

PRoNTo Manual

The *PRoNTo* Development Group

John Ashburner
Carlton Chu
Andre Marquand
Janaina Mourao-Miranda
Christophe Phillips
Jonas Richiardi
Jane Rondina
Maria J. Rosa
Jessica Schrouff

Machine Learning & Neuroimaging Laboratory
Centre for Computational Statistics and Machine Learning
Computer Science department, UCL
Malet Place, London WC1E 6BT, UK
May 20, 2012

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

Contents

1	Introduction	7
1.1	Background	7
1.2	Methods	8
1.3	Installing & launching the toolbox	9
1.3.1	Installation	9
1.3.2	Launching and batching	9
1.4	Main contributors	9
1.5	Acknowledgements	11
I	Graphical User Interface	13
2	Data & design	15
2.1	Introduction	15
2.2	Methods	16
2.2.1	Data and design input	16
2.2.2	Data and design output	16
2.2.3	Review	16
2.2.4	HRF correction	16
2.3	Graphical User interface	18
2.3.1	PRT directory	18
2.3.2	Groups	18
2.3.3	Subjects	19
2.3.4	Modalities	19
2.3.5	Masks	23
2.3.6	Review	23
2.3.7	Load, Save and Quit	23
2.4	matlabbatch interface	23
3	Prepare feature set	27
3.1	Introduction	27
3.2	Methods and resources	27
3.3	Graphical User interfaces	29
3.4	matlabbatch interface	31
4	Model Specification	33
4.1	Introduction	33
4.2	Beginning a model specification	33
4.3	Feature set	34
4.4	Model type / pattern recognition algorithm	34
4.4.1	Classification	35
4.4.2	Regression	35
4.5	Cross-validation	36
4.6	Batch interface	38

5	Model and Weights Estimation	39
5.1	Introduction	39
5.2	Methods	39
5.3	Graphical user interface	40
5.4	matlabbatch interface	40
6	Results display	43
6.1	Introduction	43
6.2	Launching results display	44
6.3	The main results display window	44
6.4	Analysing a machine's performance graphically	45
6.4.1	Predictions plot	45
6.4.2	Receiver Operating Characteristic (ROC) plot	45
6.4.3	Histogram plot	46
6.5	Statistical analysis of a machine's performance	47
6.5.1	Confusion matrix plot	47
6.5.2	The statistics table	47
6.5.3	Permutation testing	48
6.6	Visualising a weight map	48
II	Batching system	51
7	Data & Design	53
7.1	Directory	53
7.2	Groups	53
7.2.1	Group	53
7.3	Masks	55
7.3.1	Modality	55
7.4	HRF overlap	56
7.5	HRF delay	56
7.6	Review	56
8	Feature set / Kernel	57
8.1	Load PRT.mat	57
8.2	Name	57
8.3	Modalities	57
8.3.1	Modality	57
9	Specify model	59
9.1	Load PRT.mat	59
9.2	Model name	59
9.3	Use kernels	59
9.4	Feature sets	59
9.5	Model Type	59
9.5.1	Classification	59
9.5.2	Regression	60
9.6	Cross-validation type	61
9.6.1	Leave one subject out	61
9.6.2	Leave one subject per group out	61
9.6.3	Leave one block out	61
9.6.4	Leave one run/session out	61
9.6.5	Custom	62
9.7	Include all scans	62
9.8	Data operations	62
9.8.1	Mean centre features	62
9.8.2	Other Operations	62

10 Run model	63
10.1 Load PRT.mat	63
10.2 Model name	63
 III Data processing examples	 65
11 Data set 1	67
12 Data set 2	69
 IV Advanced topics	 71
13 PRT structure	73
14 List of PProNTTo functions	77
14.1 pronto.m	78
14.2 prt.m	78
14.3 prt_apply_operation.m	78
14.4 prt_check_design.m	79
14.5 prt_compute_weights.m	80
14.6 prt_cv_model.m	80
14.7 prt_cv_opt_param.m	80
14.8 prt_data_conditions.m	81
14.9 prt_data_modality.m	81
14.10 prt_data_review.m	82
14.11 prt_defaults.m	82
14.12 prt_fs.m	83
14.13 prt_func2html.m	83
14.14 prt_get_defaults.m	83
14.15 prt_get_filename.m	84
14.16 prt_init_fs.m	84
14.17 prt_init_model.m	85
14.18 prt_latex.m	86
14.19 prt_load.m	86
14.20 prt_load_blocks.m	86
14.21 prt_model.m	86
14.22 prt_normalise_kernel.m	87
14.23 prt_permutation.m	87
14.24 prt_preproc.m	88
14.25 prt_remove_confounds.m	88
14.26 prt_stats.m	88
14.27 prt_struct2latex.m	89
14.28 prt_text_input.m	89
14.29 prt_ui_compute_weights.m	89
14.30 prt_ui_cv_model.m	90
14.31 prt_ui_design.m	90
14.32 prt_ui_kernel_construction.m	91
14.33 prt_ui_main.m	91
14.34 prt_ui_model.m	92
14.35 prt_ui_prepare_data.m	92
14.36 prt_ui_prepare_datamod.m	92
14.37 prt_ui_results.m	93
14.38 prt_ui_results_help.m	93
14.39 prt_ui_reviewCV.m	94
14.40 prt_ui_reviewmodel.m	94

14.41	<code>prt_ui_select_class.m</code>	95
14.42	<code>prt_ui_select_reg.m</code>	95
14.43	<code>prt_ui_stats.m</code>	96
14.44	<code>prt_ui_sure.m</code>	96
14.45	<code>machines</code>	96
14.45.1	<code>machines\prt_KRR.m</code>	96
14.45.2	<code>machines\prt_machine.m</code>	96
14.45.3	<code>machines\prt_machine_RT_bin.m</code>	97
14.45.4	<code>machines\prt_machine_gpclap.m</code>	98
14.45.5	<code>machines\prt_machine_gpml.m</code>	98
14.45.6	<code>machines\prt_machine_gpr.m</code>	99
14.45.7	<code>machines\prt_machine_krr.m</code>	100
14.45.8	<code>machines\prt_machine_rvr.m</code>	101
14.45.9	<code>machines\prt_machine_svm_bin.m</code>	101
14.45.10	<code>machines\prt_rvr.m</code>	102
14.45.11	<code>machines\prt_weights.m</code>	103
14.45.12	<code>machines\prt_weights_bin_linkkernel.m</code>	103
14.45.13	<code>machines\prt_weights_svm_bin.m</code>	103
14.46	<code>utils</code>	104
14.46.1	<code>utils\prt_centre_kernel.m</code>	104
14.46.2	<code>utils\prt_checkAlphaNumUnder.m</code>	104
14.46.3	<code>utils\prt_normalise_kernel.m</code>	104

V Bibliography

105

Chapter 1

Introduction

Contents

1.1	Background	7
1.2	Methods	8
1.3	Installing & launching the toolbox	9
1.3.1	Installation	9
1.3.2	Launching and batching	9
1.4	Main contributors	9
1.5	Acknowledgements	11

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 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). Examples of other of imaging modalities that measure brain signals are Positron Emission Tomography (PET), Electroencephalography (EEG) recordings and Magnetoencephalography (MEG) recordings. Neuroimaging data are inherently multivariate in nature, 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 modeled. The effect of interest can be subtle and spatially distributed over the brain - a case of high-dimensional, multivariate data modeling 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 as multi-voxel pattern

analysis (MVPA) (see [11], [6] and [12]). 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 applications. For the latter, this approach can be used to predict the status of the patient scanned (healthy vs. diseased or disease A vs. B) and can provide the discriminating pattern leading to this classification.

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 modeling, 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.

1.2 Methods

PRoNTo (Pattern Recognition 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 [2]. In PRoNTo, brain scans 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 MATLABbased and includes five main modules: Data & Design, Prepare feature set, Specify and Run model, Compute weights and Display Results. In addition it has some review options to 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.

In terms of neuroimaging modalities, PRoNTo accepts NIFTI files and can be used to analyze structural and functional Magnetic Resonance Imaging and PET. It assumes that the neuroimaging data has been previously pre-processed using SPM or a similar software for neuroimaging analysis. In general, raw fMRI data should be previously corrected for movement artefact (re-aligned) 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 also applied as a pre-processing step for pattern recognition analysis, in this case the GLM coefficients (e.g. beta images in SPM) will correspond to the spatial patterns. Raw MRI data should be previously Raw PET data should be...

In PRoNTo different pattern recognition algorithms correspond to different machines. The machine library in PRoNTo v1 includes three classification models: Support Vector Machine ([3], [10]), Gaussian Process Classifier ([13], [8]), Random Forest [4] and two regression models: Kernel Ridge Regression [14] and Relevance Vector Regression [15]. New machines will be added to the library in future versions of the toolbox.

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

PRoNTo should facilitate the interaction between machine learning and 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, PRoNTTo requires 2 other softwares:

- a recent version of MATLAB. We used versions 7.5 (R2007b) to 7.14 (R2012a) to develop PRoNTTo, and it will not work with earlier versions¹.
- SPM8[9] installed on your computer².

PRoNTTo 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 PRoNTTo, the installation proceeds as follow:

1. Uncompress the zipped file in your favourite directory, for example `C:\PRoNTTo\`;
2. Launch MATLAB;
3. Go to the “File” menu → “Set path”;
4. Click on the “Add folder” button and select the PRoNTTo folder, i.e. `C:\PRoNTTo\` 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 OS’s such as: Windows XP (32 bits), Windows 7 (64 bits), Mac OS 10, 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 OS³.

1.3.2 Launching and batching

Once installed, there are three ways to call up PRoNTTo 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 PRoNTTo have been integrated into the `matlabbatch` batching system [5] (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).

1.4 Main contributors

PRoNTTo 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 reader at the Wellcome Trust Centre for Neuroimaging at the University College London Institute of Neurology. He is mainly interested in modeling 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 authored or co-authored 90 papers in international journals (h-index of 50) and written a number of book chapters;

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

²SPM8 can be downloaded from the following website: <http://www.fil.ion.ucl.ac.uk/spm/software/>. You should install it in a suitable directory, for example `C:\SPM8\`, then make sure that this directory is on the MATLABpath. No need to include the subdirectories!

³you can also contact us and we’ll try to come up with a solution for your system.

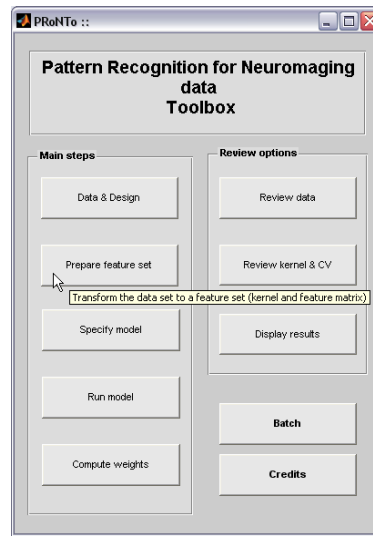


Figure 1.1: Main GUI interface: each button launches a specific processing step.

Dr. Carlton Chu is 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, New Zealand, in 2002 and the master of Biomedical Engineering from University of New South Wales, Australia, in 2004. Carlton obtained a PhD in Neuroimaging method from University College London in 2009, working in the statistical methods group at the prestigious Wellcome Trust Centre for Neuroimaging, creators of the famous “SPM” program. There he developed innovative new pattern recognition methods to automatically detect the early stages of neurodegenerative diseases such as Alzheimer’s and Huntingdon’s just 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 brain state decoding, neurodegenerative disease classification, and applying pattern recognition method to study brain networks;

Dr. Andre Marquand is a Post-Doctoral Research Fellow at the Centre for Neuroimaging Sciences, King’s College London (KCL) and an Honorary Post-Doctoral Research Fellow at the Centre for Computational Statistics and Machine Learning at University College London. His research focuses on the application of probabilistic machine learning techniques to neuroimaging data, particularly for clinical applications. His recent work includes the application of multi-class and multi-modality pattern classification methods to neuroimaging and in particular to detecting the effects of psychotropic medication on patterns of brain activity;

Dr. Janaina Mourao-Miranda is a Wellcome Trust Senior Research Fellow at Centre for Computational Statistics and Machine Learning (CSML), UCL and at the Centre for Neuroimaging Sciences (CNS), KCL. Her research focuses on developing and applying pattern recognition methods to analyze neuroimaging data, in particular brain activation and structural patterns that distinguish between controls and patients. Recent work includes the development and application of spatio-temporal SVM, one-class SVM to detect patients as outliers and in-depth studies of kernel methods for brain decoding;

Dr. Christophe Phillips is FRS-FNRS Research Associate at the Cyclotron Research Centre and adjunct Assistant 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;

Dr. Jonas Richiardi is a post-doctoral research fellow, jointly affiliated to the EPFL engineering school (Medical Image Processing Laboratory) and the Geneva university hospitals (Department of Radiology and Medical Informatics). His research interests include brain connectivity and resting-state networks analysis, interpretability of brain decoding results, functional biomarkers, learning with graphs, machine learning for neuroimaging, and the combination of imaging modalities with other biological information sources. He was co-chair of the 2010 Brain Decoding Workshop at the Int. Conf. on Pattern Recognition and a programme chair of the Pattern Recognition in NeuroImaging workshop 2011 and 2012;

Dr. Jane Rondina is a Wellcome Trust Post Doctoral Research Associate at Centre for Neuroimaging Sciences (CNS), KCL and researcher as an honorary member at Centre for Computational Statistics and Machine Learning (CSML), UCL. Her current work includes the development of a feature selection method for classification in neuroimaging and analysis of features stability. Her research interests also include the development of pattern recognition methods using data from different modalities / measures in neuroimaging;

Dr. Maria J. Rosa is a Wellcome Trust post doctoral research associate at Centre for Computational Statistics and Machine Learning (CSML), UCL. Her areas of interest include Bayesian model selection methods for fMRI and Dynamic Causal Modelling, EEG-fMRI fusion, and more recently, machine learning for neuroimaging.

Ms Jessica Schrouff has a Master in Biomedical Engineering and is currently pursuing a PhD in neuroimaging at the Cyclotron Research Centre, University of Liège, Belgium, under the supervision of Dr C. Phillips. Her project focuses on the tracking of mnesic traces of learned images in trained healthy subjects with fMRI and EEG data, as well as the classification of patients from PET images.

1.5 Acknowledgements

PRoNTo v1.1 (2012) 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

- the Department of Computer Science, University College London (<http://www.cs.ucl.ac.uk>);
- the Wellcome Trust (<http://www.wellcome.ac.uk/>);
- PASCAL2 (<http://www.pascal-network.org>) and its HARVEST programme;
- the Fonds de la Recherche Scientifique-FNRS (<http://www.fnrs.be>), Belgium;
- Fundação para a Ciência e Tecnologia - FCT (<http://www.fct.pt>), 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.

Part I

Graphical User Interface

Chapter 2

Data & design

Contents

2.1	Introduction	15
2.2	Methods	16
2.2.1	Data and design input	16
2.2.2	Data and design output	16
2.2.3	Review	16
2.2.4	HRF correction	16
2.3	Graphical User interface	18
2.3.1	PRT directory	18
2.3.2	Groups	18
2.3.3	Subjects	19
2.3.4	Modalities	19
2.3.5	Masks	23
2.3.6	Review	23
2.3.7	Load, Save and Quit	23
2.4	matlabbatch interface	23

2.1 Introduction

The first step in a statistical analysis of neuroimaging data, whether it's 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. PRoNTTo is no exception. After preprocessing the data (if required), the analysis in PRoNTTo starts with the 'Data and Design' module. It is important to note that PRoNTTo does not perform any spatial or temporal pre-processing, and if not performed with another software, pattern recognition might be affected by noise in the data.

In the 'Data and design' module the user can enter the image/scan files, experimental conditions (TR, durations and onsets of events), as well as other parameters, covariates and regression values. PRoNTTo supports multi-modality datasets and therefore it allows the user to enter more than one data modality, such fMRI, MRI, PET and ASL, per analysis. 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.

Below is a summary of what the 'Data and Design' module does. The Methods section discusses how the module is organised and what its main output is. It also mentions a few issues that need to be taken into consideration when entering the information and how they affect subsequent steps. This chapter then presents the graphical user interface (GUI) that is used to enter the data and design information and how it is used. Finally, the chapter finishes by mentioning the corresponding 'Data and Design' `matlabbatch` module, and particular issues that do not apply to the GUI.

2.2 Methods

2.2.1 Data and design input

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 the Introduction chapter.

The information that needs to be entered is almost exactly the same for both the GUI and batch (the small differences are explained below 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.

As mentioned, PRoNTo supports multi-modality analyses. Therefore the data and design module is prepared to receive as input the following types of data: fMRI, sMRI, PET and beta images (created from a previous GLM analysis). Other types of data can also be entered at the user's risk, as long as they comprise nifti files.

Regardless of which interface the user chooses to enter the data and design (GUI or batch), the organisation is very similar and starts (after choosing the directory to save `PRT.mat`) with the definition of Groups. In neuroimaging datasets, it is common to have a few subjects with a lot of images/scans 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 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 scans' option, respectively (as is shown below). If one chooses to enter the data by 'scans', PRoNTo allows the user to enter, for each modality, all subjects (one image/scan per subject) at once, which is a lot quicker than entering each subject at a time. It is important to note that when using *Regression* models this is the only way of inputting the data. As explained below, only the 'select by scans' option allows the users to enter regression values (one value per subject/image). This option however is not appropriate for modalities which have an experimental design and more than one image per subject, such as fMRI. For these datasets the user should choose the 'subjects' option. For each subject one can specify the modalities, experimental conditions and enter more than one image/scan. Both options are valid and produce exactly the same output structure (if used with the same dataset).

2.2.2 Data and design output

The output of the 'Data and Design' module is the PRT structure (as discussed in the Introduction). This structure contains subfields with all the information that is needed from the data for the subsequent analysis steps and it 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.

2.2.3 Review

The 'Data and 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. For example, the review window shows the number of subjects in each group, and for modalities with experimental design, it can be used to show and alter the number of used and unused scans (see below).

2.2.4 HRF correction

For datasets such as fMRI, 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 dispersed version of the underlying neuronal response to an experimental event (Figure 2.1). This means that, depending on the TR, the effect of the HRF can be felt over multiple scans, and therefore the acquired scans 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 PRoNTTo is to discard all overlapping scans. This is done as follows: PRoNTTo allows the user to control a delay (time it takes for the hemodynamic response to peak after the stimulus) and overlap (width of the response) parameter that determine the shape of the HRF. As can be seen in Figure 2.1, the delay means that the scans corresponding to a given condition are actually shifted in time, and the overlap means that the number of independent scans, for which the signal corresponds only to a given condition, is smaller than the total number of acquired scans for each condition. Given the delay, PRoNTTo finds which scans correspond to each condition and discards the last scans in the time-series for which the response has not yet peaked. It then uses the overlap to determine which consecutive scans 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 2.1, bottom right). The discarded scans 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, these values should be changed (e.g. to 6 seconds).

Importantly, if one wants to avoid discarding scans 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. 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 scans can be very high.

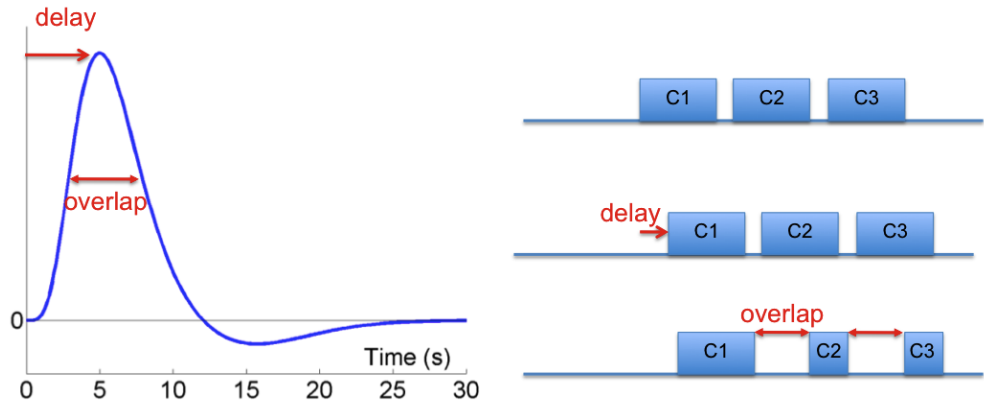


Figure 2.1: HRF correction. On the left is the standard HRF response. On the right is the effect of the delay and overlap on the number of independent scans (C1, C2 and C3 correspond to three different experimental conditions and the blue boxes correspond to various scans acquired during each condition). 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 scans/events than the ones originally acquired. In PRoNTTo, this issue is accounted for by discarding overlapping scans.

The steps to specify the information relative to the data and design using both the GUI and the `matlabbatch` system are described in the following sections.

2.3 Graphical User interface

The graphical user interface to specify the data and design is presented in Figure 2.2. This GUI can be launched by typing ‘prt’ in the Matlab window and then clicking the first button on the left, in the main steps panel.

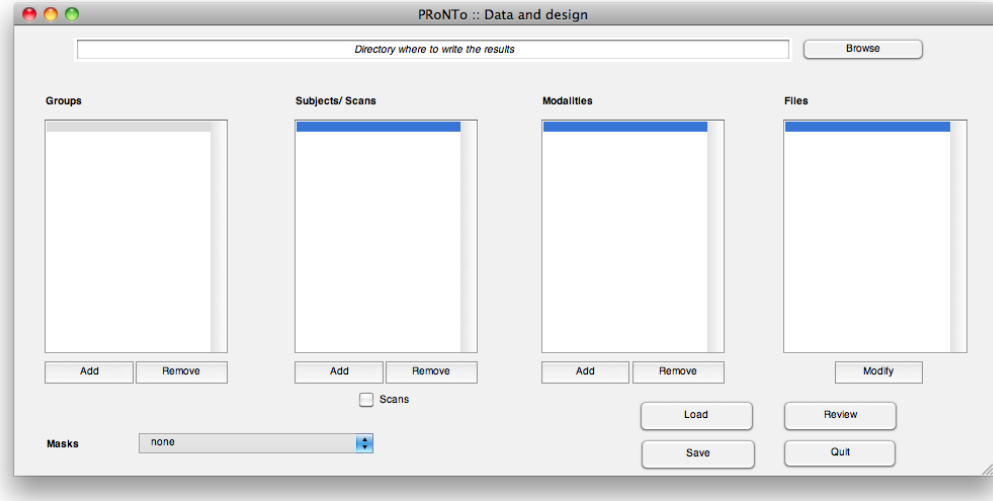


Figure 2.2: Data and design graphical user interface. This interface allows the user to enter all the information relative to the data, including the experimental design and masks. After introducing all the fields, PRoNTTo creates the PRT structure, which is saved in the specified directory, as ‘PRT.mat’ file.

2.3.1 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 and design interface (Figure 2.2). It is recommended to have different directories for different datasets (not modalities) because PRoNTTo overwrites an existing PRT in the selected directory. The later modules in PRoNTTo 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’.

2.3.2 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. PRoNTTo does not restore the deleted information and it can only be re-entered again by clicking ‘Add’.

The following panel after ‘Groups’ is ‘Subjects/Scans’. Here, as mentioned above, there are two ways of entering the data: by ‘subjects’ or by ‘scans’. The former is chosen by clicking ‘Add’ under the ‘Subjects/Scans’ panel and filling in the fields for each added subject at a time. The latter is done by clicking the tick box ‘Scans’ under the ‘Subjects/Scans’ panel. The subjects panel is then de-activated and the user can enter the modalities and files straight away. The fields to be filled under these two options are described below.

2.3.3 Subjects

Select by scans The ‘select by scans’ option allows the users to skip the subject step. To identify that this option has been selected, PRoNTo writes ‘scans’ in the subjects panel (Figure 2.3). The user can then add modalities and for each modality a new window will appear (bottom of Figure 2.3). It is important to remember that when the ‘scans’ box is clicked all the information in the subjects panel is automatically deleted! Unselecting the ‘scans’ box also deletes all the information!

Select by subjects The ‘Subjects/Scans’ 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.

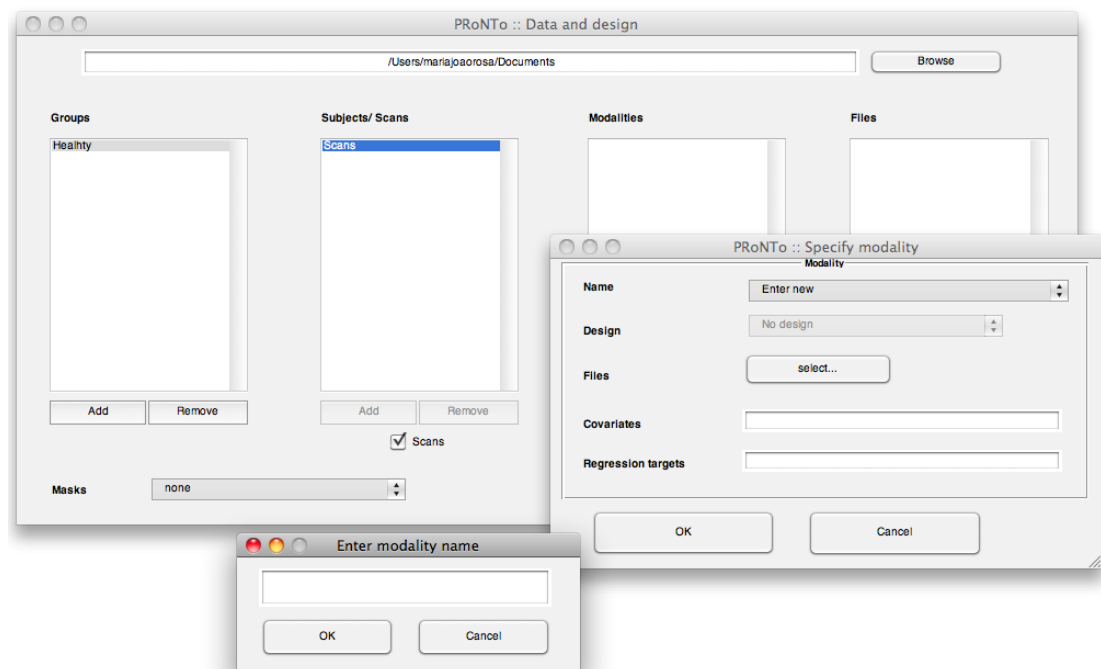


Figure 2.3: Data and design graphical user interface. If one chooses to specify everything using the ‘Scans’ option (tick box below the ‘Subjects/Scans’ panel), one can introduce the data for all subjects at once for each modality, but one cannot specify any design. This is the optimised approach when one has a lot of subjects with only one image/scan per subject, such can be the case of MRI and PET datasets.

2.3.4 Modalities

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.

The steps to enter the modality information are slightly different if one ticks the ‘scans’ box or not.

Select by scans Here the data is assumed to have been acquired without an experimental design, and therefore the ‘No design’ option is automatically selected and cannot be changed

(bottom window in Figure 2.3). However, in select by scans, the user can also introduce ‘Covariates’, i.e. a variable that covaries with the data (subjects) but of no interest to the subsequent analyses. This option is not yet functional in version 1.0 of PRoNTTo! The last empty field can be used to enter ‘Regression targets’ (Figure 2.3). This option allows the users to introduce a real number per subject to be used later for regression if that is the case. As mentioned above, this is the only way of entering the data and regression values when doing *Regression* models!

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 last option is simply ‘No design’, which means that for this modality there are no experimental conditions. The first 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 fields using the SPM software. In this case, the user does not need to specify anything else, only the files (scans/images) for this subject/modality. The design information is extracted directly from the SPM structure and saved in PRT.mat. Finally, the ‘Specify design’ option 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.

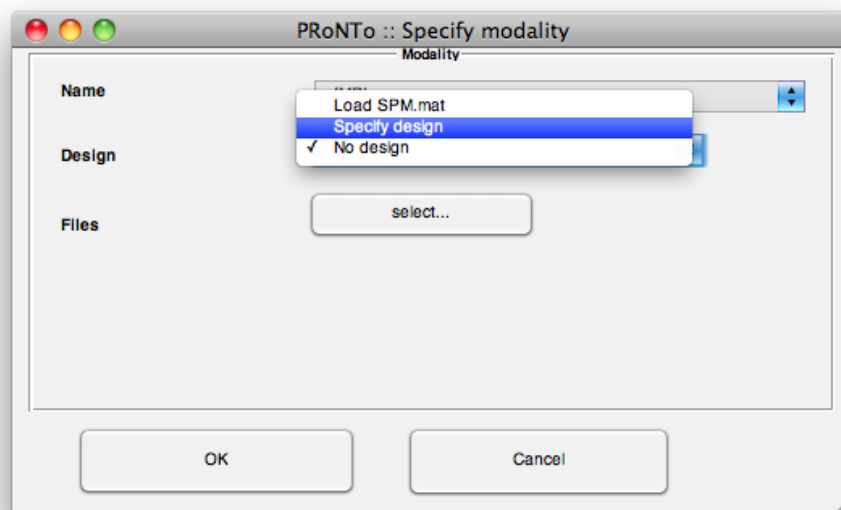


Figure 2.4: Data and design graphical user interface. The design menu in the modality window (when one uses the select by subject option) allows one to load a previously specified design from an SPM.mat file, create a new design or simply select no design, which usually applies to modalities where there is no experimental task, such as MRI or PET.

Design To create a new design one selects the option ‘Specify design’ as explained in the previous paragraph (Figure 2.4). This will then open another window (after choosing how many conditions you have) (Figure 2.5). In this window one can then write the names, onsets, and durations of each condition. 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 PRoNTTo 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 is always read in seconds. Finally, there is also an option (which is again not yet functional in version 1.0 of PRoNTTo!) to introduce covariates, which, in this case, correspond to any variable that varies along with the experimental events but of no interest for further analyses.

One issue to have in mind when specifying the design is the following: if there are more scans than experimental events, these extra scans 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.`

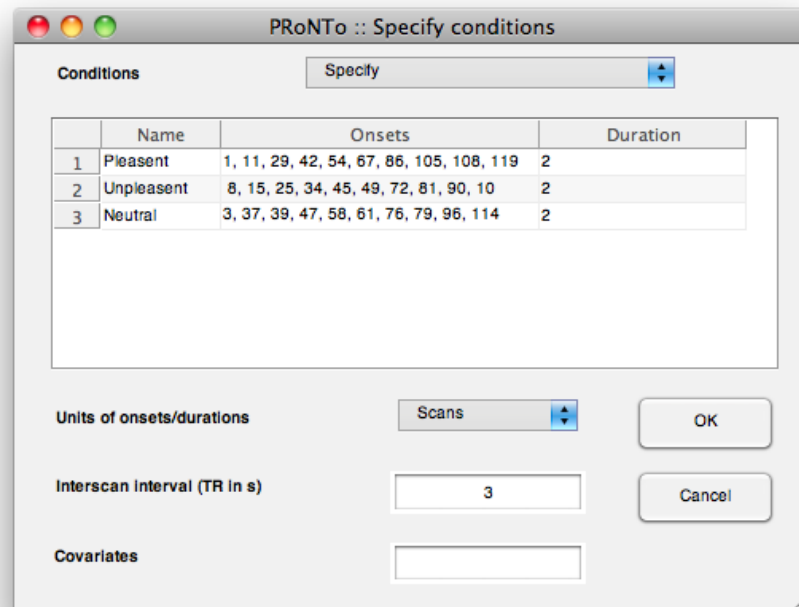


Figure 2.5: Data and design graphical user interface. The ‘specify conditions’ window is available from the modality interface when the user chooses to enter the data by subjects and clicks ‘specify design’. This window is used to enter the conditions (names, onsets and durations) as well as the units of design, TR and covariates.

Modify design The user can later modify a design by loading a PRT.mat in the Data and 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.

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 on the name of the concerned modality proposes to re-open the modality definition window. To review or modify the onsets/durations/blocks, the user can access their definition via the specify design option. Similar right-clicks allow renaming groups or subjects.

To modify the HRF parameters (delay or overlap), there is no need to load the PRT in Data and 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 anew since they do not correspond to the modified design. Changing the desired parameter and hitting return updates the PRT directly. Please remember to keep an eye on the Matlab window, since important information are displayed on the workspace!

Files Finally, independent of the way the user entered the information (by subjects or scans) the ‘Files’ option allows one to choose which image files to use (Figure 2.6). 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.

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

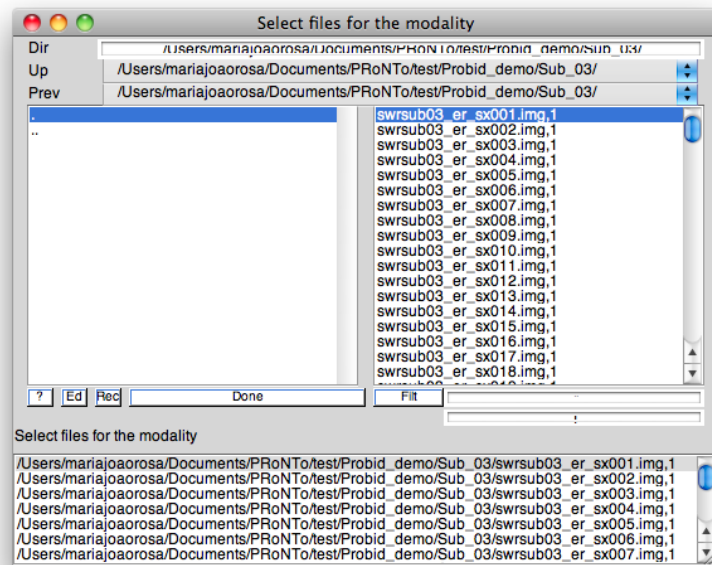


Figure 2.6: This window is called when one clicks ‘Files’ and is used to select the scans/images for each subject/modality.

clicked and no files are then selected all the previous files are deleted! Figure 2.7 shows how the data and 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 explained above.

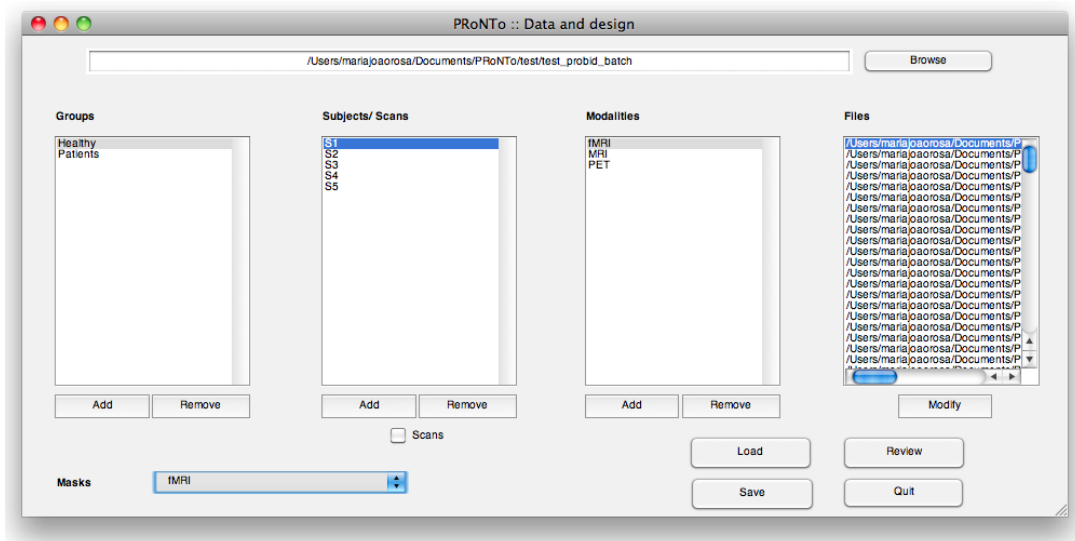


Figure 2.7: Data and design graphical user interface. After filling in all the fields using the select by subject option (the select by scans case is very similar) the data and design interface should look like this example figure.

2.3.5 Masks

This popdown menu on the bottom of the main data and design window is where the user enters a binary image mask for each modality. This mask can be previously created by the user (it has to be in MNI space) or simply chosen from a list of default masks available in the masks directory of PRoNTo. Every modality 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. 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.

2.3.6 Review

The ‘Review’ button allows one to review the data and 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. This is particularly important to check if the design is too unbalanced 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 scans plotted on the left (number of selected scans and number of discarded scans for each condition). The higher the value of the HRF peak and overlap, the higher the number of discarded scans. One can also read on the main Matlab window information regarding which group/subjects have had some scans discarded. The information below the HRF parameters corresponds to the interval between successive scans before and after the HRF delay/overlap correction. These values also change according to the changes entered in the boxes above. Please note, as mentioned in the section ‘Modify design’, that information regarding the PRT being updated after changing the HRF parameters is written on the main Matlab window. Again 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 scans 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.`

2.3.7 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.

2.4 matlabbatch interface

The ‘Data and 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 and design module.

The options presented in the ‘Data and Design’ GUI, mentioned above, are all available in the `matlabbatch` interface (Figure 2.9). However, there are a few things in the batch that differ

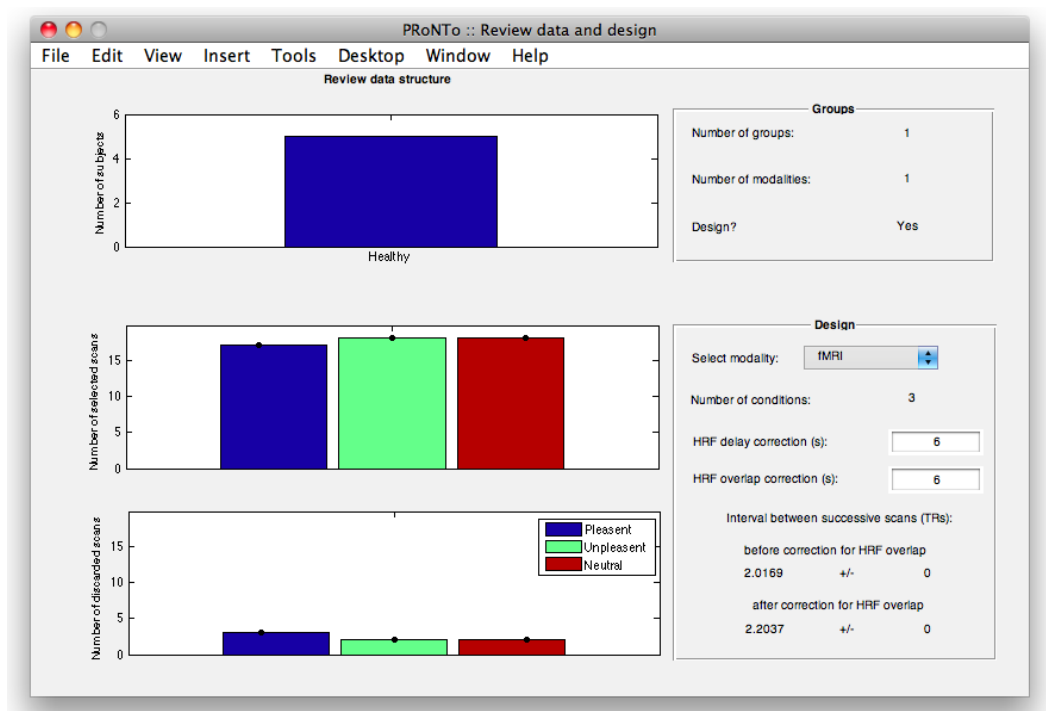


Figure 2.8: Data and design graphical user interface - ‘Review’ window. This window allows the user to check the data and design, including the number of subjects per group. It also allows the user to change the HRF delay and overlap parameters that control the number of discarded scans (appropriate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

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 scans) 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 issue 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.

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).

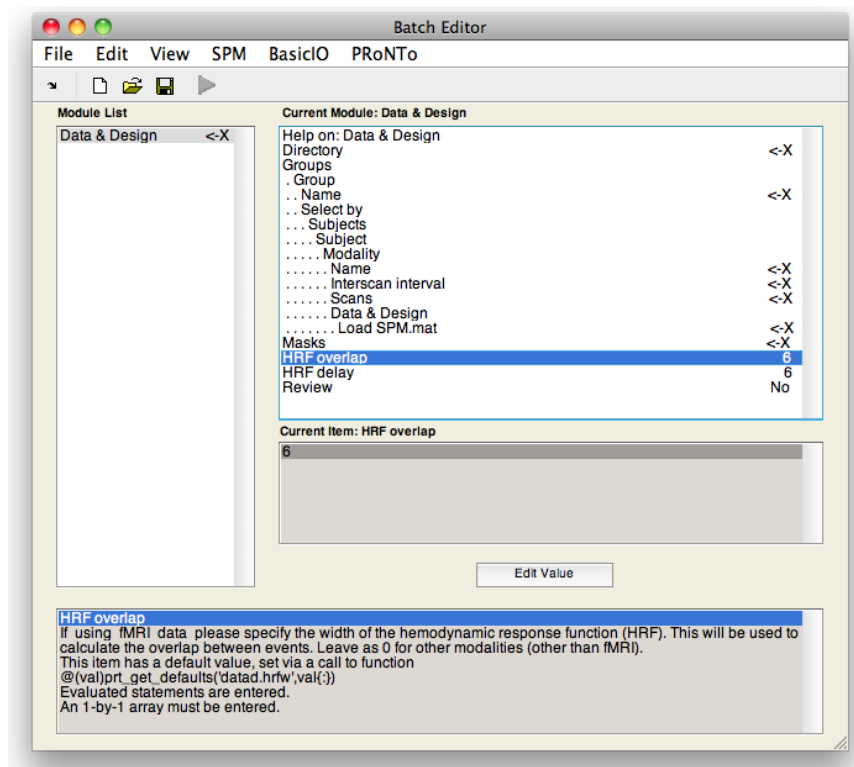


Figure 2.9: Data and design module in `matlabbatch`. The `matlabbatch` contains two extra options relative to the Data and Design interface. These options allow one to specify the delay and overlap of the HRF response (in the GUI it can only be changed in the ‘Review’ window), and which are then used to determine the number of discard scans.

Chapter 3

Prepare feature set

Contents

3.1	Introduction	27
3.2	Methods and resources	27
3.3	Graphical User interfaces	29
3.4	matlabbatch interface	31

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 values of selected features for each sample. This matrix can either be input directly into the machine or be used to compute a similarity matrix, or kernel, of the size $N_{samples} \times N_{samples}$, which is then input into the classification/regression algorithm [see “kernel trick” [7, 1]].

The “Prepare Feature Set” step computes both the feature and similarity matrices from one or more modalities, as defined in the previously built dataset (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 perform the classification/regression on specific voxels only (e.g. Regions of Interest).

Please note that only modalities containing the same number of features (i.e. selected voxels) can be included in the same FS. This will be typically the case when more than one run was acquired for each subject, the different runs being entered as different ‘modalities’ in the dataset building (e.g. modality 1 is ‘fMRI_run1’, modality 2 is ‘fMRI_run2’,...). In all other cases, a feature set has to be computed for each modality.

3.2 Methods and resources

After the selection of the dataset and of which modality to include in the FS, the toolbox accesses each image, i.e. it gets the value of the voxels which are comprised in the first level mask selected for that modality (mask specified at the data and design step, see chapter “Data and Design”). This access is performed by ‘blocks’ of features, not to overload the RAM memory. In the case of time-series, 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, ref, parameter: frequency cutoff in seconds). An example of a linear detrending (polynomial detrending of order 1) was represented in Fig. reffig:lindetrend.

For each modality, the (detrended) features are then written in a file array (SPM, ref, 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 can’t be fulfilled, we recommend

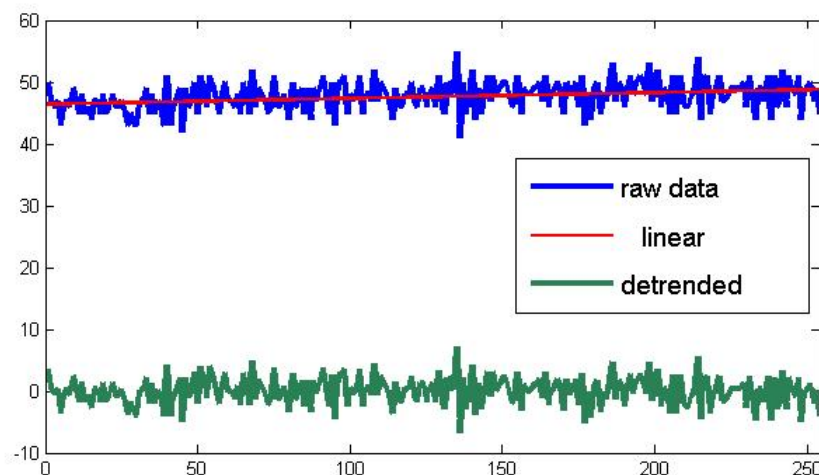


Figure 3.1: Example of detrending: the original signal over time of one feature (in blue) was approximated by a polynomial of order 1 (red line), which was then subtracted from the original signal to give the detrended signal (in green).

the use of external drives for the whole analysis. Regarding the computational expenses, we tried to minimize their effect by computing the features only once per modality: when preparing other feature sets using the same modality and detrending parameters, the same file array will be accessed.

Be careful that using the same modality but different detrending 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.

From the feature set(s), the kernel (similarity matrix) can then be computed. Different options can be specified:

- All scans/ All conds: In all scans the similarity will be computed between all scans within the time series of all subjects and in the all conds the similarity is computed only between the scans corresponding to the specified conditions of interest (see "Data and 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. We would therefore recommend this last option for cases similar to multi-subject fMRI studies with designed stimulation.
- Additional 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 voxels used to compute the kernel (and only the kernel) would be the ones contained in both the first and second levels masks. Therefore, using one first-level mask and two second-level masks would create two kernels but one file array.
- Normalisation (has to be called scaling): allows the specification of constant values to scale each scan. The user has to enter a .mat containing a variable called 'scaling' and of the same size as the number of scans in that modality. In case of quantitative modalities such as PET, this step is required since it insures the convergence of the machine learning algorithm.

These three 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.

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 link to the kernel are stored in a `fs` field (standing for “Feature Set”), which size corresponds to the number of feature sets defined by the user.

3.3 Graphical User interfaces

After clicking on the “Prepare Feature Set” button in the main interface (see Fig. 3.2), a second window will appear, allowing the user to select a dataset (Fig. 3.3.A), to name the FS (Fig. 3.3.B) and to define the number of modalities which should be included in the FS (Fig. 3.3.C), see section 3.1 for a comment on this last point).

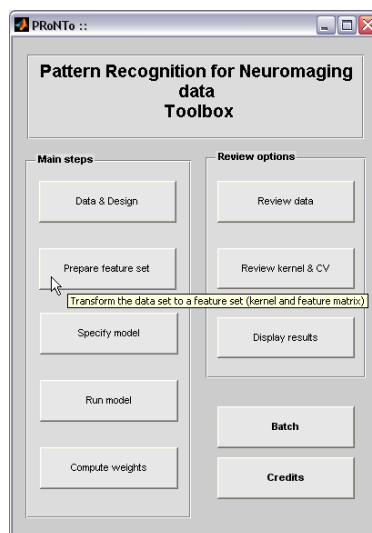


Figure 3.2: Main interface: button to launch the ‘Prepare Feature Set’ step.

To define the number of modalities to include, the user should click in the appropriate box (Fig. 3.3.C), type the number and then ‘return’ (arrow for return?). This will launch a third window, allowing the specification of the different options and parameters for each modality (Fig. 3.4). When the dataset contains only one modality, this window is launched automatically.

In this third window, the user has to choose which modality to include based on its name (Fig. 3.4.A) and which scans to use to build the kernel (all or only those linked to the design, Fig. 3.4.B). All other options are facultative. They include:

- the specification of a second-level mask (Fig. 3.4.C): type the full name (with path) of the mask or browse to select the mask image. When left empty or untouched, voxels are selected from the first-level mask specified in the data and design step.
- the detrending parameters (Fig. 3.4.D): 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.
- the scaling (Fig. 3.4.E): ‘no scaling’ is the default option. However, when dealing with quantitative modalities such as PET, the user should provide one value per scan, stored in a vector in a .mat file under the variable name ‘scaling’.

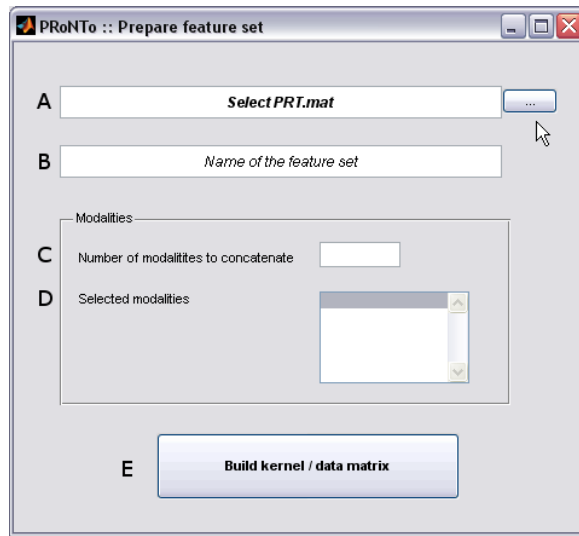


Figure 3.3: Interface of the 'Prepare Feature Set' step: A. Dataset selection: type the full name (with path) or browse to select the dataset to prepare. B. Type the FS name, which will be used to save the kernel as a .mat on the hard drive. C. Number of modalities to select. D. List containing the names of the modalities included in the FS (no user interaction possible). E. Click to build the feature set and kernel.

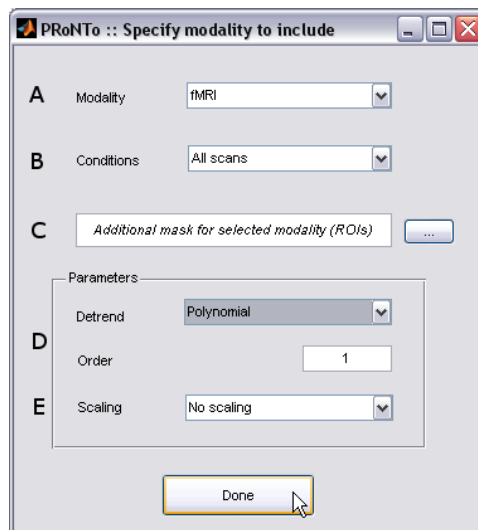


Figure 3.4: Specification of options and parameters for each modality: A. Select the modality name from a pull-down menu. B. All scans/All conds. C. Second-level mask selection. D. Detrend and its parameter. E. Scaling of the scans or not.

When working with Graphical User Interfaces (GUIs), some messages might appear in MATLABworkspace. 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 MATLABprompt when using GUIs.

3.4 matlabbatch interface

The `matlabbatch` system 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. The hierarchy for the case of a feature set containing one fMRI modality is displayed in (Fig. 3.5).

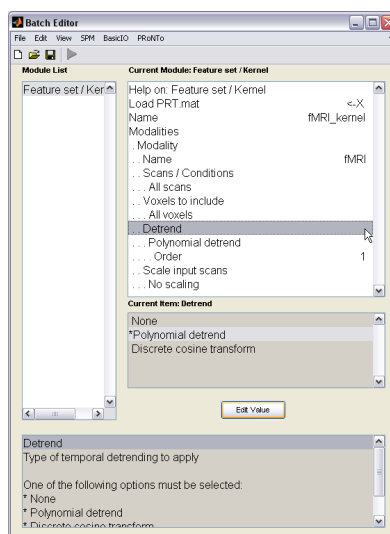


Figure 3.5: Matlabbatch GUI.

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.

Chapter 4

Model Specification

Contents

4.1	Introduction	33
4.2	Beginning a model specification	33
4.3	Feature set	34
4.4	Model type / pattern recognition algorithm	34
4.4.1	Classification	35
4.4.2	Regression	35
4.5	Cross-validation	36
4.6	Batch interface	38

4.1 Introduction

The specification of a model is the core step of the pattern recognition pipeline and entails setting up the combination of the different components making up the analysis. For example, model specification is where you select which data features to use as input (i.e. a feature set), the type of prediction to perform (e.g. classification or regression), which machine learning algorithm to employ (e.g. support vector machines, Gaussian processes, ...), which cross-validation strategy to employ (e.g. leave one subject out, leave one run out, ...) and which operations or manipulations to apply to the data before the algorithm is trained. The framework provided by PRoNTo is highly flexible and supports most types of 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 below).

4.2 Beginning a model specification

To begin a model specification with the PRoNTo user interface, select “Specify model” from the main PRoNTo window. This will launch the model specification window (Figure 4.1)

Next, select the `PRT.mat` containing your experimental parameters. 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.

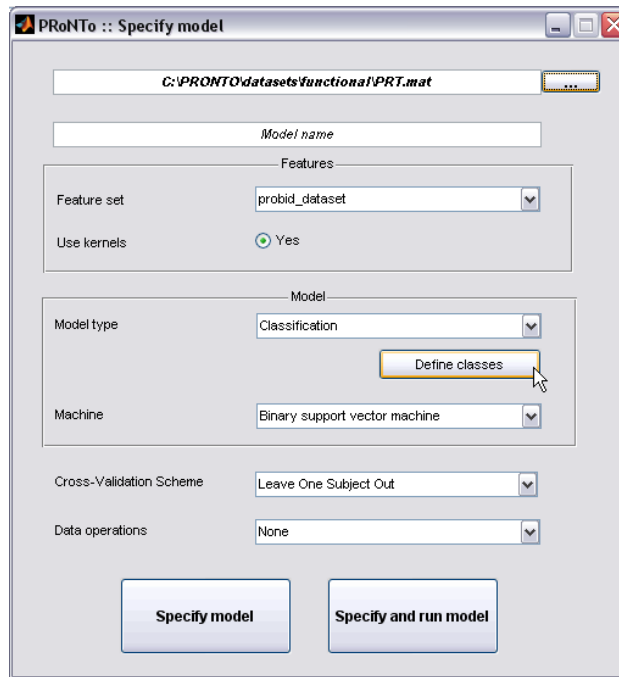


Figure 4.1: Model specification graphical user interface

4.3 Feature set

The drop-down list entitled ‘Feature set’ will be populated once a `PRT.mat` containing one or more feature sets is selected. Select the appropriate feature set from the drop-down list. Note that a single feature set may contain more than one data modality (see chapter 3). This might be useful if more than one run is available for each subject, in which case each run could be coded as an independent modality and a single-subject classifier might be specified using leave-one-run-out cross-validation.

In the current release of PRoNTTo, only kernel classifiers are supported via the user interface. The capability to support non-kernel classifiers will be added in a future release. Thus, the “Use kernel” radio button should always be set to true.

4.4 Model type / pattern recognition algorithm

In this part of the model specification input form, select the pattern recognition algorithm to employ (referred to in PRoNTTo as a “machine”). In the current release, three classification algorithms are supported (binary support vector machines, Gaussian processes (binary and multiclass) and random forests) and three multivariate regression methods (Gaussian process regression, kernel ridge regression¹ and relevance vector regression).

The PRoNTTo user interface provides a mechanism for flexible definition of which components of the 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 having 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 of 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

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

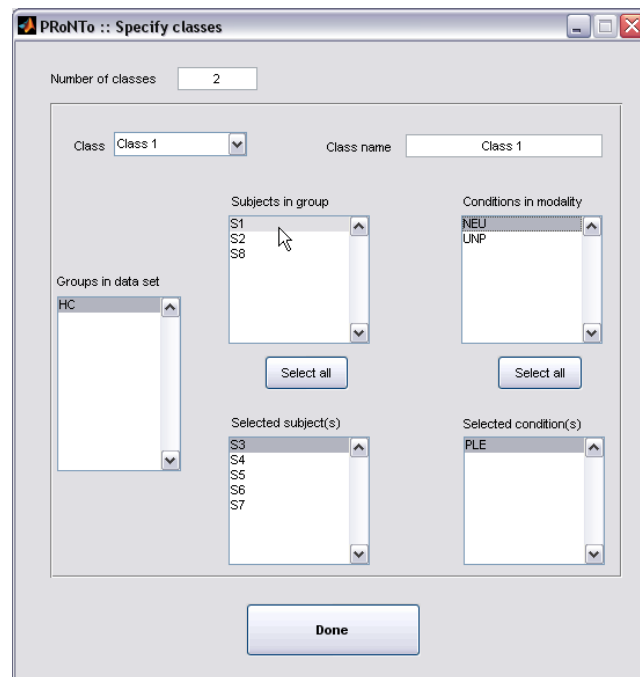


Figure 4.2: Subject / condition selection panel for classification models

defined via the user interface by clicking the “Define classes” button (for classification) or “Select subjects/scans” (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 (since by definition conditions are not identically distributed). 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.

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 data for each voxel are independently fit with the same model. Statistics test are used to make inferences on the presence of an effect at each voxel (e.g. *t-test*). Multivariate regression, on the other hand, takes into account several input variables (voxels) simultaneously, thus modeling the property of interest considering existing relations among the voxels.

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

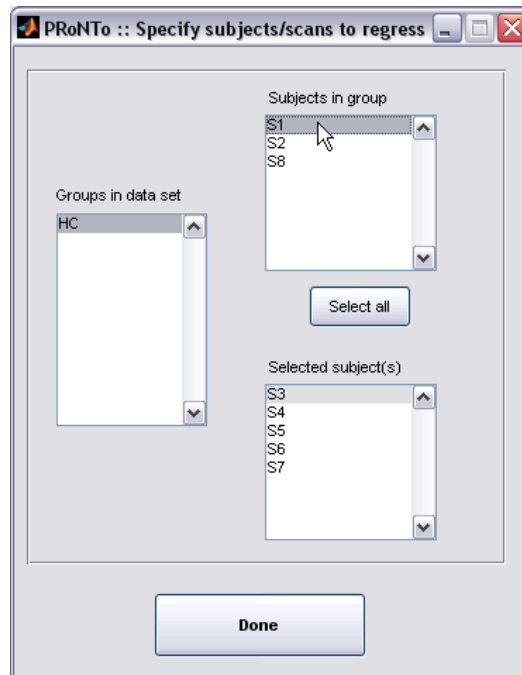


Figure 4.3: Subject / condition selection panel for regression models

to predict a continuous value instead of predicting a class to which the example belongs. These values usually refer to demographic, clinical or behavioral data (as age, blood pressure or scores resulting from a test, for example). For validation, predicted values can be correlated to the actual ones. Also, MSE (Mean Square Error) can be calculated, referring to the mean value of the squared deviations of the predictions from the true values over the cross-validation.

The specification of which subjects and scans to include in regression models is similar to that for classification, see Figure 4.3 and for the purposes of model specification in PRoNTTo, regression can be thought of as a classification problem with a single class. In the current release, regression is only supported if there is a single scan per subject (e.g. structural images or parameter estimate images from a GLM analysis). In a future release it will be possible to perform regression where an independent regression target is supplied for each trial, block or condition. To perform a regression, the regression targets are specified during the design stage. It is important to emphasize that in the current implementation, regression is only supported using the "select by scans" option (see chapter 2).

4.5 Cross-validation

In the final part of the specify model input form, select the type of cross-validation to employ. Cross-validation is a crucial part of the design and is used to assess the generalizability of the model and to ensure the model has not overfitted 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 "test 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 most common forms 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.

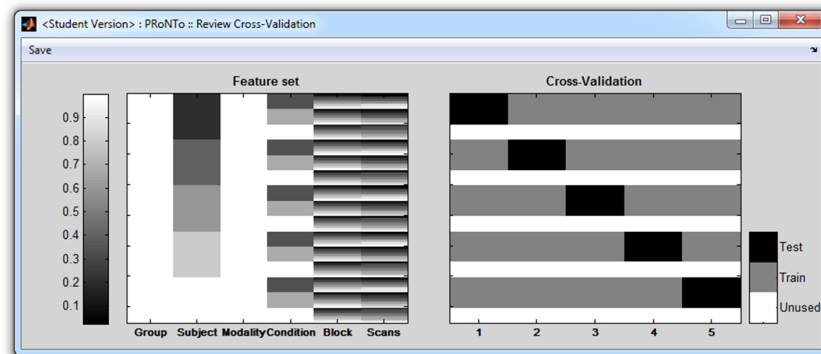


Figure 4.4: Review cross-validation matrix

The current release of PRoNTto supports each of these, and also supports leave-one-subject-per-group-out (LOGO), which is appropriate if the subjects in each group are paired or for repeated measures experimental designs. The functionality to provide arbitrary cross-validation resampling approaches will be added in a future release. 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 PRoNTto window (see 4.4 for an example). 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 test 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 is nonsensical to select a leave-one-subject-out cross-validation approach for single subject designs. It is also important to ensure that the training and test 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 autocorrelated. 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 test 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 important issue, and the user should still take care to ensure that cross-validation folds are sufficiently far apart in time (especially for LOBO cross-validation).

During this part of the model specification, it is also possible to select one or more operations to apply to the data. Each of these operations is defined below:

- **Sample averaging (within blocks):** constructs samples by computing the average of all volumes within each block or event for each subject and condition.
- **Sample averaging (within subjects):** constructs samples by computing the average of all scans within all blocks for each subject and condition
- **Mean centre features using training data:** subtract the voxel-wise mean from each data vector
- **Divide data vectors by their norm:** scales each data vector to lie on the unit hypersphere by dividing it by its Euclidean norm
- **Perform a GLM:** currently not supported

A crucial point to note is that all operations are embedded within the cross-validation structure such that they are applied independently to training and test sets. 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 test 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: (i) the order of operations is potentially important. 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. (ii) 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 over scans during cross-validation.

4.6 Batch interface

The batch module provides 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 P_{RoN}To 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 P_{RoN}To 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.

Chapter 5

Model and Weights Estimation

Contents

5.1	Introduction	39
5.2	Methods	39
5.3	Graphical user interface	40
5.4	matlabbatch interface	40

5.1 Introduction

The previous module allowed the user to specify one or more models. These include the machine to be used, the cross-validation scheme and the classification/regression problem. After model specification one needs to learn the models, i.e. use the training and test datasets to estimate the parameters of the classifiers/regressors, and test how good the model is at making predictions. This is done in the module of PRoNTo called ‘Run model(s)’.

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

5.2 Methods

After running the model specification module described above, the PRT structure (stored in `PRT.mat`) contains all the required inputs to run the model estimation. This information can be found in ‘`PRT.model(m).input`’, where m is the index of the model. The inputs, which include the cross-validation matrix, the target values or labels, and the machine (e.g. binary SVM, Random Forests, 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.

The output of a linear model includes the coefficients from the primal/dual optimisation problem. These coefficients are then multiplied by the training examples to obtain the model weights (optimal hyperplane). The vector of model weights has the same dimensions of the original voxel space, and can therefore be converted to a 3D image. This computation is done for each fold. The resulting 3D images for all folds are then assembled into a single 4D NIFTI file with dimensions $[3D \times (\text{number of folds} + 1)]$, 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`.

5.3 Graphical user interface

Using the GUI, PRoNTTo allows the user to estimate one or more models at one go. Clicking the ‘Run model’ button on PRoNTTo’s main window will launch a small window shown in Figure 5.1. The first thing that needs to be done using this window is to specify which PRT we would like to work with. PRoNTTo 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. Finally, to estimate the model(s), one needs only to click the bottom button ‘Run model(s)’.

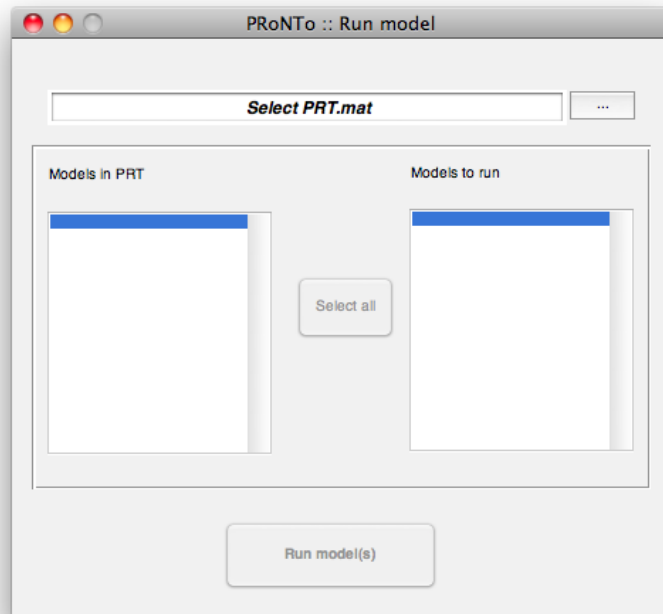


Figure 5.1: Model estimation GUI.

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 PRoNTTo window. This will launch the window shown in Fig 5.2. To estimate the weights and create the weight maps the user needs again to select a PRT.mat file. Then PRoNTTo will show the list of available models below, and the user can choose one model for which to estimate the weights. The last option refers to the name of the image file, which is saved in the same directory as PRT.mat, and that can have a name given by the user. Alternatively, if left empty PRoNTTo will name the file according to the machine corresponding to the selected model.

5.4 matlabbatch interface

The corresponding `matlabbatch` module to estimate the models can be found on PRoNTTo’s batch menu as ‘Run model’. The options are the same as in the GUI: one option to choose the PRT.mat and another to choose the model. The difference here is that the model names will not appear automatically and the user needs to write down the name (string) of the model to run. This needs to be exactly the name that was given to the model in the previous step. Another difference is the fact that only one model can be specified for each batch module. To run more than one model, more than one ‘Run model’ modules can be added to the jobs list and run sequentially.

The `matlabbatch` module to compute the weights has exactly the same options as the GUI. The only difference being that instead of listing the available models in a given PRT, it will ask

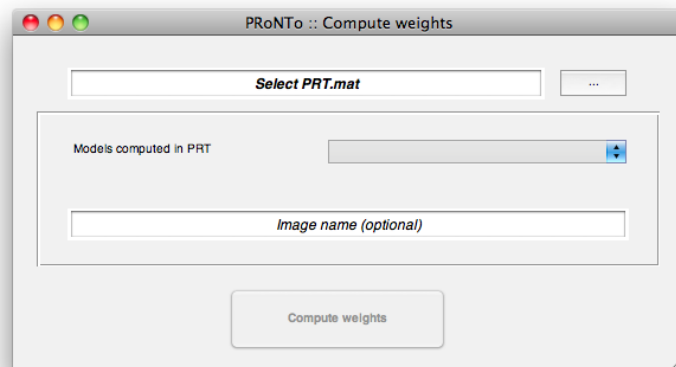


Figure 5.2: Weights computation GUI.

for the name (string) of the model to be used. Again the name of the model should be exactly the name given in 'Specify model'.

Chapter 6

Results display

Contents

6.1	Introduction	43
6.2	Launching results display	44
6.3	The main results display window	44
6.4	Analysing a machine's performance graphically	45
6.4.1	Predictions plot	45
6.4.2	Receiver Operating Characteristic (ROC) plot	45
6.4.3	Histogram plot	46
6.5	Statistical analysis of a machine's performance	47
6.5.1	Confusion matrix plot	47
6.5.2	The statistics table	47
6.5.3	Permutation testing	48
6.6	Visualising a weight map	48

6.1 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 test set in cross-validation, it is necessary to examine the outcome of the whole procedure in details. The results windows helps make various useful statistical statements about the predictive power of a machine, which (if any) subjects or conditions are modelled best, and which machine has the lowest error rate on a given dataset.

Another important aspect is to see what the machine has learned - some brain areas are probably more informative about class membership than others. For example, in a visual task, we would expect discriminative information in the occipital lobe. This is called *information mapping*, and it is of particular import to be critical at this stage - if the discriminative weight of a machine is concentrated in the eyes, for example, it is important to correct the analysis mask that was used to exclude them. The question to ask is “which voxels drive the modelling, and do they make sense with respect to the experimental paradigm and neurophysiology” ? In the case of linear kernels, the classifier/regression weight vector is a linear combination or weighted average of the training examples, and can be plotted as an image representing a weight map. The *weight map* is therefore a spatial representation of the decision function, i.e. every voxel contributes with a certain weight 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 brain pattern, rather than on individual regions or voxels, all voxels contribute to the classification or regression and no conclusions should be drawn about a particular subset of voxels in isolation.

Finally, examining model output and parameters is helpful in diagnosing the potentially bad performance of a particular mode - for example, if the machine cannot perform above chance, it

could be due to an inappropriate experimental paradigm, noisy data, insufficient amount of data, wrong choice of features, wrong choice of machine. It is important to recognise that any of these factors could cause the modelling to fail. The results window can help pinpointing the source of error.

6.2 Launching results display

Make sure all previous steps have been performed (Data and Design, Chapter 2; Prepare feature set, Chapter 3; Specify Model and Run Model, Chapter 4). Optionally, you may want to generate a weight map for your machine (Compute Weights, Chapter 5), but this is not mandatory.

In the **Review Options** panel, 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 in the right column. The window should then look something like Figure 6.1.

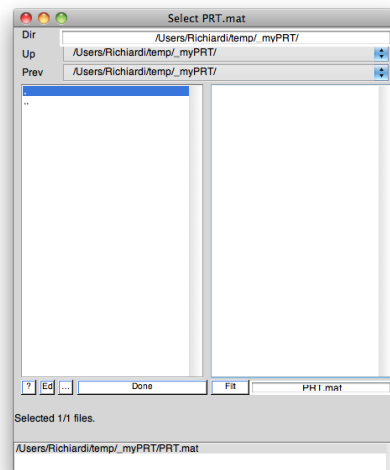


Figure 6.1: Selecting a PRT.mat for results display.

Click on **Done**, and the main results window opens (see an example initial state in Figure 6.2). In the **Model** pane in the top-right corner, you can check that you have loaded the correct **PRT.mat** by checking the list of model names appearing in the **Model** selector. For example, in Figure 6.2, we have one single model called **mySVM_AudRest**, with several cross-validation folds.

6.3 The main results display window

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

Plot : this pane displays the plots for the various analyses that can be performed on test results. With the exception of the confusion matrix plot, these cannot be interacted with.

Model : this pane allows the user to select the model to analyse, whether to analyse a particular fold or all folds at once, and which plot to produce. The **stats** sub-pane allows the user to generate a variety of statistics on the test, including accuracy statistics for classifiers, and p-values on these parameters via permutation testing.

Weight map : if a weight map has been computed (see Chapter 5) and loaded, this displays three projections of the map and allows to navigate it. If Fold: All folds / Average is selected, then this displays an average weight map across folds rather than the weight map of a single fold.

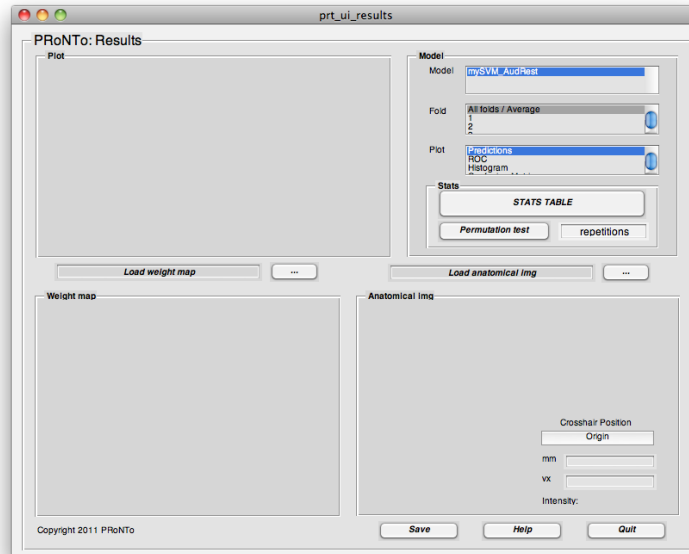


Figure 6.2: Initial state of the results display main window.

Anatomical img : if an anatomical image has been loaded, this will display three projections, and the cross-hair will be synchronised with the weight map.

To populate the Plot pane, first click on a model in the **Model** selector, then on 'all folds' (or a particular fold) in the **Fold** selector, and finally on a plot in the **Plot** selector. The next section details the plots available.

6.4 Analysing a machine's performance graphically

Looking at a machine output's graphically can often yield insights into the performance of the machine, and where modelling assumptions may prove false.

6.4.1 Predictions plot

A predictions plot displays, for a particular fold, the output value of the machine's decision function for each test sample (e.g., for a linear SVM, this could be $\mathbf{w}^T \mathbf{x}_i + b$, for a probabilistic classifier this could be a posterior probability $P(\Omega = \omega | \mathbf{x}_i)$). A well-performing classifier will yield very different function values for samples of different classes. By observing which fold have more or less overlapping function values, it is possible to understand which block / subject / condition might have a test distribution of features that departs from the training set.

On the plot, each class is represented by a different marker, and indicated in the legend. Figure 6.3 shows an example predictions plot. Here fold 1 is particularly well-behaved.

6.4.2 Receiver Operating Characteristic (ROC) plot

In two-class classification, there is always a trade-off between class 1 and class 2 errors. Indeed, a classifier predicting class 1 regardless of input would have excellent accuracy on class 1, but bad accuracy on class 2. This is also known as the sensitivity / specificity trade-off. The ROC curve is a graphical depiction of this trade-off, showing how one error rate varies as a function of the other. An ideal classifier would have an ROC passing through the top-left corner. The area under curve (AUC) is a summary measure of classifier performance, where higher is better (1 represents perfect performance, 0.5 represents random performance). As with all summary measures, the

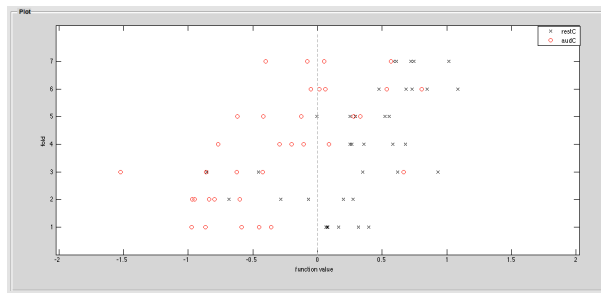


Figure 6.3: Example predictions plot for a two-class problem modelled by an SVM.

AUC is but one way of comparing performance of machines, and cannot be used alone to declare a machine statistically significantly superior to another on a given dataset.

Figure 6.4 shows an example of such a plot.

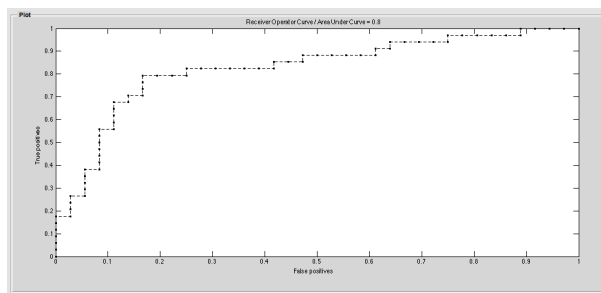


Figure 6.4: Example ROC curve for a two-class problem modelled by an SVM.

6.4.3 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 minute 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 6.5 shows an example of such a plot.

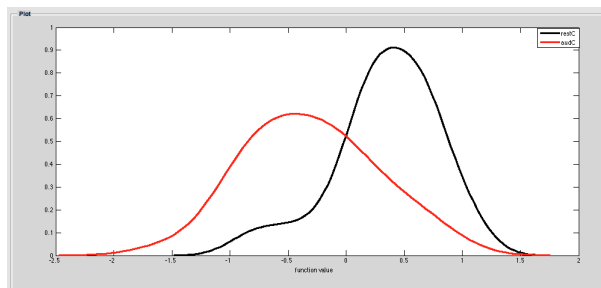


Figure 6.5: Example function values histogram curve for a two-class problem modelled by an SVM.

6.5 Statistical analysis of a machine's performance

One of the main questions to ask of a model is how precise its predictions are. In regression, goodness-of-fit is often assessed via mean squared error. In classification, a common practice is to compute prediction accuracy, both for each class and for all test data. Once an accuracy value has been obtained, it is also possible to obtain a p-value for the accuracy, reflecting how certain we are that the result is not due to chance.

6.5.1 Confusion matrix plot

The confusion matrix shows counts of predicted class labels $\hat{\omega}_n = f(\mathbf{x}_n)$ (in rows) versus true class labels ω_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 6.5 shows an example of a confusion matrix.

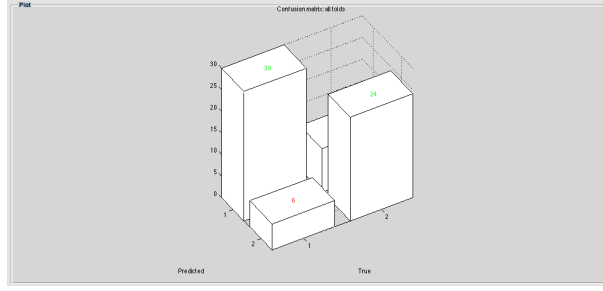


Figure 6.6: Example confusion matrix for all folds of a two-class problem modelled by an SVM.

6.5.2 The statistics table

The statistics table (see Figure 6.7 for an example in a classification setting) gives a summary of the model's performance. The accuracy p 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

$$p = 1 - \frac{1}{N} \sum_n l_{01}(\omega_n, f(\mathbf{x}_n)), \quad (6.1)$$

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

$$l_{01}(\omega_n, f(\mathbf{x}_n)) = \begin{cases} 0 & \omega_n = f(\mathbf{x}_n) \\ 1 & \omega_n \neq f(\mathbf{x}_n) \end{cases} \quad (6.2)$$

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 6.1 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 $\{p_1, \dots, p_C\}$, from which the balanced accuracy can be computed as

$$p^{bal} = \frac{1}{C} \sum p_c. \quad (6.3)$$

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 table also gives the class accuracies $\{p_1, \dots, p_C\}$, 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.

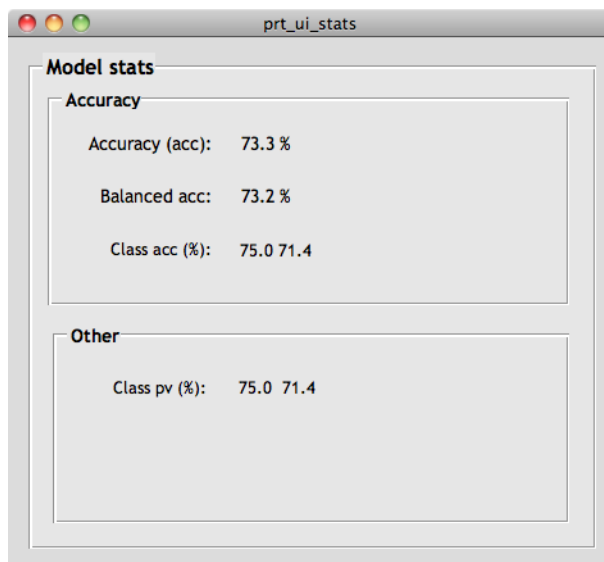


Figure 6.7: Example statistics tables for all folds of a two-class problem modelled by an SVM.

6.5.3 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 always be appropriate. Permutation testing is a non-parametric procedure that allows to obtain meaningful confidence intervals and p-values 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. After filling in the `repetitions` field with a number of repetitions R , pressing the **Permutation test** button will run for the specified number of times, and produce a p-value for accuracy statistics (see Figure 6.7). The smallest increment in p-value is proportional to $1/R$ (e.g. 20 repetitions gives increments of 0.05).

6.6 Visualising a weight map

By clicking on the [...] button next to the **Load weight map** field, a dialogue opens that allows you to select the weight map `.img` file that was computed previously (see Chapter 5 for the procedure). Similarly, a co-registered anatomical image can be loaded in the **Anatomical img** pane of the window. See Figure 6.8 for an example.

The weight map is then displayed with a cross-hair and a colorbar. The colorbar indicates the relative importance of the voxel in the decision function of the machine. This value is also indicated in the `intensity` field of the **Anatomical img** pane. Note that all voxels in the mask 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 does not make sense to isolate part of the pattern and report only on the peaks of the distribution of the decision function's weight map.*

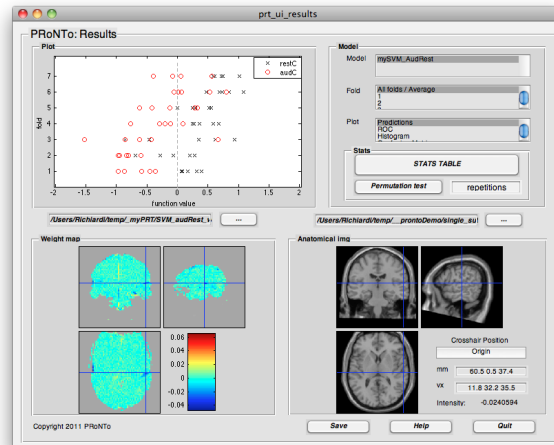


Figure 6.8: Example weight map over all folds for a two-class problem modelled by an SVM. In this rest versus auditory condition example, the voxels with the highest relative weight are located around auditory areas (notably Heschl's gyrus), bilaterally.

Part II

Batching system

Chapter 7

Data & Design

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

7.1 Directory

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

7.2 Groups

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

7.2.1 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 images (scans) per subject, such as a fMRI time series, or you may have many subjects with only one or a small number of images (scans) per subject, such as PET images. If you have many scans per subject select the option 'subjects'. If you have one scan for many subjects select the option 'scans'.

Subjects Add subjects/scans.

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.

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

SCANS Select scans (images) for this modality. They must all have the same image dimensions, orientation, voxel size etc.

DATA & DESIGN Specify data and design.

Load SPM.mat Load design from SPM.mat (if you have previously specified the experimental design with SPM).

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

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

Conditions Specify conditions. 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.

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 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.

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.

No design Do not specify design. This option can be used for modalities (e.g. structural scans) that do not have an experimental design.

Scans Depending on the type of data at hand, you may have many images (scans) per subject, such as a fMRI time series, or you may have many subjects with only one or a small number of images (scans) per subject, such as PET images. Select this option if you have many subjects per modality to spatially normalise, but there is one or a small number of scans 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 scans' 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 accross subjects/groups and the same names specified in the masks.

Files Select scans (images) for this modality. They must all have the same image dimensions, orientation, voxel size etc.

Regression targets (per scans) Enter one regression target per scan. or enter the name of a variable. This variable should be a vector [Nscans x 1], where Nscans is the number of scans/images.

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.

7.3 Masks

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

7.3.1 Modality

Specify name of modality and file 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 accross subjects/groups and the same names specified in the masks.

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.

7.4 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).

7.5 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).

7.6 Review

Choose 'Yes' if you would like to review your data and design in a separate window.

Chapter 8

Feature set / Kernel

Compute feature set according to the design specified

8.1 Load PRT.mat

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

8.2 Name

Target name for kernel matrix. This should contain only alphanumerical characters or underscores (-).

8.3 Modalities

Add modalities

8.3.1 Modality

Specify modality, such as name and data.

Name

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

Scans / Conditions

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

All scans 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

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.

Chapter 9

Specify model

Construct model according to design specified

9.1 Load PRT.mat

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

9.2 Model name

Name for model

9.3 Use kernels

Are the data for this model in the form of kernels/basis functions? If 'No' is selected, it is assumed the data are in the form of feature matrices

9.4 Feature sets

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

9.5 Model Type

Select which kind of predictive model is to be used.

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

Specify Conditions Specify the name of conditions to be included

Condition Specify condition:.

Name Name of condition to include.

All Conditions Include all conditions in this model

All scans 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

Machine

Choose a prediction machine for this model

SVM Classification Binary support vector machine.

Arguments Arguments for `prt_machine_svm_bin`. You should use `-t 4` if you selected 'use kernels' option, and `-t 0` otherwise. See libSVM documentation for details.

Gaussian Process Classification Gaussian Process Classification

Arguments Arguments for `prt_machine_gpml`

Multiclass GPC Multiclass GPC

Arguments Arguments for `prt_machine_gpclap`

Random Forest Random Forest. Breiman, Leo (2001). "Random Forests".

Machine Learning 45:5-32. This is a wrapper around Peter Geurt's implementation in his Regression Tree package.

Ntrees Number of trees in the forest.

Custom machine Choose another prediction machine

Function Choose a function that will perform prediction.

Arguments Arguments for prediction machine.

9.5.2 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

Modality name Name of modality. We only allow one modality for regression model per group at this moment

Example: 'BOLD'. Must match design specification

Machine

Choose a prediction machine for this model

Kernel Ridge Regression Kernel Ridge Regression.

Regularization Regularization for prt_machine_krr.

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

Arguments Arguments for prt_machine_gpr

Custom machine Choose another prediction machine

Function Choose a function that will perform prediction.

Arguments Arguments for prediction machine.

9.6 Cross-validation type

Choose the type of cross-validation to be used

9.6.1 Leave one subject out

Leave a single subject out each cross-validation iteration

9.6.2 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.

9.6.3 Leave one block out

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

9.6.4 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.

9.6.5 Custom

Load a cross-validation matrix. Note that an interface will be provided for this functionality in a later release

9.7 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.

9.8 Data operations

Specify operations to apply

9.8.1 Mean centre features

Select an operation to apply.

9.8.2 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 10

Run model

Trains and tests the predictive machine using the cross-validation structure specified by the model.

10.1 Load PRT.mat

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

10.2 Model name

Name of a model. Must match your entry in the
'Specify model' batch module.

Part III

Data processing examples

Chapter 11

Data set 1

This is where we explain how how to process data set 1.
This will arrive soon...

Chapter 12

Data set 2

This is where we explain how how to process data set 2.
This will arrive soon...

Part IV

Advanced topics

Chapter 13

PRT structure

This is how the main PRT structure is organised.
PRT

- group
 - gr_name
 - subject
 - subj_name()
 - modality()
 - mod_name
 - TR
 - scans
 - design
 - conds
 - cond_name()
 - onsets()
 - durations()
 - rt_trial()
 - scans()
 - blocks()
 - discardedscans()
 - hrfdiscardedscans()
 - stats
 - overlap
 - goodscans
 - discscans
 - meanovl
 - stdovl
 - mgoodovl
 - sgoodovl
 - goodovl
 - TR
 - unit
 - covar
- masks
 - mod_name

- fname
- fs
 - fs_name
 - k_file
 - id_col_names
 - fas
 - im
 - ifa
 - modality
 - mod_name
 - detrend
 - param_dt
 - mode
 - idfeat_fas
 - normalise
 - type
 - scaling
 - id_mat
- fas
 - mod_name
 - dat
 - detrend
 - param_dt
 - hdr
 - fname
 - dim
 - mat
 - pinfo
 - dt
 - n
 - descrip
 - private
 - idfeat_img
- model
 - model_name()
 - input()
 - use_kernel
 - type
 - machine
 - function
 - args
 - class
 - class_name()
 - group()
 - gr_name

- subj
 - num()
 - modality()
- fs
 - fs_name
- samp_idx
- targets
- targ_allscans
- cv_mat
- operations
- cv_type
- output()
 - fold
 - targets()
 - predictions()
 - stats()
 - con_mat
 - acc
 - c_acc
 - b_acc
 - acc_lb
 - acc_ub
 - func_val()
 - type()
 - alpha()
 - b()
 - totalSV()
- stats
 - con_mat
 - acc
 - c_acc
 - b_acc
 - acc_lb
 - acc_ub

Chapter 14

List of PRoNTo functions

Contents

14.1	<code>pronto.m</code>	78
14.2	<code>prt.m</code>	78
14.3	<code>prt_apply_operation.m</code>	78
14.4	<code>prt_check_design.m</code>	79
14.5	<code>prt_compute_weights.m</code>	80
14.6	<code>prt_cv_model.m</code>	80
14.7	<code>prt_cv_opt_param.m</code>	80
14.8	<code>prt_data_conditions.m</code>	81
14.9	<code>prt_data_modality.m</code>	81
14.10	<code>prt_data_review.m</code>	82
14.11	<code>prt_defaults.m</code>	82
14.12	<code>prt_fs.m</code>	83
14.13	<code>prt_func2html.m</code>	83
14.14	<code>prt_get_defaults.m</code>	83
14.15	<code>prt_get_filename.m</code>	84
14.16	<code>prt_init_fs.m</code>	84
14.17	<code>prt_init_model.m</code>	85
14.18	<code>prt_latex.m</code>	86
14.19	<code>prt_load.m</code>	86
14.20	<code>prt_load_blocks.m</code>	86
14.21	<code>prt_model.m</code>	86
14.22	<code>prt_normalise_kernel.m</code>	87
14.23	<code>prt_permutation.m</code>	87
14.24	<code>prt_preproc.m</code>	88
14.25	<code>prt_remove_confounds.m</code>	88
14.26	<code>prt_stats.m</code>	88
14.27	<code>prt_struct2latex.m</code>	89
14.28	<code>prt_text_input.m</code>	89
14.29	<code>prt_ui_compute_weights.m</code>	89
14.30	<code>prt_ui_cv_model.m</code>	90
14.31	<code>prt_ui_design.m</code>	90
14.32	<code>prt_ui_kernel_construction.m</code>	91
14.33	<code>prt_ui_main.m</code>	91
14.34	<code>prt_ui_model.m</code>	92
14.35	<code>prt_ui_prepare_data.m</code>	92
14.36	<code>prt_ui_prepare_datamod.m</code>	92
14.37	<code>prt_ui_results.m</code>	93

14.38	<code>prt_ui_results_help.m</code>	93
14.39	<code>prt_ui_reviewCV.m</code>	94
14.40	<code>prt_ui_reviewmodel.m</code>	94
14.41	<code>prt_ui_select_class.m</code>	95
14.42	<code>prt_ui_select_reg.m</code>	95
14.43	<code>prt_ui_stats.m</code>	96
14.44	<code>prt_ui_sure.m</code>	96
14.45	<code>machines</code>	96
14.45.1	<code>machines\prt_KRR.m</code>	96
14.45.2	<code>machines\prt_machine.m</code>	96
14.45.3	<code>machines\prt_machine_RT_bin.m</code>	97
14.45.4	<code>machines\prt_machine_gpclap.m</code>	98
14.45.5	<code>machines\prt_machine_gpml.m</code>	98
14.45.6	<code>machines\prt_machine_gpr.m</code>	99
14.45.7	<code>machines\prt_machine_krr.m</code>	100
14.45.8	<code>machines\prt_machine_rvr.m</code>	101
14.45.9	<code>machines\prt_machine_svm_bin.m</code>	101
14.45.10	<code>machines\prt_rvr.m</code>	102
14.45.11	<code>machines\prt_weights.m</code>	103
14.45.12	<code>machines\prt_weights_bin_linkernel.m</code>	103
14.45.13	<code>machines\prt_weights_svm_bin.m</code>	103
14.46	<code>utils</code>	104
14.46.1	<code>utils\prt_centre_kernel.m</code>	104
14.46.2	<code>utils\prt_checkAlphaNumUnder.m</code>	104
14.46.3	<code>utils\prt_normalise_kernel.m</code>	104

This is the list of PRoNTo functions, including the subdirectories: `machines` and `utils`.

14.1 `pronto.m`

Function launching PRoNTo (Pattern Recognition for Neuroimaging Toolbox), see `prt.m` for more details

14.2 `prt.m`

Pattern Recognition for Neuroimaging Toolbox, PRoNTo.

This function initializes things for PRoNTo and provides some low level functionalities

14.3 `prt_apply_operation.m`

function to apply a data operation to the training, test and

```

in.train:      training data
in.tr_id:      id matrix for training data

```

`in.use_kernel:` are the data in kernelised form
`in.tr_targets:` training targets (optional field)
`in.pred_type:` 'classification' or 'regression' (required for `tr_targets`)

A test set may also be specified, which require the following fields:

`in.test:` test data
`in.testcov:` test covariance (only if `use_kernel = true`)
`in.te_targets:` test targets
`in.te_id:` id matrix for test data

`opid` specifies the operation to apply, where:

- 1 = Temporal Compression
- 2 = Sample averaging (average samples for each subject/condition)
- 3 = Mean centre features over subjects
- 4 = Divide data vectors by their norm
- 5 = Perform a GLM (fMRI only)

N.B: - all operations are applied independently to training and test partitions
 - see Chu et. al (2011) for mathematical descriptions of operations 1 and 2 and Shawe-Taylor and Cristianini (2004) for a description of operation 3.

References:

Chu, C et al. (2011) Utilizing temporal information in fMRI decoding: classifier using kernel regression methods. *Neuroimage*. 58(2):560-71.
 Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for Pattern analysis*. Cambridge University Press.

14.4 prt_check_design.m

`FORMAT [conds] = prt_check.design(cond,tr,units,hrfoverlap)`

Check the design and discards scans which are either overlapping between conditions or which do not respect a minimum time interval between conditions (due to the width of the HRF function).

INPUT

- `cond` : structure containing the names, durations and onsets of the conditions
- `tr` : interscan interval (TR)
- `units` : 1 for seconds, 0 for scans
- `hrfoverlap` : value to correct for BOLD overlap (in seconds)
- `hrfdelay` : value to correct for BOLD delay (in seconds)

OUTPUT

the same `cond` structure containing supplementary fields:

- `scans` : scans retained for further classification
- `discardedscans` : scans discarded because they overlapped between conditions
- `hrfdiscardedscans` : scans discarded because they didn't respect the minimum time interval between conditions
- `blocks` : represents the grouping of the stimuli (for cross-validation)
- `stats` : struct containing the original time intervals, the

time interval with only the 'good' scans, their means and standard deviation

14.5 prt_compute_weights.m

FORMAT prt_compute_weights(PRT,in)

This function calls prt_weights to compute weights

Inputs:

PRT	- data/design/model structure (it needs to contain at least one estimated model).
in	- structure with specific information to create weights
.model_name	- model name (string)
.img_name	- (optional) name of the file to be created (string)
.pathdir	- directory path where to save weights (same as the one for PRT.mat) (string)

Output:

empty	- does not return anything (it creates an .img file)
-------	--

14.6 prt_cv_model.m

Function to run a cross-validation structure on a given model

Inputs:

PRT containing the specified model plus the following arguments:

in.filename:	filename for PRT.mat (string)
in.model_name:	name for this model (string)

Outputs:

Writes the following fields in the PRT data structure:

PRT.model(m).output.fold(i).targets:	targets for fold(i)
PRT.model(m).output.fold(i).predictions:	predictions for fold(i)
PRT.model(m).output.fold(i).stats:	statistics for fold(i)
PRT.model(m).output.fold(i).custom:	optional fields

Notes: - The PRT.model(m).input fields are set by prt_init_model, not by this function

14.7 prt_cv_opt_param.m

Function to pass optional parameters into the classifier. This is primarily used for complex data prediction methods that need to know something about the experimental design that is normally not accessible to generic prediction functions (e.g. task onsets or TR). Examples of this kind of classifier include multi-class classifier using kernel regression (MCKR) and the machine that implements nested cross-validation.

Inputs:

PRT: main data structure
 ID: id matrix for the current cross-validation fold
 CV: cross-validation structure (current fold only)

Outputs:

Provides the following fields for use by the classifier
 param.id_fold: the id matrix for this fold
 param.model_id: id for the model being computed
 param.PRT: PRT data structure

14.8 prt_data_conditions.m

PRT_DATA_CONDITIONS M-file for prt_data_conditions.fig

PRT_DATA_CONDITIONS, by itself, creates a new PRT_DATA_CONDITIONS or raises the existing singleton*.

H = PRT_DATA_CONDITIONS returns the handle to a new PRT_DATA_CONDITIONS or the handle to the existing singleton*.

PRT_DATA_CONDITIONS('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_DATA_CONDITIONS.M with the given input arguments.

PRT_DATA_CONDITIONS('Property','Value',...) creates a new PRT_DATA_CONDITIONS or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_data_conditions_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_data_conditions_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.9 prt_data_modality.m

PRT_DATA_MODALITY M-file for prt_data_modality.fig

PRT_DATA_MODALITY, by itself, creates a new PRT_DATA_MODALITY or raises the existing singleton*.

H = PRT_DATA_MODALITY returns the handle to a new PRT_DATA_MODALITY or the handle to the existing singleton*.

PRT_DATA_MODALITY('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_DATA_MODALITY.M with the given input arguments.

PRT_DATA_MODALITY('Property','Value',...) creates a new PRT_DATA_MODALITY

or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before `prt_data_modality_OpeningFcn` gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to `prt_data_modality_OpeningFcn` via `varargin`.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.10 `prt_data_review.m`

PRT_DATA_REVIEW M-file for `prt_data_review.fig`

PRT_DATA_REVIEW, by itself, creates a new PRT_DATA_REVIEW or raises the existing singleton*.

H = PRT_DATA_REVIEW returns the handle to a new PRT_DATA_REVIEW or the handle to the existing singleton*.

PRT_DATA_REVIEW('CALLBACK', hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_DATA_REVIEW.M with the given input arguments.

PRT_DATA_REVIEW('Property','Value',...) creates a new PRT_DATA_REVIEW or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before `prt_data_review_OpeningFcn` gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to `prt_data_review_OpeningFcn` via `varargin`.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.11 `prt_defaults.m`

Sets the defaults which are used by the Pattern Recognition for Neuroimaging Toolbox, aka. PRoNTo.

FORMAT `prt_defaults`

This file can be customised to any the site/person own setup. Individual users can make copies which can be stored on their own matlab path. Make sure your '`prt_defaults`' is the first one found in the path. See matlab documentation for details on setting path.

Care must be taken when modifying this file!

The structure and content of this file are largely inspired by SPM:
<http://www.fil.ion.ucl.ac.uk/spm>

14.12 prt_fs.m

Function to build file arrays containing the (linearly detrended) data and compute a linear (dot product) kernel from them

Inputs:

in.fname: filename for the PRT.mat (string)
in.fs_name: name of fs and relative path filename for the kernel matrix

in.mod(m).mod_name: name of modality to include in this kernel (string)
in.mod(m).detrend: detrend (scalar: 0 = none, 1 = linear)
in.mod(m).param_dt: parameters for the kernel detrend (e.g. DCT bases)
in.mod(m).mode: 'all_cond' or 'all_scans' (string)
in.mod(m).mask: mask file used to create the kernel
in.mod(m).normalise: 0 = none, 1 = normalise_kernel, 2 = scale modality
in.mod(m).matnorm: filename for scaling matrix

Outputs:

Calls prt_init_fs to populate basic fields in PRT.fs(f)...
Writes PRT.mat
Writes the kernel matrix to the path indicated by in.fs_name

14.13 prt_func2html.m

Script to generate the list of .m functions into html files which can be browsed around with your favourite browser.

Note that this script relies on the M2HTML package which is **NOT** distributed with PRoNTTo!

For more information, please read the M2HTML tutorial and FAQ at:
\$<\$<http://www.artefact.tk/software/matlab/m2html/>>\$

14.14 prt_get_defaults.m

Get/set the defaults values associated with an identifier

FORMAT defaults = prt_get_defaults
Return the global "defaults" variable defined in prt_defaults.m.

FORMAT defval = prt_get_defaults(defstr)
Return the defaults value associated with identifier "defstr".
Currently, this is a '.' subscript reference into the global "prt_def" variable defined in prt_defaults.m.

FORMAT prt_get_defaults(defstr, defval)
Sets the defaults value associated with identifier "defstr". The new defaults value applies immediately to:
* new modules in batch jobs
* modules in batch jobs that have not been saved yet
This value will not be saved for future sessions of PRoNTTo. To make

persistent changes, edit `prt_defaults.m`.

The structure and content of this file are largely inspired by SPM & Matlabbatch.

<http://www.fil.ion.ucl.ac.uk/spm>

<http://sourceforge.net/projects/matlabbatch/>

14.15 `prt_get_filename.m`

```
out = prt_get_filename(ids)
```

14.16 `prt_init_fs.m`

function to initialise the kernel data structure

FORMAT: Two modes are possible:

```
fid = prt_init_fs(PRT, in)
```

```
[fid, PRT, tocomp] = prt_init_fs(PRT, in)
```

USAGE 1:

function will return the id of a feature set or an error if it doesn't exist in `PRT.mat`

Input:

`in.fs_name`: name for the feature set (string)

Output:

`fid` : is the identifier for the feature set in `PRT.mat`

USAGE 2:

function will create the feature set in `PRT.mat` and overwrite it if it already exists.

Input:

`in.fs_name`: name for the feature set (string)

`in.fname`: name of `PRT.mat`

`in.mod(m).mod_name`: name of the modality

`in.mod(m).detrend`: type of detrending

`in.mod(m).mode`: 'all_scans' or 'all_cond'

`in.mod(m).mask`: mask used to create the feature set

`in.mod(m).param_dt`: parameters used for detrending (if any)

`in.mod(m).normalise`: scale the input scans or not

`in.mod(m).matnorm`: mat file used to scale the input scans

Output:

`fid` : is the identifier for the model constructed in `PRT.mat`

Populates the following fields in PRT.mat (copied from above):

```
PRT.fs(f).fs_name
PRT.fs(f).fas
PRT.fs(f).k_file
```

Also computes the following fields:

```
PRT.fs(f).id.mat:      Identifier matrix (useful later)
PRT.fs(f).id.col.names: Columns in the id matrix
```

Note: this function does not write PRT.mat. That should be done by the calling function

14.17 prt_init_model.m

function to initialise the model data structure

FORMAT: Two modes are possible:

```
mid = prt_init_model(PRT, in)
[mid, PRT] = prt_init_model(PRT, in)
```

USAGE 1:

function will return the id of a model or an error if it doesn't exist in PRT.mat

Input:

in.model_name: name of the model (string)

Output:

mid : is the identifier for the model in PRT.mat

USAGE 2:

function will create the model in PRT.mat and overwrite it if it already exists.

Input:

in.model_name: name of the model to be created (string)
in.use_kernel: use kernel or basis functions for this model (boolean)
in.machine: prediction machine to use for this model (struct)
in.type: 'classification' or 'regression'

Output:

Populates the following fields in PRT.mat (copied from above):

```
PRT.model(m).input.model_name
PRT.model(m).input.type
PRT.model(m).input.use_kernel
PRT.model(m).input.machine
```

Note: this function does not write PRT.mat. That should be done by the calling function

14.18 prt_latex.m

Extract information from the toolbox m-files and output them as usable .tex files which can be directly included in the manual.

There are 2 types of m2tex operations:

1. converting the job configuration tree, i.e. *_cfg_* files defining the batching interface into a series of .tex files.

NOTE: Only generate .tex files for each exec_branch of prt_batch.

2. converting the help header of the functions into .tex files.

These files are then included in a manually written prt_manual.tex file, which also includes chapter/sections written manually.

File derived from that of the SPM8 distribution.

<http://www.fil.ion.ucl.ac.uk/spm>

14.19 prt_load.m

Function to load the PRT.mat and check its integrity regarding the kernels and feature sets that it is supposed to contain. Updates the set feature name if needed.

input : name of the PRT.mat, path included

output : PRT structure updated

14.20 prt_load_blocks.m

Load one or more blocks of data.

This script is a effectively a wrapper function that for the routines that actually do the work (SPM nifti routines)

The syntax is either:

img = prt_load_blocks(filenamees, block_size, block_range) just to specify continuous blocks of data

or

img = prt_load_blocks(filenamees, voxel_index) to access non continuous blocks

14.21 prt_model.m

Function to configure and build the PRT.model data structure

Input:

PRT fields:

model.fs(f).fs_name: feature set(s) this CV approach is defined for

```

model.fs(f).fs_features: feature selection mode ('all' or 'mask')
model.fs(f).mask_file:   mask for this feature set (fs_features='mask')

in.fname:                filename for PRT.mat
in.model_name: name for this cross-validation structure
in.type:                 'classification' or 'regression'
in.use_kernel: does this model use kernels or features?
in.operations: operations to apply before prediction

in.fs(f).fs_name:        feature set(s) this CV approach is defined for

in.class(c).class_name
in.class(c).group(g).subj(s).num
in.class(c).group(g).subj(s).modality(m).mod_name
EITHER: in.class(c).group(g).subj(s).modality(m).conds(c).cond_name
OR:      in.class(c).group(g).subj(s).modality(m).all_scans
OR:      in.class(c).group(g).subj(s).modality(m).all_cond

in.cv.type:              type of cross-validation ('loso','logso','custom')
in.cv.mat_file: file specifying CV matrix (if type='custom');

```

Output:

```

This function performs the following functions:
1. populates basic fields in PRT.model(m).input
2. computes PRT.model(m).input.targets based on in.class(c)...
3. computes PRT.model(m).input.samp_idx based on targets
4. computes PRT.model(m).input.cv_mat based on the labels and CV spec

```

14.22 prt_normalise_kernel.m

This function normalises the kernel matrix such that each entry is divided by the product of the std deviations, i.e.

$$K_{\text{new}}(x,y) = K(x,y) / \sqrt{\text{var}(x) \cdot \text{var}(y)}$$

14.23 prt_permutation.m

Function to compute permutation test

Inputs:

```

PRT: PRT structured including model
n_permu: number of permutations
modelid: model ID

```

Outputs:

```

for classification
permutation.c_acc: Permuted accuracy per class
permutation.b_acc: Permuted balanced accuracy
permutation.pvalue_b_acc: p-value for c_acc
permutation.pvalue_c_acc: p-value for b_acc

```

```

for regression
permutation.corr: Permuted correlation
permutation.mse: Permuted mean square error
permutation.corr: p-value for corr
permutation.mse: p-value for mse

```

14.24 prt_preproc.m

Function to preprocess the images, by loading each one of them (or the ones corresponding to the selected scans when a design was specified), applying the masks on them and, if asked, detrend along each voxel along the time series.

INPUT:

```

    fname    filename and path to PRT.mat

```

OUTPUT:

```

    results are saved on disk.

```

14.25 prt_remove_confounds.m

```

[Kr, R] = prt_remove_confounds(K,C)

```

Function to remove confounds from kernel.

14.26 prt_stats.m

Function to compute predictions machine performance statistics

Inputs:

```

model.predictions: predictions derived from the predictive model
model.type:        what type of prediction machine (e.g. 'classifier','regression')

```

```

tte: true targets (test set)
ttr: true targets (training set - needed to get the number of classes)
flag: 'fold' for statistics in each fold
      'model' for statistics in each model

```

Outputs:

Classification:

```

stats.con_mat: Confusion matrix (nClasses x nClasses matrix, pred x true)
stats.acc:     Accuracy (scalar)
stats.b_acc:   Balanced accuracy (nClasses x 1 vector)
stats.c_acc:   Accuracy by class (nClasses x 1 vector)
stats.c_pv:    Predictive value for each class (nClasses x 1 vector)

```

Regression:

```

stats.mse:     Mean square error between test and prediction
stats.corr:    Correlation between test and prediction

```

14.27 prt_struct2latex.m

Function that takes in a structure S and writes down the latex code describing the whole structure and substructures recursively. The routine specifically generates the 'adv_PRTstruct.tex' file that is included, in the prt_manual.

Bits of the code and copied/inspired by spm_latex.m from the SPM8 distribution: <http://www.fil.ion.ucl.ac.uk/spm>

14.28 prt_text_input.m

PRT_TEXT_INPUT M-file for prt_text_input.fig

PRT_TEXT_INPUT, by itself, creates a new PRT_TEXT_INPUT or raises the existing singleton*.

H = PRT_TEXT_INPUT returns the handle to a new PRT_TEXT_INPUT or the handle to the existing singleton*.

PRT_TEXT_INPUT('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_TEXT_INPUT.M with the given input arguments.

PRT_TEXT_INPUT('Property','Value',...) creates a new PRT_TEXT_INPUT or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_text_input_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_text_input_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.29 prt_ui_compute_weights.m

PRT_UI_COMPUTE_WEIGHTS M-file for prt_ui_compute_weights.fig

PRT_UI_COMPUTE_WEIGHTS, by itself, creates a new PRT_UI_COMPUTE_WEIGHTS or raises the existing singleton*.

H = PRT_UI_COMPUTE_WEIGHTS returns the handle to a new PRT_UI_COMPUTE_WEIGHTS or the handle to the existing singleton*.

PRT_UI_COMPUTE_WEIGHTS('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_COMPUTE_WEIGHTS.M with the given input arguments.

PRT_UI_COMPUTE_WEIGHTS('Property','Value',...) creates a new PRT_UI_COMPUTE_WEIGHTS or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_compute_weights_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_compute_weights_OpeningFcn

via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.30 prt_ui_cv_model.m

PRT_UI_CV_MODEL M-file for prt_ui_cv_model.fig

PRT_UI_CV_MODEL, by itself, creates a new PRT_UI_CV_MODEL or raises the existing singleton*.

H = PRT_UI_CV_MODEL returns the handle to a new PRT_UI_CV_MODEL or the handle to the existing singleton*.

PRT_UI_CV_MODEL('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_CV_MODEL.M with the given input arguments.

PRT_UI_CV_MODEL('Property','Value',...) creates a new PRT_UI_CV_MODEL or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_cv_model_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_cv_model_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.31 prt_ui_design.m

PRT_UI_DESIGN M-file for prt_ui_design.fig

PRT_UI_DESIGN, by itself, creates a new PRT_UI_DESIGN or raises the existing singleton*.

H = PRT_UI_DESIGN returns the handle to a new PRT_UI_DESIGN or the handle to the existing singleton*.

PRT_UI_DESIGN('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_DESIGN.M with the given input arguments.

PRT_UI_DESIGN('Property','Value',...) creates a new PRT_UI_DESIGN or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_design_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_design_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.32 prt_ui_kernel_construction.m

PRT_UI_KERNEL MATLAB code for prt_ui_kernel.fig

PRT_UI_KERNEL, by itself, creates a new PRT_UI_KERNEL or raises the existing singleton*.

H = PRT_UI_KERNEL returns the handle to a new PRT_UI_KERNEL or the handle to the existing singleton*.

PRT_UI_KERNEL('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_KERNEL.M with the given input arguments.

PRT_UI_KERNEL('Property','Value',...) creates a new PRT_UI_KERNEL or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_kernel_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_kernel_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.33 prt_ui_main.m

PRT_UI_MAIN M-file for prt_ui_main.fig

PRT_UI_MAIN, by itself, creates a new PRT_UI_MAIN or raises the existing singleton*.

H = PRT_UI_MAIN returns the handle to a new PRT_UI_MAIN or the handle to the existing singleton*.

PRT_UI_MAIN('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_MAIN.M with the given input arguments.

PRT_UI_MAIN('Property','Value',...) creates a new PRT_UI_MAIN or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_main_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_main_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.34 prt_ui_model.m

PRT_UI_KERNEL_CONSTRUCTION M-file for prt_ui_kernel.construction.fig

PRT_UI_KERNEL_CONSTRUCTION, by itself, creates a new PRT_UI_KERNEL_CONSTRUCTION or raises the existing singleton*.

H = PRT_UI_KERNEL_CONSTRUCTION returns the handle to a new PRT_UI_KERNEL_CONSTRUCTION or the handle to the existing singleton*.

PRT_UI_KERNEL_CONSTRUCTION('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_KERNEL_CONSTRUCTION.M with the given input arguments.

PRT_UI_KERNEL_CONSTRUCTION('Property','Value',...) creates a new PRT_UI_KERNEL_CONSTRUCTION or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_kernel.construction.OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_kernel.construction.OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.35 prt_ui_prepare_data.m

PRT_UI_KERNEL MATLAB code for prt_ui_kernel.fig

PRT_UI_KERNEL, by itself, creates a new PRT_UI_KERNEL or raises the existing singleton*.

H = PRT_UI_KERNEL returns the handle to a new PRT_UI_KERNEL or the handle to the existing singleton*.

PRT_UI_KERNEL('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_KERNEL.M with the given input arguments.

PRT_UI_KERNEL('Property','Value',...) creates a new PRT_UI_KERNEL or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_kernel.OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_kernel.OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.36 prt_ui_prepare_datamod.m

PRT_UI_KERNEL_MODALITY M-file for prt_ui_kernel.modality.fig

PRT_UI_KERNEL_MODALITY, by itself, creates a new PRT_UI_KERNEL_MODALITY or raises the existing singleton*.

H = PRT_UI_KERNEL_MODALITY returns the handle to a new PRT_UI_KERNEL_MODALITY or the handle to the existing singleton*.

PRT_UI_KERNEL_MODALITY('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_KERNEL_MODALITY.M with the given input arguments.

PRT_UI_KERNEL_MODALITY('Property','Value',...) creates a new PRT_UI_KERNEL_MODALITY or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_kernel_modality_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_kernel_modality_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.37 prt_ui_results.m

PRT_UI_RESULTS MATLAB code for prt_ui_results.fig

PRT_UI_RESULTS, by itself, creates a new PRT_UI_RESULTS or raises the existing singleton*.

H = PRT_UI_RESULTS returns the handle to a new PRT_UI_RESULTS or the handle to the existing singleton*.

PRT_UI_RESULTS('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_RESULTS.M with the given input arguments.

PRT_UI_RESULTS('Property','Value',...) creates a new PRT_UI_RESULTS or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_results_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_results_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.38 prt_ui_results_help.m

PRT_UI_RESULTS_HELP MATLAB code for prt_ui_results_help.fig

PRT_UI_RESULTS_HELP, by itself, creates a new PRT_UI_RESULTS_HELP or raises the existing singleton*.

`H = PRT_UI_RESULTS_HELP` returns the handle to a new `PRT_UI_RESULTS_HELP` or the handle to the existing singleton*.

`PRT_UI_RESULTS_HELP('CALLBACK',hObject,eventData,handles,...)` calls the local function named `CALLBACK` in `PRT_UI_RESULTS_HELP.M` with the given input arguments.

`PRT_UI_RESULTS_HELP('Property','Value',...)` creates a new `PRT_UI_RESULTS_HELP` or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before `prt_ui_results_help_OpeningFcn` gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to `prt_ui_results_help_OpeningFcn` via `varargin`.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: `GUIDE`, `GUIDATA`, `GUIHANDLES`

14.39 prt_ui_reviewCV.m

`PRT_UI_REVIEWCV` M-file for `prt_ui_reviewCV.fig`

`PRT_UI_REVIEWCV`, by itself, creates a new `PRT_UI_REVIEWCV` or raises the existing singleton*.

`H = PRT_UI_REVIEWCV` returns the handle to a new `PRT_UI_REVIEWCV` or the handle to the existing singleton*.

`PRT_UI_REVIEWCV('CALLBACK',hObject,eventData,handles,...)` calls the local function named `CALLBACK` in `PRT_UI_REVIEWCV.M` with the given input arguments.

`PRT_UI_REVIEWCV('Property','Value',...)` creates a new `PRT_UI_REVIEWCV` or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before `prt_ui_reviewCV_OpeningFcn` gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to `prt_ui_reviewCV_OpeningFcn` via `varargin`.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: `GUIDE`, `GUIDATA`, `GUIHANDLES`

14.40 prt_ui_reviewmodel.m

`PRT_UI_REVIEWMODEL` M-file for `prt_ui_reviewmodel.fig`

`PRT_UI_REVIEWMODEL`, by itself, creates a new `PRT_UI_REVIEWMODEL` or raises the existing singleton*.

`H = PRT_UI_REVIEWMODEL` returns the handle to a new `PRT_UI_REVIEWMODEL` or the handle to the existing singleton*.

PRT_UI_REVIEWMODEL('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_REVIEWMODEL.M with the given input arguments.

PRT_UI_REVIEWMODEL('Property','Value',...) creates a new PRT_UI_REVIEWMODEL or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_reviewmodel_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_reviewmodel_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.41 prt_ui_select_class.m

PRT_UI_SELECT_CLASS M-file for prt_ui_select_class.fig

PRT_UI_SELECT_CLASS, by itself, creates a new PRT_UI_SELECT_CLASS or raises the existing singleton*.

H = PRT_UI_SELECT_CLASS returns the handle to a new PRT_UI_SELECT_CLASS or the handle to the existing singleton*.

PRT_UI_SELECT_CLASS('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_SELECT_CLASS.M with the given input arguments.

PRT_UI_SELECT_CLASS('Property','Value',...) creates a new PRT_UI_SELECT_CLASS or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_select_class_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_select_class_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.42 prt_ui_select_reg.m

PRT_UI_SELECT_REG M-file for prt_ui_select_reg.fig

PRT_UI_SELECT_REG, by itself, creates a new PRT_UI_SELECT_REG or raises the existing singleton*.

H = PRT_UI_SELECT_REG returns the handle to a new PRT_UI_SELECT_REG or the handle to the existing singleton*.

PRT_UI_SELECT_REG('CALLBACK',hObject,eventData,handles,...) calls the

local function named CALLBACK in PRT_UI_SELECT_REG.M with the given input arguments.

PRT_UI_SELECT_REG('Property','Value',...) creates a new PRT_UI_SELECT_REG or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_select_reg_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_select_reg_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.43 prt_ui_stats.m

PRT_UI_STATS MATLAB code for prt_ui_stats.fig

PRT_UI_STATS, by itself, creates a new PRT_UI_STATS or raises the existing singleton*.

H = PRT_UI_STATS returns the handle to a new PRT_UI_STATS or the handle to the existing singleton*.

PRT_UI_STATS('CALLBACK',hObject,eventData,handles,...) calls the local function named CALLBACK in PRT_UI_STATS.M with the given input arguments.

PRT_UI_STATS('Property','Value',...) creates a new PRT_UI_STATS or raises the existing singleton*. Starting from the left, property value pairs are applied to the GUI before prt_ui_stats_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to prt_ui_stats_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

14.44 prt_ui_sure.m

14.45 machines

14.45.1 machines\prt_KRR.m

w = prt_KRR(K,t,reg)

14.45.2 machines\prt_machine.m

Run machine function for classification or regression

FORMAT output = prt_machine(d,m)

Inputs:

- d - structure with information about the data, with fields:
 - Mandatory fields:
 - .train - training data (cell array of matrices of row vectors, each [Ntr x D]). 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 [Nte x D])
 - .tr_targets - training labels (for classification) or values (for regression) (column vector, [Ntr x 1])
 - .use_kernel - flag, is data in form of kernel matrices (true) or in form of features (false)
 - Optional fields: the machine is responsible for dealing with this optional fields (e.g. d.testcov)
- m - structure with information about the classification or regression machine to use, with fields:
 - .function - function for classification or regression (string)
 - .args - function arguments (either a string, a matrix, or a struct). This is specific to each machine, e.g. for an L2-norm linear SVM this could be the C parameter

Output:

- output - output of machine (struct).
 - Mandatory fields:
 - .predictions - predictions of classification or regression [Nte x D]
 - Optional fields: the machine is responsible for returning parameters of interest. For example for an SVM this could be the number of support vector used in the hyperplane weights computation

14.45.3 machines\prt_machine_RT_bin.m

Run binary Ensemble of Regression Tree - wrapper for Pierre Geurt's RT code

FORMAT output = prt_machine_RT_bin(d,args)

Inputs:

- d - structure with data information, with mandatory fields:
 - .train - training data (cell array of matrices of row vectors, each [Ntr x D]). 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 [Nte x D])
 - .tr_targets - training labels (for classification) or values (for regression) (column vector, [Ntr x 1])
 - .use_kernel - flag, is data in form of kernel matrices (true) or in form of features (false)
- args - vector of RT arguments
 - args(1) - number of trees (default: 501)

Output:

- output - output of machine (struct).
 - * Mandatory fields:
 - .predictions - predictions of classification or regression [Nte x D]
 - * Optional fields:
 - .func_val - value of the decision function

.type - which type of machine this is (here, 'classifier')

14.45.4 machines\prt_machine_gpclap.m

Run multiclass Gaussian process classification (Laplace approximation)

FORMAT output = prt_machine_gpclap(d,args)

Inputs:

d - structure with data information, with mandatory fields:

- .train - training data (cell array of matrices of row vectors, each [Ntr x D]). 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 [Nte x D])
- .testcov - testing covariance (cell array of matrices row vectors, each [Nte x Nte])
- .tr_targets - training labels (for classification) or values (for regression) (column vector, [Ntr x 1])
- .use_kernel - flag, is data in form of kernel matrices (true) or in form of features (false)

args - argument string, where

- h - optimise hyperparameters (otherwise don't)
- c covfun - covariance function:
 - 'covLINKcell' - simple dot product
 - 'covLINGlm' - construct a GLM

experimental args (use at your own risk):

- p - use priors for the hyperparameters. If specified, this indicates that a maximum a posteriori (MAP) approach will be used to set covariance function hyperparameters. The priors are obtained by calling prt_gp_priors('covFuncName')

N.B.: for the arguments specifying functions, pass in a string, not a function handle. This script will generate a function handle

Output:

output - output of machine (struct).

- * Mandatory fields:
 - .predictions - predictions of classification or regression [Nte x D]
- * Optional fields:
 - .type - which type of machine this is (here, 'classifier')
 - .func_val - predictive probabilities
 - .loghyper - log hyperparameters
 - .nlml - negative log marginal likelihood
 - .mu - test latent means
 - .s2 - test latent variances
 - .alpha - GP weighting coefficients

14.45.5 machines\prt_machine_gpml.m

Run Gaussian process model - wrapper for gpml toolbox

FORMAT output = prt_machine_gpml(d,args)

Inputs:

d - structure with data information, with mandatory fields:

- .train - training data (cell array of matrices of row vectors,

each [Ntr x D]). 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 [Nte x D])
.testcov   - testing covariance (cell array of matrices row vectors, each [Nte x Nte])
.tr_targets - training labels (for classification) or values (for regression) (column vector, [Ntr x 1])
.use_kernel - flag, is data in form of kernel matrices (true) or in form of features (false)

args      - argument string, where
  -h      - optimise hyperparameters (otherwise don't)
  -f iter  - max # iterations for optimiser (ignored if -h not set)
  -l likfun - likelihood function:
              'likErf' - erf/probit likelihood (binary only)
  -c covfun - covariance function:
              'covLINKcell' - simple dot product
              'covLINGlm'   - construct a GLM
  -m meanfun - mean function:
              'meanConstcell' - suitable for dot product
              'meanConstglm'  - suitable for GLM
  -i inffun - inference function:
              'prt_infEP' - Expectation Propagation

experimental args (use at your own risk):
  -p      - use priors for the hyperparameters. If specified, this indicates that a maximum a posteriori (MAP) approach will be used to set covariance function hyperparameters. The priors are obtained by calling prt_gp_priors('covFuncName')
```

N.B.: for the arguments specifying functions, pass in a string, not a function handle. This script will generate a function handle

Output:

```
output - output of machine (struct).
* Mandatory fields:
.predictions - predictions of classification or regression [Nte x D]
* Optional fields:
.type      - which type of machine this is (here, 'classifier')
.func_val  - predictive probabilities
.mu        - test latent means
.s2        - test latent variances
.loghyper  - log hyperparameters
.nlnl      - negative log marginal likelihood
.alpha     - GP weighting coefficients
.sW        - likelihood matrix (see Rasmussen & Williams, 2006)
.L         - Cholesky factor
```

14.45.6 machines\prt_machine_gpr.m

Run Gaussian process regression - meta-wrapper for regression with gpml FORMAT output = pr

Inputs:

```
d      - structure with data information, with mandatory fields:
.train  - training data (cell array of matrices of row vectors, each [Ntr x D]). 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 [Nte x D])
- .testcov - testing covariance (cell array of matrices row vectors, each [Nte x Nte])
- .tr_targets - training labels (for classification) or values (for regression) (column vector, [Ntr x 1])
- .use_kernel - flag, is data in form of kernel matrices (true) or in form of features (false)

args - argument string, where

- h - optimise hyperparameters (otherwise don't)
- f iter - max # iterations for optimiser (ignored if -h not set)
- l likfun - likelihood function:
 - 'likErf' - erf/probit likelihood (binary only)
- c covfun - covariance function:
 - 'covLINKcell' - simple dot product
 - 'covLINGlm' - construct a GLM
- m meanfun - mean function:
 - 'meanConstcell' - suitable for dot product
 - 'meanConstglm' - suitable for GLM
- i inffun - inference function:
 - 'prt_infEP' - Expectation Propagation

experimental args (use at your own risk):

- p - use priors for the hyperparameters. If specified, this indicates that a maximum a posteriori (MAP) approach will be used to set covariance function hyperparameters. The priors are obtained by calling `prt_gp_priors('covFuncName')`

N.B.: for the arguments specifying functions, pass in a string, not a function handle. This script will generate a function handle

Output:

output - output of machine (struct).

- * Mandatory fields:
 - .predictions - predictions of classification or regression [Nte x D]
- * Optional fields:
 - .type - which type of machine this is (here, 'classifier')
 - .func_val - predictive probabilities
 - .mu - test latent means
 - .s2 - test latent variances
 - .loghyper - log hyperparameters
 - .nlml - negative log marginal likelihood
 - .alpha - GP weighting coefficients
 - .sW - likelihood matrix (see Rasmussen & Williams, 2006)
 - .L - Cholesky factor

14.45.7 machines\prt_machine_krr.m

Kernel ridge regression

FORMAT output = prt_machine_svm_bin(d,args)

Inputs:

- d - structure with data information, with mandatory fields:
 - .train - training data (cell array of matrices of row vectors, each [Ntr x D]). 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
              [Nte x D])
.tr_targets - training labels (for classification) or values (for
              regression) (column vector, [Ntr x 1])
.use_kernel - flag, is data in form of kernel matrices (true) or in
              form of features (false)
args       - libSVM arguments
Output:
output     - output of machine (struct).
* Mandatory fields:
.predictions - predictions of classification or regression [Nte x D]
* Optional fields:
.func_val   - value of the decision function
.type      - which type of machine this is (here, 'classifier')

```

14.45.8 machines\prt_machine_rvr.m

Relevance vector regression (training and testing)

FORMAT output = prt_machine_svm_bin(d,args)

Inputs:

```

d          - structure with data information, with mandatory fields:
.train     - training data (cell array of matrices of row vectors,
              each [Ntr x D]). 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
              [Nte x D])
.tr_targets - training labels (for classification) or values (for
              regression) (column vector, [Ntr x 1])
.use_kernel - flag, is data in form of kernel matrices (true) or in
              form of features (false)
args       - libSVM arguments

```

Output:

```

output     - output of machine (struct).
* Mandatory fields:
.predictions - predictions of classification or regression [Nte x D]
* Optional fields:
.func_val   - value of the decision function
.type      - which type of machine this is (here, 'classifier')

```

14.45.9 machines\prt_machine_svm_bin.m

Run binary SVM - wrapper for libSVM

FORMAT output = prt_machine_svm_bin(d,args)

Inputs:

```

d          - structure with data information, with mandatory fields:
.train     - training data (cell array of matrices of row vectors,
              each [Ntr x D]). 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
              [Nte x D])
.tr_targets - training labels (for classification) or values (for

```

```

        regression) (column vector, [Ntr x 1])
    .use_kernel - flag, is data in form of kernel matrices (true) of in
                  form of features (false)
    args      - libSVM arguments
Output:
    output - output of machine (struct).
    * Mandatory fields:
    .predictions - predictions of classification or regression [Nte x D]
    * Optional fields:
    .func_val - value of the decision function
    .type     - which type of machine this is (here, 'classifier')
```

14.45.10 machines\prt_rvr.m

Optimisation for Relevance Vector Regression

```

[w,alpha,beta,ll] = prt_rvr(Phi,t)
Phi - MxM matrix derived from kernel function of vector pairs
t    - the values to be matched
w    - weights
alpha - 1/variance for the prior part of the model
beta  - 1/variance for the likelihood part of the model
ll    - the negative log-likelihood.

[w,alpha,beta,nu,ll]=spm_rvr(K,t,opt)
K    - a cell-array of MxM dot-product matrices.
t    - the values to be matched
opt  - either 'Linear' or 'Gaussian RBF'
        'Linear'      is for linear regression models, where
                        the optimal kernel is generated by
                        [nu(1)*K1 + nu(1)*K2... ones(size(K1,1),1)]
        'Gaussian RBF' is for regression using Gaussian radial basis
                        functions. The kernel is generated from
                        P1 = nu(1)*K1 + nu(1)*K2 ... ;
                        P2 = repmat(diag(P1) ,1,size(P1,2)) +...
                        repmat(diag(P1)',size(P1,1),1) - 2*P1;
                        Phi = exp([-0.5*P2 ones(size(P1,1),1)]);
w    - weights
alpha - 1/variance for the prior part of the model
beta  - 1/variance for the likelihood part of the model
nu    - parameters that convert the dot-product matrices into
        a kernel matrix (Phi).
ll    - the negative log-likelihood.
```

The first way of calling the routine simply optimises the weights. This involves estimating a restricted maximum likelihood (REML) solution, which maximises $P(\alpha, \beta | t, \Phi)$. Note that REML is also known as Type II Maximum Likelihood (ML-II). The ML-II solution tends towards infinite weights for some the regularisation terms (i.e. $1/\alpha(i)$ approaches 0). The appropriate columns are removed from the model when this happens.

The second way of calling the routine also estimates additional input scale parameters as described in Appendix C of Tipping (2001). This method is much slower, as a full optimisation for the scale

parameters is done after each update of the alphas and beta.

see: <http://research.microsoft.com/mlp/RVM/relevance.htm>

Refs:

The Relevance Vector Machine.

In S. A. Solla, T. K. Leen, and K.-R. Mller (Eds.),
Advances in Neural Information Processing Systems 12,
pp. 652-658. Cambridge, Mass: MIT Press.

Michael E. Tipping

Sparse Bayesian Learning and the Relevance Vector Machine
Journal of Machine Learning Research 1 (2001) 211-244

14.45.11 machines\prt_weights.m

Run function to compute weights

FORMAT weights = prt_weights(d,m)

Inputs:

d - data structure
(fields of .d can vary depending on weights function)
m - machine structure
.function - function to compute weights (string)
.args - function arguments

Output:

weights - weights vector [Nfeatures x 1]

14.45.12 machines\prt_weights_bin linkernel.m

Run function to compute weights for linear kernel binary classifiers

FORMAT weights = prt_weights_bin.linkernel (d,args)

Inputs:

d - data structure
.datamat - data matrix [Nfeatures x Nexamples]
.coeffs - coefficients vector [Nexamples x 1]
args - function arguments (can be empty)

Output:

weights - vector with weights [Nfeatures x 1]

14.45.13 machines\prt_weights_svm_bin.m

Run function to compute weights for binary SVM

FORMAT weights = prt_weights_svm_bin (d,args)

Inputs:

d - data structure
.datamat - data matrix [Nfeatures x Nexamples]
.coeffs - coefficients vector [Nexamples x 1]
args - function arguments (can be left empty)

Output:

weights - vector with weights [Nfeatures x 1]

14.46 utils

14.46.1 utils\prt_centre_kernel.m

This function centres the kernel matrix, respecting the independence of training and test partitions. See Shawe-Taylor and Cristianini for background on this approach.

Shawe-Taylor, J. and Cristianini, N. (2004). Kernel methods for Pattern analysis. Cambridge University Press.

14.46.2 utils\prt_checkAlphaNumUnder.m

check whether a given string is alphanumerical or underscore

FORMAT out = prt_checkAlphaNumUnder(s)

Inputs:

s - a string of arbitrary length to check

Output:

out - logical 1 if the all chars in the string are alphanumerical
logical 0 otherwise

Based on isalpha_num in the identification toolbox

14.46.3 utils\prt_normalise_kernel.m

This function normalises the kernel matrix such that each entry is divided by the product of the std deviations, i.e.

$K_{\text{new}}(x,y) = K(x,y) / \sqrt{\text{var}(x)*\text{var}(y)}$

Part V

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] Christopher M. Bishop. *Pattern Recognition and Machine learning*. Springer, 2006.
- [3] Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.
- [4] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.
- [5] Volkmar Glauche. MATLAB batch system. <http://sourceforge.net/projects/matlabbatch/>.
- [6] J. D. Haynes and G. Rees. Decoding mental states from brain activity in humans. *Nat. Rev. Neurosci.*, 7:523–534, 2006.
- [7] Thomas Hofmann, Bernhard Scholkopf, and Alexander J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220, 2008.
- [8] A. Marquand, M. Howard, M. Brammer, C. Chu, S. Coen, and J. Mourao-Miranda. Quantitative prediction of subjective pain intensity from whole-brain fMRI data using Gaussian processes. *Neuroimage*, 49:2178–2189, 2010.
- [9] Members and collaborators of the Wellcome Trust Centre for Neuroimaging. Statistical Parametric Mapping, SPM8. <http://www.fil.ion.ucl.ac.uk/spm>, 2008. Wellcome Trust Centre for Neuroimaging, University College London, UK.
- [10] Janaina Mourao-Miranda, Karl J Friston, and Michael Brammer. Dynamic discrimination analysis: a spatial-temporal svm. *Neuroimage*, 36(1):88–99, 2007.
- [11] K. A. Norman, S. M. Polyn, G. J. Detre, and J. V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fMRI data. *Trends Cogn. Sci. (Regul. Ed.)*, 10:424–430, 2006.
- [12] F. Pereira, T. Mitchell, and M. Botvinick. Machine learning classifiers and fMRI: a tutorial overview. *Neuroimage*, 45:199–209, 2009.
- [13] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. the MIT Press, 2006.
- [14] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
- [15] Michael E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.