Universitat Politècnica de Catalunya

Bachelor's Degree Final Project

Image Processing Group

Analysis of the dynamics of gray matter reduction in Alzheimer's Disease

Author:
Asier Aduriz Berasategi

Director:
Verónica Vilaplana Besler

May 13, 2016

Abstract

This project attempts to study the cerebral atrophy patterns in gray matter across the different stages of the Alzheimer's Disease (AD), or more specifically, along the entire AD continuum, in a voxelwise approach. To this end, we propose and implement an extensible toolbox that allows to fit different models to the data, hence defining a curve for each voxel that shows the evolution of the gray matter volume in the respective region as compared to the progression of the disease. The toolbox also includes several evaluation methods to estimate how closely the proposed model fits the data for each particular voxel. The resulting values, namely *fitting-scores*, serve as a base to achieve two different goals: **a)** to identify the regions within the brain that are (most) likely to follow the curve-shape specified in a given model, and **b)** to depict the model that best describes the behavior of the gray matter volume in each voxel from a fixed set of models.

Este proyecto trata de estudiar los patrones de atrofia cerebral de materia gris en las diferentes etapas de la enfermedad de Alzheimer (AD), y más concretamente, a lo largo del contínuo del alzhéimer, llevando a cabo un análisis a nivel de vóxel. Con este objetivo, proponemos e implementamos un paquete de herramientas extensible que permite ajustar diferentes modelos a los datos, definiendo así una curva para cada vóxel que muestra la evolución del volumen de materia gris en la respectiva región con respecto al progreso de la enfermedad. El paquete también incluye varios métodos de evaluación para estimar la exactitud con la que el modelo propuesto encaja con los datos para cada vóxel particular. Los valores resultantes, denominados fitting-scores (calificaciones de ajuste), sirven como base para obtener dos objetivos diferentes: ***a)*** *identificar las regiones del cerebro que (más) probablemente siguen la forma de las curvas especificadas en un modelo dado, y* ***b)*** *seleccionar el modelo que mejor describe las pautas de la materia gris en cada voxel de un conjunto prefijado de modelos.*

Aquest projecte tracta d'estudiar els patrons d'atrofia cerebral de matèria grisa en les diferentes etapes de la malaltia d'Alzheimer (AD), i més concretament, al llarg del continu de l'alzheimer, duent a terme una anàlisi a nivell de vòxel. Amb aquest objectiu, proposem i implementem un paquet d'eines extensible que permet ajustar diferents models a les dades, definint així una corba per a cada vòxel que mostra l'evolució del volum de matèria grisa en la respectiva regió a mesura que la malaltia progressa. El paquet també inclou diversos mètodes d'avaluació per tal d'estimar l'exactitud amb la qual el model proposat encaixa amb les dades per a cada vòxel particular. Els valors resultants , anomenats fitting-scores (qualificacions d'ajust), serveixen com a base per obtenir dos objectius diferents: ***a)*** *identificar les regions del cervell que (més) probablement segueixen la forma de les corbes especificades en un model donat, i* ***b)*** *seleccionar el model que millor descriu les pautes de la matèria grisa en cada vòxel d'un conjunt prefixat de models.*

Contents

Chapter 1

Context of the project

1.1 Introduction

This project is the result of the collaboration between the *Image Processing Group (GPI)*[1] at the *Technical University of Catalonia (UPC)*[2] and the *Pasqual Maragall Foundation (FPM)*[3]. The main objective is to build a toolbox that helps researchers at the aforementioned foundation, and in the neuroimage community as a whole, to better understand how the gray matter volume across the brain of a patient evolves as they develop Alzheimer's Disease, and hopefully contribute to discover more effective treatments for such disease and/or its symptoms.

1.2 Alzheimer's Disease

Alzheimer's disease (AD) is a chronic neurodegenerative disorder that is characterized by progressive neuropathology and cognitive decline. According to the *Alzheimer's Association*[4], it currently affects more than 36 million people in the world, accounting for 60% to 80% of dementia cases. Patients suffering from such pathology experience various symptoms, among which memory loss and reduction of other mental abilities can be found, that worsen over time, getting severe enough to interfere with their daily lives.

Unfortunately, even though treatments that slow down the progress of the mentioned symptoms are available and a worldwide effort is being put on finding better ways to fight the disease, as for today no cure for AD is known to mankind.

[1] Official website: *https://imatge.upc.edu/*
[2] Official website: *http://www.upc.edu/*
[3] Official website: *https://fpmaragall.org/*
[4] Official website: *http://www.alz.org/*

On the other hand, it is well known that a patient's brain suffers changes during the earliest stages of the disease and long before showing any clinical symptoms. For that reason, researchers focus their efforts towards defining which changes occur and where they take place, with the goal of detecting indicators to predict the development of the disease.

As explained in [Gispert et al., 2015], the characteristic progressive memory impairment in AD is related to a pattern of atrophy in the volume of gray matter in various regions of the brain. Along with this, cerebrospinal fluid (CSF) concentrations of β-amyloid 1-42 (Aβ42) and total tau (t-tau) proteins, among others, have shown to serve as in vivo proxy measures of the central neuropathological hallmarks of AD ([Braak et al., 2013]). Therefore, studying the relationship between CSF biomarkers and regional changes on the brain's structure may contribute to understanding the pathological mechanisms of the disease.

1.3 AD-CSF Index

To be able to analyze and represent the evolution of gray matter volume as the Alzheimer's Disease progresses, a concept that is based in biological measurements and captures such progression as a continuous incremental value must be introduced. The AD-CSF index presented in [Molinuevo et al., 2013] is an indicator composed by the sum of the normalized CSF levels of Aβ42 and t-tau proteins that reflects the degree of the pathology, therefore determining where the patient is along the AD continuum.

This index has shown higher diagnostic capability than the individual biomarkers and other progression indices in a multicenter validation study ([Molinuevo et al., 2013]) and in autopsy-confirmed patients with AD ([Struyfs et al., 2014]). However, the process to obtain the data necessary to compute such index is rather expensive and highly intrusive (even painful) for the patient. As a result, there is a strong interest in understanding the dynamics of other more affordable and less intrusive biomarkers, such as the brain region connectivity or the gray matter volume, and their potential to predict the patient's state along the AD continuum.

1.4 State of the art

Several approaches have been used to establish the association between CSF biomarkers and brain atrophy. The most frequently used method relies on linear models that assume a uniform association of both variables across the whole AD spectrum, i.e., regardless of whether these reach pathological values or not ([Insel et al., 2014]). Another approach consists of grouping the range of CSF measurements into 'positive' and 'negative' categories using thresholds that are generally derived from their diagnostic capacity

([Molinuevo et al., 2013]). The latter method generally assumes constant atrophy rates in each group with no transition between them.

Although these models allow for a simple interpretation, they are unlikely to represent the relationship between CSF biomarkers and brain atrophy realistically ([Insel et al., 2014]). Moreover, there is prior evidence that the trajectories of CSF biomarkers and cortical atrophy are nonlinear ([Sabuncu et al., 2011]) and affected by interactions with age and other complex factors ([Jack Jr et al., 2012]). Therefore, linear models are not the best suited to describe the associations between biomarkers known to have nonlinear dynamics ([Jack Jr et al., 2013]).

As regards nonlinear methods, several studies have used them to model the relationship between biomarkers and neuroimaging-derived measurements. For example, polynomial regression was used in [Fjell et al., 2010] to model age-related atrophy in relation to CSF Aβ levels using a quadratic term. Other methods do not need to explicitly model the parametric form of the association: generalized additive models (GAM) have been used to model the effect of aging ([Schuff et al., 2012]), whereas splines was used to model the relationship between brain atrophy and CSF Aβ levels in [Insel et al., 2014], and local regression methods were employed to track the evolution of several biomarkers in dominantly inherited AD as a function of the estimated years from expected symptom onset in [Bateman et al., 2012].

Finally, a polynomial approach was used in [Gispert et al., 2015] to model the dynamics of gray matter reduction associated to progression through the AD biological continuum, reporting significant nonlinear dependencies in specific memory-related areas of the brain as a result of the analysis.

Chapter 2

Proposed approach

2.1 Definition of the main objective

Now that we have introduced a few key concepts, let us define the goal of the current project as follows: we will aim at developing an extensible toolbox in *python* that allows to fit arbitrary models that may explain how the gray matter data relates to the AD-CSF index -defining thus the gray matter volume as a function of the stage of the disease–, and assess the results by means of an arbitrary evaluation function. The procedure will be almost completely based in a voxelwise approach, and we will mainly focus on analyzing various nonlinear fittings.

2.2 Data

We will study the processes of the brain all along the disease's stages using images obtained through different MRI techniques, provided to us by the Pasqual Maragall Foundation (FPM). These images have already been treated and normalized by the FPM to fit a reference template such that the stored volumetric data represents the amount (percentage) of gray matter for each voxel (3-dimensional pixel corresponding to a cube of 1.5mm x 1.5mm x 1.5mm) of the brain. The exact procedure followed to obtain such volumetric data is described in more detail in [Gispert et al., 2015]. Each of the 129 subjects in the study has also been diagnosed to be in one of 4 classes: NC (control, no indicators of the disease), PC (pre-clinical, some indicators but without cognitive decline), MCI (mild cognitive impairment, with a slight cognitive decline), and AD (Alzheimer's disease, already having the disease). Furthermore, information regarding other biomarkers is also available for each subject, including but not limited to age, sex, AD-CSF index value, APOE4 protein level, and education level. A detailed list of available measurements and their description is attached in appendix A.

2.3 Hypotheses

Given the lack of data as regards the behavior of the aforementioned factors over time, some assumptions must be made in order to obtain statistically significant results. These are listed below:

- The AD-CSF index is a good indicator of the stage of the Alzheimer's Disease regardless of such stage.
- The gray matter volume in each voxel follows well-defined patterns along the AD-continuum described by the AD-CSF index.
- The distribution of the template-normalized and sex- and age-regressed gray matter volume values is the same for control patients that will never develop the disease and those that will evolve into PC, MCI, and/or AD patients.
- As a generalization of the previous point, the distribution of the aforementioned gray matter volumes for a patient that is in a certain stage in the AD-continuum (described by the AD-CSF index) is the same regardless of the degree to which such patient will develop the disease in the future.
- There are enough subjects for each interval in the AD-continuum for the results to be statistically significant, i.e., the sample of the population is large enough to correct the influences of the outliers and yield sound statistical results.

2.4 Workflow

Figure 2.1 shows the execution pipeline for which the toolbox has been designed.

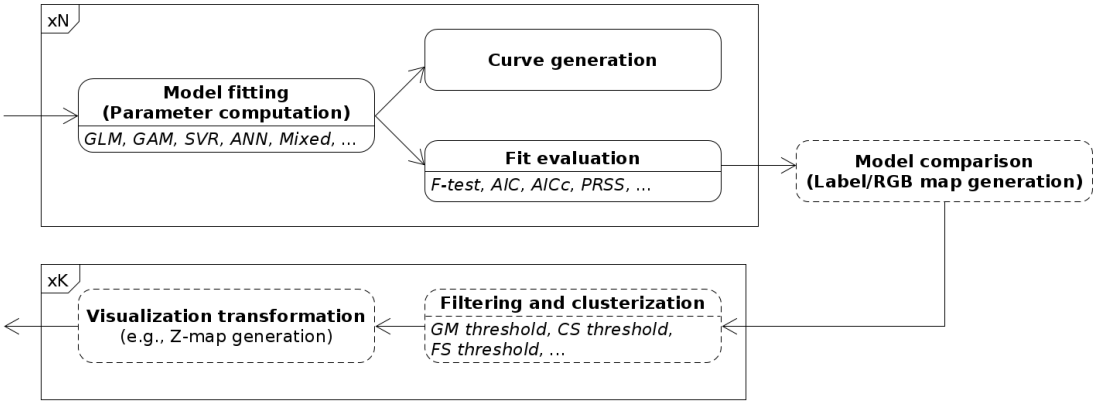

Figure 2.1: General execution pipeline.

In it, the first block is fed with the information of all 129 subjects, as described in section 2.2. Then, N models are selected, and for each of them their parameters are set so that they fit the data optimally according to an arbitrary criterion defined at runtime. Once computed, the results are either displayed in the form of curves (one for each voxel and model), or evaluated with an arbitrary evaluation function.

In the latter case, N 3-dimensional maps are generated, one for each model, in which each voxel describes how well the selected model fits the information obtained from the subjects. This way, each of the generated maps can be used to detect the regions of the brain where the data behaves as hypothesized by the respective model. Another option is to compare the N models against each other to generate new maps (e.g., by selecting the maximum value and the model to which this corresponds for each voxel), in order to decide which ones are most likely to explain the behavior of the data, and to what extent they are able to do so.

In any case, we may have the need to correct possible outlier voxels and treat the final results so that they can be appropriately visualized. To this end, we include two additional blocks in the pipeline, that can be iterated an arbitrary number K of times with different thresholds and/or visualization transformations.

The generated maps are stored in the form of NIfTI-1 image files, and can be visualized using external tools, such as FSLView[1]. On the other hand, given the amount of space that it would require to explicitly store the values of each curve for each voxel and model, we decided to integrate a method that would interactively generate and return such values for the specified voxels so that they can be easily plotted (or even stored, if wanted) with a few lines of code.

2.4.1 Model fitting block

The first block of the pipeline constitutes the core of the toolbox. In it, extensibility is key to ensure that anyone who wants to contribute to the project or just use it for their own experiments can do it without too much effort. As a consequence, this block must be general enough to include as many scenarios as possible, but simple enough to be easily understandable and usable. We propose a workflow in which two separate main blocks are present: in the first stage, the effect of the covariates, i.e., independent variables that are not of interest but may affect the observations, is subtracted from the original observations, therefore *correcting* the data and only leaving the behavior that is due to the variables of interest and random or unexplained causes; the second stage tries to predict such behavior by using the variables of interest. For this reason, in the scope of this project the covariates are also called *correctors*, whereas the variables of interest are often referred to as *predictors*.

[1] http://fsl.fmrib.ox.ac.uk/fsl/fslview/

Figure 2.2 depicts the complete process, in which it can be observed that the correction parameters and the prediction parameters are optimized separately, and for two potentially different models.

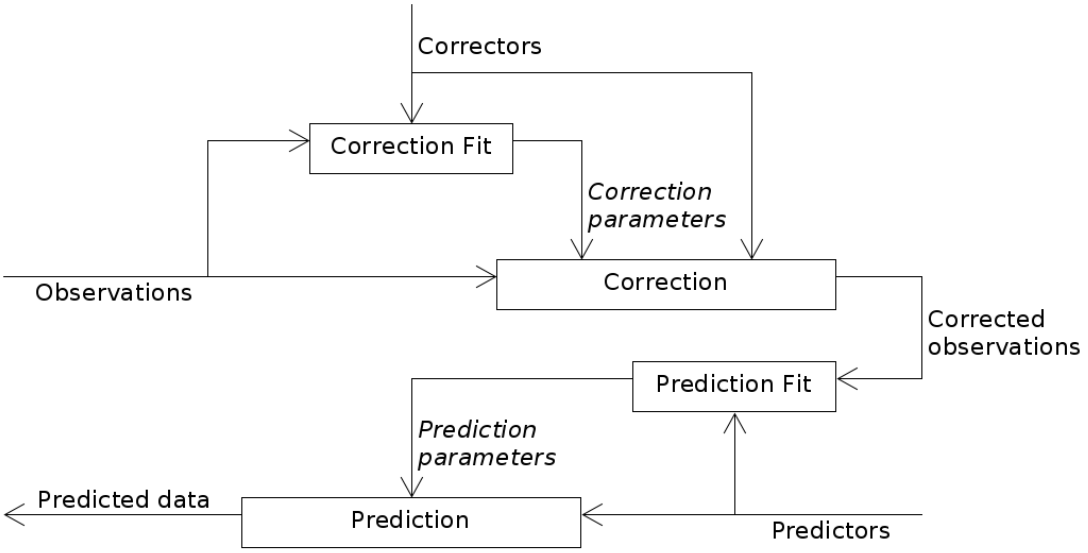

Figure 2.2: Execution pipeline for model fitting.

Although the toolbox has been designed to support this rather general scheme, in the context of this projects some restrictions will be applied:

- Since our goal is to analyze the dynamics of gray matter along the AD continuum, our observations will always correspond to gray matter volume values, and we will only have one predictor, corresponding to the value of the AD-CSF index.
- For the sake of simplicity, we will consider that the correction for any fitter is always of the following form:

$$Correction\,(Y, C, CP) = Y - Prediction\,(C, CP) \tag{2.1}$$

  where Y is the observations variable, C is the set of correctors, and CP is the set of correction parameters.
- We will also want the results to be comparable among themselves and to the ones presented in [Gispert et al., 2015]. To this end, we will always correct the data with the General Linear Model fitter (section 2.6.1). The correctors will also be the same in all experiments, that is, the sex of the subject and both, the linear and the quadratic terms of their age.

2.4.2 Fit evaluation block

The fit evaluation block is an important element of the pipeline as well, since it allows to easily assess the quality of the fit for each model and voxel. In general, a fit evaluation method may not be suitable for all the models in the system, as it may assume some conditions that are not always fulfilled. However, various methods should still be available to allow for different experiments to be tested against the data.

On the other hand, if the model comparison block is to be used in an experiment, employing the same method to test all the input models is recommended, given that results yielded by different tests might not be comparable neither in range, nor in statistical properties.

2.5 The toolbox

We present a system that is composed of an independent *fitting library*, which comprises the *curve fitting* classes and the *fit evaluation* methods, and a *processing* block that interacts with this library by feeding it the adapted data from the database (see fig 2.3).

To deal with the rather large amount of data corresponding to the gray matter volume maps (there are >2M voxels/subject, and >270M voxels in total), a *data hub* has also been incorporated to the system.

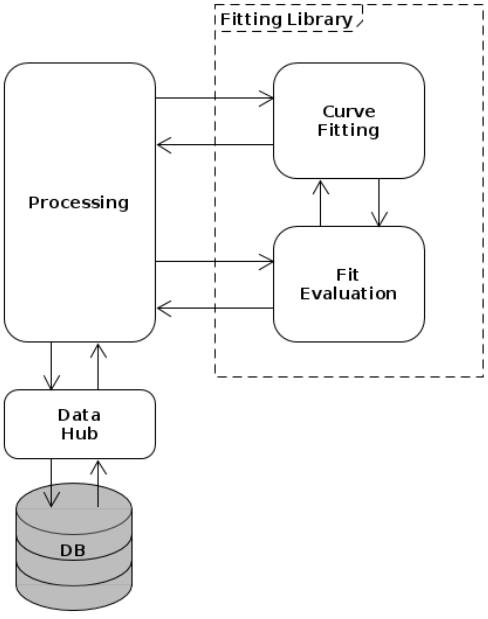

Figure 2.3: General implementation scheme.

The whole framework is built over the NumPy[2] package and its structures, but additional libraries are also imported whenever needed inside each module.

2.5.1 Data hub

The data hub block contains three modules, called *niftiIO*, *Subject* and *Database*.

The *niftiIO* module is a tool that facilitates the management of potentially heavy NIfTI-1 image files by using the Nibabel[3] library.

The *Subject* module makes use of the *niftiIO* module to implement an auxiliary method that allows to load the gray matter volumes chunk by chunk, specifying the maximum size of each chunk (in MB), and also defines the *Subject* class, which contains all the attributes necessary to represent the different measurements available in the database for each subject.

Finally, the *Database* module is the responsible of accessing the *xls*-formatted file in which most of the measurements are, in order to initialize a list of Subjects to work with and link each subject to the NIfTI-1 image file of the file system that contains their gray matter volume data.

This way, the user only must edit the *user_paths.py* file to introduce the paths of both, the *xls* file and the directory that contains all the gray matter data files in their environment, and call the *get_data* method in the *Database* module to load the subjects and the *chunks* method in the *Subject* module to go through the gray matter data chunk by chunk when needed.

2.5.2 Processing

The processing block is designed so that one *processor* exists for each *fitter* in the fitting library. We consider that all processors should have some specific functions, such as a *process* method that would query the computation of the correction and prediction parameters for all the voxels in a region delimited by two 3-dimensional points, or a *curve* method that would return the values of the prediction function for t_{points} equally distributed points between t_1 and t_2 for the same region.

Hence, an abstract *Processor* class has been implemented with such methods and other useful tools, from which all processors shall inherit and override the required model-dependent methods to connect each particular fitter and make it usable with the same interface. This idea is portrayed in the simplified UML diagram in appendix B.

[2] http://www.numpy.org/
[3] http://nipy.org/nibabel/

A detailed description of each public method of the *Processor* class, with its arguments and return values is given below (arguments surrounded by brackets are optional):

- ***subjects*** : List of Subject instances representing the subjects of the current Processor instance.
- ***correctors*** : NxC (2-dimensional) matrix, representing the values of the features of the subjects that are to be used as predictors in the fitter, where N is the number of subjects and C the number of correctors.
- ***predictors*** : NxP (2-dimensional) matrix, representing the values of the features of the subjects that are to be used as predictors in the fitter, where N is the number of subjects and P the number of predictors.
- ***progress*** : float, indicating the progress (percentage of data processed) of the last call to *process*. If it has not been called yet, this will be 0.0, whereas if the task is already completed it will be 100.0. Useful if the *process* method is called inside a thread/process and another thread/process wants to know the state of the call, e.g. in a GUI.
- ***process****([(x1:x2, y1:y2, z1:z2) : Region], [mem_usage : float])* : processes the correction and prediction parameters for the specified regions, loading chunks of at most mem_usage MB each time, and returns them in the form of a *Processor.Results* instance.
- ***user_defined_parameters*** : tuple of floats that encodes the options selected by the user. This is useful to reproduce the curves or compute the evaluation of the fittings in a later execution, by saving the return value and using it to initialize the *Processor* instance in the next execution.
- ***corrected_values****(correction_parameters : array-like, [(x1:x2, y1:y2, z1:z2) : Region], [(origx, origy, origz) : Point3D], [mem_usage : float])* : given the correction parameters, an *origin* point from which the correction parameters were computed, and a region specified by coordinates relative to the origin point, loads and corrects the data of such region in chunks of at most mem_usage MB, and returns the corrected values in the form of an numpy array.
- ***curve****(prediction_parameters : array-like, [(x1:x2, y1:y2, z1:z2) : Region], [(t1, t2, tpoints) : Timeline])* : given the prediction parameters, a region specified by coordinates relative to the origin point from which the prediction parameters were computed, and a timeline specified by its two extremes and the number of points between them, returns a list of length *tpoints* that contains the uniformly distributed values between *t1* and *t2* and an array-like structure containing the values of the curve in such points for each voxel.
- static ***evaluate_fit****(evaluation_function : function, correction_processor : Processor, correction_parameters : array-like, prediction_processor : Processor, prediction_parameters : array-like, [(x1:x2, y1:y2, z1:z2) : Region], [gm_threshold : float],*

[filter_nans : bool], [default_value : float], [mem_usage : float]) : returns an array-like structure that contains the score obtained by the fitting for each voxel in the specified region, ensuring that the data is loaded by chunks that do not exceed mem_usage MB in each step. Such score is computed by applying the evaluation function to the correction and prediction processors with the correction and prediction parameters specified by the arguments. The results are filtered by setting each voxel to the default value if any of the following conditions are fulfilled: the mean (over the subjects) gray matter volume in such voxel does not reach the specified GM threshold, or the result of the evaluation is not a number and the *filter_nans* flag is set.

- static ***cluster****(fitting_scores : array-like, [default_value : float], [fit_lower_threshold : float], [fit_upper_threshold : float], [cluster_threshold : int], [produce_labels : bool])* : returns the fitting thresholds after filtering them as specified by the upper and lower thresholds at the voxel level, clustering them and removing any clusters below the cluster threshold. Any voxel that is not in the allowed range and does not pertain to a cluster of sufficient size is set to the default value specified in the arguments. Finally, if the *produce_labels* flag is set, a second map that indicates the cluster to which it voxel pertains is also returned.

Notice that, since the aforelisted methods are generic and applicable to any processor, they must internally call some 'private' methods that are model-specific. For this reason, these other methods have been specified as abstract functions and must consequently be overridden in the subclasses to ensure their correct functioning. Additionally, some other helper functions that might be useful to facilitate the creation of new *processors* have been included, such as the *__get*__* methods that make the interaction with the user transparent to the subclasses (see below):

- abstract ***__fitter__****(user_defined_parameters : tuple)* : given the parameters obtained either by calling *__read_user_defined_parameters__* in this same execution or by loading them from a file in which they were stored as returned by *__user_defined_parameters__* in a previous execution, this method returns a fully initialized instance of a subclass of *CurveFitter*. This method is always called in the initialization of a *Processor* instance.

- abstract ***__user_defined_parameters__****(fitter : CurveFitter)* : given the fitter obtained from the *__fitter__* method, returns a tuple of floats representing the user defined parameters that would be required in such method to initialize an identical fitter from scratch.

- abstract ***__read_user_defined_parameters__****(predictor_names : iterable<Subject.Attribute>, corrector_names : iterable<Subject.Attribute>)* : reads the parameters that are necessary to successfully initialize a new instance of a fitter from the user, and returns a tuple of floats that encode such parameters. The *__fitter__* method is then called with this tuple 'as is' as argument. The *__get*__* functions should be used here together with the built-in *super* function to ensure that the user interface is coherent

and benefits from all the future improvements.

- ***__curve__**(fitter : CurveFitter, predictor : list<float>, prediction_parameters : array-like)* : given a fully initialized fitter, a list of floats (the *axis*) and the prediction parameters of each voxel, returns the value of the curve evaluated in each point of the axis for each voxel. This is not an abstract method as it calls the *predict* function of the fitter with the prediction parameters and the predictors by default, but this behavior should be overridden if it does not yield the desired output.
- ***__corrected_values__**(fitter : CurveFitter, observations : array-like, correction_parameters : array-like)* : given a fully initialized fitter, a map of observations and a map of correction parameters, returns an array-like structure containing the corrected values of the observations. This is not an abstract method because it calls the *correct* function of the fitter with the observations and the correction parameters by default, but this behavior should be overridden if it does not yield the desired output.
- ***__getint__**(...)*, ***__getfloat__**(...)*, ***__getoneof__**(...)*, ***__getoneinrange__**(...)*, ***__getyesorno__**(...)*, ... : helper methods to obtain input of different types from the user in execution time. Each of these functions is based on the *__processor_get__* static method, and has several optional parameters to personalize the data retrieval with a default value, the number of trials, the range of acceptable values, the text to show to request the data, etc.
- static ***__processor_get__**([obtain_input_from], [apply_function], [try_ntimes], [default_value], [show_text], [show_error_text])* : base function to interact with the user for input requests. The data retrieval will be obtained from the specified source (for instance, the command line or a GUI component) by showing the indicated text. Then a function will be applied to the obtained value, and in case the process raises any errors the data retrieval will be retried as many times as specified in the arguments, showing the error text in each case. If the input could not be correctly obtained, an error will be raised. The default value will be returned if the user does not fill the corresponding field. All parameters are optional.

Having defined this class as described allows to modify the base class as desired (for instance, by adding a GUI) and automatically obtain the benefits of the modification on all its subclasses by just applying inheritance properties.

2.5.3 Curve Fitting

In the context of this project, a fitter is an algorithm that, given a *target variable*, one or more *variables of interest* that might explain the behavior of such target and possibly some *covariates* (variables that are not of interest) that may also contribute to such behavior, all of them potentially random, adjusts the *set of parameters* of a parametric function so

that once accounted for the contribution of the covariates, the function with the variables of interest as input fits the observations of the target variable in the best possible manner according to a certain *quality measure*.

In other words, a fitter is an algorithm that, given a parametric function f (the prediction function) with parameters p_1 to p_k, a quality-measure Q, a dependent variable y whose behavior is unknown and we want to predict (the *target*), a set of variables x_1 to x_n that might explain the behavior of y (the *variables of interest/predictors*), and another set of variables c_1 to c_m whose contribution to y we want to remove (the *covariates/correctors*), computes the following expression:

$$\hat{p}_1, \ldots, \hat{p}_k = arg_{p_1,\ldots,p_k} max \left[Q \left(f_{p_1,\ldots,p_k} \left(x_1, \ldots, x_n \right), g \left(y, c_1, \ldots, c_m \right) \right) \right], \tag{2.2}$$

where g denotes a function that *corrects* the target variable so that the information contributed by the covariates is removed from it.

Given that the variables are potentially random and we might be unaware of their statistical properties, their behavior will be estimated through a number of measurements obtained as follows:

- The target variable y will be observed N times, having thus a vector y_{Nx1} of N observations that we want to explain. It is important to notice that such observations can be sampled in a completely arbitrary way from any kind of set, e.g., moments in time, points in space, samples of a population, etc.
- Both, the variables of interest and the covariates will be observed at the same times (with the same samples) as those of the target variable and gathered in two matrices, X_{Nxn} and C_{Nxm}, respectively, such that the i^{th} column of X_{Nxn} contains the N observations of the i^{th} variable of interest, that is, x_i, and the j^{th} column of C_{Nxm} contains the N observations of the j^{th} covariate, c_j.

Since all the fitters are oriented to obtaining the optimal parameters for a specific function f (a.k.a. *model*), we will name each fitter with the name of such function.

As in the *Processing* block, we define an abstract *CurveFitter* class that will implement the methods that are potentially common to every fitter. Again, the simplified UML diagram of such class and its connection with the *Processing* block can be observed in appendix B. The details of such methods are given below:

- ***correctors*** : NxC (2-dimensional) matrix, representing the correctors of the model, where N is the number of observations and C the number of correctors.
- ***predictors*** : NxP (2-dimensional) matrix, representing the predictors of the model, where N is the number of observations and P the number of predictors.
- ***features*** : Nx(C+P) (2-dimensional) matrix, representing the features (correctors and predictors) of the model (see the *correctors* and *predictors* attributes).

- ***correction_parameters*** : Array-like structure of shape (K_c, X_1, ..., X_n), representing the correction parameters for which the correctors best explain the observational data passed as argument in the last call to *fit*, where K_c is the number of parameters for each variable in such observations, and X_1, ..., X_n are the dimensions of the *observations* argument in the last call to *fit* (there are $X_1 * ... * X_n$ target variables).
- ***prediction_parameters*** : Array-like structure of shape (K_p, X_1, ..., X_n), representing the prediction parameters for which the predictors best explain the observational data passed as argument in the last call to *fit*, where K_p is the number of parameters for each variable in such observations, and X_1, ..., X_n are the dimensions of the *observations* argument in the last call to *fit* (there are $X_1 * ... * X_n$ target variables).
- ***orthogonalize_correctors()***, ***normalize_correctors()***, ***orthonormalize_correctors()***, ***orthogonalize_predictors()***, ***normalize_predictors()***, ***orthonormalize_predictors()***, ***orthogonalize_all()***, ***normalize_all()***, ***orthonormalize_all()*** : Orthogonalizes/Normalizes/Orthonormalizes each corrector/predictor with respect to all the previous features applying the Gram-Schmidt algorithm[4] and returns a *deorthonormalization matrix* that right-multiplied with the new, orthonormalized feature matrix yields the original feature matrix.
- ***fit****(observations : array-like)* : Computes the correction and prediction parameters that best fit the observations. The results are accessible by calling the *correction_parameters* and *prediction_parameters* attributes of the class. This method returns a reference to self, i.e., the CurveFitter instance in which it is called.
- ***correct****(observations : array-like, [correctors : 2D array-like (matrix)], [correction_parameters : array-like])* : Returns an array-like structure of the same shape as the *observations* argument, containing the values of the data after accounting for the correctors by using the correction parameters. Any optional arguments that is omitted is replaced with the corresponding structures stored as internal attributes. If the shape of the *observations* is $(N, X_1, ..., X_n)$, then the *correctors* must be a NxC matrix and the *correction parameters* must be of shape $(K_c, X_1, ..., X_n)$, where N is the number of observations for each variable, C is the number of correctors, $M = X_1 * ... * X_n$ is the number of target variables and K_c is the number of correction parameters for each target variable.
- ***predict****([predictors : 2D array-like (matrix)], [prediction_parameters : array-like])* : Returns an array-like structure of shape $(N, X_1, ..., X_n)$ that contains the prediction computed by using the predictors together with the prediction parameters. The *predictors* argument must be a NxP (2-dimensional) matrix, whereas the *prediction_parameters* argument must be of shape $(K_p, X_1, ..., X_n)$, where N is the number of observations for each variable, P is the number of predictors, $M = X_1 * ... * X_n$ is the number of target variables and K_p is the number of prediction parameters for each variable. If any of the arguments is omitted, it is replaced by the corresponding

[4] https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

structure in the internal attributes of the instance.

Analogously to the *Processor* class, here too the generic methods must call abstract, model-specific functions (implemented in the subclasses) to be able to work correctly. Those functions do not need to check for generic shape incompatibilities among the arguments, since these are already inspected in the corresponding generic methods of the *CurveFitter* class. The list below describes the methods that must be implemented in the subclasses[5]:

- abstract static *__**fit**__(correctors : 2D array-like (matrix), predictors : 2D array-like (matrix), observations : 2D array-like (matrix))* : Computes and returns the correction and prediction parameters that optimally fit the features (correctors and predictors) to the observations. The *correctors* argument is a NxC matrix, whereas *predictors* is a NxP matrix and *observations* is a NxM matrix. The result should be a tuple of two elements, in which the first element is the K_cxM matrix containing the correction parameters and the second element is the K_rxM matrix of prediction parameters. M is the number of target variables, N is the number of observations for each variable, C the number of correctors, P the number of predictors, K_c the number of correction parameters for each target variable and K_p the number of prediction parameters for each target variable.
- abstract static *__**correct**__(observations : 2D array-like (matrix), correctors : 2D array-like (matrix), correction_parameters : 2D array-like (matrix))* : Computes and returns the values of the observations after accounting for the correctors by using the correction parameters. Here, *observations* is a NxM matrix representing the observational data to be corrected, *correctors* is a NxC matrix that contains the covariates (features in which we are not interested), and *correction_parameters* is a K_cxM matrix that stores the parameters that best fit the correctors to the observations for each variable. M is the number of target variables, N is the number of observations for each variable, C the number of correctors, and K_c the number of correction parameters for each target variable.
- abstract static *__**predict**__(predictors : 2D array-like (matrix), prediction_parameters : 2D array-like (matrix))* : Computes and returns a prediction made using the predictors together with the prediction parameters. The *predictors* argument is a NxP matrix containing the features to be used to try to explain/predict the corrected observational data, whereas *prediction_parameters* is a K_pxM matrix that stores the parameters that best fit the predictors to such data for each variable. M is the number of target variables, N is the number of observations for each variable, P is the number of predictors and K_p the number of prediction parameters for each target variable.

[5]Here the *static* keyword in means that a method is allowed to be implemented as a static function in the subclass

A *MixedFitter* metaclass has also been defined to allow for the creation of fitters that correct the data following a given pattern and predict the corrected data applying a different model[6].

Finally, a subclass that defines the *correct* method as stated in equation 2.1 (see section 2.4.1), namely *AdditiveCurveFitter*, has also been implemented to be used in the scope of this project.

2.5.4 Fit evaluation

The fit evaluation block is composed of various functions that may have nothing in common. For this reason, there is not an abstract *FitEvaluator* class from which all fit evaluation methods inherit, but rather a set of functions that define the properties of each of them. These functions are:

- ***requires****(evaluation_method : function, target_method_name : string, target_method_description : string)* : sets a dependence on a target method (with its name and description) for the evaluation method.
- ***default****(evaluation_method : function, target_method_name : string, target_method_description : string, default_method : function)* : indicates that *default_routine* is being used in the evaluation method in place of the target method, but this could be overridden if wanted.
- ***bind****(fitter : CurveFitter, evaluation_method : function, target_method_name : string, target_method : function)*: binds the target method to the evaluation method, designating that this should be used whenever a call to *target_method_name* is executed when evaluating the indicated fitter.

Note: This is NOT the current implementation of the methods, since implementing this block as described above would require very advanced python skills and a huge amount of time and testing to ensure its correctness and liability. Instead, the required methods have been implemented in the CurveFitter class, raising a NotImplementedError by default, and are called from each evaluation method without checking for errors. This way, if an evaluation method is not compatible with a particular fitter, an error will be raised when the user tries to combine them.

2.5.5 Model comparison

We have implemented two paradigms of comparison among different models, namely *RGB* and *BEST*.

[6] Although the implementation of this class is still incomplete, a first version has already been created.

In the *RGB* mode, the comparison is conducted by generating a 3-dimensional image of 3 channels (i.e., each voxel would comprise 3 elements) in which each channel corresponds to the results of one model, therefore making it possible to visualize the fitting scores of each model as one of the red, green or blue components of the color for each voxel. This idea could possibly be generalized to support other color spaces (e.g., HSV), or even more colors -by keeping 3 *RGB* channels in each voxel, but combining (adding up) an arbitrary number K of *distant/complementary* colors[7] (one for each model) weighted by the value of the corresponding model in such voxel–. However, this would result in much less intuitive maps being generated, which would lead to more difficulties to interpret the results.

In the *BEST* mode, on the other hand, two maps are generated: *BestFitScores* and *BestLabels*. *BestFitScores* is computed by selecting the best fitting score over the different models for each voxel, whereas *BestLabels* stores the index of the model to which such score corresponds. Please, notice that the 'best' fitting score must be decided according to an appropriate criterion for each evaluation method. As an example, the Akaike Information Criterion (section 2.7.4) outputs lower values for better models, whereas the R-squared method (see section 2.7.2) assigns a higher value to more preferable models. This method creates a map of best fits together with a mask that can be colorized so that each color represents one model. This way, when superposing the grayscale image corresponding to the *BestFitScores* map with a colorized mask based on the *BestLabels* map, we obtain a visual representation in which it is easy to see which model explains the data best and to what degree it is able to do so.

Table 2.1 summarizes the advantages and disadvantages of each paradigm:

Paradigm	**Advantages**	**Disadvantages**
RGB	- Easy to visually compare how better a model is as compared to the others. - Goodness of fit correlated with brightness of colors. - Information of all models in same picture simultaneously.	- Only up to 3 models supported for each comparison.
BEST	- Best model visually clear. - Goodness of fit also clearly visible. - Suitable for virtually as many models as wanted.	- No way to compare how better a fit is as compared to the others (for a particular voxel). - Boundaries between dominant models prone to errors.

Table 2.1: Advantages and disadvantages of model comparison paradigms

Both methods have been implemented as separate scripts.

[7]For instance, a possible selection for $K = 4$ would be orange, cyan, purple and yellow

2.5.6 Filtering and clustering

The filtering and clustering performed through calls to the *cluster* function in *Processor* uses a module named *graphlib*, that has been implemented to support a few useful graph-based methods. The exception to this is the gray-matter filtering, whose implementation is in the *evaluate_fit* function of the *Processor* class (refer to section 2.5.2 for more information on this class). This code 'drops' the voxels that do not have a large enough mean gray volume right after the evaluation, meaning that the returned fitting-scores have already been filtered based on the mentioned criterion.

The graphlib module contains the definition of an abstract class called *Graph*, that only requires the implementation of two methods to become concrete: *nodes()*, which yields (generates) the nodes of the graph one by one, and *neighbors(node)*, that given a node yields its neighbors. This class then automatically adds the *sccs* method, that detects the *Strongly Connected Components*[8] of the graph using an iterative version of Tarjan's algorithm[9], and the *rec_sccs* method that contains its recursive version.

On top of this base class, the *UndirectedClusterization3DGraph* defines an undirected graph that is represented as a 3-dimensional matrix in which each voxel corresponds to a node. In this graph, the (undirected) edges are defined as follows: a node will have as neighbors all its surrounding voxels (e.g., 26 if it is a central voxel) if and only if the node itself fulfills a specific condition, and no neighbors at all otherwise. The condition is specified through a function that receives a node and returns a boolean value, which is fed to the initialization method when instantiating the graph. Given that this is an undirected graph, the *sccs* method is overridden and Tarjan's algorithm is replaced by a more efficient and simple *Breadth First Search (BFS)*[10].

Finally, a class called *NiftiGraph* is built on the *UndirectedClusterization3DGraph* by defining the condition function as the fact of pertaining to a certain range, i.e., given a minimum and/or a maximum threshold(s), a voxel fulfills the condition if and only if its value is greater or equal to the lower threshold and strictly smaller than the upper threshold, being both values defined when instantiating the graph (if omitted, the lower and upper thresholds are set to minus infinite and plus infinite respectively).

In addition to these structures, a generic graph was implemented in the form of a class, namely *GenericGraph*, in this same module, in order to test the correctness and performance of the aforementioned graph algorithms. This class is not used in the toolbox, but can be imported and employed in any other application if wanted.

Currently, the toolbox supports filtering by mean gray matter volume (*GM filtering*) using a lower threshold, by fitting score (*FS filtering*) by means of lower and upper thresholds, and by cluster size (*CS filtering*) through a lower threshold.

[8] https://en.wikipedia.org/wiki/Strongly_connected_component
[9] https://en.wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm
[10] http://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm

2.5.7 Visualization transformation

In some particular situations, the final data of an experiment must be adapted to the visualization tools used to display it so that the results are easily understandable. This is the case, for instance, of the maps of p-values generated by the *F-test* evaluation method (see section 2.7.3), which can be easily interpreted when displayed in the form of numbers, but are not suited to be mapped to gray-scale values and displayed as they are.

To solve this, a transformation function can be applied to the data, obtaining this way new values that the visualization tool will display much more clearly. In the example of the *F-test*, obtaining the z-values corresponding to the inverted p-values (after clusterization) has shown to be a good way of adapting the data to be displayed with *FSLView*.

This block is not integrated in the toolbox, and has been written for particular cases in the form of scripts, since the transformation depends on the specific properties of the data (distribution, range, etc.) and the tool for which it is being adapted (e.g., *FSLView*).

2.6 Fitters

2.6.1 General Linear Model: GLM

The General Linear Model (GLM) is based on the assumption that the target variable is linearly related to the variables of interest and the covariates, as expressed by the following expression:

$$y = \delta_1 c_1 + \ldots + \delta_m c_m + \beta_1 x_1 + \ldots + \beta_n x_n + \epsilon \tag{2.3}$$

As we can see, the target variable is just a linear combination of the variables of interest and the covariates plus a certain error term (ϵ), which is considered to be a gaussian i.i.d. random variable (white noise).

In this model, the parameters to be optimized are the coefficients of both, the covariates and the variables of interest, having as optimization criteria the minimization of the MSE, or which is the same, the minimization of the square of the module of ϵ.

To do so, we must first substitute each variable in equation 2.3 with its observations. This gives N expressions, that can be rewritten in vector form as:

$$\begin{gathered} y_{Nx1} = C_{Nxm}\Delta_{mx1} + X_{Nxn}\beta_{nx1} + \epsilon_{Nx1} \\ \Delta_{mx1} = \begin{pmatrix} \delta_1 \\ \vdots \\ \delta_m \end{pmatrix}; \beta_{nx1} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} \end{gathered} \tag{2.4}$$

We can then proceed to define a new matrix $A_{Nx(m+n)}$, denoted *design matrix* (a.k.a. model matrix), and a new vector $\Gamma_{(m+n)x1}$, and express equation 2.4 in terms of such variables:

$$A_{Nx(m+n)} = (C_{Nxm}|X_{Nxn})\,; \Gamma_{(m+n)x1} = \begin{pmatrix} \Delta_{mx1} \\ - - - \\ \beta_{nx1} \end{pmatrix} \qquad (2.5)$$

$$y_{Nx1} = A_{Nx(m+n)}\Gamma_{(m+n)x1} + \epsilon_{Nx1}$$

Finally, we minimize the square of the module of ϵ by differentiating it w.r.t the Γ^T vector and equaling the resulting expression to zero (this is an Ordinary Least Squares – OLS - problem). The demonstration of this equality has been attached in appendix C.1.

$$\frac{\partial}{\partial \Gamma^T}\|\epsilon\|^2 = 0 \iff \Gamma = \left(A^T A\right)^{-1} A^T y \qquad (2.6)$$

If we interpret these results geometrically, we will observe that $A\Gamma$ is actually a vector in the subspace defined by the independent variables (variables of interest and covariates, the columns of A), and what the solution is trying to do is minimize the distance between this vector and the observations of the target variable y, which is accomplished when $A\Gamma$ is the projection of y in the mentioned subspace, i.e., when $\epsilon \perp A\Gamma$ (see figure 2.4).

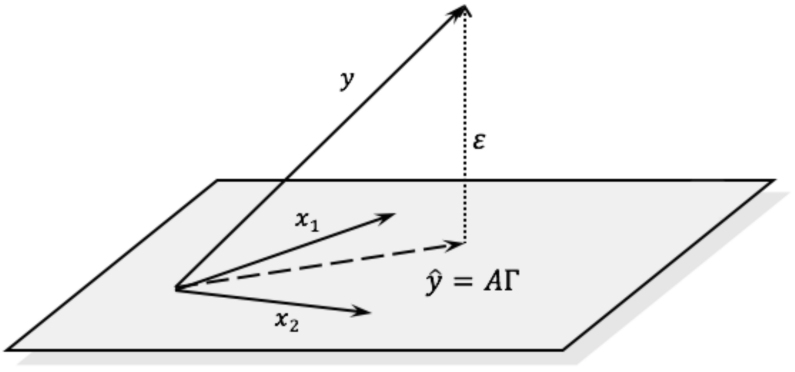

Figure 2.4: Geometrical interpretation of GLM fitting

GLM with correlated error

Until now, we have assumed that the error of each sample had a gaussian distribution and was uncorrelated with the error of the other samples. However, the General Linear Model can also treat the case in which the error is correlated among samples. Having a covariance matrix of the form $Cov_{NxN} = \sigma^2 V_{NxN}$, where $V_{NxN} = \sum_i \lambda_i Q^i_{NxN}$, and being Q^i_{NxN} the matrix corresponding to the i^{th} covariance component of the model, the

optimization problem becomes a Weighted Least Square (WLS) problem, whose solution is:

$$\Gamma = \left(A^T V^{-1} A\right)^{-1} A^T V^{-1} y \tag{2.7}$$

If we make $V^{-1} = W^T W$, the solution transforms to:

$$\Gamma = \left((WA)^T WA\right)^{-1} (WA)^T Wy = \left({A_w}^T A_w\right)^{-1} {A_w}^T y_w \tag{2.8}$$

where $A_w = WA; y_w = Wy$.

That is, the result is the same as in the case of i.i.d. error, but substituting the design matrix and the observations with their *whitened* versions.

The General Linear Model as a fitter

This fitter assumes that the data has already been whitened and the variables of interest have been orthogonalized w.r.t. the covariates (refer to equation 2.2 in section 2.5.3 to see what each of these functions means).

- $Q(a, b) = -\|b - a\|^2$ (negative of MSE)
- $f_{p_1,\ldots,p_k}(x_1, \ldots, x_n) = p_1 x_1 + \ldots + p_k x_n \qquad (k = n)$
- $g(y, c_1, \ldots, c_m) = y - f_{\hat{\delta}_1,\ldots,\hat{\delta}_m}(c_1, \ldots, c_m)$, where
  - $\hat{\delta}_1, \ldots, \hat{\delta}_m = arg_{\delta_1,\ldots,\delta_m} max\left[Q\left(f_{\delta_1,\ldots,\delta_m}(c_1, \ldots, c_m), y\right)\right]$

The corresponding class, called *GLM*, has been implemented as a wrapper of the *Linear-Regression* class from the *scikit-learn*[11] python library.

It is important to notice that, as stated, the *GLM* class implements the fitter for the *General Linear Model*, not to be confused with the *Generalized Linear Model*, which is a generalization of the former, that can fit non-normal distributions of the exponential family. A more detailed description has been included in appendix D.

2.6.2 Generalized Additive Model: GAM

Although attractively simple, the traditional linear model discussed earlier is often not optimal, since most inter-variable dependencies in real-world gathered data are not linear. Here a more automatic and flexible statistical method that may be used to identify and characterize nonlinear regression effects is described.

[11] http://scikit-learn.org/

The Generalized Additive Model is a generalization of the Generalized Linear Model, in which the observations depend linearly on unknown smooth functions of the prediction variables. Introduced in [Hastie and Tibshirani, 1990], this model blends properties of the GLM with the benefits of additive models, allowing non-parametric fits with relaxed assumptions on the relationship between independent and dependent variables at the cost of loosing some interpretability in the results.

The GAM model relates an outcome Y to various prediction variables X_i through unspecified smooth functions f_j. If these functions were a basis expansion of some kind, the problem would be easily solved by ordinary least-squares. However, this approach is different: we use non-parametric functions fitted using scatter-plot smoothers (*splines*, *kernel smoother*, etc.). The model assumes that the observations vector Y is sampled from an exponential family distribution (*gaussian*, *binomial*, ...) and that its conditional mean $E[Y|X] = \mu(X)$ is related to the prediction variables through a link function $g(\mu(X))$ defined as the sum of several weighted functions, as shown next:

$$g(\mu(X)) = \alpha + f_1(X_1) + f_2(X_2) + \ldots + f_N(X_p) = \alpha + \sum_{j=1}^{P} f_j(X_j),$$

where p is the number of predictors.

Each function f_j can be defined by specifying their parametric form (for example, polynomials of a given degree), or in a non-parametric way, by indicating the type of *smooth function* to which they correspond. As a result, we can easily combine linear functions with nonlinear ones assuming that the dependence of the observations on the prediction variables is of the form $g(\mu) = X_A^\top \beta + \alpha + \sum_{j=|A|}^{N} f_j(X_j)$, where X_A is the set of predictors to be modeled linearly and $X_B = \{X_j \mid j \geq |X_A|\}$ is the set of predictors whose effect in the observations is hypothesized to be potentially nonlinear.

The functions f_i are estimated iteratively by means of an algorithm called *backfitting*, whose main block is a scatter-plot smoothing. However, a few restrictions must be applied and/or fulfilled to ensure that the algorithm converges ([Hastie et al., 2009, chap. 9.1]):

- $\alpha = \text{mean}(Y)$. The constant α is not uniquely identifiable, since each function in the model may have its own mean. Therefore, the standard convention is to assume that for any given function f_j, the mean over the images of the predictors for the function is zero, i.e., $\sum_{i=1}^{N} f_j(x_{ij}) = 0 \quad \forall j$.
- The feature matrix X, which comprises all the prediction vectors X_j as columns, is not singular.

If those conditions are fulfilled, the problem is strictly convex and the solution is unique. As a result, the backfitting algorithm (described in algorithm 1) will yield the optimal functions for this model.

Algorithm 1 Backfitting algorithm

$\hat{\alpha} \leftarrow \frac{1}{N}\sum_{i=1}^{N} y_i$ ▷ Initialize elements
for $j = 1 \dots P$ **do**
  $\hat{f}_j \leftarrow 0$
end for
while $\hat{f}_j$ has not converged for some j **do**
  for $j = 1 \dots P$ **do**
    $\hat{f}_j \leftarrow \mathbf{S_j}\left[y - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k\left(X_k\right)\right]$ ▷ Smooth function
    $\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N}\sum_{i=1}^{N} \hat{f}_j(x_{ij})$
  end for
end while

Moreover, this procedure can accommodate several fitting methods in exactly the same way, by specifying the appropriate smoothing operator $\mathbf{S_j}$:

- Other univariate regression smoothers such as local polynomial regression and kernel methods.
- Linear regression operators yielding polynomial fits, piecewise constant fits, parametric spline fits, series and Fourier fits.
- More complicated operators such as surface smoothers for second or higher-order interactions or periodic smoothers for seasonal effects.

The Generalized Additive Model as a fitter

- $Q\left(a, b\right) = -\|b - a\|^2$ (negative of MSE)
  - Alternatively, $Q\left(a, b\right) = -\|b - a\|^2 - \lambda * \int a''(x)\partial x$ (negative of PRSS)
- $f_{p_0, p_1, \dots, p_k}\left(x_1, \dots, x_n\right) = p_0 + p_1 f_1\left(x_1\right) + \dots + p_k f_n\left(x_n\right)$
- $g\left(y, c_1, \dots, c_m\right) = y - f_{\hat{\delta}_1, \dots, \hat{\delta}_m}\left(c_1, \dots, c_m\right)$, where
  - $\hat{\delta}_1, \dots, \hat{\delta}_m = arg_{\delta_1, \dots, \delta_m} max\left[Q\left(f_{\delta_1, \dots, \delta_m}\left(c_1, \dots, c_m\right), y\right)\right]$

The integration of this model into the toolbox is being performed by Adrià Casamitjana Díaz, a PhD student at the Image Processing Group.

2.6.3 Support Vector Regression: SVR

The Support Vector Regression is a variant of the Support Vector Machine for regression. In this model, a potentially infinite set of basis functions whose values do not need to be

known, $h_m(x)$ for $m = 1, \ldots, M$, is defined through the inner product in the transformed space, also known as *kernel function*:

$$K(x, y) = \langle h(x), h(y) \rangle = \sum_{m=1}^{M} h_m(x) h_m(y) \tag{2.9}$$

The dependent variable in this case is assumed to be a linear transformation of the independent variable in the transformed space, i.e.:

$$f(x) = \sum_{m=1}^{M} \beta_m h_m(x) + \beta_0 \tag{2.10}$$

where the β_m coefficients (for $m = 0 \ldots M$) are estimated by minimizing the expression:

$$H(\beta) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2} \sum_{m=0}^{M} \beta_m^2 \tag{2.11}$$

for some general error measure $V(r)$. For any choice of such measure, the solution can be shown to have the form ([Hastie et al., 2009, chap. 12.3]:

$$\hat{f}(x) = \sum_{m=1}^{M} \hat{\beta}_m h_m(x) + \hat{\beta}_0 = \sum_{i=1}^{N} \hat{\alpha}_i K(x, x_i) \tag{2.12}$$

where $\hat{\alpha}_i$ also depends on the basis functions through the kernel function alone. A demonstration for the particular case of $V(r) = r^2$ has been included in appendix C.2. Moreover, due to the nature of the optimization problem, typically only some of the solution coefficients $\hat{\alpha}_i$ are nonzero, and the associated data values (x_i) are called the support vectors.

A frequently used error function is depicted in figure 2.5, together with its mathematical expression (on the left).

$$V_\epsilon(r) = \begin{cases} 0, & if \ \|r\| < \epsilon \\ \|r\| - \epsilon, & otherwise. \end{cases}$$

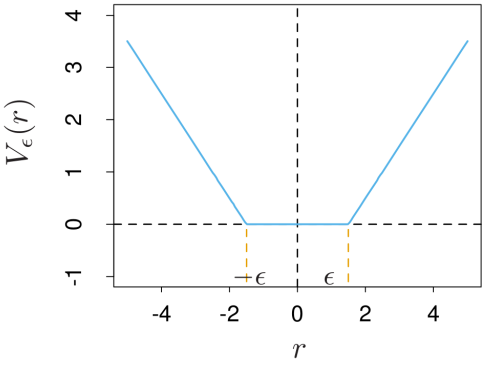

Figure 2.5: ϵ-insensitive error function $V_\epsilon(r)$

This error function does not penalize the predictions that fall within the *tube* of radius ϵ around the real observations, and the penalization for the rest of the predictions is linear on the distance to the hyperplane defined to a distance ϵ from the observations. The role of ϵ is to set the error-tolerance of the prediction, i.e., a higher value for this parameter will lead to a more flexible model, and thus to a higher error (in general). Meanwhile, the λ parameter works as a regularization parameter to penalize the complexity of the model and avoid overfitting.

Three typical choices for the kernel functions in the literature ([Hastie et al., 2009, chap. 12.3]) are also listed below:

$$\begin{aligned} \textbf{d}^{\textbf{th}}\textbf{-Degree polynomial:} \quad & K(x,x') = (1+\langle x,x'\rangle)^d, \\ \textbf{Radial basis:} \quad & K(x,x') = exp\left(-\gamma\|x-x'\|^2\right), \\ \textbf{Neural network:} \quad & K(x,x') = tanh\left(\kappa_1\langle x,x'\rangle + \kappa_2\right). \end{aligned}$$

The Support Vector Regression Machine as a fitter

- $Q(a,b) = -\left(\sum_{i=1}^{N} V(b_i - a_i) + \frac{\lambda}{2}\sum_{l=0}^{K} p_l^2\right)$ (negative of the aforementioned H function)
- $f_{p_0,p_1,\ldots,p_k}(x_1,\ldots,x_n) = \left\{\sum_{l=1}^{k} p_l h_l(x_j) + p_0\right\}_{j=1}^{n} = \left\{\sum_{i=1}^{N} \hat{\alpha}_i K(x_j,x_i)\right\}_{j=1}^{n}$
- $g(y,c_1,\ldots,c_m) = y - f_{\hat{\delta}_1,\ldots,\hat{\delta}_m}(c_1,\ldots,c_m)$, where
  - $\hat{\delta}_1,\ldots,\hat{\delta}_m = arg_{\delta_1,\ldots,\delta_m} max\left[Q\left(f_{\delta_1,\ldots,\delta_m}(c_1,\ldots,c_m),y\right)\right]$

The integration of this model into the toolbox is being performed by Santi Puch Giner, a bachelor's degree student at the Image Processing Group.

2.7 Fit evaluation methods

As previously stated, several evaluation methods have been included in the toolbox to assess the quality of the fitted curves for each voxel and be able to compare different models among each other.

2.7.1 Mean Squared Error: MSE

The Mean Squared Error is likely the simplest measure in the toolbox, and is computed as follows:

$$MSE = \frac{1}{N} \|y^c - \hat{y}\|^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i^c - \hat{y}_i)^2 \tag{2.13}$$

where y^c is the corrected observations vector and $\hat{y}$ is the prediction vector (both of length N).

The main disadvantage of this method is that, if the fitted model is sufficiently flexible or complex, the overfitting problem may arise, since this measure does not penalize neither the complexity of the model nor the abruptness of the resulting curves. However, this is still an appropriate indicator of the quality of the fit for cases in which the models are very simple or a huge amount of data is available (i.e., when we can ensure that there will not be overfitting).

2.7.2 R-squared

This method is basically a range-adjusted version of the MSE method, and it is computed as follows:

$$R^2 = 1 - \frac{MSE}{\frac{1}{N} \sum_{i=1}^{N} (y_i^c - \bar{y}^c)^2} \tag{2.14}$$

where y^c denotes the corrected observations vector and $\bar{y}^c = \frac{1}{N} \sum_{i=1}^{N} y_i^c$ is the mean value of such vector.

This method suffers from the same disadvantage as the MSE; however, the R-squared measure is ensured to be in the range $[0, 1]$, which can be useful when applying the filtering, clustering and visualization transformation blocks.

2.7.3 F-test

An F-test is any statistical hypothesis test in which the test statistic follows an F-distribution under the null hypothesis. For instance, the ratio of two appropriately scaled chi-squared distributed variables follows an F-distribution that is determined by the degrees of freedom (df) of each of those variables $F(df_{numerator}, df_{denominator})$. This test can be used in regression problems to determine whether a particular part of a model is significantly improving the overall performance of the rest of the model.

Consider two models, $M_{restricted}$ and M_{full}, such that $M_{restricted}$ is nested within M_{full}, that is, $M_{restricted}$ is a *submodel* of M_{full}, namely the *restricted model*. Then, we can measure how much the inclusion of $M_{full} - M_{restricted}$ in the model is enhancing the fit by comparing the variance of the fitting error when using the complete model, M_{full}, as

opposed to using the restricted model, $M_{restricted}$. Typically, model M_{full} will yield better results (a smaller error) than $M_{restricted}$, but what we want to measure is whether the difference is significant or not (we want to test whether the submodel $M_{full} - M_{restricted}$ is relevant or not). Thus, we will perform an F-test where the null hypothesis states that the variances of both errors are equal, while the alternative hypothesis claims that the variance of the error when using the complete model (M_{full}) is smaller than when using the partial model ($M_{restricted}$):

- H_0: $Var\left(MSE_{M_{restricted}}\right) = Var\left(MSE_{M_{full}}\right)$
- H_A: $Var\left(MSE_{M_{restricted}}\right) > Var\left(MSE_{M_{full}}\right)$

The F-statistic is then computed as follows:

$$F = \frac{\left(\frac{RSS_{restricted} - RSS_{full}}{df_{restricted} - df_{full}}\right)}{\left(\frac{RSS_{full}}{df_{full}}\right)} \tag{2.15}$$

where $RSS_x = N * MSE_x = \sum_{i=1}^{N} \left(y_i - \hat{y}_i^x\right)^2$ is the *Residual Sum of Squares* of model M_x - being $\hat{y}^x$ the prediction obtained by using such model and y the vector of observations– and df_x is the number of degrees of freedom of the error signal of model M_x. The computation of the degrees of freedom of a model is generally non-trivial and model-dependent; as a consequence, it must be implemented inside each fitter separately[12].

On the other hand, if the fitting has been performed in a weighted fashion, the expression remains the same except for the fact that the residual sum of squares for model M_x is computed as follows: $RSS_x = \sum_{i=1}^{N} w_i \left(y_i - \hat{y}_i^x\right)^2$, where w_i is the weight given to the i^{th} sample in the fitting process.

Under the null hypothesis, F will follow an $F\left(df_{restricted} - df_{full}, df_{full}\right)$ distribution.

The toolbox provides two methods based on this measure: the *fstat* method returns the value of the F-statistic itself, whereas the *ftest* function returns the corresponding p-value, that is, the value of the cumulative density function of the mentioned distribution evaluated in the value of the F-statistic.

2.7.4 Akaike Information Criterion: AIC

The Akaike Information Criterion is yet another measure to evaluate the quality of the fit for a regression problem. However, this one is oriented to the comparison of different models. The formula to compute it is shown below:

[12]The computation of the degrees of freedom for the GAM fitter has been implemented by Adrià Casamitjana Díaz (see section 2.6.2), whereas such implementation for the SVR fitter has been performed by Santi Puch Giner (see section 2.6.3)

$$AIC = 2k - 2ln\left(L\right) \tag{2.16}$$

where k is the number of estimated parameters in the model, and L is the maximum value of the likelihood function for such model.

When comparing different models with this method, the ones with the minimum resulting values are the ones that best fit the data.

2.7.5 Corrected Akaike Information Criterion: AICc

The corrected AIC is a variant of the AIC that adds a higher penalization to the complexity of the model. The formula for this measure is model-specific. The explicit expression for univariate linear models with normally-distributed residuals is shown below.

$$AICc = AIC + \frac{2k\left(k+1\right)}{n-k-1} \tag{2.17}$$

Again, lower values are preferred over higher ones.

2.7.6 Penalized Residual Sum of Squares: PRSS

The Penalized Residual Sum of Squares ([Hastie et al., 2009, chap. 2.8]) is the only method in the toolbox that directly penalizes the roughness of the predicted curve itself instead of the model complexity. The expression of such measure is described next:

$$PRSS = N * MSE + \lambda \int \left[f''\left(x\right)\right]^2 \partial x \tag{2.18}$$

where f'' is the second derivative of the prediction function, and the integral is limited to the range of the input predictors.

The second derivative indicates how rapidly the slope of the prediction function changes, being thus an indicator of the abruptness of the curve. A variant of this measure is also proposed for the toolbox, in which the integral is replaced by the maximum value of the square of the second derivative in the region of interest (the range of the input predictors).

Chapter 3

Results

3.1 Experiments

As explained in section 2.4.1, all the experiments have been performed so that the results can be comparable to those of [Gispert et al., 2015] and among themselves. In particular, the gray matter data has always been corrected with the *GLM* fitter, having the linear term of the patients' sex and the linear and quadratic terms of their age as correctors, and then predicted with the AD-CSF index as the only predictor.

The first experiment consists in testing the overall dependence of the corrected gray matter volume on the AD-CSF index in order to identify the regions of the brain in which such dependence is strong. This will allow us to check which parts of the brain are actually affected during the progression of the disease. We do this by following the same methodology as in [Gispert et al., 2015], where a 3^{rd} order polynomial of the AD-CSF index is introduced in the GLM, to then apply an F-test and keep only the values whose p-value is lower than 0.001. The results (after clusterization of >100 voxels and visualization transformation) are shown in appendix E.1, where we can observe that both, the relevant regions detected by our software and their shapes match closely the ones found by [Gispert et al., 2015].

The second experiment also reproduces the result of such paper. In this case, we compare the effect of the nonlinear (quadratic and cubic) terms of the AD-CSF index as opposed to the linear term, applying the *RGB* paradigm of fit comparison (section 2.5.5) on the values of the f-statistic for the linear and the nonlinear models, being both of them General Linear Models. Once again, we can observe that the results are coherent with those of [Gispert et al., 2015], showing strong nonlinear tendencies in the gray matter volume throughout the AD continuum expressed in terms of the AD-CSF index (see appendix E.2).

The third experiment shows a comparison between the three terms of the 3^{rd} order polynomial of the AD-CSF index, i.e., all of the linear, quadratic and cubic terms are fed to

three different GLM instances, being in each case two of them correctors and the other one the predictor. This way, each model expresses the *additional* information given by each of the terms with respect to the other two. Then, the resulting fitting-scores are compared with the *BEST* paradigm described in section 2.5.5. The results are filtered to show only the regions that are relevant (with p-value < 0.01). In this case, we can observe (appendix E.3) that both of the nonlinear terms are clearly dominant over the linear one, with no significant difference in relevance between the quadratic and the linear terms.

The fourth and last experiment consists in generating five fitters: one for the GLM model with a 3^{rd} degree polynomial on the AD-CSF index, a second one for the polynomial GAM model of same characteristics, another one for the splines-based GAM model, one more for the polynomial SVR, and a last one for the gaussian SVR. Then, all the parameters are optimized for each model, and five curves are displayed for three relevant voxels (according to the previous experiments). These curves can be observed in appendix E.4, being clearly visible, once more, that the results are coherent with those reported in [Gispert et al., 2015].

3.2 Conclusions

The toolbox has been successfully implemented and is already almost fully operative. Moreover, thanks to its design, the proposed structure seems to be flexible enough to support virtually any data-driven learning approach, and apply it in problems of different fields through the use of the *Fitting* library, and particularly in experiments related to neuroimaging through the *Processing* library.

Proof of it is that two students at the Image Processing Group, namely Santi Puch Giner and Adrià Casamitjana Díaz, have already (partially) integrated two quite different fitters (with two types of submodels each) into the toolbox, and are using them to successfully obtain consistent results, hence proving the flexibility and usability of the package. Besides, Santi has also been able to build an automatic hyperparameter look-up and selection feature into the system (used in the fourth experiment of section 3.1), thus verifying its extensibility. Finally, we have contrasted the maps yielded by the *GLM* fitter against the results published in [Gispert et al., 2015], consequently confirming them and showing the correctness of the structures and techniques implemented in this project, and have also been able to generate different curves that seem to faithfully fit the data in each voxel.

As a result, and despite the fact that certain parts of the library have not yet been completed, we can conclude that the toolbox is already a viable solution to fit nonlinear patterns over any data that meets the implicit requirements set by the curve fitting block (see sections 2.4.1 and 2.5.3), and particularly, to analyze the dynamics in gray matter reduction throughout the Alzheimer's Disease continuum in a flexible manner.

Appendices

Appendix A.

Information table for each subject

Attribute	**Description**
ID	Integer value that uniquely identifies the subject in the database.
GM Data	A separate NIfTI-1 image file containing a 3-dimensional ($121x145x121$) matrix that represents the brain of the subject. Each element of the matrix contains the gray matter volume of the corresponding region of the brain.
Diagnostic	An integer representing the diagnostic given to the subject, as follows: 0 = NC (control, no indicators of the disease). 1 = PC (pre-clinical, some indicators but without cognitive decline). 2 = MCI (mild cognitive impairment, with a slight cognitive decline). 3 = AD (Alzheimer's disease, already having the disease).
Age	An integer indicating the age of the subject in years.
Sex	A binary value indicating the genre of the subject: 0 = Female. 1 = Male.
APOE4	An integer representing the status of the APOE4 gene for this subject.
Education	An integer that indicates the academic level of the subject.
Aβ42	An integer representing the level of Aβ42 protein measured for this subject.
T-tau	An integer representing the level of t-tau protein measured for this subject.
P-tau	An integer representing the level of p-tau protein measured for this subject.
AD-CSF index	A floating point value corresponding to the computed AD-CSF index value for this subject.

Table A.1: Available information of each subject

For more information on how these measurements were done, please refer to [Gispert et al., 2015].

Appendix B.

Simplified UML diagram

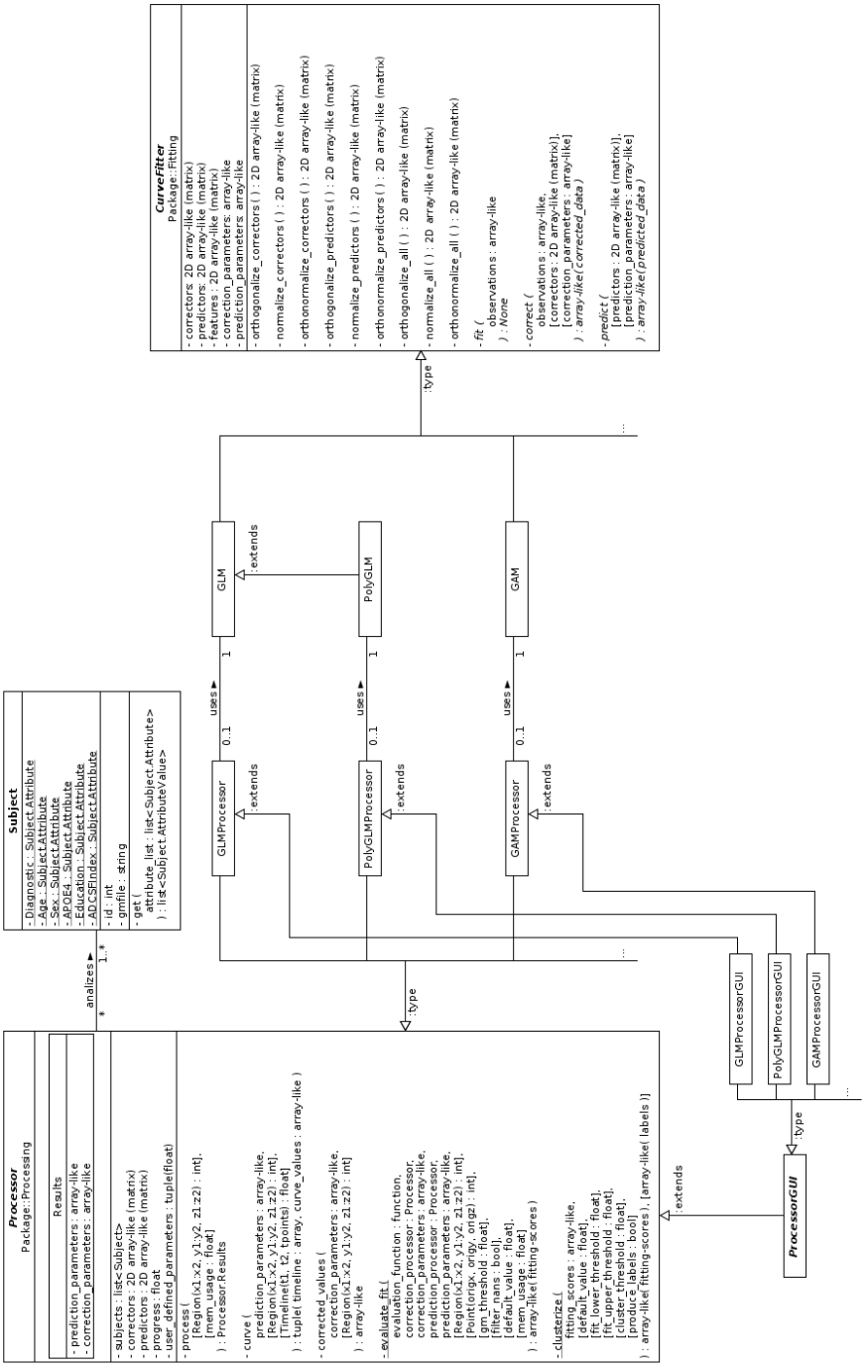

Figure B.1: Simplified UML diagram

Appendix C.

Mathematical demonstrations

C.1 Ordinary Least Squares solution

$$\begin{gathered}\frac{\partial}{\partial \Gamma^T}\|\epsilon\|^2 = \frac{\partial}{\partial \Gamma^T}\left(\epsilon^T\epsilon\right) = \frac{\partial}{\partial \Gamma^T}\left[(y - A\Gamma)^T(y - A\Gamma)\right] = \\ \frac{\partial}{\partial \Gamma^T}\left(\Gamma^T A^T A\Gamma - \Gamma^T A^T y - y^T A\Gamma\right) = \frac{\partial}{\partial \Gamma^T}\left(\Gamma^T A^T A\Gamma - 2y^T A\Gamma\right) = \\ A^T A\Gamma + \left(\Gamma^T A^T A\right)^T - 2\left(y^T A\right)^T = 2\left(A^T A\Gamma - A^T y\right)\end{gathered} \tag{C.1}$$

$$\begin{gathered}\frac{\partial}{\partial \Gamma^T}\|\epsilon\|^2 = 2\left(A^T A\Gamma - A^T y\right) = 0 \iff A^T A\Gamma = A^T y \\ \frac{\partial}{\partial \Gamma^T}\|\epsilon\|^2 = 0 \iff \Gamma = \left(A^T A\right)^{-1} A^T y\end{gathered} \tag{C.2}$$

C.2 Support Vector Regression solution

Please, refer to section 2.6.3 for more information on the *Support Vector Regression*. For concreteness, the case $V(r) = r^2$ has been selected. This proof has been extracted from [Hastie et al., 2009, chap. 12.3].

Let $\mathbf{H}$ be the N x M basis matrix in which element (i, m) corresponds to $h_m(x_i)$, and suppose that $M > N$ is large. For simplicity, we assume that $\beta_0 = 0$, or that the constant is absorbed in h.

We estimate β by minimizing the penalized least squares criterion

$$H(\beta) = (\mathbf{y} - \mathbf{H}\beta)^T(\mathbf{y} - \mathbf{H}\beta) + \lambda\|\beta\|^2. \tag{C.3}$$

The solution is $\hat{\mathbf{y}} = \mathbf{H}\hat{\beta}$, with $\hat{\beta}$ determined by

$$-\mathbf{H}^T\left(\mathbf{y} - \mathbf{H}\hat{\beta}\right) + \lambda\hat{\beta} = 0 \implies \hat{\beta} = \left(\mathbf{H}^T\mathbf{H} - \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{y}. \tag{C.4}$$

From this it appears that we need to evaluate the M x M matrix of inner products in the transformed space ($\mathbf{H}^T\mathbf{H}$ term in the solution). However, we can pre-multiply the expression by $\mathbf{H}$ to give

$$-\mathbf{H}\mathbf{H}^T\left(\mathbf{y}-\mathbf{H}\hat{\beta}\right)+\lambda\mathbf{H}\hat{\beta}=0 \implies \left(\mathbf{H}\mathbf{H}^T+\lambda\mathbf{I}\right)\mathbf{H}\hat{\beta}=\mathbf{H}\mathbf{H}^T\mathbf{y}$$

$$\mathbf{H}\hat{\beta}=\left(\mathbf{H}\mathbf{H}^T+\lambda\mathbf{I}\right)^{-1}\mathbf{H}\mathbf{H}^T\mathbf{y}. \tag{C.5}$$

The N x N matrix $\mathbf{H}\mathbf{H}^T$ consists of inner products between pairs of observations (i, i'); that is, the evaluation of an inner product kernel $\left\{\mathbf{H}\mathbf{H}^T\right\}_{i,i'}=K\left(x_i, x_{i'}\right)$.

In this case, the predicted value will be

$$\hat{f}\left(x\right)=h\left(x\right)^T\hat{\beta}=\sum_{i=1}^{N}\hat{\alpha}_iK\left(x, x_i\right), \tag{C.6}$$

where $\hat{\alpha}=\left(\mathbf{H}\mathbf{H}^T+\lambda\mathbf{I}\right)^{-1}\mathbf{y}$.

Appendix D.

Generalized Linear Model

The General Linear Model (section 2.6.1) can optimally fit the coefficients of the model when the error has a normal distribution. However, this is often not the case in data collected from natural environments, which causes the General Linear Model not to be suitable for many real-world problems.

Fortunately, if the observations follow a probability density function that pertains to the exponential family, the Generalized Linear Model can be used to optimally fit the parameters of the model to the observations. The exponential family is characterized by the following parameterized expression:

$$f_y\left(y|\theta,\tau\right) = h\left(y,\tau\right) exp\left(\frac{b\left(\theta\right)^T T\left(y\right) - A\left(\theta\right)}{d\left(\tau\right)}\right) \tag{D.1}$$

In the Generalized Linear Model, the mean of the target variable depends on the independent variables through:

$$E\left(y\right) = \mu = g^{-1}\left(A\Gamma\right) \tag{D.2}$$

where g denotes the *link function*, whereas the variance is given as a function of the mean:

$$Var\left(y\right) = V\left(E\left(y\right)\right) = V\left(\mu\right) \tag{D.3}$$

for some function V.

The link function provides the relationship between the linear predictor and the mean of the distribution function, i.e., it linearizes the dependence of the target variable w.r.t. the variables of interest and the covariates.

Hopefully, an example will help to clarify these concepts. Let's imagine we are trying to predict the number of people that will go to the beach (dependent variable) based uniquely on the temperature (independent variable). We could try to use a linear model, but we would see that, although the predictions would be correct for a certain range of temperatures, values below a certain threshold would eventually produce negative predictions, which would be inappropriate for this case (-5 people going to the beach does not make any sense), that is, this model would not escalate well for small values. Instead, we could try to fit an exponential model, so that whenever the temperature raises a fixed amount of degrees, the model doubles the value of the prediction. Obviously, the prediction would need to be rounded, but this model would never output a prediction below 0 people, no

matter how much the temperature drops, giving thus a much better performance for small values. In this case,

$$E\left(y\right) = \mu = g^{-1}\left(A\Gamma\right) = exp\left(A\Gamma\right) = exp\left(\beta T\right)$$

being T the temperature, and therefore having as the link function:

$$g\left(A\Gamma\right) = log\left(A\Gamma\right)$$

The unknown parameters Γ in the Generalized Linear Model are typically estimated with maximum-likelihood, maximum quasi-likelihood or Bayesian techniques, by substituting the mean (μ) with the observations.

Appendix E.

Results of the experiments

E.1 Heat map of brain regions affected by AD

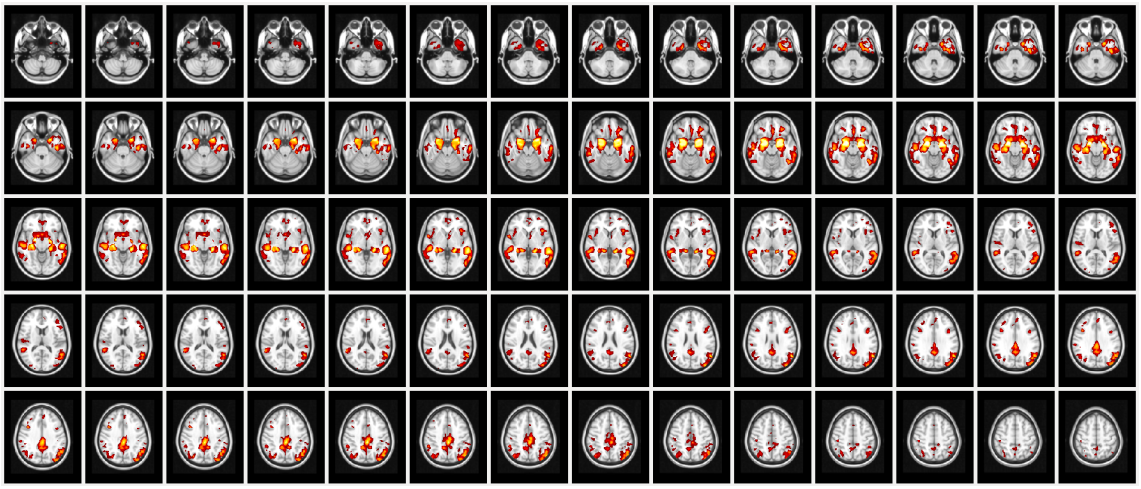

Figure E.1: Brain areas showing gray matter attrophy in association with AD-CSF index. Results for GLM from our toolbox. Only the axial view is available.

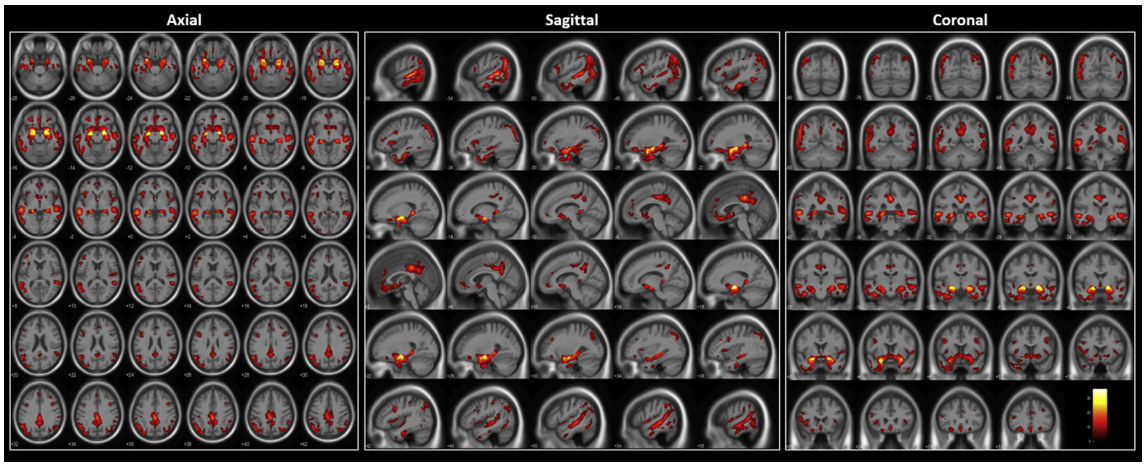

Figure E.2: Brain areas showing gray matter attrophy in association with AD-CSF index. Results from [Gispert et al., 2015]. Please, notice that all scans are flipped horizontally with respect to figure E.1.

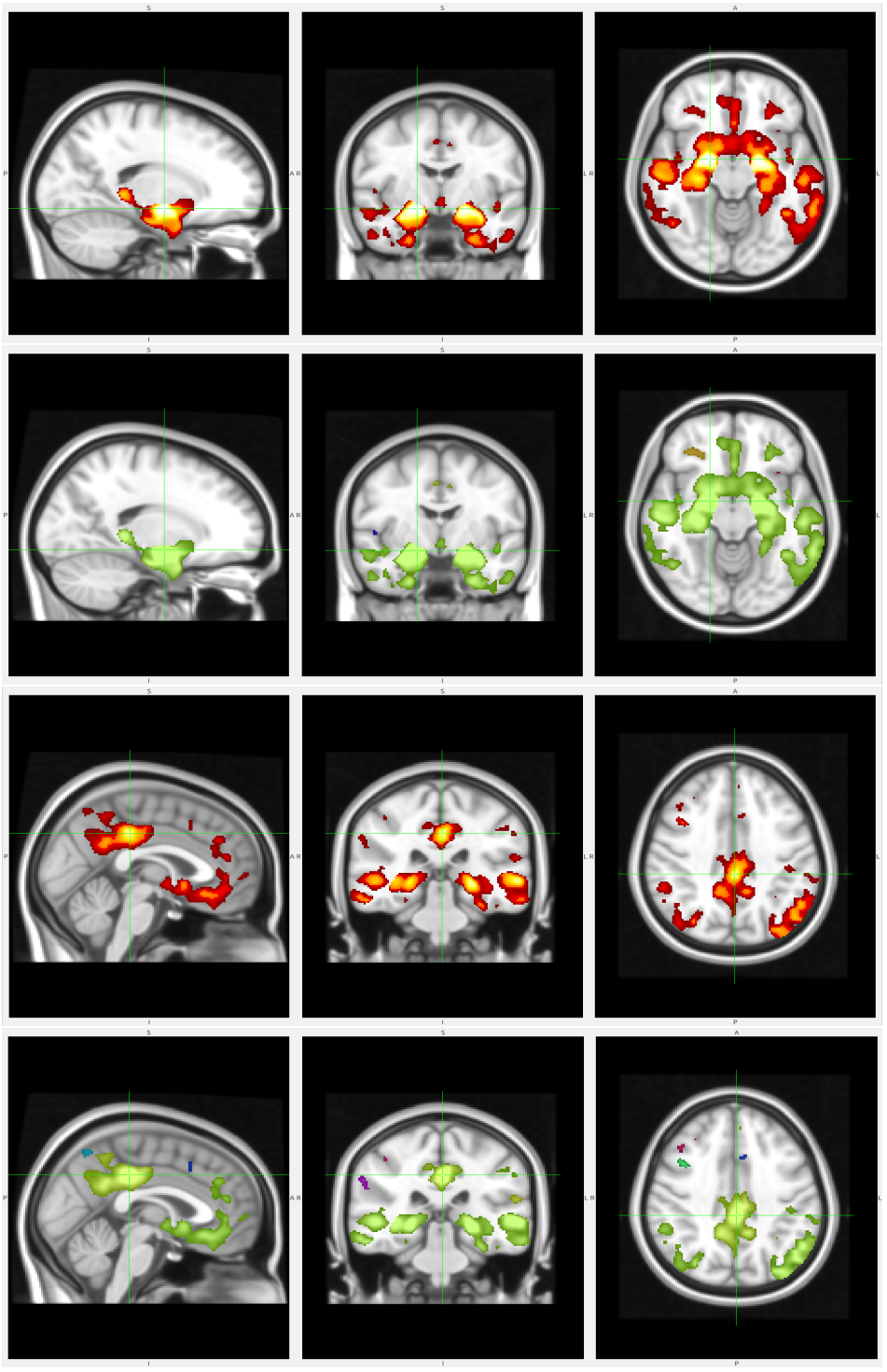

Figure E.3: Sagittal, coronal, and axial view of relevant regions centered in two voxels that maximize the fitting-score for their respective cluster. The first and third rows correspond to heatmaps showing the value of the fitting-scores, whereas the second and fourth regions correspond to masked versions of such maps, in which the different colors indicate the cluster to which each voxel pertains.

E.2 Linear vs. Nonlinear fit comparison

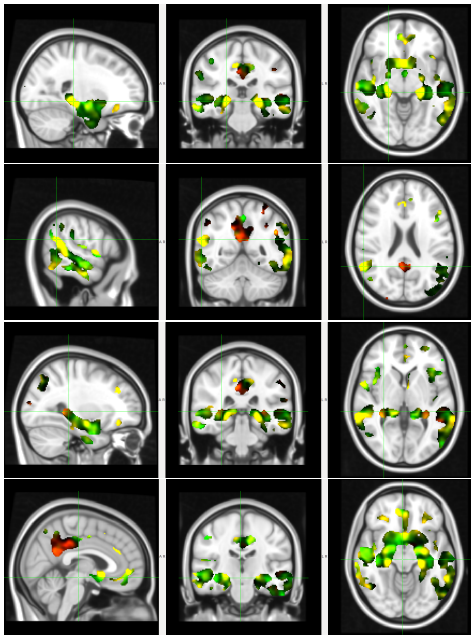

Figure E.4: Additive fusion of the F-maps associated to linear (red) and nonlinear (green) components of the regression against the AD-CSF index. Results from our toolbox.

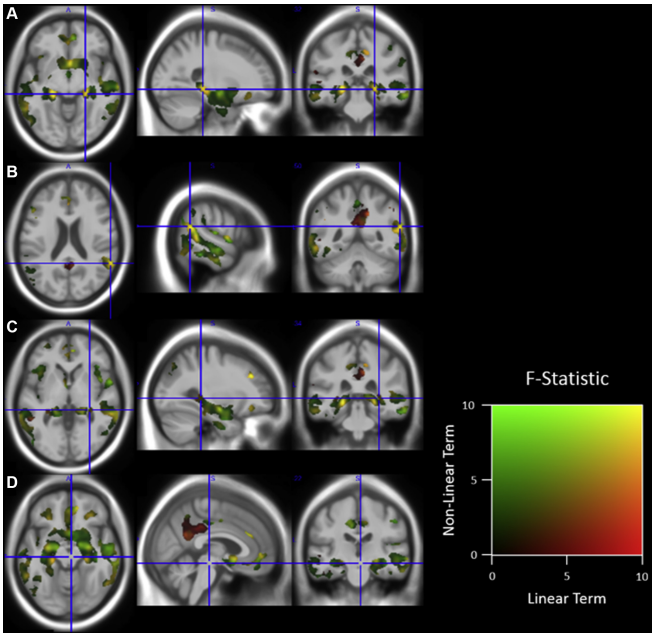

Figure E.5: Additive fusion of the F-maps associated to linear (red) and nonlinear (green) components of the regression against the AD-CSF index. Results from [Gispert et al., 2015]. Please, notice that all axial and coronal scans are flipped horizontally w.r.t. figure E.4

E.3 Single term fit comparison

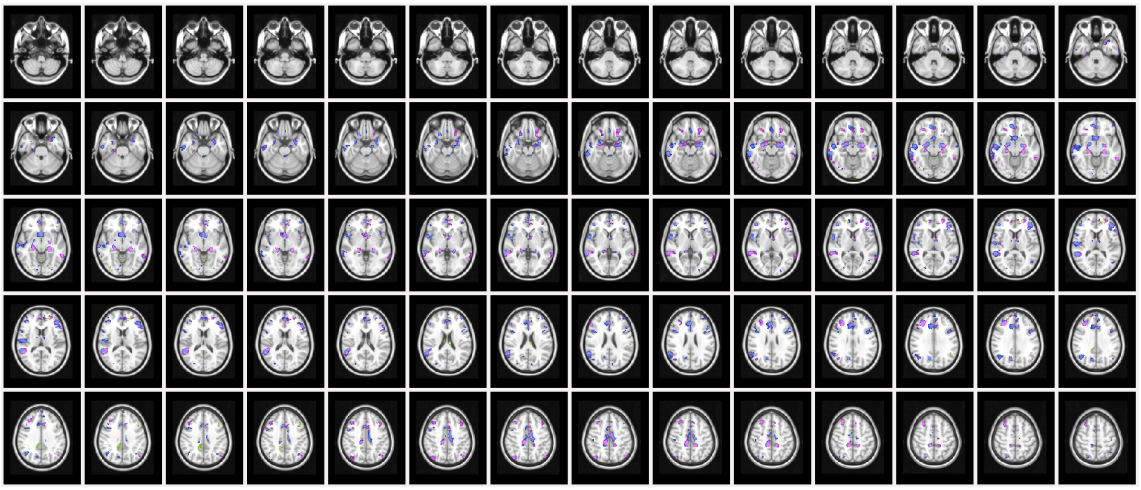

Figure E.6: Best model selection between linear (green), quadratic (purple) and cubic (blue) terms of the AD-CSF index after having accounted for the other two terms.

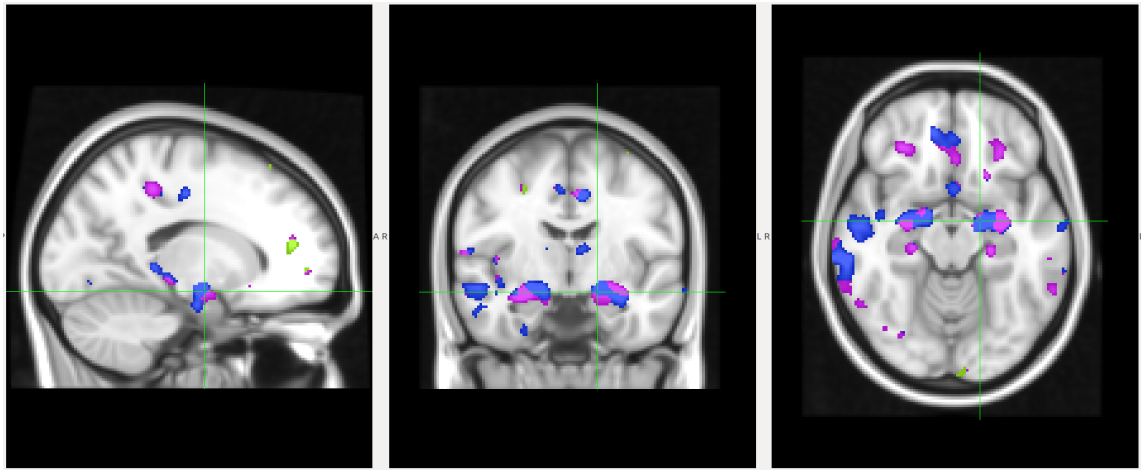

Figure E.7: Sagittal, coronal and axial view of the left Hippocampus. Notice the clear dominance of the nonlinear terms (purple and blue) over the linear one (green).

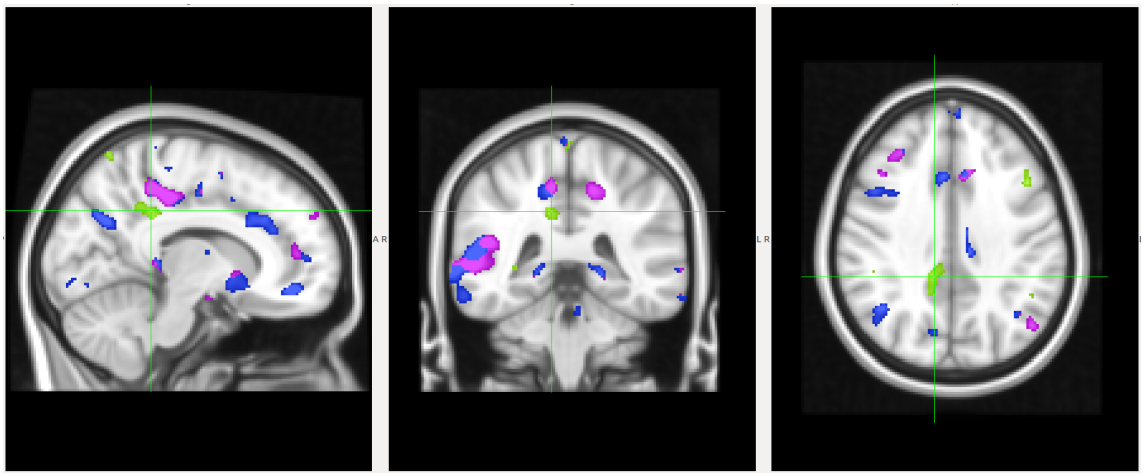

Figure E.8: Sagittal, coronal and axial view of the largest cluster where the linear term provides more *additional* information than any of the nonlinear terms separately.

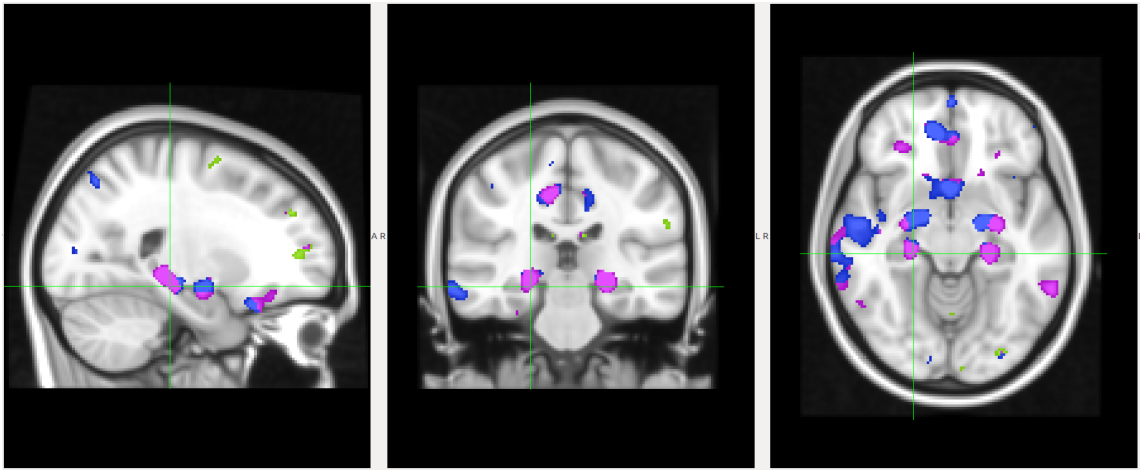

Figure E.9: Sagittal, coronal and axial view of the right ParaHippocampal. Once again, the nonlinear fittings are clearly dominant over the linear one.

E.4 Curve generation and visualization

Note: The curve obtained from the GLM fitter does not seem to be present in any of the graphs generated by our toolbox. This is because the curve produced by the polynomial GAM fitter is actually the same, which is logical given that the Generalized Additive Model is indeed a generalization of the General Linear Model, and its polynomial version is identical to the GLM except for the way in which they are fitted (they produce exactly the same output).

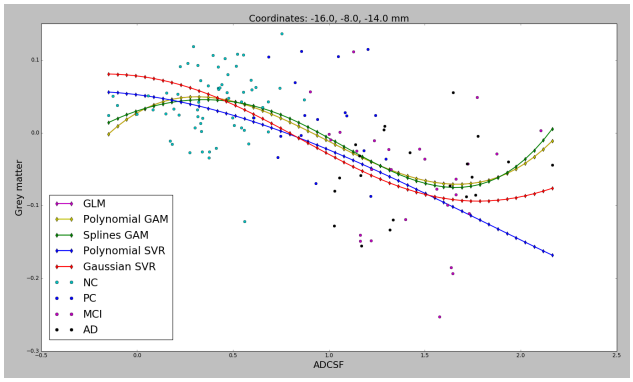

Figure E.10: Nonlinear fit of different models by using our toolbox for a voxel in the left Hippocampus (brain region).

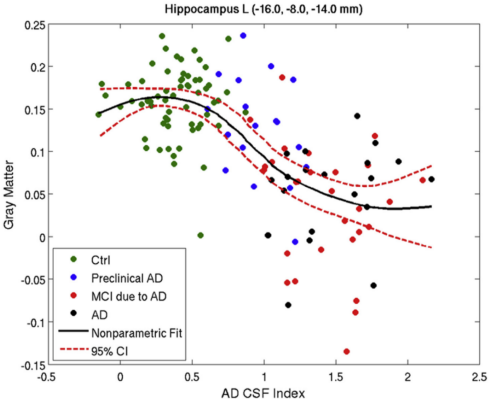

Figure E.11: Nonlinear fit using Matlab's lowess function for a voxel in the left Hippocampus (brain region). Extracted from [Gispert et al., 2015].

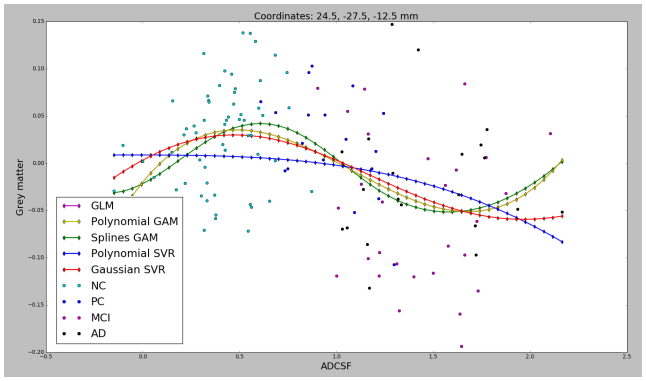

Figure E.12: Nonlinear fit of different models by using our toolbox for a voxel in the right ParaHippocampal (brain region).

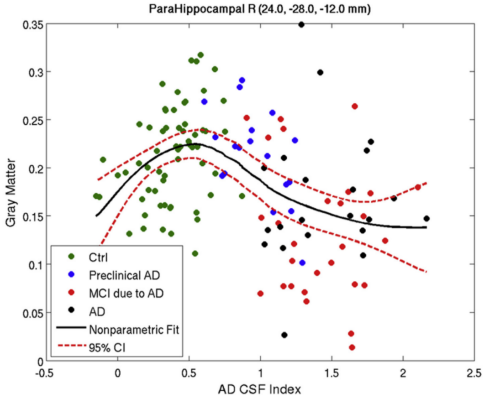

Figure E.13: Nonlinear fit using Matlab's lowess function for a voxel in the right ParaHippocampal (brain region). Extracted from [Gispert et al., 2015].

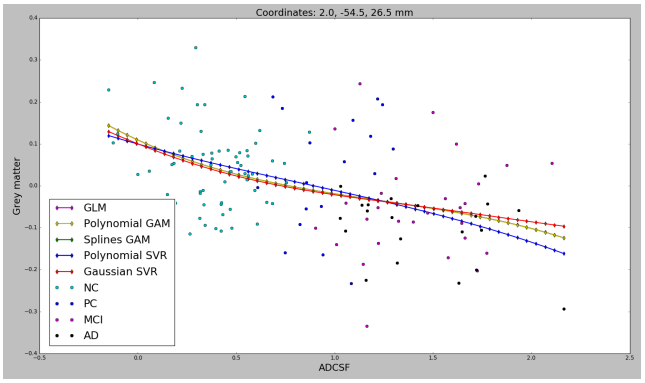

Figure E.14: Nonlinear fit of different models by using our toolbox for a voxel in the right Precuneus (brain region).

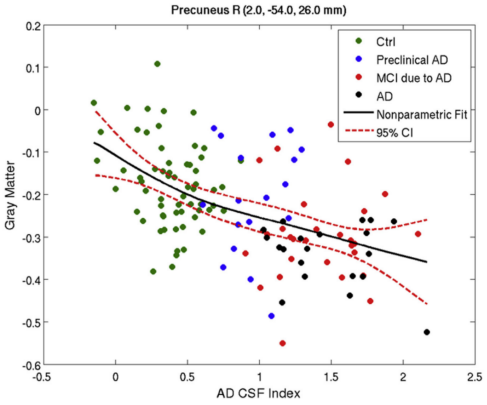

Figure E.15: Nonlinear fit using Matlab's lowess function for a voxel in the right Precuneus (brain region). Extracted from [Gispert et al., 2015].

Appendix F.

Gantt Chart

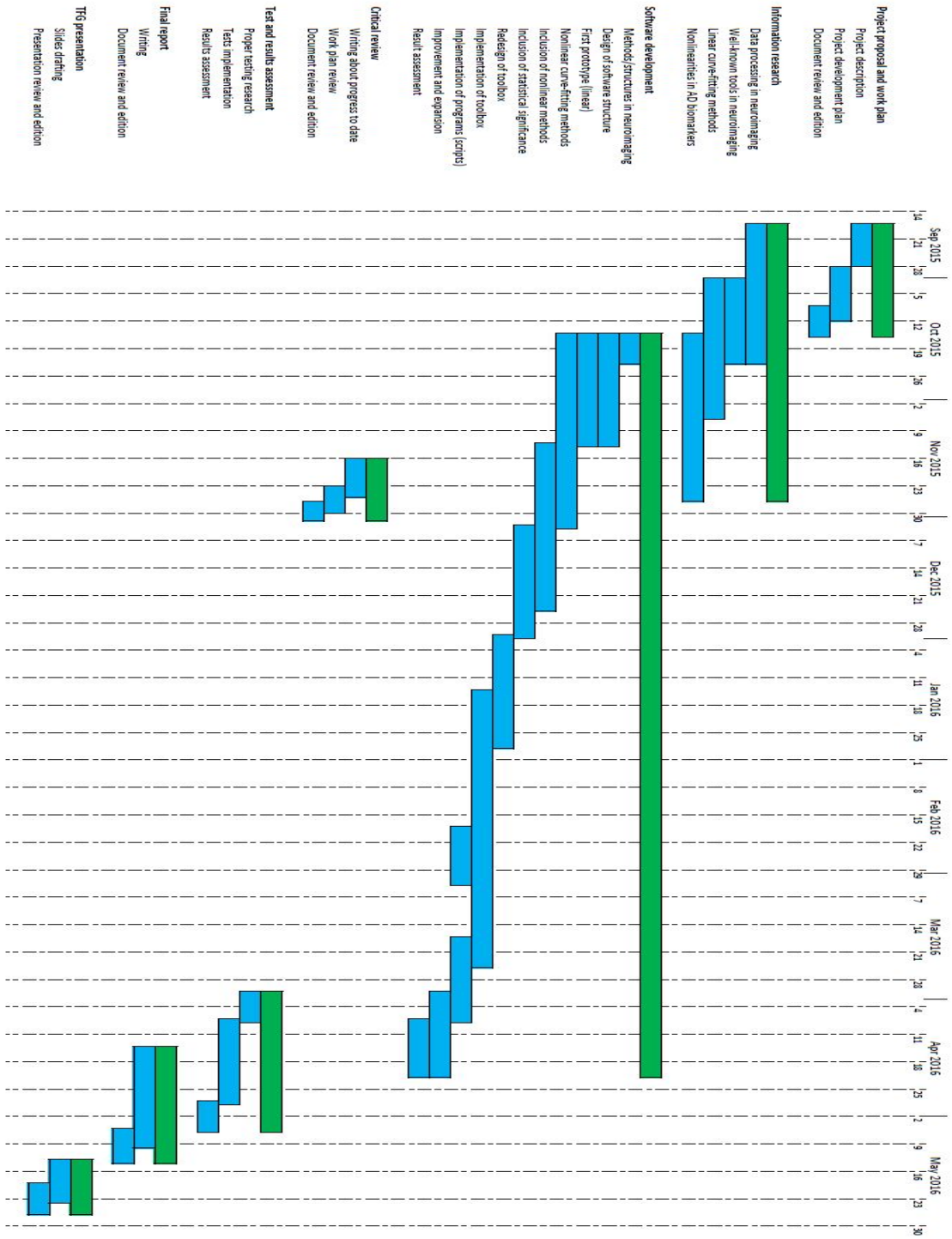

Figure F.1: Gantt chart that illustrates the schedule followed during the project.

Bibliography

[Bateman et al., 2012] Bateman, R. J., Xiong, C., Benzinger, T. L. S., Fagan, A. M., Goate, A., Fox, N. C., Marcus, D. S., Cairns, N. J., Xie, X., Blazey, T. M., Holtzman, D. M., Santacruz, A., Buckles, V., Oliver, A., Moulder, K., Aisen, P. S., Ghetti, B., Klunk, W. E., McDade, E., Martins, R. N., Masters, C. L., Mayeux, R., Ringman, J. M., Rossor, M. N., Schofield, P. R., Sperling, R. A., Salloway, S., and Morris, J. C. (2012). Clinical and biomarker changes in dominantly inherited alzheimer's disease. *The New England Journal of Medicine*, 367:795–804.

[Braak et al., 2013] Braak, H., Zetterberg, H., Tredici, K. D., and Blennow, K. (2013). Intraneuronal tau aggregation precedes diffuse plaque deposition, but amyloid-β changes occur before increases of tau in cerebrospinal fluid. *Acta Neuropathologica*, 126(5):631–641.

[Fjell et al., 2010] Fjell, A. M., Walhovd, K. B., Fennema-Notestine, C., McEvoy, L. K., Hagler, D. J., Holland, D., Blennow, K., Brewer, J. B., and Dale, A. M. (2010). Brain atrophy in healthy aging is related to csf levels of aβ1-42. *Cerebral Cortex*, 20(9):2069–2079.

[Gispert et al., 2015] Gispert, J. D., Rami, L., Sánchez-Benavides, G., Falcon, C., Tucholka, A., Rojas, S., and Molinuevo, J. L. (2015). Nonlinear cerebral atrophy patterns across the alzheimer's disease continuum: impact of apoe4 genotype. *Neurobiology of Aging*, 36(10):2687–2701.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* Springer, second edition edition.

[Hastie and Tibshirani, 1990] Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models*, volume 43 of *Monographs on Statistics and Applied Probability*. Chapman & Hall/CRC.

[Insel et al., 2014] Insel, P. S., Mattsson, N., Donohue, M. C., Mackin, R. S., Aisen, P. S., Jack Jr, C. R., Shaw, L. M., Trojanowski, J. Q., and Weiner, M. W. (2014). The transitional association between β-amyloid pathology and regional brain atrophy. *Alzheimer's & dementia*, 11(10):1131–1260.

[Jack Jr et al., 2013] Jack Jr, C. R., Knopman, D. S., Jagust, W. J., Petersen, R. C., Weiner, M. W., Aisen, P. S., Shaw, L. M., Vemuri, P., Wiste, H. J., Weigand, S. D., Lesnick, T. G., Pankratz, V. S., Donohue, M. C., and Trojanowski, J. Q. (2013). Tracking pathophysiological processes in alzheimer's disease: an updated hypothetical model of dynamic biomarkers. *The Lancet Neurology*, 12(2):207–216.

[Jack Jr et al., 2012] Jack Jr, C. R., Vemuri, P., Wiste, H. J., Weigand, S. D., Lesnick, T. G., Lowe, V., Kantarci, K., Bernstein, M. A., Senjem, M. L., Gunter, J. L., Boeve, B. F., Trojanowski, J. Q., Shaw, L. M., Aisen, P. S., Weiner, M. W., Petersen, R. C., and Knopman, D. S. (2012). Shapes of the trajectories of 5 major biomarkers of alzheimer disease. *Archives of Neurology*, 69(7):856–867.

[Molinuevo et al., 2013] Molinuevo, J. L., Gispert, J. D., Dubois, B., Heneka, M., Lleo, A., Engelborghs, S., Pujol, J., de Souza, L. C., Alcolea, D., Jessen, F., Sarazin, M., Lamari, F., Balasa, M., Antonell, A., and Rami, L. (2013). The ad-csf-index discriminates alzheimer's disease patients from healthy controls: a validation study. *Journal of Alzheimer's Disease*, 36(1):67–77.

[Sabuncu et al., 2011] Sabuncu, M. R., Desikan, R. S., Sepulcre, J., Yeo, B. T. T., Liu, H., Schmansky, N. J., Reuter, M., Weiner, M. W., Buckner, R. L., Sperling, R. A., and Fischl, B. (2011). The dynamics of cortical and hippocampal atrophy in alzheimer disease. *Archives of Neurology*, 68(8):1040–1048.

[Schuff et al., 2012] Schuff, N., Tosun, D., Insel, P. S., Chiang, G. C., Truran, D., Aisen, P. S., Jack Jr, C. R., and Weiner, M. W. (2012). Nonlinear time course of brain volume loss in cognitively normal and impaired elders. *Neurobiology of Aging*, 33(5):845–855.

[Struyfs et al., 2014] Struyfs, H., Molinuevo, J. L., Martin, J.-J., Deyn, P. P. D., and Engelborghs, S. (2014). Validation of the ad-csf-index in autopsy-confirmed alzheimer's disease patients and healthy controls. *Journal of Alzheimer's Disease*, 41(3):903–909.