in ms).

As we can see in figure 18.19, the MKL model mostly identifies time windows within the [0 1000]ms window of simulated signal. However, the MKL only provides a sparse solution, i.e. it does not recover all time windows that have discrimination information in the A vs B classification.

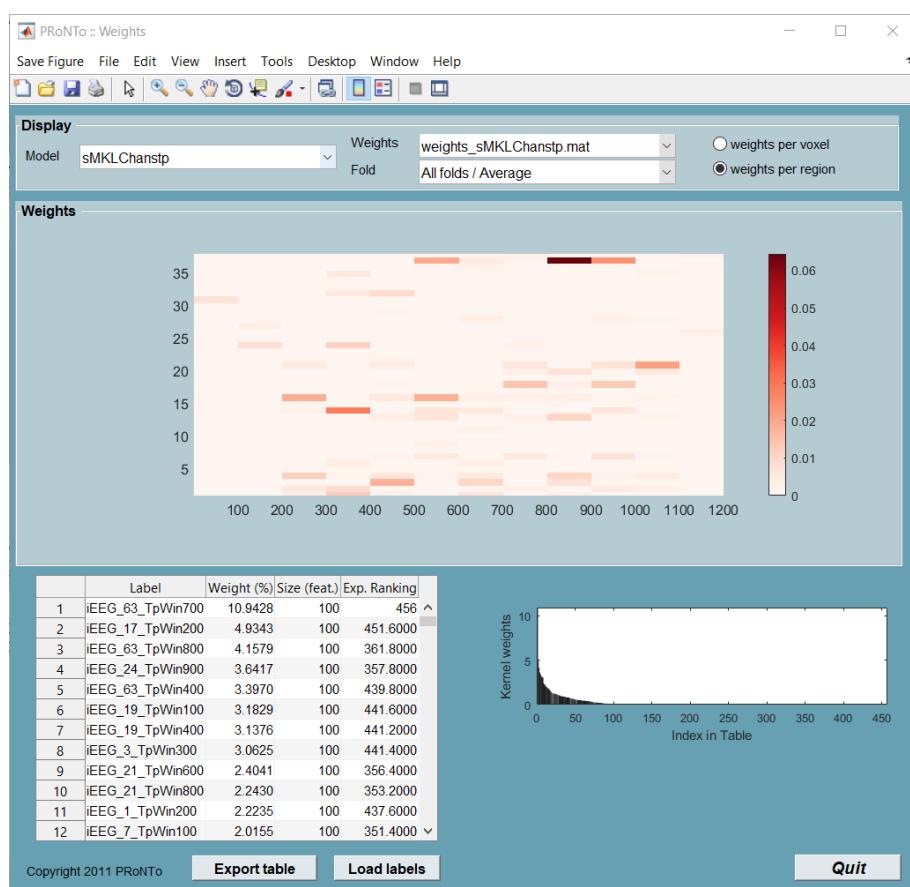


Figure 18.19: ‘Display weights’ GUI.

Chapter 19

Non-kernel machine example

Contents

19.1	GUI analysis	205
19.1.1	Data & Design	205
19.1.2	Prepare feature set	205
19.1.3	Model: Specify new	206
19.1.4	Model: Specify from	206
19.1.5	Model: Run	206
19.1.6	Display results	207
19.1.7	Compute weights	207
19.1.8	Display weights	207
19.2	Batch analysis	208
19.2.1	Data & Design	208
19.2.2	Feature set / Kernel	208
19.2.3	Model: Specify new	208
19.2.4	Model: Specify from	208
19.2.5	Model: Run & Display results	210
19.2.6	Compute & Display weights	210

In this example, we will use the same dataset used in Chapter 8 and a non-kernel machine to classify conditions ‘A’ and ‘B’. As non-kernel machines build the model based on the original features and not based on the similarity between samples they can be computationally expensive if the number of features is very large. However, some algorithms only exist in primal form and could be of interest (for example LASSO, Tibshirani, 1996). In this tutorial, we will select a single channel (one on which there is signal) of the same ECoG data we used in the previous chapter and use the L1-SVM machine that selects a subset of informative features during model estimation.

This tutorial will be quite short as most information is already introduced in chapter 18 and in this tutorial we build on top of that. So the reader is strongly advised to do that tutorial first before proceeding to this one.

19.1 GUI analysis

19.1.1 Data & Design

You will need to replicate, exactly, the procedure followed in the previous chapter. Follow step by step subsection 18.1.1 in order to finish inputting the data.

19.1.2 Prepare feature set

- First replicate, exactly, the procedure followed in the previous chapter in order to build the first feature set, ‘All’, used in that chapter. Follow step by step subsection 18.1.2 in order to do this.

- Then build a second feature set ('Chan3'), that will include only channel 3 and time points the same as before, [-100 1100]ms around onset. The final configuration of this feature set should look like figure 19.1

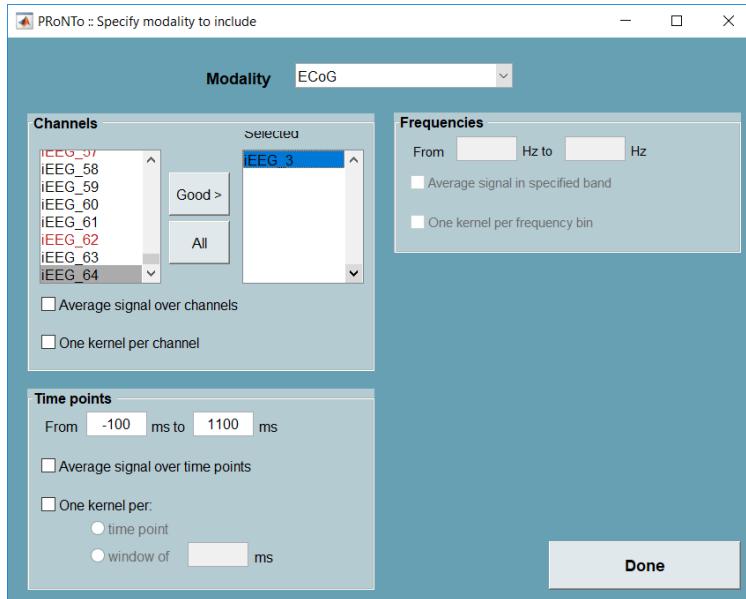


Figure 19.1: 'Prepare feature set' GUI for the 'Chan3' feature set.

19.1.3 Model: Specify new

Replicate, exactly, the procedure followed in the previous chapter in order to build the first of the two models, 'SVM', used in that chapter. Follow step by step subsection 18.1.3 in order to do this.

19.1.4 Model: Specify from

- Choose the 'SVM' model as a basis (default).
- Remove the selected feature set and instead select our new 'Chan3' feature set.
- Untick the 'Use kernels' radio button. The 'machine' list will be updated and now includes Binary L2-SVM, Binary L1-SVM, Multiclass SVM, L2-Logistic Regression and L1-Logistic Regression (all interfaced from LIBLINEAR).
- Select 'Binary L1-SVM' and optimize the hyper-parameter using a 'k-folds on Block per Class' cross-validation with k=4.
- Two new options appear in 'Data operations' for non-kernel methods: 'Normalize features' or 'Z-score features'. Lets z-score and finally specify the model.

The final configuration should look like figure 19.2

19.1.5 Model: Run

- It is now time to run our models so open the 'Run' module.
- After you select the 'PRT.mat' file you will see the 2 models we have created so far. You can either select and run them one by one, or you can select all and run them all together.
- In 'Permutations' select 'Perform permutation test', with 100 repetitions. Be aware that 100 repetitions is a small number when running permutation tests and we are only using it for demonstration purposes. 1000 repetitions (or even more) is a more realistic number if you want your results to have a good statistical power.

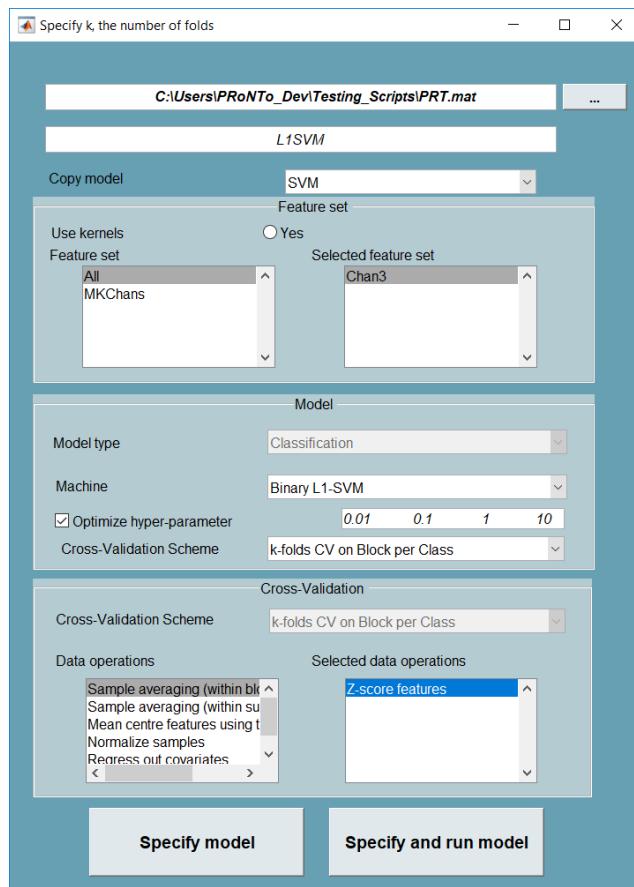


Figure 19.2: ‘Model: Specify from’ GUI for the new ‘L1SVM’ model.

The ‘Run’ window should look similar to the one in figure 17.13 of Chapter 17 with the ‘Save permutation parameters’ option unchecked.

19.1.6 Display results

The ‘L1SVM’ model gives 67.21% total accuracy and a AUC of 0.76, which is worse than the kernel ‘SVM’ model that had a 93.86% total accuracy and a AUC of 0.99. The results for the ‘L1SVM’ model are shown in figure 19.3.

19.1.7 Compute weights

We can now build the weights for the ‘L1SVM’ model. In this case, select the PRT.mat, the ‘L1SVM’ model and leave the other options as default.

19.1.8 Display weights

The weights for the ‘L1SVM’ model are displayed in figure 19.4. As this is an L1-SVM, only a few time points on channel 3 should have non-null weights. Even though the L1SVM provides sparse weights for each fold, the average weight across folds, shown in figure 19.4, is not sparse. This reflects the fact that different features are being selected across folds therefore the selection of features across fold is very unstable.

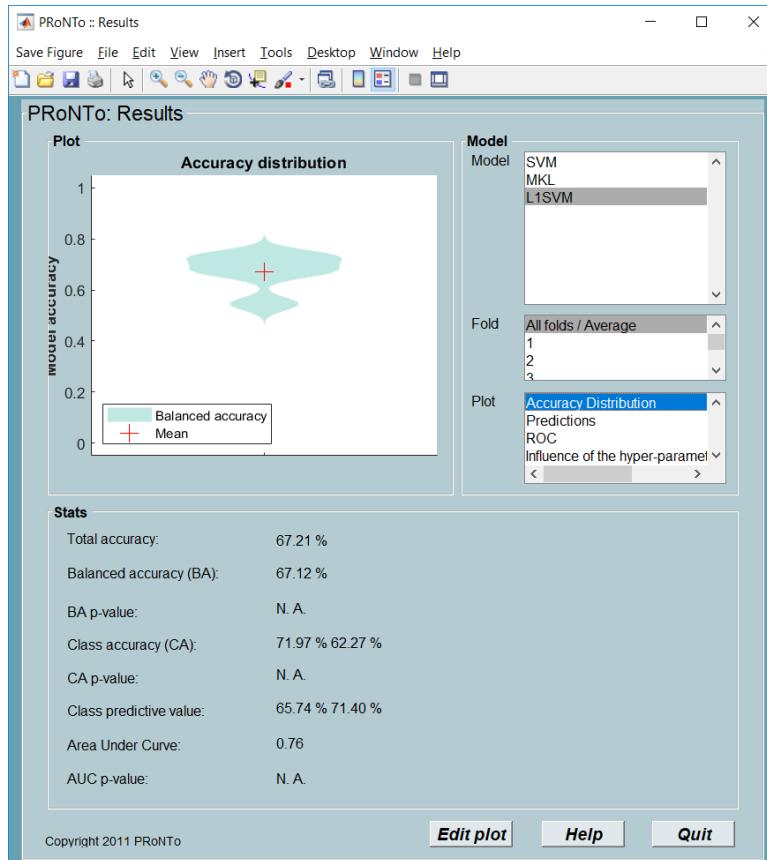


Figure 19.3: ‘Display results’ GUI for the new ‘L1SVM’ model.

19.2 Batch analysis

19.2.1 Data & Design

You will need to replicate, exactly, the procedure followed in the previous chapter. Follow step by step subsection [18.2.1](#) in order to finish inputting the data.

19.2.2 Feature set / Kernel

- First replicate, exactly, the procedure followed in the previous chapter in order to build the first of the two feature sets, ‘All’, used in that chapter. Follow step by step subsection [18.2.2](#) in order to do this.
- Then build a second feature set (‘Chan3’), that will include only channel 3 and time points the same as before, [-100 1100]ms around onset. The final configuration of this feature set should look like figure [19.5](#)

19.2.3 Model: Specify new

Replicate, exactly, the procedure followed in the previous chapter in order to build the first of the two models, ‘SVM’, used in that chapter. Follow step by step subsection [18.1.3](#) in order to do this.

19.2.4 Model: Specify from

We will try the ‘Model: Specify from’ `matlabbatch` module and the Binary L2-SVM machine this time. Remember that in the ‘Model: Specify from’ `matlabbatch` module you only need to specify the fields you’re modifying from the copied model, and all the rest will remain the same.

- Specify the appropriate ‘PRT.mat’.

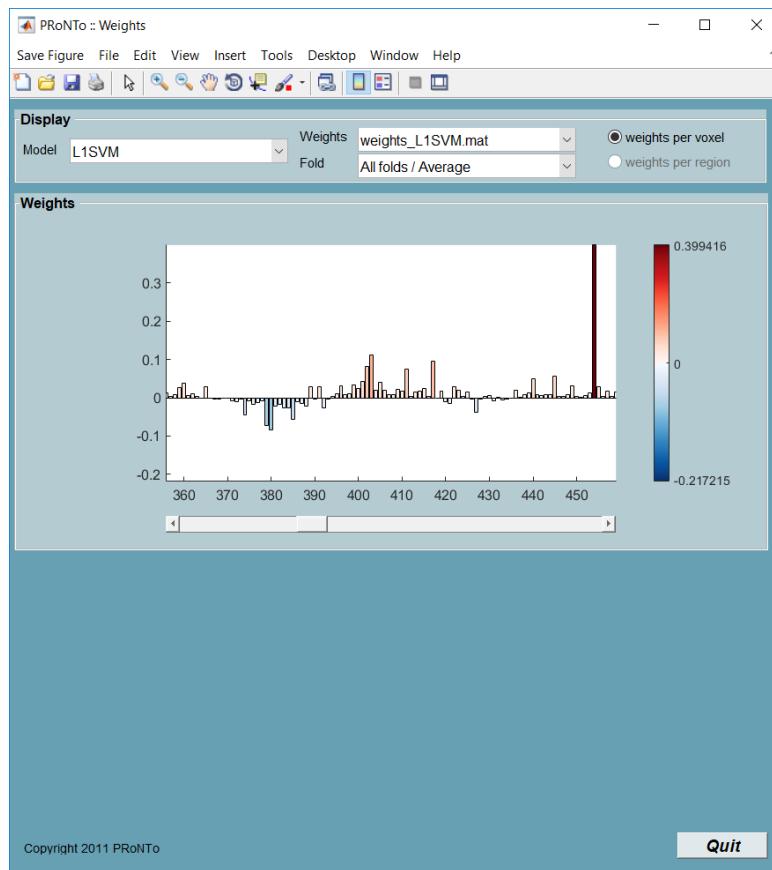
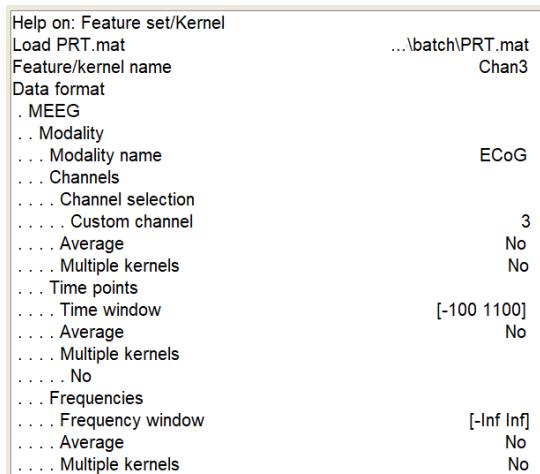


Figure 19.4: ‘Display weights’ GUI. ‘L1SVM’ model.

Figure 19.5: ‘Feature set / Kernel’ module in `matlabbatch` for the ‘Chan3’ feature set.

- Give a name to the new model (‘L2SVM’) and choose the model to copy from (‘SVM’).
- In the ‘Fields to modify’ select the ‘New: Field’ option.
 - In the ‘Field’, select the ‘Model type’ option, then choose ‘Classification’, ‘Non-kernel machine’ and select the ‘Binary L2-SVM’.
 - In the ‘Machine optimization and parameters’, select ‘Optimize hyper-parameter’ and choose ‘k-folds CV on block per class’ with a k=4 in the ‘Cross-validation type for hyper-parameter optimization’ field.

- In the ‘Fields to modify’ select once more the ‘New: Field’ option.
 - In the ‘Field’, select the ‘Feature sets’ option, select a ‘New: Name’ and specify it (‘Chan3’).
- In the ‘Fields to modify’ select once more the ‘New: Field’ option.
 - In the ‘Field’, select the ‘Data operations’ option. Then in ‘Other operations’ select ‘Select operations’, select the ‘New: Operation’ and choose the ‘Z-score features (Non-kernel only)’ option.

After you have specified everything, the final configuration should look like the one in figure 19.6.

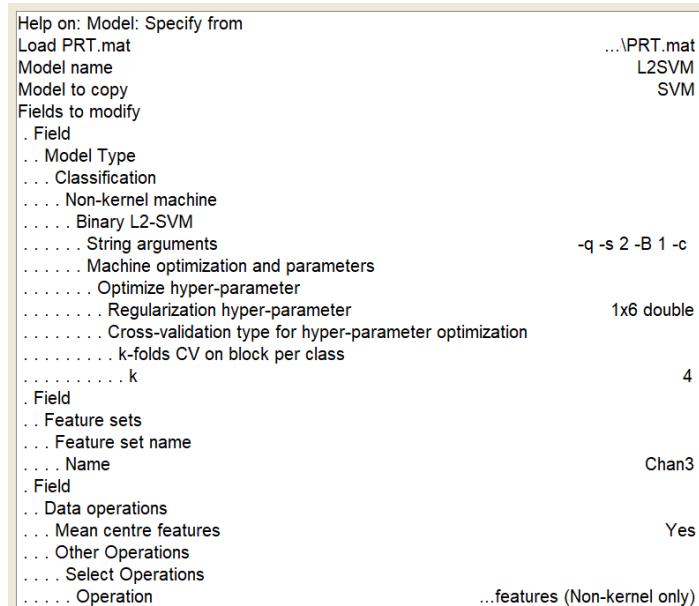


Figure 19.6: ‘Model: Specify from’ module in `matlabbatch`.

19.2.5 Model: Run & Display results

Simply load your ‘PRT.mat’ and run all the models with no permutations. The model displays an accuracy very similar to the ‘L1SVM’ model.

19.2.6 Compute & Display weights

Open the ‘Compute weights’ `matlabbatch` module, load the PRT.mat, specify the model name (‘L2SVM’) and leave all the rest at their default values.

In figure 19.7 you see that the weights for this model are quite smooth compared to the L1-norm. That was expected since the L2-norm produces non-sparse weights.

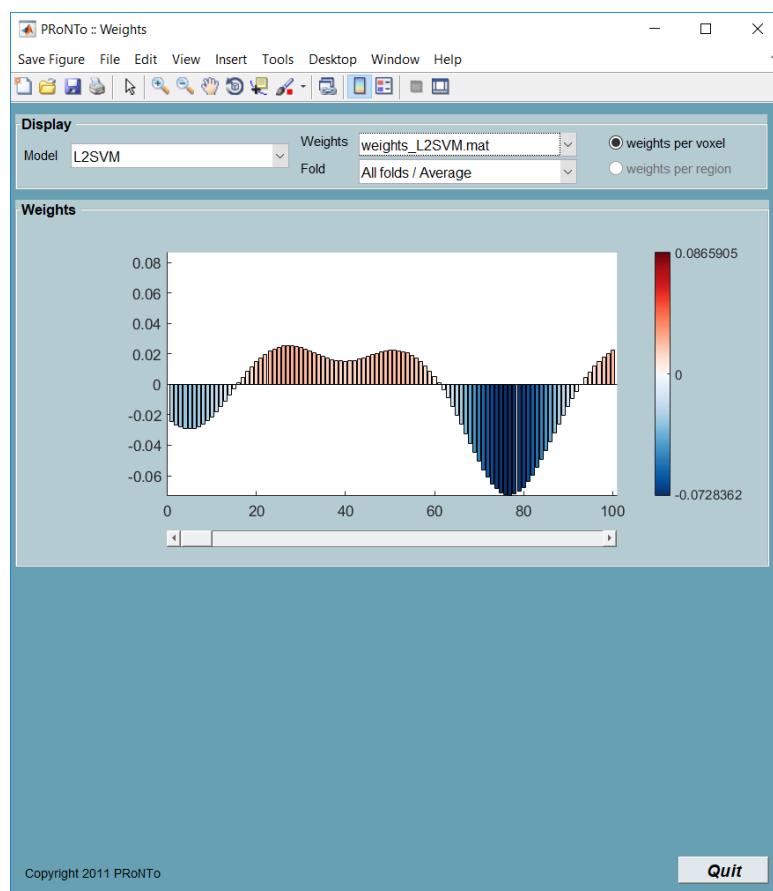


Figure 19.7: ‘Display weights’ GUI for the ‘L2SVM’ model.

Chapter 20

Within-subject Regression

Contents

20.1	GUI analysis	214
20.1.1	Data & Design	214
20.1.2	Prepare feature set	215
20.1.3	Model: Specify new	215
20.1.4	Model: Run	216
20.1.5	Display results	217
20.2	Batch analysis	218
20.2.1	Data & Design	218
20.2.2	Feature set / Kernel	218
20.2.3	Model: Specify new	219
20.2.4	Model: Run & Display results	220

In this example, we will use the semi-simulated ECoG data with random regression targets and covariates to illustrate within subject regression with MEEG data. Most of what follows also applies to nifti and .mat although there are small differences at the Data and Design stage.

For each condition, we will create random targets by using a fixed number (1 for condition A, 3 for condition B) to which we add uniform random noise between 0 and 1. There are 73 epochs for condition A, so for trials of condition ‘A’ we have:

```
> rt_trial = ones(1, 73) + rand(1, 73);
```

And for trials of condition ‘B’:

```
> rt_trial = 3 * ones(1, 73) + rand(1, 73);
```

The covariates will then model the ground truth of those targets, such that when removing confounds, we would remove all target differences between the 2 conditions. We can either simply use

```
> R = ones(73, 1); save('Confound_A.mat', 'R') % for condition A
```

```
> R = 3 * ones(73, 1); save('Confound_B.mat', 'R') % for condition B
```

On the other hand, we could consider that this is actually a categorical covariate, i.e. being from condition A or from condition B. This needs to be one-hot encoded according to:

```
> R = repmat([10], 73, 1); save('Confound_A.mat', 'R') % for condition A
```

```
> R = repmat([01], 73, 1); save('Confound_B.mat', 'R') % for condition B
```

In the present ‘dummy’ application, this doesn’t make much of a difference and both options correctly remove the main condition effect. In practice, we recommend using one-hot encoding of categorical variables when these are clearly unordered.

20.1 GUI analysis

20.1.1 Data & Design

- Starting as before, create one group ‘G1’, add one subject and one new modality called ‘ECoG’.
 - Set the type of the modality as ‘MEEG’ and select the .mat file corresponding to the ECoG simulated recording (found in the *SimulatedECoG/Data/* directory).
 - Once we select the appropriate .mat file, the experimental design will automatically load from that file. It can be reviewed by clicking the ‘Events in file’ option in the ‘Design’ section (figure 18.1 from chapter 18) where a new window will appear, like the one in figure 18.2 from chapter 18.
 - As you see the condition names, onsets and durations have already been filled. There are 2 more columns that can be edited: the 4th corresponds to regression targets per trial and the last to covariates per trial. As the design is automatically loaded from the MEEG data, these 2 fields have to be entered manually, for each file in the design. The expressions above can be entered in the corresponding table cells to obtain a design similar to (when using the value covariates) the one of figure 20.1.

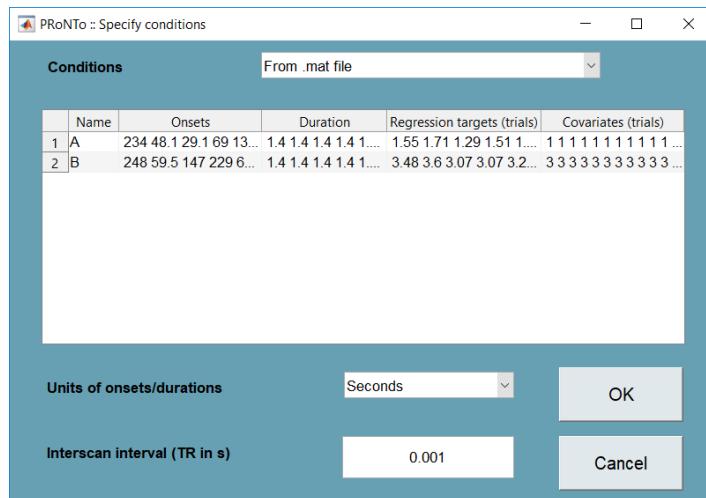


Figure 20.1: ‘Specify conditions’ GUI.

Important notes:

- The number of targets must correspond to the number of trials in each condition, including bad trials for MEEG data (hence the 73 instead of 60 and 56).
 - Multiple covariates can be entered in the form of a # trials x # covariates matrix. As the table can only display one line, the matrix is vectorised before display. This does not affect the matrix in the PRT structure.
 - For .mat or nifti formats, a file can be specified to fill in the whole table, instead of manually inputting the regression targets and the covariates. This file should contain the ‘names’, ‘onsets’ and ‘durations’ variables (as in previous PRoNTo versions), as well as a ‘rt_trial’ and ‘R’ cell arrays, with one cell per condition.

Please note that there is no need to specify a mask file for the MEEG modality type as all the useful information is already contained inside the MEEG file.

- After clicking ‘OK’ on both windows, you can save the data into a PRT.mat in the directory of your choice, and also review the data, either prior to saving them through the ‘Review’ button in the ‘Data & Design’ window, or after saving them, using the ‘Review data’ button in PRoNTo’s main window.

You can check the values were passed correctly by typing:

```
> load('PRT.mat')
> PRT.group.subject.modality.design.conds(2).cov_trial % or rt_trials
```

20.1.2 Prepare feature set

The feature set you’ll be using is the ‘MKChans’ feature set from chapter 18. So follow step by step subsection 18.1.2 in order to replicate, exactly, the procedure followed in the that chapter in order to build the ‘MKChans’ feature set that is used there. The final configuration of this feature set should look like figure 18.5.

20.1.3 Model: Specify new

You will first build a new model to perform the within-subject regression and see whether a pattern can be found for this ‘artificial’ regression model. Hence, you will need to build one model with the covariates, and one without them and compare.

- Choose the appropriate PRT.mat file.
- Specify the name of our model (e.g. ‘RegKRRnoCov’).
- Choose the ‘MKChans’ feature set and change the ‘Model type’ to ‘Regression’.
- Click ‘Select Subjects/scans’, add subject S1 and both conditions A and B to the model (i.e. we want to include all epochs in the regression model) and click ‘Done’. The window should look like figure 20.2.

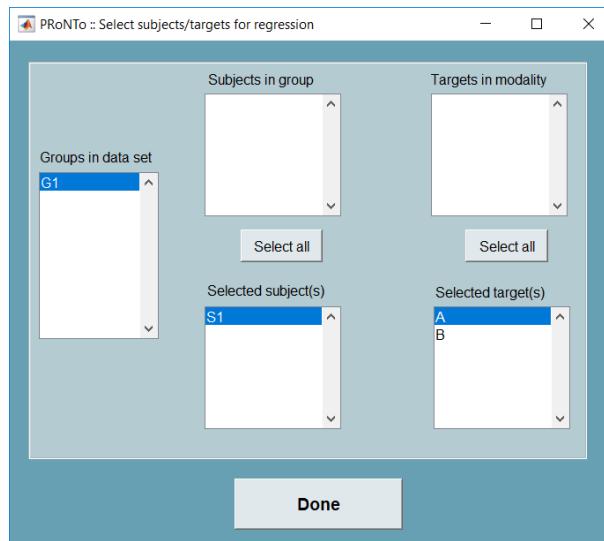


Figure 20.2: ‘Select Subjects/targets for regression’ GUI.

- Choose the Kernel Ridge Regression machine and optimize the hyper-parameter C (range: [0.01 0.1 1 10 100 1000]).
- The nested cross-validation scheme (inner loop) is ‘k-folds on Block’ with k=4.
- Choose the same scheme for the outer CV with k=5.
- Finally, mean centre the kernel and specify the model.

After you have specified everything, the final configuration should look like the one in figure 20.3.

Next you will need to create a second model, similar to the first one, with the main difference that this time you will regress out the covariates, in order to compare and contrast the differences of the two models.

- So similarly create another model called RegKRRcov where you replicate everything from the first model.
- You only have to add the ‘Regress out covariates’ operation. Please note that we could not have used the ‘Model:Specify from’ option as the covariates are only loaded in the model if the ‘Regress out covariates’ operation is selected.

After you have specified everything, the final configuration should look like the one in figure 20.4.

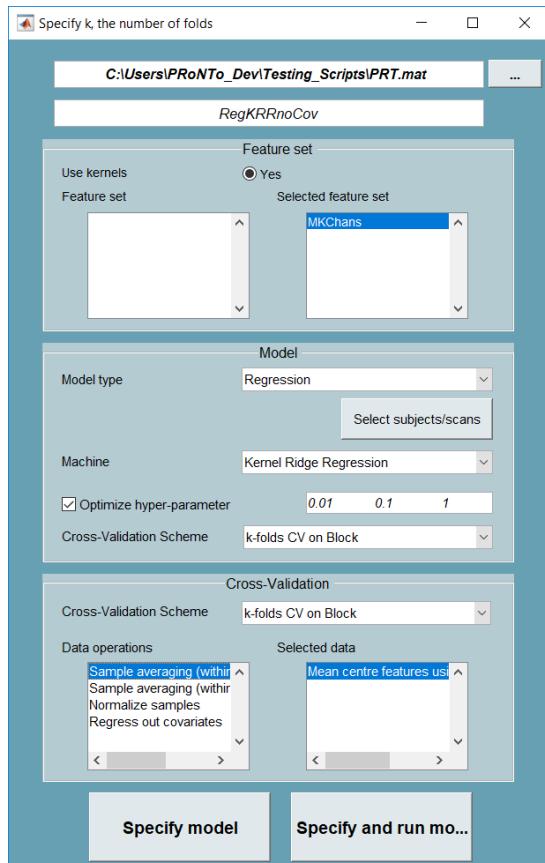


Figure 20.3: ‘Specify model’ GUI.
‘RegKRRnoCov’ model.

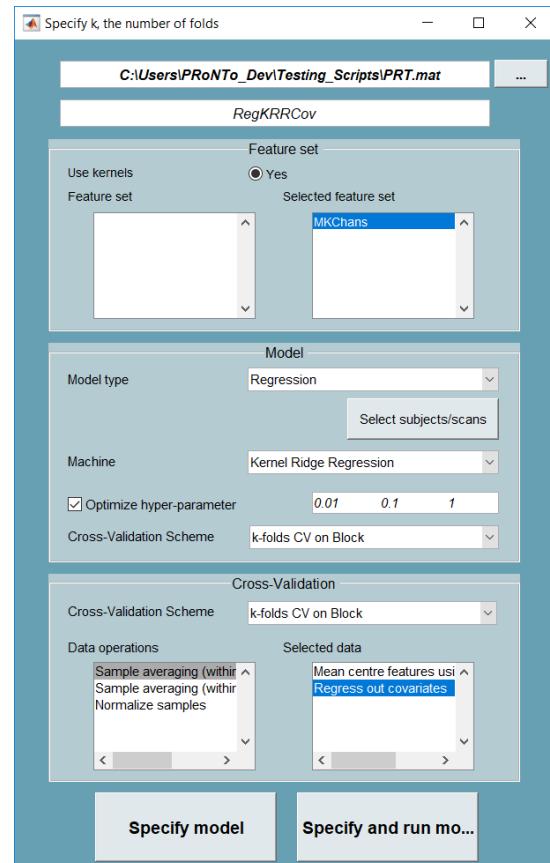


Figure 20.4: ‘Specify model’ GUI.
‘RegKRRcov’ model.

20.1.4 Model: Run

- It is now time to run our models so open the ‘Run’ module.
- After you select the ‘PRT.mat’ file you will see the 2 models we have created so far. You can either select and run them one by one, or you can select all and run them all together.
- In ‘Permutations’ select ‘Perform permutation test’, with 100 repetitions. Be aware that 100 repetitions is a small number when running permutation tests and we are only using it for demonstration purposes. 1000 repetitions (or even more) is a more realistic number if you want your results to have a good statistical power.

The ‘Run’ window should look similar to the one in figure 17.13 of Chapter 17 with the ‘Save permutation parameters’ option unchecked.

20.1.5 Display results

The results for both models are displayed in figure 20.5. The first model somewhat identifies that trials from condition A have lower regression targets than epochs from condition B. We see however that the model predicts a relatively central value for all epochs, close to 2.5 (the average between all targets when taking the random noise into account). Removing the covariates leads to a slightly worse model, that clearly predicts the target average all of the time.

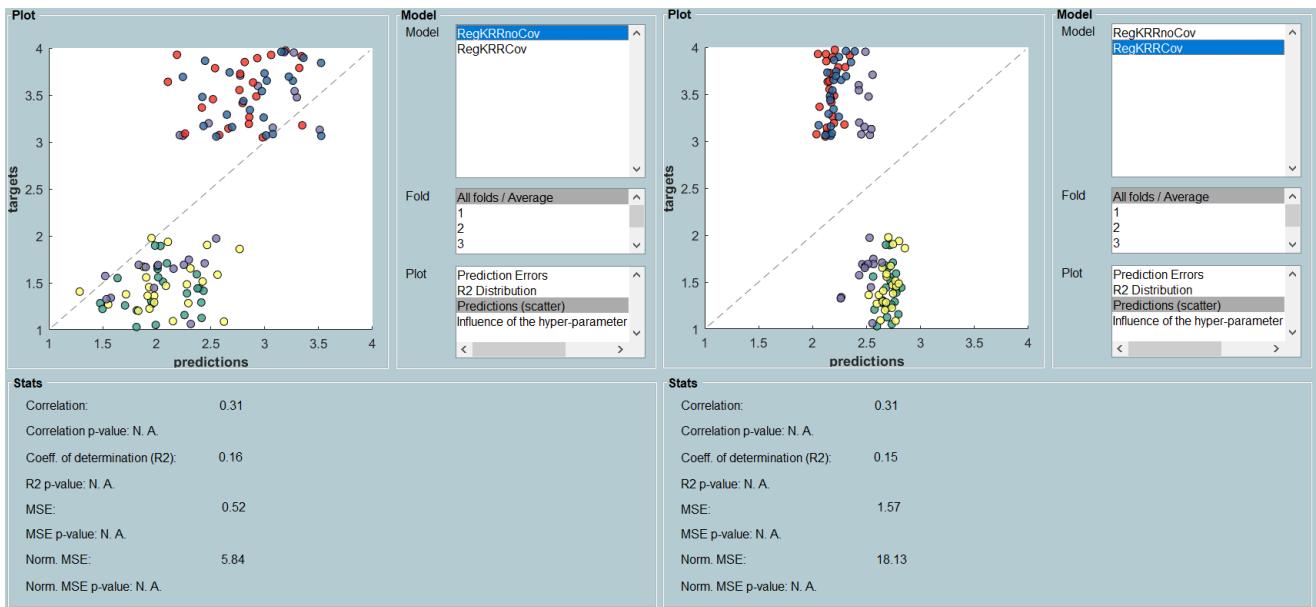


Figure 20.5: ‘Display results’ GUI for regression (left) and for the same regression after having removed confounds that are heavily correlated with the targets (right).

As for classification, the plots have been modified according to the emphasis that ‘All folds’ only reflects the average across the different folds (i.e. different models estimated). A ‘Prediction Errors’ plot was added to reflect, for each point, the difference between the target and the predictions. A perfect regression model would lead to all points being located on the ‘0’ vertical axis.

Similarly to the ‘Accuracy distribution’ plot, a ‘R2’ plot displays the distribution of R2 across the different folds in a violin plot. The plot is symmetric if no permutations were performed while it will contain the ‘true label’ R2 distribution on its left and the ‘permuted label’ R2 distribution on its right if permutations were estimated (figure 20.6).

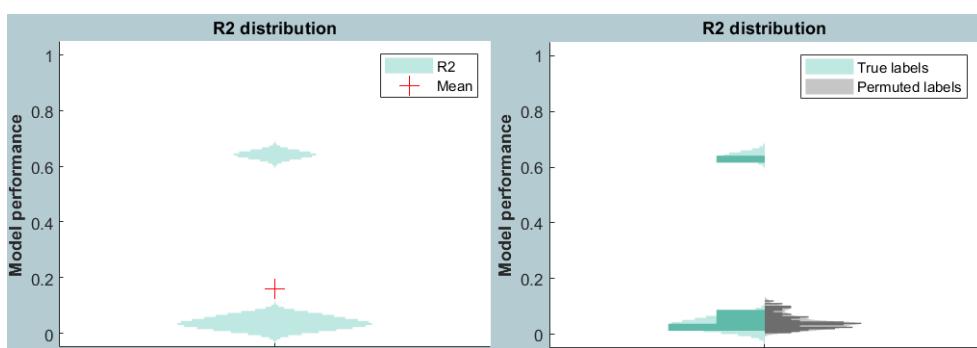


Figure 20.6: R2 distribution plot without (left) and with (right) permutations.

Important note: The pre-processing steps sort the 2 conditions A and B. This means that the targets are also sorted, with all the lower values in the 1st half of the samples and the higher values in the 2nd half. This clearly

affects the cross-validation: in the first fold, we only leave lower values out, making the train and test sets very imbalanced. Hence, care should be taken to avoid this kind of situations and to ensure that the cross-validation train and test sets are balanced in terms of target variances and ranges. This of course was not an issue during the classification as we were able to use the ‘k-folds on Blocks per Class Out’ cross-validation scheme, leading to balanced train and test sets.

20.2 Batch analysis

20.2.1 Data & Design

- First, replicate exactly, the procedure followed in Chapter 18. Follow step by step subsection 18.2.1 in order to finish inputting the data.
The only thing you need to modify is to add the covariates and the regression targets.
- Go to the ‘Add regression targets/covariates’ field, select the ‘Conditions’ option and add a ‘New: Condition’.
- For each condition, you then need to:
 - Specify the conditions name (such that PRoNTo knows which condition to relate the inputs to, e.g. ‘A’)
 - Input the regression targets as an expression (a vector or expression, e.g. ‘ones(1,73)+rand(1,73)’)
 - And finally load the covariates from a file (saved in a variable called ‘R’ as described in the start of this chapter).

After you have specified everything, the final configuration should look like the one in figure 20.7.

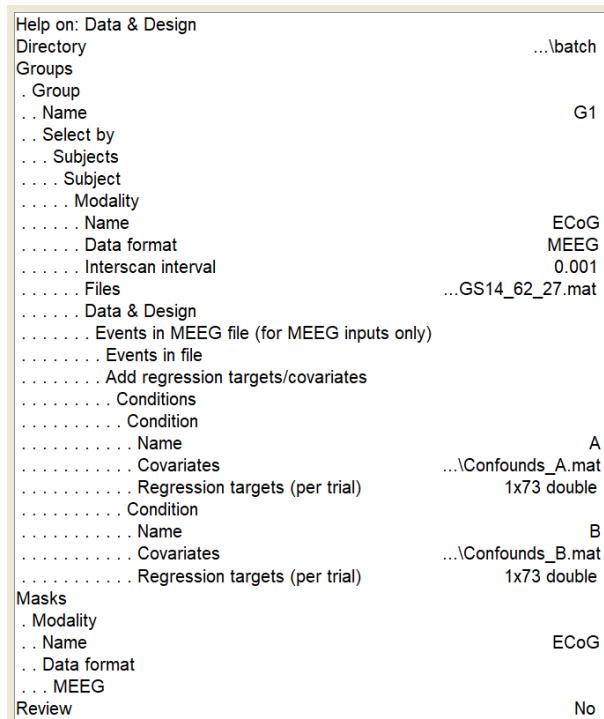


Figure 20.7: ‘Data & Design’ module in `matlabbatch`.

20.2.2 Feature set / Kernel

As for the GUI analysis, there is nothing specific to within-subject regression at the feature set level and the instructions from the previous chapters can be followed.

- First replicate, exactly, the procedure followed in the chapter 18 in order to build the first of the two feature sets, ‘All’, used in that chapter, but name it ‘MKChans’. Follow step by step subsection 18.2.2 in order to do this.
- The only thing you have to modify is the ‘Multiple kernels’ field, where you need to change this from ‘No’ to ‘Yes’.

The final configuration of this feature set should look like figure 20.8.

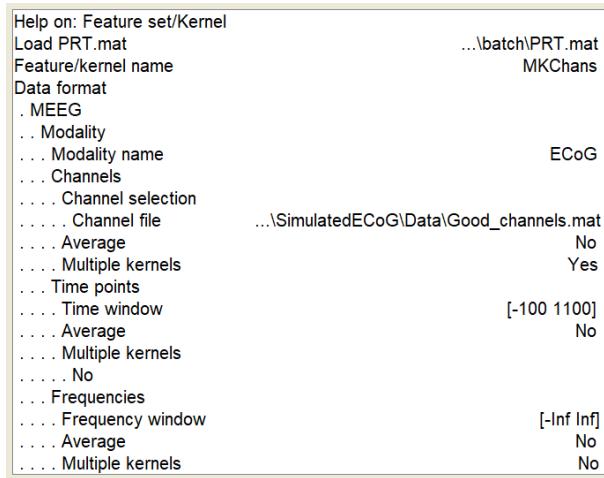


Figure 20.8: ‘Feature set / Kernel’ module in `matlabbatch`.

20.2.3 Model: Specify new

As in the GUI analysis, you have to specify two versions of the same model: one without additional operations and one removing the confounds.

- Specify a new model (‘RegGRnoCov’) that accesses your PRT.mat and choose the ‘MKLChans’ feature set.
- Change the ‘Model type’ to ‘Regression’, add a new group (‘G1’) and since we have only 1 subject type ‘1’ in the ‘Subjects’ field.
- In the ‘Conditions / Samples’ select ‘All Conditions’.
- In the ‘Machine Type’ select the ‘Kernel machine’ and choose ‘Gaussian Process Regression’.
- There is no option for hyper-parameter optimization as this is performed in a Bayesian framework. There is however a cross-validation scheme for model performance evaluation (outer loop). Choose the ‘k-folds CV on blocks cross-validation scheme with k=5’.
- Leave the rest at their defaults.

After you have specified everything, the final configuration should look like the one in figure 20.9

You will now create the second model with covariates removed. First right-click on the ‘Model: Specify new’ module you just finished and click ‘Replicate Module’. That way you will only need to modify two things.

- Change the name of this model to ‘RegGRCov’.
- In ‘Other operations’ choose ‘Select Operations’, add a ‘New: Operation’, and in the ‘Operation’ field choose the ‘Regress out covariates’ option.

After you have specified everything, the final configuration should look like the one in figure 20.9 with only the two aforementioned differences.

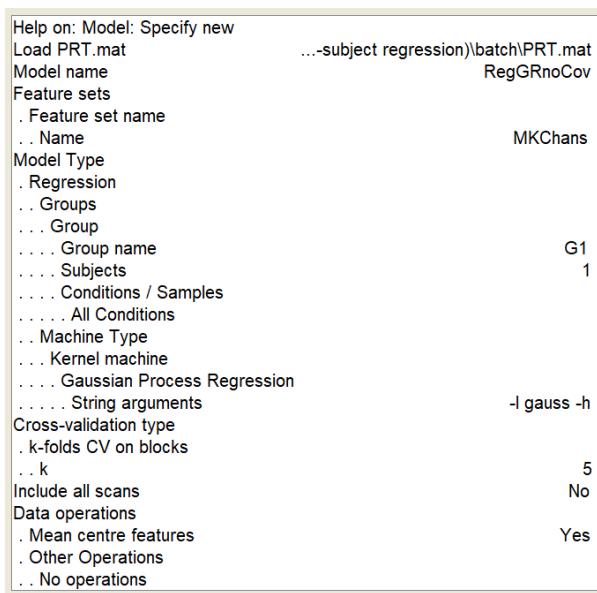


Figure 20.9: ‘Model: Specify new’ module in `matlabbatch`.

20.2.4 Model: Run & Display results

Simply open two ‘Model: Run’ modules, choose your PRT.mat, type the name of the two modes as chosen in the previous step and leave the rest at their default values.

If you’ve done everything properly the results should be quite similar to the GUI analysis with KRR.

Chapter 21

New Machine Tutorial

Contents

21.1	Introduction	221
21.2	prt_new_machine.m	222
21.2.1	Inputs	222
21.2.2	Outputs	223
21.3	How to import and test your new machine in Batch	223
21.4	How to import your new machine in GUI	227
21.4.1	prt_defaults.m	227
21.4.2	prt_get_machine_ui.m	228
21.4.3	prt_ui_copy_model.m	228
21.4.4	prt_plot_nested_cv.m	229
21.4.5	prt_weights_*.m	229
21.4.6	Running your new machine	229

21.1 Introduction

Many research groups from both the machine learning and the neuroscience communities are actively involved in the implementation of new methods for analyzing neuroscience data. In an effort to promote science and the collaboration between researchers across the world, the developers of PRoNTo, while having it as an open-source software right from the start, have also been constantly trying to make it more user-friendly and easy to use. To that end, the whole structure of PRoNTo has been built in such a way, that it makes it relatively easy for a developer or an experienced user to interfere in all steps of the process and customize them to their needs. However, while the customization of all the other steps of the process is much more straight-forward, importing a new method (or ‘machine’, as they are called in PRoNTo), or a wrapper for a library, is a bit more intricate.

This tutorial is intended as a semi-standalone tutorial with the purpose of walking a developer through all the steps required to import their own method into PRoNTo both in Batch and GUI. It will provide all the details on how your machine in your script needs to be structured, as well as the modifications required in all the scripts of PRoNTo to fully port a new method in it and we expect that users with programming experience should be able to use nothing more than this tutorial to import a new method. We will follow closely Chapter 14, which uses the IXI dataset, therefore even though not necessary, the readers are advised to go through that tutorial first. The IXI dataset can be found on PRoNTo’s website <http://www.mlnl.cs.ucl.ac.uk/pronto/prtdata.html> (data set #3). The machine we will use as a toy is the Kernel Ridge Regression (*prt_machine_krr*), so some parts (like the arguments passed) will resemble the ones in the *prt_machine_krr* script. You will also need to download a compressed copy of the *new_machine* folder, found in http://www.mlnl.cs.ucl.ac.uk/pronto/new_machine, unzip it and put the whole folder (*new_machine*) inside the *PRoNTo/machines* folder. The *new_machine* folder includes all the files needed for you to run this tutorial. Inside it you will find:

- A file named ‘prt_new_machine.m’, which will be the new method tested.

- A *d* structure example, which is a structure containing all the information of the data that the ‘prt_new_machine.m’ accepts as input.
- And *new_machine_batch.mat*, which is a .mat file containing all PRoNTo Batch modules to test your new machine in case you only want to focus on testing your method.

Note: Importing a new machine is far more straight-forward through the Batch System, compared to GUI, so users familiar with it are strongly advised to first try their machines in Batch, and only go through the process of porting it in GUI if they intend to use GUI for their analyses.

Note: Any reference to specific lines in the scripts follow PRoNTo v3.0, so the reader is advised to download that version in order to help themselves find the parts of the code that need to be modified.

PRoNTo is MATLAB-based and includes six main modules: ‘Data & Design’, ‘Prepare feature set’, ‘Model: Specify new’, ‘Model: Specify from’, ‘Model: Run’ and ‘Compute weights’. For a specific model PRoNTo can display results in terms of performance as well as the model’s weights. Additional review options enable the user to review information about the data, features and models. All modules were implemented using a graphical user interface (GUI) and the MATLAB SPM Batch System. Using the MATLAB Batch System the user can run each module as batch jobs, and you can also extract the jobs you created in .mat files and run them using only scripts, which enables a very efficient analysis framework. Hence, people primarily interested in testing their method, should only focus on section 21.3 which uses the Batch.

All information about the data, experimental design, models and results are saved in a structure called PRT inside a .mat file called ‘PRT.mat’. The PRT.mat is first created when you run the ‘Data & Design’ module. PRoNTo also creates additional files, or overwrites existing ones, during each step (module) of the analysis. Each step of the analysis also updates the PRT.mat itself accordingly.

In the following section we are first going to explain how to create your own ‘machine’ script according to PRoNTo’s standards, together with the structure of the input data, as well as the structure that the output data have to have. In Section 21.3 we will briefly explain the Batch system, and how to import and run your new machine with it. And finally in Section 21.4 we will explain how to import your new machine in PRoNTo’s GUI.

21.2 prt_new_machine.m

First of all the user has to create their own machine script. Once they have created it according to the specifications of PRoNTo, they have to put it inside the *PRoNTo/machines* folder. A template script named ‘prt_new_machine’ should already exists inside the folder *PRoNTo/machines/new_machine*, so the user is advised to use this as a template. Two things are important here. The way the input is structured, so that the user knows how to modify their method to accept the data as they enter their script, as well as the way the user has to structure the output of their script so that it is compatible with the rest of the PRoNTo structures.

Important note: The full path of the custom machine must be manually added to the Matlab Path.

21.2.1 Inputs

There are two input structures to the machine function that contain all the relevant information of the data, *d* and *args*.

1. **d:** This is the structure containing all the information of the data. The *d* structure contains more fields than what are needed. The most important ones are the following:

- **.tr_targets:** training labels (for classification) or values (for regression). This is a column vector of [Ntr x 1] dimensions, where Ntr is the number of targets of the training data.
- **.te_targets:** testing labels (for classification) or values (for regression). This is a column vector of [Nte x 1] dimensions, where Nte is the number of targets of the testing data.

- **.use_kernel:** Whether the data are in form of kernel matrices (true) or in form of features (false).
- **.pred_type:** Prediction type, ‘Classification’ or ‘Regression’.
- **.train:** Training data (cell array of matrices of row vectors, each having [Ntr x D] dimensions if we input the raw data, or [Ntr x Ntr] if we input their kernels]). Each matrix contains one representation of the data. This is useful for approaches such as multiple kernel learning.
- **.test:** Testing data (cell array of matrices row vectors, each having [Nte x D] dimensions if we input the raw data, or [Nte x Nte] if we input their kernels]).
- **.testcov:** Testing covariance (cell array of matrices row vectors, each [Nte x Nte]).

Irrespective of the original type of the data (NIfTI, MEEG or .mat), the data are extracted and/or converted to the d structure. While in .mat you could visualize your data right from the start, in the cases of NIfTI and MEEG there is no PRoNTTo function that lets you visualize your data. A test d structure is provided inside the *PRoNTTo/machines/new_machine* folder to help the readers visualize better how the input structure is.

2. **args:** These are arguments required by each machine. These arguments can be strings, alphanumeric, or numeric characters, they are usually specific to the machine and they are mostly defined as defaults elsewhere.

21.2.2 Outputs

In order for a new machine to be fully compatible with PRoNTTo the outputs of the machine have to be structured in a specific way. The outputs of the machine are all gathered under one structure, named *output*, which has the following fields:

- **.predictions:** Predictions of classification or regression [Nte x D]. This field is mandatory.
- **.func_val:** Values of the decision function [Nte x D]. This field is not always mandatory, but the reader is advised to always have it on their output.

All the other outputs are optional in most cases. The user is advised to go through the different existing machines, in order to get an idea of how they are structured.

Important note regarding weights: Depending on your machine, in order for it to be able to visualize the contribution (weights) of different regions/features, some extra outputs might be mandatory. For example, while in kernel machines the weights are computed directly, for non-kernel machines they need to be explicitly computed. Hence, if you want to create a non-kernel machine, it is mandatory for either the coefficients to be in the output structure in order to be computed by PRoNTTo, or to compute the weights internally and contain them in the output. A thorough look at the existing machines will give the user a good sense of what is needed when.

21.3 How to import and test your new machine in Batch

In the introduction of this chapter it was mentioned that importing a new machine in Batch is more straightforward when done through Batch, compared to the GUI. That is because the PRoNTTo code regarding the Batch system is already prepared to accept a new custom machine, which is done when defining a model in the ‘Model: Specify new’ and ‘Model: Specify from’ modules so there are absolutely no code modifications required. We assume, however, that the reader is already familiar with the PRoNTTo Batch from the previous sections, and if not enough, we strongly advise the reader to go through some tutorials first. More specifically, prior to try importing this new machine with Batch, even though not necessary, it will strongly help the readers if they have first completed the tutorial of Chapter 14, since we are going to use the same PRT.mat (which is the main structure format file of PRoNTTo and everything is done and stored in there) and the same feature set to run the new machine. Finally we remind the reader that the toy machine (KRR) of this tutorial, *prt_new_machine.m*, is nothing more than a copy of the *prt_machine_krr.m*, used here with a generic name for instructional purposes.

We will discuss quite briefly the main parts needed in order for someone not interested in details to directly proceed, with minimal exposure, to import their own machine in PRoNTTo. So anyone wanting to get a better

understanding should go through the rest of the manual.

To start with, to launch the toolbox GUI, just type `prt` or `pronto` at the MATLAB prompt and the main GUI figure will pop up, see Fig. 1.1. From there on simply click on the processing step needed. Most functions of PRoNTo have been integrated into the `matlabbatch` batching system [8] (like SPM8) and the batching GUI is launched from the main GUI by clicking on the `Batch` button. Once in the Batch system, click the ‘Load Batch’ button to load the file named ‘`new_machine_test_batch.mat`’ and you will see 5 modules in the Module List.

As you can see in the PRoNTo GUI, each job (one job being for example to specify the details of a model, or to ‘run’ [train&test] a model) requires you to input all the information every time you want to run this job. This can be tiresome, and prone to errors since after running the job you have no practical way of retrieving fast the information that you just put. So the whole idea behind Batch is to create a principled way to group a lot of jobs together, be able to run them whenever you want, and be able to save them in a .mat file (using the ‘Save Batch’ button) in order to be able to review and revise them whenever you want to. Each module in the list represents one job. The whole module list is run when you click the ‘Run Batch’ button. Now let’s proceed by briefly explaining the 5 different modules in the list.

Note: All fields of all modules have a help section which can be found at the bottom of the GUI of the Batch system after clicking on the desired field.

1. **Data & Design:** In this module the user imports all the information related to the data, such as, for example, the type of the data, the regression values for regression problems, covariates, and masks, as well as information about the experimental design.

- In this module the user has to change the ‘Directory’ field to the directory they desire to run their analysis. It is in this directory where the PRT.mat file will be created. You first have to right-click and ‘Unselect All’ of the previously saved directory.
- Change the ‘Files’ field to the directory you have stored the 102 subjects of the IXI data set. They should be in the *IXI/Data/aged/Guys* directory. You first have to right-click and ‘Unselect All’ of the previously saved directories of the 102 subjects.
- Change the ‘From file’ under ‘Regression targets (subject)’ to the appropriate path where the ‘Age_old_Guys.mat’ is located. These are the regression target values of the 102 subjects. You first have to right-click and ‘Unselect All’ of the previously saved directory of the ‘Age_old_Guys.mat’ file.
- And finally change the ‘File’ under ‘Masks’ to the appropriate path where the ‘SPM_mask_noeyes.img’ is located. Masks are usually used if we want to throw away some irrelevant information prior to the analysis. For example here the fMRI information from the eyes is irrelevant to our analysis, so we use a standard ‘no_eyes’ mask to throw away the data from the eyes. You first have to right-click and ‘Unselect All’ of the previously saved directory of the ‘SPM_mask_noeyes.img’ mask image file.

2. **Feature set/Kernel:** In this module the user creates the feature sets from the loaded data in the previous module, in order to be used later on when we build our models. Depending on the type of your data the fields are different, but generally in this module, after defining the data format, you can choose whether to include based on samples or conditions, for example in the case of NIfTI images whether to include all voxels or part of them, some simple extra preprocessing steps, etc.

- Here you only need to make sure you have the appropriate directory in the ‘Load PRT.mat’ field.

Note: When you first create a batch of jobs in the Batch, there is no ‘PRT.mat’ yet created. In that case you can either run the first module by itself, and then click ‘Specify’ to choose the already created PRT.mat, and then run the rest of the modules. Or you can run them all together by using the ‘Dependency’ button. By clicking the ‘Dependency’ button you can choose for example the ‘Data & Design’ module, which instructs the ‘Load PRT.mat’ field of the ‘Feature set/Kernel’ module to harvest the ‘PRT.mat’ of the ‘Data & Design’ module to be run as the PRT.mat of this module (‘Feature set/Kernel’).

3. **Model: Specify new:** In this module the user imports all the information related to the machine learning models they want to build, such as whether you want to do classification or regression, which feature sets to use, as well as all the information about the machine learning algorithm itself and the model selection and cross-validation schemes.

- First make sure you have the appropriate directory in the ‘Load PRT.mat’ field. Please note the instructions in the previous module about how to use the ‘Dependency’ button.
 - Change the ‘Function’ under the ‘Custom machine’ to the appropriate path where the ‘prt_new_machine.m’ is located.
4. **Model: Run:** The purpose of this module is pretty simple. Here the user only defines which models they want to run, and whether or not to do permutation test or not.
- Here you only have to make sure you have the appropriate directory in the ‘Load PRT.mat’ field.
5. **Compute weights:** The previous modules allow the user to specify and run one or more models. These include the machine to be used, the cross-validation scheme and the classification/regression problem. The estimation of those models led to predictions on unseen/test data (in each fold), from which measures of performance of the model can be derived and displayed. In addition, as PRoNTo primarily uses linear models, it provides the option of recovering the model weights in the original feature space, and transforming the weights vector into an image, or map. These maps contain at each feature the corresponding weight of the linear model (that together define the predictive function), and which related to how much this particular feature contributed to the classification/regression task in question. The purpose of this module is to compute those model weights.
- Here you only have to make sure you have the appropriate directory in the ‘Load PRT.mat’ field.

If everything was done properly, clicking the green ‘Run Batch’ button will finish smoothly and give you the same results as the KRR model in Chapter 14.

At this stage the user should be familiar enough with the Batch in order to do a first test, running the template new machine we provide. It is strongly advisable that the users go in debug mode and enable a breakpoint in the start of the *prt_new_machine.m* script to study the two input main input structures (*d* and *args*) directly through the MATLAB workspace in order to gain a better understanding.

Create your own Batch from scratch

The users that want to create the whole batch from scratch by themselves should first follow and finish the regression tutorial of Chapter 14 and, after that, proceed to the additional instructions below.

- First open the Batch and fill it in as instructed in 14.2.3 but without filling in the ‘Machine type’ field. For practical purposes, change the ‘Cross-validation type’ from ‘Leave one subject out’ to ‘k-folds CV on subjects’ with ‘k=5’.

The ‘Model: Specify new’ module at this point should look similar to figure 21.1.

- Now we will describe how to fill the additional field that we skipped before, ‘Machine type’. Since KRR uses kernels, select the ‘Kernel machine’ option, and, after that, in the ‘Kernel machine’ field, select the ‘Custom machine’ option. There are in total 3 fields here, 2 of which are new.

1. **Function:** You need to fill in the full path of your machine’s file. In this particular case we have put the machine inside a folder named ‘new_machine’ so the full path would be something like ‘PRoNTo/machines/new_machine/prt_new_machine.m’.

2. **Custom machine string arguments:** These are string parameters specific to each machine. They are usually required if the machine calls an external library for computations. These are the same string parameters mentioned in 21.4.1, so the reader is referred to this subsubsection for a further explanation. In our case, our custom machine requires no string arguments, so leave this field as it is.

3. **Custom machine optimization and parameters:** This is the same as the ‘Machine optimization and parameters’ of the rest of the machines, with the only exception that no values are auto-filled. First select ‘Optimize hyper-parameter’ and in the ‘Regularization hyper-parameter’ insert the values [0.010.1110100100]. Finally change the ‘Cross-validation type for hyper-parameter optimization’ to ‘k-folds CV on subjects’ with k=4.

Help on: Model: Specify new	
Load PRT.mat	... PRT.mat file
Model name	New_Machine
Feature sets	
. Feature set name	
. . Name	Scalar_Momentum
Model Type	
. Regression	
. . Groups	
. . . Group	
. . . . Group name	Aged
. . . . Subjects	102x1 double
. . . . Conditions / Samples	
. Target	
. Name	age
. . . . Machine Type	<-X
Cross-validation type	
. k-folds CV on subjects	
. . k	5
Include all scans	No
Data operations	
. Mean centre features	Yes
. Other Operations	
. . No operations	

Figure 21.1: ‘Model: Specify new’ module.

- In case you do not want to do a nested cross-validation, you have to select the ‘No optimization’, and in the ‘No optimization’ field you should fill in any default parameter values that your machine requires. In our case that is the value of the λ parameter, and is equal to 1. For further information regarding default parameter values the reader is referred to [21.4.1](#).

The ‘Model: Specify new’ module finally at this point should look similar to figure [21.2](#).

Help on: Model: Specify new	
Load PRT.mat	... PRT.mat file
Model name	New_Machine
Feature sets	
. Feature set name	
. . Name	Scalar_Momentum
Model Type	
. Regression	
. . Groups	
. . . Group	
. . . . Group name	Aged
. . . . Subjects	102x1 double
. . . . Conditions / Samples	
. Target	
. Name	age
. . . . Machine Type	
. . . Kernel machine	
. . . Custom machine	
. . . . Function	... \prt_new_machine.m
. Custom machine string argument	"
. Custom machine optimization and parameters	
. Optimize hyper-parameter	
. Regularization hyper-parameter	1x6 double
. Cross-validation type for hyper-parameter optimization	
. k-folds CV on subjects	
. k	4
Cross-validation type	
. k-folds CV on subjects	
. . k	5
Include all scans	No
Data operations	
. Mean centre features	Yes
. Other Operations	
. . No operations	

Figure 21.2: Final configuration of the ‘Model: Specify new’ module.

21.4 How to import your new machine in GUI

21.4.1 prt_defaults.m

This is the script that sets most of the defaults used by PRoNTo. Each machine and/or library has its own parameters that might need to be set as defaults. You have to modify this script to include any default parameters your machine requires. There are 3 things you have to modify in this script.

Machine lists for GUI

Here we provide the GUI of PRoNTo with a list of names for each machine available. All names can be found inside `prt_def.machine`. The names of the machines are grouped according to whether the machine is for classification or regression, whether it uses kernels or not, and whether it is a multi-kernel method or not. In our case since our toy method is KRR (named ‘New Machine’), which is for regression and uses kernels, we will need to add it to `prt_def.machine.reg_K`.

```
93  prt_def.machine.reg_K = { 'Kernel Ridge Regression', 'New Machine', ...
94      'Relevance Vector Regression', 'Gaussian Process Regression', ...
95      'epsilon-SVR' };
96  prt_def.machine.reg_NK = { 'epsilon-SVR' };
```

String parameters

Some times machines require string arguments in their input, especially machine that are wrapper for libraries. Any string arguments your machine might require need to be defined here. Our toy machine (KRR) does not require any string arguments, so for an example we will present the string arguments required to run an L2-SVM as well as an ϵ -SVR using the LIBSVM library.

```
105 %LIBSVM machines
106 % Classification - dual
107 prt_def.model.libsvm_sargs = '-q -s 0 -t 4 -c'; %L2 SVM
108 % Regression - dual
109 prt_def.model.libeSVR_sargs = '-q -s 3 -t 4 -c'; % e-SVR
```

If your machine required some string arguments to work, you would need to define the field `prt_def.model.new_machine_sargs` together with its proper string arguments. You should remember the name you give to this field (`.new_machine_sargs`) because you are going to use it later on.

Default parameter values

Besides string arguments, some machines may also require numeric values to be set as defaults. For example:

```
128 prt_def.model.rtargs = 601;
129 prt_def.model.l1MKLmaxitr = 250;
130 prt_def.model.wipargs = [1 0.5];
131
132 %LIBSVM and LIBLINEAR defaults
133 prt_def.model.libsvmargs = 1;
```

In this particular case, the default parameter values required by our toy machine (KRR) are the same as the LIBSVM and LIBLINEAR defaults shown above, so there is no need to add new ones explicitly. We are going to use this when asked in the next subsection. In case you need to specify new ones, you have to put it inside the `prt_def.model` structure. So your newly defined defaults would need to be inside a structure `prt_def.model.name_args` where ‘name_args’ can be whatever name you want to give to your machine’s arguments. You should remember this name because you are going to also use it later on.

Important Note: The order with which the default parameter values will be harvested is the same as the one that are defined. So the user is advised to pay attention to that order since the machine will most likely not hit an error if run with transposed default parameter values.

21.4.2 prt_get_machine_ui.m

This script is called when you use the GUI, to harvest the defaults you previously defined for each machine¹. You have to modify these scripts to properly harvest your machines defaults by including your machine in both of them. Be careful, however, because the GUI and the Batch have small differences in how they define things. Below is an example of how we defined our new machine, which is the same as KRR. First is an example from *prt_get_machine_ui.m*, and then from *prt_get_machine.m*.

```

110      % New Machine
111      elseif any(strfind(name, 'New Machine'))
112          machine.function='prt_new_machine';
113          machine.args = def.libsvmargs;
114
115      %Relevance Vector Machine
116      elseif any(strfind(name, 'Relevance'))
117          machine.function='prt_machine_rvr';
118
119      % Gaussian process
120      elseif any(strfind(name, 'Process Regression'))
121          machine.function='prt_machine_gpr';
122          machine.s_args= def.gpr_sargs;

```

As we see in the code above, it is in this part that we are asking for the default parameter values that were defined in the previous section. If a new machine required different default parameter values, the user would have defined in the previous section, giving them a specific name, which would have been used in place of *def.libsvmargs* in line #113.

21.4.3 prt_ui_copy_model.m

This is the script responsible for the GUI of the ‘Model: Specify from’ module. Just as in 21.4.1, the user has to put the name of their machine here as well, in two instances, inside the if-loops #970-1002 and #1232-1296.

First instance

```

993 elseif val==2
994     handles.type='regression';
995     set(handles.butts_defclass, 'String', 'Select subjects/scans')
996     %set the list of machines
997     set(handles.pop_machine, 'String', {'Kernel Ridge Regression', 'New Machine',
998         ...
999         'Relevance Vector Regression', 'Gaussian Process Regression', 'Multi-
1000         Kernel Regression'})
1000     set(handles.pop_machine, 'Value', 1)
1001     handles.machine.function='prt_machine_krr';
1001     handles.machine.args=handles.def.krrargs;
1002 end

```

Second instance

```

1271 elseif strcmpi(handles.dat.model(indmod).input.type, 'regression')
1272     handles.type='regression';
1273     if nk % Kernel regression machines
1274         list = handles.reg_K;
1275         if handles.multimod || handles.multiroi || ...
1276             numel(sel)>1
1277             list = [list, handles.MK];

```

¹*prt_get_machine.m* is the equivalent of *prt_get_machine_ui.m* but for Batch. Modifications to *prt_get_machine.m* will be discussed in the next section.

```

1278     end
1279     if strcmpi(handles.dat.model(indmod).input.machine.function , '
1280         'prt_machine_krr')
1281         mach = 'Kernel Ridge Regression';
1282     elseif strcmpi(handles.dat.model(indmod).input.machine.function , '
1283         'prt_new_machine')
1284         mach = 'New Machine';
1285     elseif strcmpi(handles.dat.model(indmod).input.machine.function , '
1286         'prt_machine_rvr')
1287         mach = 'Relevance Vector Regression';
1288     elseif strcmpi(handles.dat.model(indmod).input.machine.function , '
1289         'prt_machine_gpr')
1290         mach = 'Gaussian Process Regression';
1291     elseif strcmpi(handles.dat.model(indmod).input.machine.function , '
1292         'prt_machine_sMKL_reg')
1293         mach = 'L1 Multi-Kernel Learning';
1294     elseif strcmpi(handles.dat.model(indmod).input.machine.function , '
1295         'prt_machine_svm_bin')
1296         mach = 'epsilon-SVR';
1297     end

```

21.4.4 prt_plot_nested_cv.m

This script plots the results of the nested cv that appear on *prt_ui_results*. The user has to modify the script, by including an extra case in the switch-loop of lines #27-132, with their machine. Since our toy machine for this tutorial is KRR, we will be using logscale as well, and the x and y labels are also going to be the same.

```

66     case 'prt_new_machine'
67         xlabel = 'Lambda';
68         ylabel = 'NMSE';
69
70         %If no axes_handle is given, create a new window
71         if ~exist('axes_handle', 'var')
72             figure;
73             axes_handle = axes;
74             logscale = 1;
75         else
76             % Clear EVERYTHING in the UI before defining the axes
77             cla(axes_handle, 'reset');
78             logscale = 1;
79         end

```

21.4.5 prt_weights_*.m

As mentioned in 21.2.2 in the note regarding weights, depending on your machine, in order for it to be able to visualize the contribution of different regions/features, some extra outputs might be mandatory. In general, in most cases weights are computed by *prt_weights_linkernel.m*. So depending on your machine you might either use one of the existing scripts, or you might need to create your own new *prt_weights_*.m* for building the weights of your new custom machine. That, however, is depended on your machine so the user is advised to go through the existing *prt_weights_*.m* scripts to further understand how they might need to structure theirs.

21.4.6 Running your new machine

If your machine's script is properly defined, and if everything was done accordingly, when you open the 'Model: Specify new/from' modules, you are going to see your new custom machine as one of the options in the drop-down list like in figure 21.3.

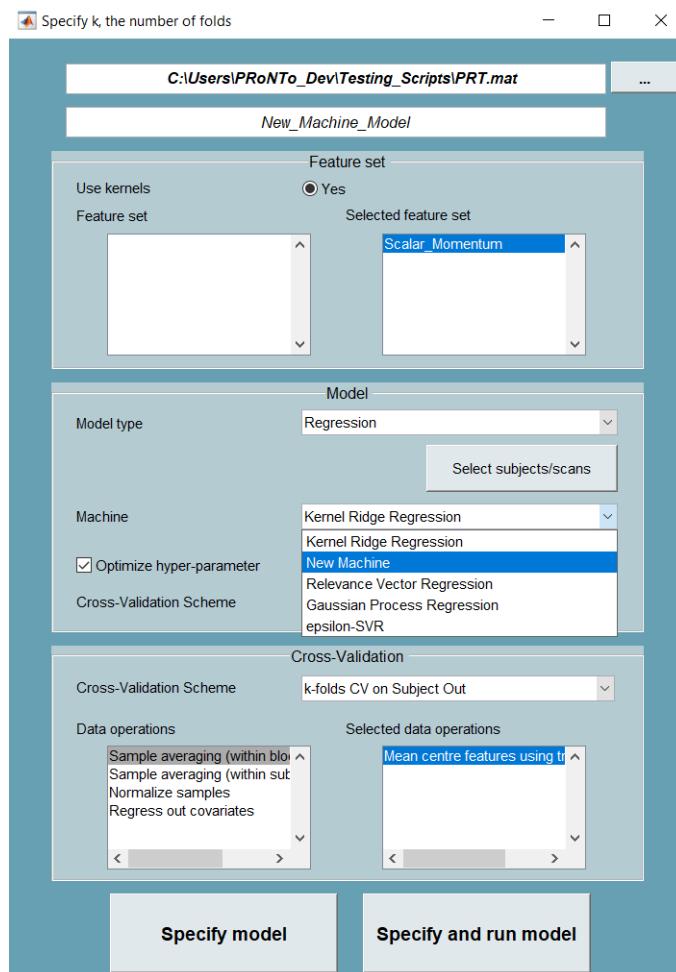


Figure 21.3: ‘Model: Specify new’ module in GUI with the new machine option

Part IV

Advanced topics

Chapter 22

Developer's manual

Contents

22.1	Introduction	233
22.2	PRoNTo folder structure	234
22.3	Data & Design	235
22.3.1	PRT fields created	235
22.3.2	Files created	235
22.3.3	GUI behaviour	235
22.3.4	Batch behaviour	236
22.3.5	Functions called	236
22.4	Prepare feature set	237
22.4.1	PRT fields created	237
22.4.2	Files created	237
22.4.3	GUI behaviour	238
22.4.4	Batch behaviour	238
22.4.5	Functions called	238
22.5	Model: Specify new/from	239
22.5.1	PRT fields created	240
22.5.2	Files created	241
22.5.3	GUI behaviour	241
22.5.4	Batch behaviour	241
22.5.5	Functions called	242
22.6	Model: Run	243
22.6.1	PRT fields created	245
22.6.2	Functions called	245
22.7	Compute weights	246
22.7.1	PRT fields created	247
22.7.2	Files created	247
22.7.3	Functions called	247

22.1 Introduction

There is an increasing interest in applying machine learning methods to detect predictive patterns in neuroimaging data. The accumulative data size in neuroimaging, on the other hand, starts reaching a big data scale and opening opportunities for more engineers and data scientists interested in mental health and neuroscience to apply their expertise to this field. Collaborations between neuroscience and machine learning communities show promises to discover novel and informative biomarkers for brain diseases. To meet the needs from both communities, we add new contents about PRoNTo in this v3.0 manual for users who are interested in understanding it at a deeper level. Therefore, this chapter will describe the main framework of PRoNTo and its backend core

functions in order to provide a roadmap for developers.

This chapter will begin with descriptions on the folder structure, which explains how the files and library folders are organised, and some naming conventions we use. Then we will explain the five functional modules: ‘Data & Design’, ‘Prepare feature set’, ‘Model: Specify new’ and ‘Model: Specify from’, ‘Model: Run’, and finally ‘Compute weights’. Specifically, we will provide details for

- How the GUI and Batch interfaces are built.
- What functions are called to arrange the user inputs to PRoNTo accepted structures.
- How the data structures are passed to core functions carrying on computations and model estimations.
- How cross-validations are implemented.
- How machines from third-party libraries are called.
- How weights are computed for visualization.

We will also provide detailed information on specific functions implementing each module for reference. We think that understanding the structure of PRoNTo shall be enough for developers to implement and interface their own functionalities, hence further details on particular components (e.g., implementing a new data type) beyond the framework backbone are not included here.

22.2 PRoNTo folder structure

PRoNTo contains several subfolders under the main PRoNTo folder. In this section, we will provide a brief description of each subfolder.

- The *_devUtils* and *_unitTests* folders are particular tools for testing the software. The *atlas* folder contains two atlases: the AAL atlas with labels, and the Brodmann atlas without labels. If the developer wishes to add atlases, this is the recommended place for that. Atlases in NIfTI format can be created easily by using SPM or manually by the user. Please refer to the last point in ‘Features’ in the Section 3.3.1 of the user manual. If you use multiple kernel learning (MKL) and want to see ROI contributions in ‘Display weights’, a .mat file storing ROI labels should be created alongside with the atlas. This .mat file should comprise a cell array, and each cell should contain a label of a specific region. Please refer to point ‘ROI Label’ in Section 7.2.4 in the user manual for more details on how to create and use this .mat file. Atlases in .mat format can be created by users manually.
- The *batch* folder contains the files for the Batch module, which includes both the configuration (with ‘cfg’ in the function names) and job execution (with ‘run’ in the function names) functions. For each module in PRoNTo, we have corresponding configuration and execution files. The configuration functions generate the trees defining the interfaces of the modules in Batch. The job execution functions pass user inputs to lower level functions in PRoNTo for calculations and estimations. GUI and Batch share the same lower functions for computations and estimations. This modular design eases the development of new machines and functions in each module.
- The *machines* folder contains the different regression and classification algorithms. There are functions from several third-party libraries (e.g., libsvm, liblinear and SimpleMKL), ‘machines’ which are wrapper functions (with ‘prt_machine’ in the function names) interfacing the libraries to PRoNTo, and functions computing weights for each machine. If the developer wishes to add a new machine, they can open these functions and read their input and output arguments. There are a few naming conventions: ‘gp’ Gaussian processes, ‘cla’ classification, ‘krr’ kernel ridge regression. Names of these algorithms also refer to libraries, for example, ‘gp’ refers to ‘gpml’ library, ‘MKL’ refers to simpleMKL library.
- The *manual* folder contains latex files for the manual, images and other contents. Developers can add to the documentation here.
- The *masks* folder contains one first-level mask in the Data & Design module until before version 3. This changed in PRoNTo v3.0 and more masks (in NIfTI format) can be added.

- In *utils* folder, there are several lower level functions e.g. checking if inputs are alphanumeric, or performing operations on kernels, such as splitting data in the kernel and normalising kernels.
- Beyond the above mentioned subfolders, functions with ‘ui’ in their names define the GUI interfaces, i.e. *_ui_* .fig and *_ui_* .m file pairs define the figure and how the GUI behaves, respectively. We used GUIDE (GUI development environment) to create GUI and add custom functions and codes to extend these interface functions.
- The rest of the ‘prt_’ functions perform the computations and estimations defined in each module.

22.3 Data & Design

22.3.1 PRT fields created

The PRT structure is an important data structure (a .mat file) created and repeatedly used by PRoNTo. It is firstly created after the ‘Data & Design’ module is initiated / specified by the user. At first, it consists of two fields: ‘group’ and ‘masks’. Users input all their data, experimental design (e.g., modalities, runs, conditions) and other related information here (e.g. covariates, regression values). After this step, data information is stored in the group field of PRT.mat, and the first level masks, HRF overlap and delay parameters are stored in the masks field of the PRT.mat. Below is a list of subfields in [PRT.group](#) and [PRT.masks](#):

- [PRT.group](#)
 - [PRT.group.gr_name](#): Storing group names.
 - [PRT.group.subject](#): Storing subject information.
 - * [PRT.group.subject.subj_name](#): Storing subject names.
 - * [PRT.group.subject.modality](#): Subfields containing information on regression targets, covariates, modality name, design and scans. Note: classification labels are specified in ‘Model: Specify new/from’ modules later.
- [PRT.masks](#)
 - [PRT.masks.mod_name](#): Name of corresponding modality.
 - [PRT.masks.type](#): Storing the data type.
 - [PRT.masks.fname](#): The full path of the specified mask.
 - [PRT.masks.hrfoverlap](#): Storing information on HRF overlap (default: 0).
 - [PRT.masks.hrfdelay](#): Storing information on HRF delay (default: 0).

22.3.2 Files created

PRT structure, saved in a PRT.mat file in the path specified by the user.

22.3.3 GUI behaviour

GUI of Data & Design module provides interfaces for users to specify where to store the PRT structure and files, data information, experimental design, masks and all other related information.

- Save button creates the PRT structure.
- Load button loads previously created PRT.mat files and checks their compatibility.
- Review button provides a graphical review of all the input information and the interface for modifying some HRF parameters.
- [prt.ui.main.m](#) and [prt.ui.main.fig](#) function pair define the GUI of the main window.
- Button click on ‘Data & Design’ in the main window will call [prt.ui.design.m](#) and [prt.ui.design.fig](#) to open the Data & Design module.

22.3.4 Batch behaviour

The Batch of Data & Design module provides users similar functions and interfaces as GUI. However, there are three major differences.

- Firstly, the batch job can be saved as a .mat file or a .m function. Both users and developers can make changes directly in these files.
- Secondly, both users and developers should be careful about the names of modalities when using the Batch, i.e. the names needs to be consistent.
- Finally, the HRF delay and overlap can be changed directly in the Batch of Data & Design. In GUI, they can be modified in Review.

22.3.5 Functions called

The GUI functions related to the Data & Design module can be found under the root PRoNTo folder, whereas the Batch functions of this module are under the *batch* folder.

GUI:

1. `prt_ui_design.m` and `prt_ui_design.fig`

These functions define GUI interface and other functions for the Data & Design module. The function '`prt_ui_design.m`' initializes the GUI window, executes the commands associated with each button, checks if the inputs are correct, and creates outputs in MATLAB prompt. If the developer chooses to make changes in this GUI module (e.g., a buttons or menus functions), you can find related sub-functions with corresponding names in `prt_ui_design.m`. Some important functions called by `prt_ui_design.m` are summarized as follows.

- `prt_data_modality.m` and `prt_data_modality.fig`

The two files define the GUI panel for specifying the information of the modality, experimental design, covariates and regression targets. The `prt_data_modality.m` also calls:

- `prt_data_conditions.m` and `prt_data_conditions.fig`, for receiving inputs for conditions, i.e. experimental design.
- `prt_check_design.m`, for checking the design and discards scans according to overlapping conditions or the minimum time interval between conditions based on the HRF function and specified HRF parameters.
- `prt_data_targets.m` and `prt_data_targets.fig`, for opening the target window to examine and/or modify targets.
- `prt_get_design_MEEG.m`, for filling the design from MEEG file. In v3.0, new data types (.mat and MEEG) are allowed, hence relevant codes are added to interface functions, and/or core functions carrying out the backend computations.

- `prt_get_defaults.m` and `prt_defaults.m`

Default settings of PRoNTo are stored in `prt_defaults.m`, including global defaults such as colour, parameters for the data and design, preprocessing, default atlas for ROI definition, arguments of different machines, and parallelization of the code. The function `prt_get_defaults.m` gets the relevant default values from `prt_defaults.m` for this module. For instance, the code '`color = prt_get_defaults('color')`' sets colors of backgrounds and figure parameters for Data & Design module. `prt_defaults.m` is constantly called by many functions in PRoNTo.

- `prt_data_review.m` and `prt_data_review.fig`

These two functions define the GUI for reviewing the data.

2. `prt_load.m` and `prt_struct.m`

This function is called when clicking the Load button. It loads PRT.mat and checks its fields for backwards compatibility. If you wish to add a field to the PRT structure, and that field is necessary for display or further analyses, the `prt_struct` file should be amended to set default values for that field to ensure that older PRT files can still be read and modified.

Batch:3. `prt_cfg_design.m` and `prt_run_design.m`

- The function `prt_cfg_design.m` is the Data & Design module configuration file that builds the PRT.mat data and design structure.
- The function `prt_run_design.m` is the job execution function, which takes a harvested job data, rearranges them into PRT structure and saves it.

At this stage, the GUI and batch have very little code in common. Hence, changes in the GUI have to be performed independently in the batch and conversely.

22.4 Prepare feature set

22.4.1 PRT fields created

There are two fields created in PRT.mat in this step, too. Field *fs* (feature set) and *fas* (file array structure). The information for the two fields is specified by users in the ‘Specify modality to include’ panel.

For each selected modality, a file array is created containing the data (detrended if specified). The field `PRT.fas` is linked to this file array. In *fas* field, the information stored refers to the modality (as defined in Data & Design). This means that multiple MRI sessions, entered as different runs, will each create a specific *fas*. A *fas* is linked to a *.dat* file created at this stage, which is with dimensions of #images x #features within the first-level mask. A *fas* structure also stores feature operations (detrend and detrend parameters), header of the first image for recovering images if needed, and indices of all features within the first-level mask.

The field *fs* refers to the kernel/feature set that is created. It is a structure of size number of feature sets created by the user. This structure of *fs* stores information on the feature set name, kernel file, ID matrix, information of using multiple kernels, second-level mask and atlas. This field also contains information about the second-level mask and/or the atlas (used to build the kernel).

One important subfield is the ID matrix, which contains information about the modalities selected in the feature set arranged in six different columns: group, subject, modality, condition, block and scan.

- The name of the columns are stored in `PRT.fs.id_col_names`.
- The group numbers, subject numbers, modality labels, condition numbers, block labels and scan numbers themselves are stored in `PRT.fs.id_mat`, whose dimension is the number of samples x 6. Columns in `PRT.fs.id_col_names` and `PRT.fs.id_mat` are in corresponding order.

The ID matrix is referred to in the ‘Review kernel & CV’ reviewing tool in PRoNTo, where a graphical display shows the structure of this matrix.

- In addition, there is a *fas* subfield in `PRT.fs`, that is different from `PRT.fas`. This subfield in *fs* refers to the modality and scan indices of the feature set modalities in the *fas* structure. It is useful when modalities are concatenated in samples (e.g. accessing the correct scans from multiple runs in an MRI experiment, with each run linked to a different `PRT.fas` file array).

As mentioned in the Data & Design module, in v3.0, different data formats (.mat, NIfTI and MEEG) are supported. Users can specify which data type and modality shall be used to build the feature sets. Some changes according to these are made in both GUI and Batch of this module, which will be more described as follows.

22.4.2 Files created

- FAS .dat file(s).
- A kernel matrix (.mat with variable Phi, a cell array with a kernel in each cell).

- Updated masks and/or atlases (NIfTI resampled at the size of the input data images).

The .dat file refers to one `PRT.fas` structure and stores all features in the first-level mask for each modality. It is a binary file. If computing multiple kernels using a second-level atlas is chosen by the user, the kernel matrix will contain multiple cells. Each cell stores information of one kernel.

22.4.3 GUI behaviour

The GUI of this module takes the PRT.mat created in the Data & Design module. It provides interfaces for users to further specify feature sets, their names, selected modalities, operations on the feature sets and atlas for defining kernels.

In v3.0, users may specify three different data types (NIfTI, .mat and MEEG) in the Data & Design module. If only the NIfTI or .mat file is specified, after loading PRT.mat, GUI calls for the Specify modality to include panel. If more than two of the data types are specified, e.g. NIfTI and .mat, or all of them, GUI calls for Specify type of data after loading PRT.mat. The reason for this design is that Specify modality to include panel is different between MEEG and the other two types. The function `prt_ui_main.m` checks the type of data (modalities) in the loaded PRT.mat and calls for these panels. Buttons on the panels have names of the modalities. Their functions are defined using corresponding GUI files, such as `prt_ui_fs_alldatatype.m` and `prt_ui_fs_alldatatype.fig`. Sub-functions like `prt_ui_fs_alldatatype.m` will gather modality information based on the button name, and pass it to `prt_ui_prepare_data.m`. This function then calls for the right Specify modality to include panel for the selected data type/modality. This panel accepts user inputs on which modalities to be included and several other related inputs, then Prepare feature set window is called. After filling in the fields of this window by users, `prt_ui_prepare_data.m` passes user inputs to `prt_fs.m` function.

Main functions in the Prepare feature set module, such as building the file array and computing feature set matrices (including multiple kernels) are completed by `prt_fs.m` or `prt_fs_EEG.m`. Here, we take `prt_fs.m` as an example for some further illustrations. This function calls `prt_init_fs.m` to initialize the file arrays, kernels and feature set parameters. Then it calls `prt_fs_modality.m` to build file arrays and kernels. The .dat files linked to file arrays are created here. A sub-function within `prt_fs.m` named as `prt_compute_ROI_kernels` builds kernels based on the second-level atlas.

22.4.4 Batch behaviour

The ‘Prepare feature set’ module in the Batch is similar to GUI. However, the batch requires consistent names across modules. Running important modules altogether will overwrite the PRT.mat, hence delete the links between the PRT.mat and the computed feature set(s) and kernel(s). *FAS* will be recreated each time the Batch is launched. Function `prt_cfg_fs.m` defines the tree of this Batch module. The function `prt_run_fs.m` harvests user inputs from the tree. It checks the modality and data types, then calls `prt_fs.m` or `prt_fs_EEG.m` to execute computations on file arrays, kernels and feature set. After loading PRT.mat, types of data can be selected under Data format field.

22.4.5 Functions called

Core:

1. `prt_fs.m`

The function gathers information on the modalities to build file arrays containing the (detrended) data and computes the linear kernels. It resizes the 2nd level masks and atlases if needed. The 2nd-level masks are used here as an important tool. It is used to define more precise scope for building the feature sets, for example, restricting the analysis to certain brain regions. Multiple kernels can be built for multiple modalities, or multiple regions of a specified atlas. `prt_fs.m` calls `prt_fs_modality.m` directly to compute multiple kernels for different modalities, but calls `prt_compute_ROI_kernels` to build ROI defined kernels for multiple regions in the atlas. A list of functions `prt_fs.m` calls is as follows:

- `prt_init_fs.m` to populate basic fields in `prt.fs`, like the file array structure, kernel data structure and feature set parameters.

- `prt_fs_modality.m` to write the kernel matrix to the user-defined path as well as the feature set if needed.
- `prt_compute_ROI_kernels`. This is a sub-function within `prt_fs.m`. It reads voxel indexes within both the 2nd-level mask and the 1st-level mask first. It then computes kernel for each region in the user-specified atlas and stores indices in the image for the weight computation.

2. `prt_init_fs.m`

The function initialises the file arrays, kernels and feature set parameters.

3. `prt_fs_modality.m`

Code in this function are divided in two parts: building file arrays and computing kernels. If a file array has been already built for a given modality, it won't be built again, but read from the existing file. Operations like scaling are applied on the kernel only. To build multiple kernels, `prt_fs.m` calls `prt_fs_modality.m` as many times as the number of kernels defined. It calls `prt_get_defaults.m` to get defaults of `fs`, and `prt_load_blocks.m` to access one or more blocks of data to avoid memory overload.

GUI:

4. `prt_ui_prepare_data.fig` and `prt_ui_prepare_data.m`

This pair of functions define the GUI to specify how many feature sets will be used. These modalities can either be concatenated (e.g. multiple runs of an experiment), or combined in multiple kernel settings (e.g. sMRI grey and white matter). For each modality entered, the function calls `prt_ui_prepare_datamod.m` to let users specify parameters for the selected modality, or data types in .mat or NIfTI formats. It calls `prt_ui_prepare_dataMEEG.m` and `prt_ui_prepare_dataMEEG.fig` to let users specify parameters for MEEG data type.

5. `prt_ui_prepare_datamod.fig` and `prt_ui_prepare_datamod.m`

This pair of files are called by `prt_ui_prepare_data.m` to accept user inputs about modalities, NIfTI and .mat data types to compute the file array, kernels and feature set.

6. `prt_ui_prepare_dataMEEG.m` and `prt_ui_prepare_dataMEEG.fig`

This pair of files are called by `prt_ui_prepare_data.m` to accept user inputs about MEEG data to compute file array, kernels and feature sets.

Batch:

7. `prt_cfg_fs.m` and `prt_run_fs.m`

The `fs` configuration file constructs the tree structure for this module. It provides user interfaces for specifications of the feature set for each data type and modality. The job execution function takes all inputs from `prt_cfg_fs.m`. It arranges them into proper data structure and stores this structure in a variable called ‘input’. It then passes PRT.mat and ‘input’ to `prt_fs.m` or `prt_fs_EEG.m` to build file arrays and compute kernels.

22.5 Model: Specify new/from

In v3.0, the ‘Specify model’ module of versions 2.X is split to ‘Model: Specify new’ and ‘Model: Specify from’ modules. They compute the inputs for the models but don’t estimate/run the models. The main inputs for this module are: feature sets, where users select which feature set will be used to build the model, whether to use kernel or not, model type (regression or classification), selection of samples and classes, machines, cross validation (CV) strategy and data operations. ‘Model: Specify new’ is used when users want to specify a new model to run. ‘Model: Specify from’ is used when users want to specify models similar to previous ones.

22.5.1 PRT fields created

Model: Specify new

Two subfields in `PRT.model` are created and filled: `PRT.model.model_name`, which stores the name of the model and `PRT.model.input` which stores all the other information. Another subfield `PRT.model.output` is only created but is not filled here. It will be filled after the model is estimated ('Model: Run' module). Below is an illustration of the information stored in `PRT.model.input`:

- `PRT.model.input.use_kernel`: Whether to use kernel methods (1) or not (0, for non-kernel methods). Non-kernel methods are available from v3.0.
- `PRT.model.input.type`: Regression or classification. This field is called very often within PRoNTo.
- `PRT.model.input.machine`: Stores information of the chosen machine. The algorithms can be found in the subfolder `PRoNTo/machines/`. More details on how to call the machines can be found in the following 'Model: Run' section.
- `PRT.model.input.use_nested_cv`: Whether to optimize hyperparameters (1) or not (0).
- `PRT.model.input.nested_param`: A vector of user-defined parameters if `PRT.model.input.use_nested_cv` is 1. In versions 2.X, this can also be empty, which means to leave it for the algorithm to set values by default. The default values are specified in `prt_nested_CV.m` at the 'Model: Run' step. In v3.0, this has changed. All default parameters are automatically filled in the corresponding fields of GUI or Batch, once 'Optimize hyperparameter' field is activated. Hence from v3.0 they are taken as inputs by default from user interfaces.
- `PRT.model.input.cv_type_nested`: The CV strategy for the inner CV loop.
- `PRT.model.input.cv_k_nested`: Value of k for the k-fold inner CV loop. For instance, if 10 is input, it means that the strategy takes 90% of the data for training, and 10% of the data for test.
 - Leave-one-out has k=0.
- `PRT.model.input.subsample`: Whether to subsample the different classes accordingly or not, in order to have balanced classes.
- `PRT.model.input.class`: Contains the user-specified selection for each class.
 - It's a structure of size number of classes (classification) or 1 (regression). It specifies the groups, subjects within that group and conditions for each class, with a similar structure as in `PRT.group` (without the modalities). And we identify which indices in the global ID matrix are selected to build the specified class.
 - The CV matrix and everything else are built from here. This can be viewed in Review kernel & CV. Note: its always good to have a check on the CV matrix to ensure the inputs do what you want. You can also make a CV matrix yourself and input via Custom choice in Cross-validation scheme. In the Review module, you can view the classes specified.
- `PRT.model.input.samp_idx`: Index of the selected samples for this model, in the corresponding feature set.
- `PRT.model.input.targets`: Prediction targets for building the model.
- `PRT.model.input.covar`: Values of the confounders to be regressed out.
- `PRT.model.input.cv_k`: Value of k for the k-fold outer CV loop (i.e. the main or external CV).
- `PRT.model.input.cv_mat`: CV matrix for the external CV.
- `PRT.model.input.cv_type`: CV type for the external CV.
- `PRT.model.input.operations`: Stores user selected data/kernel operations as an integer vector. The operations will be applied according to this order.

Model: Specify from

‘Model: Specify from’ module allows users to define models similar to previously specified models, instead of filling in all parameters multiple times. In GUI, this module is implemented in `prt_ui_copy_model.m` and `prt_ui_copy_model.fig`. In Batch, fields that can be modified are the feature sets, model type and data operations. The batch functions define this module are `prt_cfg_copy_model.m` and `prt_run_copy_model.m`. ‘Model: Specify from’ creates another structure in `PRT.model`. Similar to ‘Model: Specify new’, the model name and input fields are filled after this step. All the subfields inside `PRT.model.input` structure are the same to those created by ‘Model: Specify new’.

The two modules create and estimate inputs for estimating the model. The higher-level users and developers can copy `PRT.model` and change any of the sub-fields listed above when it is needed manually. CV matrix is computed based on CV schemes specified by the user.

22.5.2 Files created

No new files are created at this stage.

22.5.3 GUI behaviour

Model: Specify new

In GUI, users can choose only to specify a model, or to specify and run a model. It computes the sample indices based on the classes, computes CV matrix based on the ID matrix (and labels if ‘Leave-per-Group-Out’) and CV specifications, and it gathers the targets and covariates based on the class definition. ID matrix is used here as a reference to obtain these sub-groups of indices, targets and covariates for building a particular model. CV matrix can also be input by users via Custom CV. In v3.0, it also contains default hyperparameters of the selected machines for building the model. The function pair of `prt_ui_model.m` and `prt_ui_model.fig` define the GUI, arrange user inputs into appropriate structures and send them to several core functions (e.g., `prt_model.m`) to implement the above computations.

Model: Specify from

In GUI, it is the same as the Specify new module, users can choose Specify model alone or Specify and run model. After loading `PRT.mat`, users can select which previously built model is used as a reference. Some parts of the window is automatically filled, i.e. these information is copied from the selected referential models. For example, model type, and the main CV scheme. Users can change other activated fields, such as feature sets, machines, inner CV scheme, and data operations. Since this module copies information from the referential model, it does not carry out computations described in Specify new module. User defined changes for the model will be filled in corresponding fields in `PRT.model`. The function pair of `prt_ui_copy_model.m` and `prt_ui_copy_model.fig` define this module, organise user inputs, and updates PRT.

22.5.4 Batch behaviour

Model: Specify new

Batch provides similar interfaces and functions to the GUI, but the Specify model and run model modules are separated. This module calls `prt_model.m` to configure and build `PRT.model` structure too. `prt_cfg_model.m` and `prt_run_model.m` carry out this module in batch.

Model: Specify from

Unlike GUI, the list of built models is not automatically shown in the batch. Users shall input the exact name of the previous model to copy from. Three fields can be changed: Feature sets, Model type and Data operations. This module calls `prt_run_copy_model.m` to update `PRT.model` structure.

22.5.5 Functions called

Core:

1. `prt_model.m`

This function initialises the model data structure (by calling `prt_init_model.m`), computes targets and the sample index.

- In PRoNTo v2.1, classification and regression are addressed differently. In classification, the codes visit information stored in the design. For regression, only subject-level analysis is available, so the codes do not go into the design. The sub-functions `compute_targets` and `compute_target_reg` inside `prt_model.m` compute the prediction targets for classification and regression, respectively. Targets that are not selected for this model are put to zeros. From PRoNTo v3.0 both classification and regression compute the prediction targets using an updated version of the `compute_targets` sub-function.

One note is that the labels of different classes are 1,2,3,..., here referring to Class 1, Class 2, Class 3, ... respectively. They will be converted to class labels accepted by the actual machines inside the machine function, if needed. For example, if Faces and Houses are selected by the user as Class 1 and Class 2, respectively, and if we use SVM, the labels 1 and 2 will be converted to -1 and 1 in the machine `prt_machine_svm_bin.m`.

2. `prt_compute_cv_mat.m`

It computes the CV matrix. Inner and external CV loops call the same function. This function is accessed in the Specify model and ‘Model: Run’ modules. In PRoNTo, CV schemes are shown in abbreviations, for example:

- LOSO: leave-one-subject-out. If we have 56 subjects, and 10 folds. Then 5 subjects in each fold, the remaining 6 will be distributed to the first 6 folds. As a result, 6 subjects in the 1 to 6 folds, 5 subjects in the remaining 4 folds. The reason to do this is for distributing subjects more evenly.
- LORO:leave-one-run-out. There is no leave-multiple-run-out option in PRoNTo, only leave-one-run-out.

Other CV schemes can be found in Section 4.5 in this manual. PRoNTo also provides interfaces for users to input their own CV design, i.e. the custom CV. Users can input their own CV matrix via this interface. The custom CV saves the model and calls the `compute_cv_mat.m` if the user chooses a ‘basis’. Please note that nested CV will not be computed in ‘Model: Specify new’ step, but during the model estimation.

GUI:

3. `prt_ui_model.m` and `prt_ui_model.fig`

The GUI functions that define the interfaces of Specify model module. It calls several other functions among which:

4. `prt_ui_select_class.m` and `prt_ui_select_class.fig`

This function provide interfaces to users to select classes for classification problems.

5. `prt_ui_select_reg_new.m` and `prt_ui_select_reg_new.fig`

This function provide interfaces to users to select data for regression analysis in regression problems.

6. `prt_ui_specify_CV_basis.m` and `prt_ui_specify_CV_basis.fig`

This function provides interfaces for users to input their custom CV information. It lets users load their own CV design, or build their own CV based on several basic CV schemes available in PRoNTo, such as LOBO. These inputs serve as ‘basis’ for users to further modify CV design manually in a pop-up window after this step where `prt_ui_specify_CV_basis.m` calls `prt_ui_custom_CV.m` and `prt_ui_custom_CV.fig` to allow more custom CV specification manually.

7. `prt_model.m`

This function configures and builds the `PRT.model` structure.

8. `prt_cv_model.m`

This function runs a CV on a given model.

9. `prt_ui_copy_model.m` and `prt_ui_copy_model.fig`

This function pair defines the ‘Model: Specify from’ module.

Batch:

10. `prt_cfg_model.m` and `prt_run_model.m`

The configuration function defines the tree of the ‘Model: Specify new’ module and the interface behaviours in Batch. It provides interfaces for users to specify which feature sets to use, the model type (classification and regression), machine type (kernel or non-kernel), inner and outer CV schemes, and data operations. Default values for machine hyperparameters and arguments are automatically filled in correspondent fields. Users can change these values. It gathers these information, then passes them to `prt_run_model.m`. `prt_run_model.m` takes the user inputs and arranges them into a proper model structure. It passes `PRT.mat` and this model structure to `prt_model.m`.

11. `prt_cfg_copy_model.m` and `prt_run_copy_model.m`

The configuration file defines the tree of the Specify from module and the interface behaviours in Batch. It provides interfaces for users to choose which previously built model to copy, and which fields in the model to modify. It passes these information to `prt_run_copy_model.m`, which arranges these user inputs to a model structure with the new model name (user-specified). `prt_run_copy_model.m` stores this new model to and updates `PRT.mat`.

22.6 Model: Run

‘Model: Specify new’ and ‘Model: Specify from’ modules set up the configuration of the model and build the model structure. ‘Model: Run’ module performs the training and testing. It includes the selection of which model to run, hyperparameter optimization by using inner CV schemes, model training and testing using external CV schemes, computing statistics for model performance measures, and `PRT.mat` updates. In v3.0, we have moved permutations from the ‘Display results’ to this module. In GUI, this module can be run directly from specify model stage. In Batch, it is separated from the ‘Model: Specify new/from’ modules.

Cross-validation

CV schemes are important to model training and testing. In PRoNTo, this is mainly performed by `prt_cv_model.m`, which executes both external and inner CV. After some initialization steps, it begins the main/external CV loops. If the user chooses to optimize hyperparameters, the external CV sends data information of each fold to the inner CV. Users can choose different CV strategies for inner and external CV. The function `prt_nested_cv.m` is called to optimize hyperparameters using the inner CV scheme. After this, the best hyperparameter is returned to `prt_cv_model.m` within each external fold. Whether the hyperparameter is optimized or not, one model is estimated per fold by calling `prt_cv_fold.m`. `prt_cv_model.m` passes `PRT.mat` file and a data structure `fdata` to `prt_cv_fold.m`. The `fdata` structure contains information within each fold on ID matrix, model index, CV matrix, class labels or targets, and kernels. `prt_cv_fold.m` returns the estimated model, related parameter values and targets. This is followed by statistic computations for model performance measures per external fold, by calling `prt_stats.m`. In `prt_cv_model.m`, statistics are calculated and stored at fold level and model level. Model level statistical metrics are obtained by concatenating results across folds in versions 2.X, but by averaging metrics across folds in v3.0. `PRT.mat` is updated and saved at the end of `prt_cv_model.m`.

Hyperparameter optimization

In many situations, it is preferable to optimize hyperparameters to increase the predictive model performance. If users choose this step, each external fold constructs `fdata` structure and sends it alongside with `PRT.mat` to `prt_nested_cv.m`, which implements the inner CV loop. The training data in the external fold is

splitted into training and testing sets inside `prt_nested_cv.m` according to the user-specified inner CV scheme. Before looping over inner CV folds, `prt_nested_cv.m` checks hyperparameter ranges. Hence, this function first loops over hyperparameters within the range and then loops over inner folds for each hyperparameter. It calls `prt_cv_fold.m` to perform the model training and testing. One model is created for each inner fold, generating predictions. Within every loop using a certain hyperparameter, model performance metrics are calculated for each inner fold. Statistic calculation is done by calling `prt_stats.m`. In version 2, we concatenate predictions from each inner fold to obtain the model-level performance measures. In version 3, we average metrics across folds. Among the metrics, we use balanced accuracy for classification and MSE for regression. This process generates one metric value per hyperparameter. After completing the inner CV for all values of the hyperparameter, we choose the best one according to the metrics, i.e. the hyperparameter generating the maximum balanced accuracy is selected as the best one for classification and the hyperparameter generating the minimum MSE is selected as the best one for regression. `prt_nested_cv.m` returns the best hyperparameter to `prt_cv_model.m` for the outer model training and testing. In v3.0, hyperparameter optimization can be performed on models with more than one hyperparameters.

The machines

Model training and testing for both external and inner CV are done by calling `prt_cv_fold.m`. This function configures model parameters and applies user-specified data operations before feeding them to the machines (e.g., `prt_prepare_task_input_STL.m`). It calls `prt_machine.m` which will then distribute to the specific machine after some input checks. The machine-specific functions are in the *machines* folder, with machine in their names. Some of them are wrapper functions of the core functions in several third-party libraries, such as Libsvm, Liblinear and SimpleMKL. PRoNTo uses wrapper functions to organise inputs (data, class labels, targets, function arguments and hyperparameters) to formats accepted by actual machines in those libraries, and organize outputs from these machines to PRoNTo accepted formats. For instance, `prt_machine_svm_bin.m` is a wrapper of binary SVM in Libsvm. PRoNTo takes the hyperparameter range from GUI or Batch, then combines it with other machine arguments required by Libsvm. Training and testing datasets are organized according to Libsvm conventions. Predictions and model parameters from the binary SVM machine are stored according to PRoNTo conventions. Machine arguments include options provided by third-party libraries and hyperparameters (e.g., C for SVM). We define machine options in `prt_defaults.m`.

Generally speaking, each `prt_machine_*.m` function accepts two inputs, a structure `d` containing the data information and a structure `args` containing machine-specific information including options and hyperparameter values. In structure `d`: `d.train` and `d.test` contain the training and testing data respectively. The `d.tr_targets` refers to targets of the training data. There is no need to pass `d.te_targets` as model performance is evaluated outside of the machine, in PRoNTo `prt_stats.m`. One flag `d.use_kernel` contains information about whether the machine is a kernel machine or not. While this is functional from v3.0, in versions 2.X, this is by default always true. The output of each machine contains prediction values, function values from the decision function, model parameters and the type of the machine (classifier or regression).

Permutations

In v3.0, permutations have been moved to the ‘Model: Run’ module. `prt_permutation.m` implements the permutation test and saves the results. It accepts PRT.mat, number of permutations specified by the user, model ID (for which model to permute), path to save the results and whether to compute weights for models created by the permutations. If this option is selected, an extra field will be created inside `PRT.model`, `PRT.model.output.permutation`. This structure contains relevant information of models using permuted data, such as performance metrics, statistics, fold-level information (e.g., model parameters, function values, predictions) and the permuted data used for each permutation. Saving the permutation parameters is only optional. However if a user wants to build weights on the permutations then it is required to first save the permutation parameters.

Results

During ‘Model: Run’ module, predictions, decision function values, weights and metrics measuring the model performance are computed. These results are stored in several subfields in `PRT.output`.

Within each main CV fold, statistics are computed at fold level. In versions 2.X, after looping over all folds, we concatenate predictions from each fold and pass them to `prt_stats.m` together with true targets to compute model-level statistics. In v3.0, after looping over all folds, we compute the model level statistics by averaging fold-level stats across folds. In `prt_stats.m`, for classification, we compute the confusion matrix, accuracy, balanced accuracy, accuracy by class, predictive value for each class, as well as AUC under ROC curve for binary classification. They are implemented by `compute_stats_classifier` function. They call `prt_tpr_fpr.m` to compute AUC under ROC. For regression, we compute mean square error (MSE), normalized MSE, correlation between test and prediction, and squared correlation. They are implemented by `compute_stats_regression` function. `prt_stats.m` stores the values in structure stats and returns it to `prt_cv_model.m` and `prt_nested_cv.m`.

22.6.1 PRT fields created

Two new fields are created in PRT.mat in this module: `PRT.model.output.fold` and `PRT.model.output.stats`. In `PRT.model.output.fold`, several subfields are generated to store information of hyperparameter effects, prediction values in each fold, fold-level statistics, decision function values, model parameters and other information specific to some machines (e.g. kernel contributions for MKL models).

22.6.2 Functions called

Core:

1. `prt_cv_model.m`, `prt_nested_cv.m` and `prt_cv_fold.m`

`prt_cv_model.m` is where CV schemes are implemented within PRoNTo. It gets PRT.mat and user inputs from the interface. Before looping over CV folds, it gets the model index, configure variables based on the CV matrix, gets values and variables for model estimation, such as user-specified targets, class numbers, kernels, and machine arguments. It also initializes outputs. `prt_cv_model.m` calls `prt_getKernelModel` to loads kernels/features for kernel/non-kernel machines. `prt_cv_model.m` calls `prt_nested_cv.m` to optimize hyperparameters using inner CV by passing PRT.mat and the aforementioned fdata structure. `prt_nested_cv.m` returns the best hyperparameter value for external CV loops. `prt_cv_model.m` calls `prt_cv_fold.m` to train and test the predictive models by passing PRT.mat and fdata structure. `prt_nested_cv.m` also calls `prt_cv_fold.m` within each inner fold to estimate models. `prt_cv_fold.m` returns the model parameters and targets/labels. `prt_cv_model.m` also computes statistics for each fold and averaged statistics for the model by calling `prt_stats.m`.

2. `prt_stats.m`

This function is called by `prt_cv_model.m` and `prt_nested_cv.m` to calculate fold-level and model-level metrics measuring the model performance. It takes predictions, model type, test targets, and class numbers as inputs. For classification, the confusion matrix, accuracy, balanced accuracy, class accuracy, predictive value and AUC under ROC are calculated. For regression, MSE (and normalized MSE), correlation and squared correlation are calculated. These metric values are passed as outputs in a stats structure.

3. `prt_machine.m` and `prt_machine_*.m` functions

These functions are in the *machines* folder carrying out the actual modeling. `prt_machine.m` is called by `prt_cv_fold.m` and passes data information and machine arguments to the machine-specific functions `prt_machine_*.m`, which are generally wrapper functions of the actual machines provided by several third-party libraries. These wrapper functions perform some initial checks on the inputs, re-structure data and arguments to formats accepted by these machines, then checks outputs from the estimated models and organize them into PRoNTo accepted structures. If developers want to add your own machines, make sure the input and output are in formats compatible to both your own machine and PRoNTo.

4. `prt_permutation.m`

This function carries out the permutation tests. It accepts user inputs from GUI or Batch interfaces, configures variables (e.g., CV matrix, fold numbers, and ID matrix), loads kernels/features for kernel/non-kernel machines, permutes data according to user inputs in Data & Design module, runs models on the permuted data, calculates statistics for model performance, computes p-value and saves the results.

GUI:

5. `prt_ui_cv_model.m` and `prt_ui_cv_model.fig`

This function pair define and implement the GUI interface for ‘Model: Run’. It loads the specified models from the PRT and fills a list that the user can select from. It calls `prt_cv_model.m` and `prt_permutation.m` to execute the ‘Model: Run’ module. `prt_cv_model` is called sequentially on the models selected for estimation.

Batch:

6. `prt_cfg_cv_model.m` and `prt_run_cv_model.m`

The configuration file defines the tree structure of ‘Model: Run’ in Batch. It provides interfaces for users to load PRT.mat, fills in permutation fields for users to choose, and provides fields for users to specify related parameters. It passes user inputs to `prt_run_cv_model.m` for execution. `prt_run_cv_model.m` loads PRT.mat and configure the specified model according to user inputs, then calls `prt_cv_model.m` to run model estimation. If permutation is selected, it calls `prt_permutation.m` to perform the permutation tests.

22.7 Compute weights

Weights are computed in two steps. They are carried out by `prt_compute_weights.m`, and `prt_weights_*.m` for specific machines. For NIfTI data, weight maps can be built as voxel-wise images or ROI-wise images. For mat and MEEG data types, weights are built in similar manners to those for NIfTI, though some specific changes have been made in related functions for the two types. Hence this section focus on the weight maps of NIfTI images in the following descriptions. For multiple kernel learning, maps indicating kernel contributions with ranking information can be obtained.

Weight computation

The function `prt_compute_weights.m` accepts information from PRT.mat and some user-defined information, such as which model the weight image is for, where to save, flags indicating whether to build weights for each permutation and/or for each ROI, and atlas for ROI-wise weight image if it is selected. If different modalities were used without concatenating the samples, one weight map is built for each modality.

For each modality, weight computation is classified into classification and regression problems. One weight image is built for each class in case of multi-class problems, whilst only one weight map is built for binary classification and regression problems. These are implemented in the scripts `prt_compute_weights_class.m` and `prt_compute_weights_regr.m`, respectively. Processing routines for classification and regression are similar to each other. Take classification as an example to illustrate the routine. `prt_compute_weights_class.m` finds the type of data (NIfTI, .mat or MEEG) and machine first, then creates image names. One part needs some care is the multiple indexing step. In the ‘Data & Design’ module, users specify a 1st-level mask. The indices of features (e.g., voxels in NIfTI files) based on this mask are stored in `PRT.fas` structure. In the feature set preparation step, if users have specified a second-level mask and/or atlas, indices of features used to build feature sets/kernels based on these are stored in `PRT.fs` structure. For example, in the ‘Prepare feature set’ module, we need a subset of images to build the file array, from which we use potentially a subset to build a kernel. Furthermore, to avoid overloading the memory, the weight image is built slice by slice along the z axis, since most images are 3-D files. So we need to take indices from that slice, within both the second-level and the first-level masks. The three levels of indices are necessary to build weight maps. Hence, there is a need to perform the multi-level indexing search. This includes getting the correct indices from the 1st-level file array (.dat file) linking to each modality, from the 2nd-level feature sets (`fs` structure), and from each slice along z axis, potentially within ROIs defined by 2nd-level mask and/or atlas.

The next step is to create maps and check if weight images need to be created for the permutation test. Image initialization is performed before the weight computation. Weight images are created along z axis. One image is built for each fold, which results in a 4-D image where the last map is an average over all folds. Data operations are applied before weight computation. Then the data and information related to the machine used to generate the model are sent to `prt_weights.m`. The code performing actual multiplication for weights is done by `prt_weights.m` in the *machines* folder within `prt_compute_weights_class.m` and `prt_compute_weights_regr.m`.

This function calls other `prt_weights_*.m` in the *machines* folder. Since different machines have different formulations, computing weights using outputs from these machines may vary too. Take `prt_weights_sMKL_cla.m` as an example. The inputs are structure `d` and args similar to inputs to the machines. But the data loaded in `prt_compute_weights_class.m` and `prt_compute_weights_regr.m` is `d.datamat`. By taking coefficients from structure `d` and data from `d.datamat`, multiplications for weight values are carried out in these `prt_weights_*.m` functions. `prt_weights.m` returns a weight vector to `prt_compute_weights_class.m`. Following procedures are to build the weights, reshape them into the correct dimensions, normalize the weights for visualization and save the images. In `prt_compute_weights.m`, the variable `flag2` tells if we want to get one weight per region instead of one weight per voxel, so weight computation will be changed according to this.

22.7.1 PRT fields created

In `PRT.mat`, extra fields are created: `PRT.model.output.weight_img`, `PRT.model.output.weight_ROI` if building weights for ROIs is selected, subfields saving ranking information of weights, and `PRT.model.output_weight_MOD` if weights for MKL are selected.

22.7.2 Files created

This module creates weight image files: weight maps with ‘weights_’ and/or ‘ROI_weights_’ in their names. The default naming rule is to use ‘weights_’ appended with the model name. If there are multiple classes or modalities, the class number or modality name will be appended to that name. Maps of ROIs are separated from voxel-wise maps which can be identified by names. If building weight images for permutations is selected, extra folders (with names `perm_*` for voxelwise weights and `perm_ROI_*` for ROI-wise weights, respectively) containing weight maps per permutation are created.

22.7.3 Functions called

1. `prt_compute_weights.m`, `prt_compute_weights_class.m`, and `prt_compute_weights_regr.m`

PRoNTo calls `prt_compute_weights.m` to implement weight image computation. It accepts several inputs: `PRT.mat`, user defined inputs including model name, image name, path to save the maps, atlas, one flag indicating whether to build weight images for permutations, and another flag indicating whether to build ROI-wise weight images. It outputs image files with specific names. This function computes the number of images needs to be built according to the number of modalities, and whether ROI-wise summary weight maps are selected by users. It finds the right model, loops over feature sets and/or modalities (over feature sets in v3.0, over modalities in versions 2.X, if building one kernel per modality) and saves `PRT.mat`. Within feature set loops, it further assesses the number of images according to whether it is a classification problem (class numbers) or regression problem. The actual weight map construction is carried out by `prt_compute_weights_class.m` and `prt_compute_weights_regr.m` for classification and regression problems, respectively. Computing weights for each voxel for a whole brain and for ROIs are processed differently by calling the two functions. If ROI-wise weight images are selected to be built, a flag ‘`flag2`’ will be set to 1 and sent to the two functions.

- `prt_compute_weights_class.m` and `prt_compute_weights_regr.m` also deal with weight computations for permutations. The following procedures are initializing the image, building images slice by slice along the z axis, building one image per fold, applying operations, building weights (a 4D image, the last dimension is the number of folds plus one average map across folds), normalizing the weights for visualization and save the images. They call `prt_weights.m` and `prt_weights_*.m` to perform the actual multiplications for weight computations.
- `prt_compute_weights_class.m` and `prt_compute_weights_regr.m` save weight maps and return the image names to `prt_compute_weights.m`. `prt_compute_weights.m` fills weight fields in `PRT.model.output` and updates `PRT.mat`.

2. `prt_weights.m`, `prt_weights_bin_linkernel.m`, and `prt_weights_*.m` for other machines

These functions are in the *machines* folder. Formulations of different machines may vary, so weight computations may be machine-specific. For the kernel machines, the coefficients are obtained from the kernel space, then mapped back to the original space. When adding new machines, corresponding weight computations shall be considered and added too. There are some general cases in PRoNTo,

where `prt_weights_bin_linkernel.m` can be used. Other `prt_weights_*.m` functions implement multiplications for machines beyond the linear kernel case.

One example of the machine-specific weight function is `prt_weights_sMKL_cla.m`. It accepts input structures `d` and `args`, creates weight vector as the output. A list of subfields in structure `d` is shown as follows:

- `d.datamat`: the data being loaded in `prt_compute_weights_class.m` and `prt_compute_weights_regr.m` of dimension number of #features x #number of examples, corresponding to the features in the training set.
- `d.coeffs`: coefficients, such as alphas generated by using SVM.
- `d.betas`: this is the kernel contributions, specific to sMKL.
- `d.idfeat_img`: voxel indices of the features in the image.

Chapter 23

PRoNTo functions and the PRT structure

Contents

23.1	List of PRoNTo functions	249
23.2	The PRT structure	249

23.1 List of PRoNTo functions

For the users interested in delving deeper into the PRoNTo functions, there is a set of HTML files of all the functions inside the *PRoNTo/manual/html_doc* directory that can be opened with any application that supports HTML.

The *index.html* is the main HTML file which has a short description of all the PRoNTo functions. Scrolling to the end of it, the user will also find the index to all the subdirectories.

Clicking into a specific PRoNTo function, for example *prt_cv_model*, one could see:

- A synopsis section, which informs us that this is a function related to cross-validation for a given model,
- A further description section, with the way the inputs and the outputs of this function are structured, potential notes, etc.
- A cross-reference information section, which informs us of the functions that *prt_cv_model* calls, as well as the functions that call *prt_cv_model*.
- A subfunctions section, which lists all the subfunctions in our main function.
- And finally a source code section, which is just the function itself.

Users interested in customizing PRoNTo functions to their needs and/or further developing, are advised to make use of the HTML file indexing to get a better sense of the relations between the PRoNTo functions.

23.2 The PRT structure

Another useful thing to have is the whole picture of the PRT structure. Exploring the PRT structure and learning your way around it, will certainly prove to be quite useful for various reasons.

The most straight-forward way of exploring the PRT structure is through the MATLAB Workspace. But this way has a couple of difficulties, and usually you cannot see the whole picture of the structure because of MATLAB limitations.

Another much more practical way of visualizing the PRT structure is by converting it to a JSON file format and then opening it with any compatible application. There are various different software programs with which you can visualize the structure in a much more meaningful way. You can even find Internet browsers (for example *Mozilla Firefox*) that have built-in JSON viewer where simply dragging & dropping the *.json* file in the browser will open it. A couple of other useful advantage of the JSON format is that:

- You can use it to store and/or transfer the information of a PRT structure without using much space (as a JSON file format is just an array of strings), and without requiring MATLAB to be installed in order to open it.
- You can parse the PRT structure quite easily.

The most straight-forward way of converting a PRT structure to a JSON file format and vice versa is through the *spm_jsonwrite()* (to encode) and the *jsondecode()* (to decode) commands.

- **spm_jsonwrite('prt_struct.json',PRT)** will encode your current PRT in a JSON file format with the name *prt_struct* and directly save it in your current folder. That is an SPM function and hence requires you to have SPM in your path.
- **PRT_rec = jsondecode(fileread('prt_struct.json'))** will decodes it and recreate the PRT structure in a struct called *PRT_rec*. That is a MATLAB built-in function introduced with version R2016b.

There are of course many more ways of encoding/decoding MATLAB to JSON and vice versa, such as for example JSONlab, or even using the MATLAB built-in function *jsonencode* and then saving it by yourself.

Inside the *PRoNTo/manual* you can find a *prt_struct.json* file that we created based on the PRT structure of the tutorial in Chapter 13.

Part V

Appendix

Chapter 24

Appendix

24.1 One data file per subject

For behavioural data, it is common to save data from all subjects in a single file, with each row being a sample (subject or trial) and each column being a variable (or feature). PRoNTo however does not accept this format as multiple rows could arise from another input dimensionality of the data that the user wishes to keep (e.g. 2D connectivity matrix instead of vectorized).

We hence provide a script that transform a #samples x #features matrix into a series of .mat files. If the data for each subject comes from a distance measure (e.g. correlation), an additional flag allows to turn the vector back to its original 2D form.

The script takes in the matrix as well as a true-false statement (represented as 1 and 0 numericals) to decide whether to infer 2D matrix from a distance vector (1 to try to perform this operation, 0 to keep the vector, default: 0). Example: variable ‘fmri’ contains the connectivity vector (upper triangular only) for each of 293 subjects.

```
> [path_files] = script_create_one_file_per_row(fmri, 1);
```

The output is a new folder called Samples_mat that contains files ‘Samples_001.mat’, ..., ‘Samples_293.mat’ (here for 293 subjects). Each file contains one variable called ‘data’ that represents the data for this sample. In the present case, ‘data’ is a 349*349 matrix.

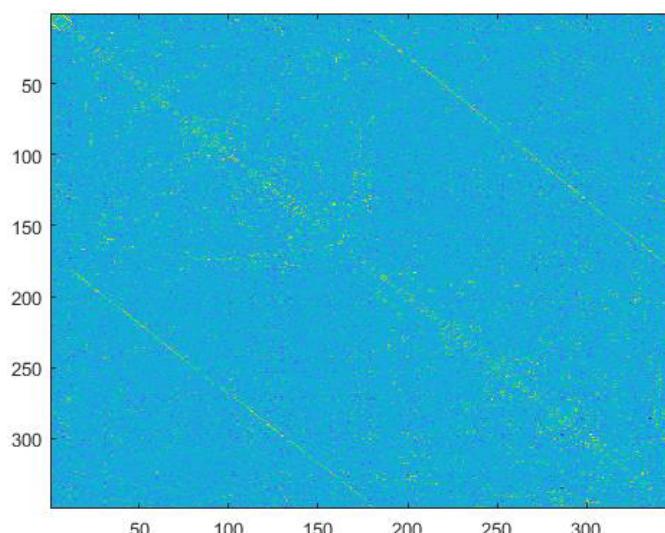


Figure 24.1

Script:

```

1 function [ path_files ] = script_create_one_file_per_row( data_matrix , tosquare )
2 % Creates one file per row of a data matrix. The files will be saved in a
3 % subfolder , with the names 'Sample_ ' prepended. The second input 'tosquare '
4 % reflects whether the data should be turned into a symmetric matrix or not
5 % (0 , default value)
6 %
7 % Written for PRoNTo v3.0 by J. Schrouff
8
9 if nargin<1 || isempty( data_matrix )
10    beep
11    disp( 'At least one variable must be specified , the data matrix to convert ' )
12    return
13 end
14 if nargin<2 || isempty( tosquare )
15    tosquare = 0;
16 end
17 nsamp = size( data_matrix ,1 );
18 d = dir( '.' );
19 path_files = fullfile(d(1).folder , 'Samples_mat' );
20 if ~exist( path_files , 'dir' )
21    mkdir( path_files );
22 end
23 cd( path_files );
24 fprintf([ 'Sample (out of %d): ' , repmat( ' ' ,1 ,ceil(log10(nsamp))) , '%d' ] ,nsamp , 1);
25 for i = 1:nsamp
26
27    % Subject counter
28    if i>1
29        for idisp = 1:ceil(log10(i)) % delete previous counter display
30            fprintf( '\b' );
31        end
32        fprintf( '%d' ,i );
33    end
34    % Access sample's data
35    datas = data_matrix(i ,: );
36    if tosquare
37        try
38            data = squareform( datas );
39        catch
40            data = datas;
41        end
42    else
43        data = datas;
44    end
45    prep = [];
46    for idisp = 1:floor(log10(nsamp))-floor(log10(i)) % Add zeros in front of
47        name for easier access
48        prep = [ prep , '0' ];
49    end
50    fname = [ 'Sample_ ' ,prep ,num2str(i) , '.mat' ];
51    save(fname , 'data' );
52 end
53 fprintf( '\n' );

```

24.2 Compute atlas for connectivity matrix

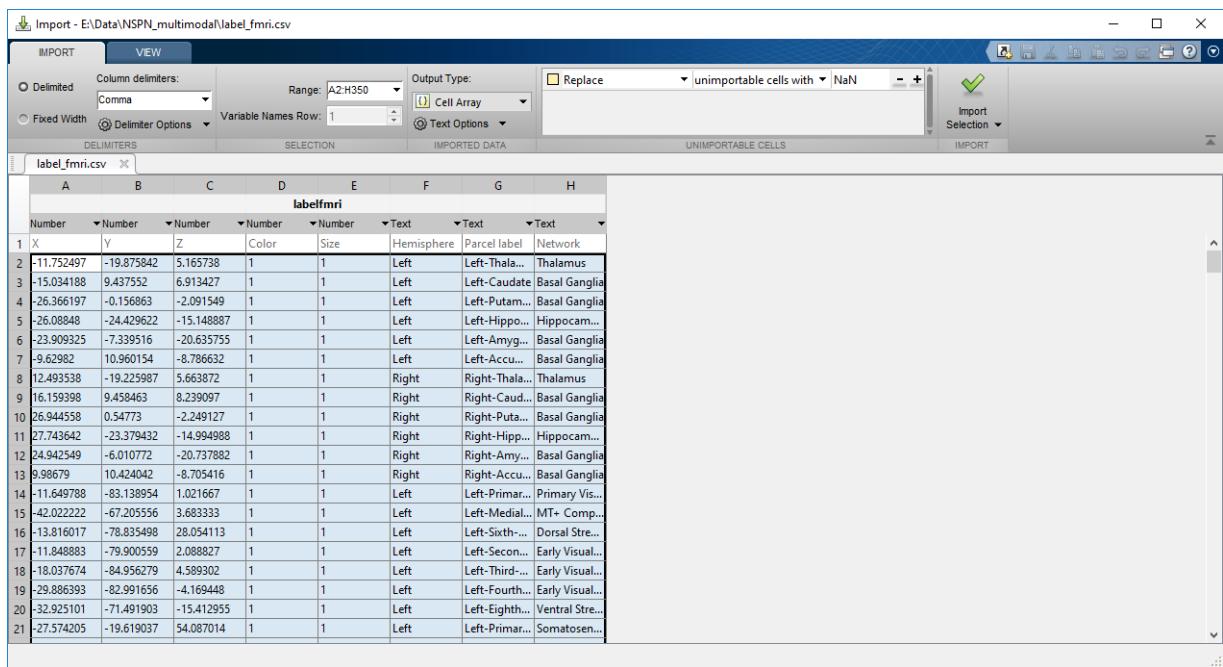
In order to build connectivity matrices based on time series extracted from a brain parcellation, you first need to have a list of anatomical parcels, originating from some brain parcellation, and also to have each of them associated to a larger ‘network’.

In the example here, there are 349 anatomical parcels with their characteristics (columns A through F) and their names (column G), and they are all associated to larger ‘networks’ (column H). You then need to save your file in a .csv (or an Excel) file.

	A	B	C	D	E	F	G	H
1	X	Y	Z	Color	Size	Hemisphere	Parcel label	Network
2	-11.7525	-19.8758	5.165738		1	1	Left-Thalamus-Proper	Thalamus
3	-15.0342	9.437552	6.913427		1	1	Left-Caudate	Basal Ganglia
4	-26.3662	-0.156863	-2.09155		1	1	Left-Putamen	Basal Ganglia
5	-26.0885	-24.4296	-15.1489		1	1	Left-Hippocampus	Hippocampus
6	-23.9093	-7.33952	-20.6358		1	1	Left-Amygdala	Basal Ganglia
7	-9.62982	10.96015	-8.78663		1	1	Left-Accumbens-area	Basal Ganglia
8	12.49354	-19.226	5.663872		1	1	Right-Thalamus-Proper	Thalamus
9	16.1594	9.458463	8.239097		1	1	Right-Caudate	Basal Ganglia
10	26.94456	0.54773	-2.24913		1	1	Right-Putamen	Basal Ganglia
11	27.74364	-23.3794	-14.995		1	1	Right-Hippocampus	Hippocampus
12	24.94255	-6.01077	-20.7379		1	1	Right-Amygdala	Basal Ganglia
13	9.98679	10.424042	-8.705416		1	1	Right-Accumbens-area	Basal Ganglia

Figure 24.2

This file is then loaded in Matlab (double-clicking or ‘uiopen’) as a cell array. **Note:** Be careful to modify the ‘Output Type’ field whenever needed.



	A	B	C	D	E	F	G	H
1	labelfmri							
Number	Number	Number	Number	Number	Number	Text	Text	Text
2	-11.752497	-19.875842	5.165738	1	1	Left	Left-Thala...	Thalamus
3	-15.034188	9.437552	6.913427	1	1	Left	Left-Caudate	Basal Ganglia
4	-26.366197	-0.156863	-2.091549	1	1	Left	Left-Putam...	Basal Ganglia
5	-26.08848	-24.429622	-15.148887	1	1	Left	Left-Hippo...	Hippocampus
6	-23.909325	-7.339516	-20.635755	1	1	Left	Left-Amyg...	Basal Ganglia
7	-9.62982	10.960154	-8.786632	1	1	Left	Left-Accu...	Basal Ganglia
8	12.493538	-19.225987	5.663872	1	1	Right	Right-Thala...	Thalamus
9	16.159396	9.458463	8.239097	1	1	Right	Right-Caud...	Basal Ganglia
10	26.944558	0.54773	-2.249127	1	1	Right	Right-Puta...	Basal Ganglia
11	27.743642	-23.379432	-14.994988	1	1	Right	Right-Hipp...	Hippocampus
12	24.942549	-6.01077	-20.737882	1	1	Right	Right-Amy...	Basal Ganglia
13	9.98679	10.424042	-8.705416	1	1	Right	Right-Accu...	Basal Ganglia
14	-11.649788	-83.138954	1.021667	1	1	Left	Left-Primar...	Primary Vis...
15	-42.022222	-67.205556	3.683333	1	1	Left	Left-Medial...	MT+ Comp...
16	-13.816017	-78.835498	28.054113	1	1	Left	Left-Sixth...	Dorsal Stre...
17	-11.848883	-79.900559	2.088827	1	1	Left	Left-Secon...	Early Visual...
18	-18.037674	-84.956279	4.589302	1	1	Left	Left-Third...	Early Visual...
19	-29.886393	-82.991656	-4.169448	1	1	Left	Left-Fourth...	Early Visual...
20	-32.925101	-71.491903	-15.412955	1	1	Left	Left-Eighth...	Ventral Stre...
21	-27.574205	-19.619037	54.087014	1	1	Left	Left-Primar...	Somatosens...

Figure 24.3

The last column of the cell array will then represent the ‘network’ each time series belongs to. The variable name of the whole cell array in this example is ‘labelfmri’, and it is only the last column (*labelfmri(:, end)*) containing the ‘networks’ that you need to pass to the script.

```
> [atl_mat, ROI_names] = script_build_atlas_from_cell_array(labelfmri(:, end));
```

This outputs and saves the atlas file and its corresponding labels for use in PRoNTo. **Note:** Please ensure to rename both of these files if you need to create multiple atlases (otherwise they will be over-written).

The atlas is a 2D matrix of size 349*349 that contains an integer for each network i-network j interaction. Here, 25 ‘networks’ were defined, giving 325 pairwise interactions ($n*(n-1)/2$, with n the number of networks). The matrix is upper diagonal to ensure to mask out redundant features (as correlations are undirected). **Note:** For directed interactions, the full matrix would need to be generated (and the script to be modified).



Figure 24.4

Script:

```

1 function [atl_mat,ROI_names] = script_build_atlas_from_cell_array(region_list)
2 if nargin<1 || isempty(region_list)
3     beep;
4     disp('Import excel file into cell array first')
5     return
6 end
7 if size(region_list,2)>1
8     beep
9     disp('Please only provide the list of regions, no other variable')
10    return
11 end
12 if ~iscell(region_list(1))
13     beep
14     disp('List of regions must be a cell array')
15     return
16 end
17 if isstring(region_list{1})
18     region_list = cellstr(region_list); % Convert to cell array of characters
19 end
20
21 % Gather ROI names and labels
22 labs = unique(region_list);

```

```

23 n = numel(labs);
24 % Build atlas (squareform)
25 % _____
26 % Initialize matrices
27 cnt=1;
28 a = zeros(numel(region_list),numel(region_list));
29 c = zeros(size(a));
30
31 % Loop over each pair of 'networks' and give it a unique number (i.e.
32 % the same unique number to all regions pertaining to the pair)
33 labels = cell((n*(n-1))/2 + n,1);
34 for i=1:n
35     indi = ismember(region_list ,labs{i});
36     for j=i:n
37         indj = ismember(region_list ,labs{j});
38         nroi1 = nnz(indi);
39         nroi2 = nnz(indj);
40         roi = cnt*ones(nroi1 ,nroi2);
41
42         % Build diagonal matrix (i.e. (i,i) interactions)
43         if j==i
44             c(indi ,indj)= roi;
45         end
46         a(indi ,indj)= roi;
47         labels{cnt}=[labs{i} , '-' , labs{j }];
48         cnt=cnt+1;
49     end
50 end
51
52 % Maybe regions were not sorted before we built the matrix. This will
53 % result in a matrix a that is not upper triangular (not symmetric). We
54 % correct for this with the following:
55 b = a+a'-c;
56
57 % b is now symmetric, and we can extract its upper triangle to match the
58 % inputs from the connectivity matrix
59 mask = triu(true(size(b)),1);
60 atl_mat = b.*mask;
61 save('atlas.mat','atl_mat');
62
63 % Save them in a file for display with the weights
64 ROI_names = labels;
65 save('Labels_atlas.mat','ROI_names')

```

24.3 Connectivity matrix from MEEG

Below is a script provided, if one wants to create a connectivity matrix from averaged EEG or MEG trials (here both). The script as it is, goes through all the subjects of the Multimodal SPM dataset, computes correlations between all variables and creates the connectivity matrix, which is then saved inside each subject's folder. These connectivity matrices created by this script are also the ones we use in the tutorial. In order for this script to run, you only need to modify the 'prepath' accordingly.

```

1 prepath = '**\tutorials_v3_with_data\Multimodal_SPM_preprocessed\data\';
2 subpath={'\EEG\','\MEG\'};
```

```

3 fname = 'mapMcbspmeeg_run_01_sss.mat';
4 type = { 'EEG' , 'MEG' };
5
6 for i = 1:16
7     disp(['Computing subject S', num2str(i)])
8     for t = 1:2
9         fullp = [ prepath , 'S' , num2str(i) , subpath{t} , type{t} ];
10        filename = [ fullp , fname ];
11        D = spm_eeg_load(filename);
12        dim = D.size();
13        for c = 1:dim(end)
14            conn_mat = corr(D(:, :, c)', D(:, :, c)');
15            mask = triu(true(size(conn_mat)), 1);
16            conn_mat = conn_mat(mask);
17            cond = D.conditions(c);
18            savename = [ fullp , cond{1} , '_connectivity_matrix.mat' ];
19            save(savename, 'conn_mat');
20            disp(type{t})
21            disp(['Size: ', num2str(size(conn_mat))])
22        end
23    end
24 end

```

Alternatively, the matrices can be saved as 2D arrays instead of taking only the upper triangular part (remove line ‘`conn_mat = conn_mat(mask)`’). In this case, a second-level mask should however be provided (save the defined mask in a separate file, i.e. `save(mask.mat , 'mask')`).

24.4 Connectivity ROI weights

The reader is referred to 17.1.9 for information related to the use of these scripts. In order for thesee scripts to run, you only need to modify the ‘`prepath`’ accordingly.

Script 1: Gather the atlas information (hard-coded here) and compute a summarized ‘network’ matrix (i.e. #networks x #networks instead of #channels x #channels).

```

1 % Load atlas to access the mask
2 prepath = '**\tutorials_v3_with_data\Multimodal_SPM_preprocessed\data\';
3 addpath(prepath);
4 load('EEG_atlas.mat')
5 id = find(mask);
6
7 % Atlas info for display and summarization
8 labs = { 'OF' , 'LF' , 'CF' , 'RF' , 'LC' , 'CC' , 'RC' , 'LP' , 'CP' , 'RP' , 'O' };
9 OF = [2 ,4:8];
10 LF=[9:11 ,18:21];
11 CF=[12:14 ,22:24];
12 RF=[15:17 ,25:28];
13 LC=[29:32 ,41:43];
14 CC=[33:35 ,44:46];
15 RC=[36:39 ,47:49];
16 LP=[40 ,51:53 ,62 ,63];
17 CP=[54:58 ,64];
18 RP=[50 ,59 ,60 ,61 ,65 ,66];
19 O=[1 ,3 ,67:70];
20 ind = {OF,LF,CF,RF,LC,CC,RC,LP,CP,RP,O};

```

```

21 % load weights and access the average fold
22 % (this file should be in the same folder as your PRT.mat)
23 load('ROI_weights_ConnEEG_atlas_MKL.mat')
24 w_av = squeeze(weights(:,:,:,:));
25
26 % Weights, reconstructed on the original matrix
27 or_weights = zeros(size(mask,1),size(mask,2));
28 or_weights(id) = w_av;
29
30 % Summarize weights in each ROI as a single value for plotting
31 sum_weights = zeros(length(ind),length(ind));
32 w_sym = or_weights + or_weights';
33 for i = 1:length(ind)
34     for j=1:length(ind)
35         sum_weights(i,j) = max(max(w_sym(ind{i},ind{j})));
36     end
37 end
38
39 % Imagesc matrix
40 cc = cbrewer('seq','Reds',200);
41 figure;
42 imagesc(sum_weights);
43 if min(min(sum_weights))==0 %if some weights are perfectly 0
44     cc(1,:)=[0.8 0.8 0.8];
45 end
46 colormap(cc);
47 colorbar;
48 set(gca,'XTick',1:numel(labs))
49 set(gca,'YTick',1:numel(labs))
50 set(gca,'XTickLabel',labs)
51 set(gca,'YTickLabel',labs)
52
53 % Use modified version of schemaball to plot the summarized weights
54 schemaball_JS(sum_weights,labs,cc,[0 1 1])

```

Script 2: Plot schemaball of kernel contributions from a symmetric 2D matrix (first input), the network labels (a cell array, 2nd input), the colormap to plot the curves in (3rd input) and the color of the nodes to represent within-network contributions (4th input). This code was modified from the schemaball.m written by Oleg Komarov found on Matlabs file exchange.

```

1 function h = schemaball_JS(r, lbls, ccolor, ncolor)
2 % SCHEMABALL Plots correlation matrix as a schemaball
3 %
4 % SCHEMABALL(R) R is a square numeric matrix with values in [-1,1].
5 %
6 % NOTE: only the off-diagonal lower triangular section of R is
7 % considered, i.e. tril(r,-1).
8 %
9 % SCHEMABALL(..., LBLS, CCOLOR, NCOLOR) Plot schemaball with optional
10 % arguments (accepts empty args).
11 %
12 % - LBLS Plot a schemaball with custom labels at each node.
13 % LBLS is either a cellstring of length M, where
14 % M = size(r,1), or a M by N char array, where each
15 % row is a label.

```

```

16 %
17 % - CCOLOR Supply an RGB triplet that specifies the color of
18 % the curves. CURVECOLOR can also be a 2 by 3 matrix
19 % with the color in the first row for negative
20 % correlations and the color in the second row for
21 % positive correlations.
22 %
23 % - NCOLOR Change color of the nodes with an RGB triplet.
24 %
25 %
26 % H = SCHEMABALL(...) Returns a structure with handles to the graphic objects
27 %
28 % h.l handles to the curves (line objects), one per color shade.
29 % If no curves fall into a color shade that handle will be NaN.
30 % h.s handle to the nodes (scattergroup object)
31 % h.t handles to the node text labels (text objects)
32 %
33 %
34 % Examples
35 %
36 % % Base demo
37 % schemaball
38 %
39 % % Supply your own correlation matrix (only lower off-diagonal triangular part
40 % is considered)
41 % x = rand(10).^3;
42 % x(:,3) = 1.3*mean(x,2);
43 % schemaball(x)
44 %
45 % % Supply custom labels as [ 'aa'; 'bb'; 'cc'; ...] or { 'Hi', 'how', 'are' ,...}
46 % schemaball(x, repmat(( 'a': 'j' ),1,2))
47 % schemaball(x, { 'Hi' , 'how' , 'is' , 'your' , 'day?' , 'Do' , 'you' , 'like' , 'schemaballs'
48 % ? , 'NO!!' })
49 %
50 % % Customize curve colors
51 % schemaball([],[],[1,0,1;1 1 0])
52 %
53 % % Customize node color
54 % schemaball([],[],[],[0,1,0])
55 %
56 % % Customize manually other aspects
57 % h = schemaball;
58 % set(h.l(~isnan(h.l)), 'LineWidth',1.2)
59 % set(h.s, 'MarkerEdgeColor','red','LineWidth',2,'SizeData',100)
60 %
61 % Additional features:
62 % - <a href="matlab: web('http://www.mathworks.com/matlabcentral/fileexchange
63 % /42279-schemaball',' -browser')">FEX schemaball page</a>
64 % - <a href="matlab: web('http://www.stackoverflow.com/questions/17038377/how-to
65 % -visualize-correlation-matrix-as-a-schemaball-in-matlab/17111675',' -browser')">Origin: question on Stackoverflow.com</a>
66 % - <a href="matlab: web('https://github.com/GuntherStruyf/matlab-tools/blob/
67 % master/schemaball.m',' -browser')">Schemaball by Gunther Struyf</a>
68 %
69 % See also: CORR, CORRPLOT
70 % Author: Oleg Komarov (oleg.komarov@hotmail.it)

```

```

67 % Tested on R2013a Win7 64 and Vista 32
68 % 15 jun 2013 - Created
69
70 %% Parameters
71 % Tweak these only
72 % Number of color shades/buckets (large N simply creates many perceptually
    indifferent color shades)
73 N = 100;
74 % Points in [0, 1] for bezier curves: leave space at the extremes to detach a
    bit the nodes.
75 % Smaller step will use more points to plot the curves.
76 t = (0.025: 0.05 :1)';
77 % Nodes edge color
78 % ecolor = [.25 .103922 .012745];
79 ecolor = [0 0 0];
80 % Text color
81 tc当地 = [.3 .3 .3];
82
83 %% Checks
84
85 % Ninput
86 narginchk(0,4)
87
88 % Some defaults
89 if nargin < 1 || isempty(r); r = (rand(50)*2-1).^29; end
90 sz = size(r);
91 if nargin < 2 || isempty(lbls); lbls = cellstr(reshape(sprintf('%-4d', 1:sz(1)))
    ,4,sz(1))'; end
92 if nargin < 4 || isempty(ncolor); ncolor = [0 0 1]; end
93
94 % R
95 if ~isnumeric(r) || any(abs(r(:)) > 1) || sz(1) ~= sz(2) || numel(sz) > 2 || sz
    (1) < 3
    error('schemaball:validR', 'R should be a square numeric matrix with values
        in [-1, 1].')
96
97 end
98
99 % Lbls
100 if (~ischar(lbls) || size(lbls,1) ~= sz(1)) && (~iscellstr(lbls) || ~isvector(
    lbls) || length(lbls) ~= sz(1))
    error('schemaball:validLbls', 'LBLS should either be an M by N char array or
        a cellstring of length M, where M is size(R,1).')
101
102 end
103 if ischar(lbls)
    lbls = cellstr(lbls);
104 end
105
106
107 % Ccolor
108 if nargin < 3 || isempty(ccolor)
    ccolor = hsv2rgb([[linspace(.8333, .95, N); ones(1, N); linspace(1,0,N)] ,...
109     [linspace(.03, .1666, N); ones(1, N); linspace(0,1,N)] ]');
110 else
    szC = size(ccolor);
    if ~isnumeric(ccolor) || szC(2) ~= 3
        error('schemaball:validCcolor', 'CCOLOR should be a 1 by 3 or 2 by 3
            numeric matrix with RGB colors.')
111 elseif szC(1) == 1

```

```

116     ccolor = [ccolor; ccolor];
117     ccolor = rgb2hsv(ccolor);
118     ccolor = hsv2rgb([repmat(ccolor(1,1:2),N,1), linspace(ccolor(1,end),0,N)
119                      ;
120                      repmat(ccolor(2,1:2),N,1), linspace(0,ccolor(2,end),N)']);
121 elseif szC(1) == 2
122     ccolor = rgb2hsv(ccolor);
123     ccolor = hsv2rgb([repmat(ccolor(1,1:2),N,1), linspace(ccolor(1,end),0,N)
124                      ;
125                      repmat(ccolor(2,1:2),N,1), linspace(0,ccolor(2,end),N)']);
126 else
127     N = floor(size(ccolor,1)/2);
128 end
129 end
130 % Ncolor
131 szN = size(ncolor);
132 if ~isnumeric(ncolor) || szN(2) ~= 3 || szN(1) > 1
133     error('schemaball:validNcolor', 'NCOLOR should be a single RGB color, i.e. a
134         numeric row triplet.')
135 end
136 ncolor = rgb2hsv(ncolor);
137 %% Engine
138 %% Create figure
139 figure('renderer','zbuffer','visible','off')
140 axes('NextPlot','add')
141 %% Index only low triangular matrix without main diag
142 tf = tril(true(sz),-1);
143 %% Index correlations into bucketed colormap to determine plotting order (darkest
144 %% to brightest)
145 N2 = 2*N;
146 [n, isrt] = histc(r(tf), linspace(min(r(tf)),max(r(tf)) + eps(100),N2 + 1));
147 plotorder = reshape([N:-1:1; N+1:N2],N2,1);
148 %% Retrieve pairings of nodes
149 [row,col] = find(tf);
150 %% Use tau http://tauday.com/tau-manifesto
151 tau = 2*pi;
152 %% Positions of nodes on the circle starting from (0,-1), useful later for label
153 %% orientation
154 step = tau/sz(1);
155 theta = -.25*tau : step : .75*tau - step;
156 %% Get cartesian x-y coordinates of the nodes
157 x = cos(theta);
158 y = sin(theta);
159 %% PLOT BEZIER CURVES
160 %% Calculate Bx and By positions of quadratic Bezier curves with P1 at (0,0)
161 %% B(t) = (1-t)^2*P0 + t^2*P2 where t is a vector of points in [0, 1] and
162 %% determines, i.e.

```

```

167 % how many points are used for each curve , and P0–P2 is the node pair with (x,y)
168 % coordinates .
169 t2 = [1-t , t ].^ 2 ;
170 s . l = NaN(N2,1) ;
171
172 % LOOP per color bucket
173 for c = 1:N2
174     pos = plotorder(c) ;
175     idx = isrt == pos ;
176     if nnz(idx)
177         Bx = [t2*[x(col(idx)); x(row(idx))]; NaN(1,n(pos))] ;
178         By = [t2*[y(col(idx)); y(row(idx))]; NaN(1,n(pos))] ;
179         s . l(c) = plot(Bx(:),By(:), 'Color' ,ccolor(pos,:), 'LineWidth' ,1) ;
180     end
181 end
182
183 % PLOT NODES
184 % Do not rely that r is symmetric and base the mean on lower triangular part
185 % only
186 [row,col] = find(tf(end:-1:1,end:-1:1) | tf) ;
187 subs = col ;
188 iswap = row < col ;
189 tmp = row(iswap) ;
190 row(iswap) = col(iswap) ;
191 col(iswap) = tmp ;
192
193 % Plot in brighter color those nodes with larger within connections
194 [Z,isrt] = sort(diag(r)) ;
195 Z = (Z-min(Z)+0.01)/(max(Z)-min(Z)+0.01) ;
196 ncolor = hsv2rgb([ repmat(ncolor(1:2), sz(1),1) Z*ncolor(3)]) ;
197 s . s = scatter(x(isrt),y(isrt),round(Z*48), ncolor, 'fill') ;
198
199 % PLACE TEXT LABELS such that you always read 'left to right'
200 ipos = x > 0;
201 s . t = zeros(sz(1),1);
202 s . t( ipos ) = text(x( ipos )*1.1, y( ipos )*1.1, lbls( ipos ), 'Color' ,tcolor);
203 set(s.t( ipos ),{ 'Rotation' }, num2cell(theta(ipos)'/tau*360))
204 s . t(~ipos) = text(x(~ipos)*1.1, y(~ipos)*1.1, lbls(~ipos), 'Color' ,tcolor);
205 set(s.t(~ipos ),{ 'Rotation' }, num2cell(theta(~ipos)'/tau*360 - 180), 'Horiz' ,
206 right )
207
208 % ADJUST FIGURE height width to fit text labels
209 xtn = cell2mat(get(s.t, 'extent')) ;
210 post = cell2mat(get(s.t, 'pos')) ;
211 sg = sign(post(:,2)) ;
212 posfa = cell2mat(get([ gcf gca ], 'pos')) ;
213
214 % Calculate xlim and ylim in data units as x (y) position + extension along x (y)
215 ylims = post(:,2) + xtn(:,4).*sg ;
216 ylims = [ min(ylims) , max(ylims) ] ;
217 xlims = post(:,1) + xtn(:,3).*sg ;
218 xlims = [ min(xlims) , max(xlims) ] ;
219
220 % Stretch figure
221 posfa(1,3) = (( diff(xlims)/2 - 1)*posfa(2,3) + 1) * posfa(1,3) ;
222 posfa(1,4) = (( diff(ylims)/2 - 1)*posfa(2,4) + 1) * posfa(1,4) ;

```

```
220 % Position it a bit lower (movegui slow)
221 posfa(1,2) = 100;
222
223 % Axis settings
224 set(gca, 'Xlim', xlims, 'Ylim', ylims, 'XColor', 'none', 'YColor', 'none', ...
225     'clim',[min(r(tf)),max(r(tf))])
226 set(gcf, 'pos', posfa(1,:),'Visible','on')
227 axis equal
228
229 % Set colormap
230 colormap(gca,ccolor);
231 if nargout == 1
232     h = s;
233 end
234
235 end
```

Part VI

Bibliography

Bibliography

- [1] M Aizerman, E Braverman, and L Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.
- [2] Francis R Bach, Gert R G Lanckriet, and Michael I Jordan. Multiple Kernel Learning, Conic Duality, and the SMO Algorithm. In *Proceedings of the Twenty-first International Conference on Machine Learning, ICML '04*, pages 6—, New York, NY, USA, 2004. ACM.
- [3] Christopher M Bishop. *Pattern Recognition and Machine learning*. Springer, 2006.
- [4] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.
- [5] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, Cambridge, 2000.
- [6] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.
- [7] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.
- [8] Volkmar Glauche. {MATLAB} batch system. <http://sourceforge.net/projects/matlabbatch/>.
- [9] Stefan Haufe, Frank Meinecke, Kai Görzen, Sven Dähne, John-Dylan Haynes, Benjamin Blankertz, and Felix Bießmann. On the interpretation of weight vectors of linear models in multivariate neuroimaging. *Neuroimage*, 87:96–110, 2014.
- [10] J V Haxby, M I Gobbini, M L Furey, A Ishai, J L Schouten, and P Pietrini. Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science (New York, N.Y.)*, 293(5539):2425–30, sep 2001.
- [11] J D Haynes and G Rees. {D}ecoding mental states from brain activity in humans. *Nat. Rev. Neurosci.*, 7:523–534, 2006.
- [12] Thomas Hofmann, Bernhard Scholkopf, and Alexander J Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220, 2008.
- [13] Max A Little, Gael Varoquaux, Sohrab Saeb, Luca Lonini, Arun Jayaraman, David C Mohr, and Konrad P Kording. Using and understanding cross-validation strategies. perspectives on saeb et al. *GigaScience*, 6(5):1–6, 2017.
- [14] A Marquand, M Howard, M Brammer, C Chu, S Coen, and J Mourao-Miranda. {Q}uantitative prediction of subjective pain intensity from whole-brain f{M}{R}{I} data using {G}aussian processes. *Neuroimage*, 49:2178–2189, 2010.
- [15] Members and collaborators of the Wellcome Trust Centre for Neuroimaging. Statistical {P}arametric {M}apping, {SPM8}. <http://www.fil.ion.ucl.ac.uk/spm>, 2008.
- [16] Gregory A Miller and Jean P Chapman. Misunderstanding Analysis of Covariance. *Journal of Abnormal Psychology*, 110(1):40–48, 2001.

- [17] Janaina Mourao-Miranda, Karl J Friston, and Michael Brammer. Dynamic discrimination analysis: a spatial-temporal SVM. *Neuroimage*, 36(1):88–99, 2007.
- [18] K A Norman, S M Polyn, G J Detre, and J V Haxby. {B}eyond mind-reading: multi-voxel pattern analysis of f{M}{R}{I} data. *Trends Cogn. Sci. (Regul. Ed.)*, 10:424–430, 2006.
- [19] F Pereira, T Mitchell, and M Botvinick. {M}achine learning classifiers and f{M}{R}{I}: a tutorial overview. *Neuroimage*, 45:199–209, 2009.
- [20] Alain Rakotomamonjy, Francis Bach, Stephane Canu, and Yves Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.
- [21] Anil Rao and Janaina Mourao-Miranda. Feature adjustment in kernel space when using cross-validation. *Research Note - UCL Department of Computer Science*, 2017.
- [22] Carl Edward. Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.
- [23] Jessica Schrouff, Julien Cremers, Gaetan Garraux, L Baldassarre, Janaina Mourao-Miranda, and Christophe Phillips. Localizing and comparing weight maps generated from linear kernel machine learning models. In *Pattern Recognition in Neuroimaging (PRNI), 2013 International Workshop on*, pages 124–127. IEEE, 2013.
- [24] Jessica Schrouff, J. M. Monteiro, L. Portugal, M. J. Rosa, C. Phillips, and J. Mourão-Miranda. Embedding Anatomical or Functional Knowledge in Whole-Brain Multiple Kernel Learning Models. *Neuroinformatics*, 16(1):117–143, jan 2018.
- [25] Jessica Schrouff and Janaina Mourao-Miranda. Interpreting weight maps in terms of cognitive or clinical neuroscience: nonsense? In *2018 International Workshop on Pattern Recognition in Neuroimaging (PRNI)*, pages 1–4. IEEE, 2018.
- [26] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, 2004.
- [27] Daniel B Suits. Use of dummy variables in regression equations. *Journal of the American Statistical Association*, 52(280):548–551, 1957.
- [28] Michael E. Tipping. Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, 1(Jun):211–244, 2001.
- [29] Sebastian Weichwald, Timm Meyer, Ozan Özdenizci, Bernhard Schölkopf, Tonio Ball, and Moritz Grosse-Wentrup. Causal interpretation rules for encoding and decoding models in neuroimaging. *Neuroimage*, 110:48–59, 2015.