DEPARTMENT OF PHYSICS E. FERMI
Master's degree in physics

Functional connectivity measures using machine learning techniques

Supervisors:
Prof. Alessandra Retico
Prof. Piernicola Oliva

Candidate:
Federico Campo

Academic year: 2022/2023

Contents

Chapter 1

ASD = previous Introduction

Autism spectrum disorder (ASD) is becoming a growing social issue, drawing more and more attention expecially during the last decades, both for its social impact on families and society and because of the raise in number of diagnosied cases.

ASD is classified as a neurodevelopmental behavioural disorder [4] [7] and refers to a broad range of conditions manifesting as deficits in social communication and interaction such as reduced sociability or empathy, repetitive behaviours and resistance to changes, and sometimes speech difficulties.[10]

Early signs start appearing by the age of 2 or 3, but the most accurate assessment procedures so far are comportamental, for a toddler, parents answer a set of questions about every-day behaviour of their children like the checklist for Autism in Toddler [13] where a list of question such as "When you point at something, your child watch in that direction? "or "des your children shows interest towards other children? "

However, since there isn't a definitive biological test, often ASD is discovered even after the adolescence always through a comportamental assessment test. Currently ADI-R (Autism Diagnostic Interview - Revised) and ADOS (Autism Diagnostic Observation Schedule) are considered the 'gold standard' tools for diagnosis of ASD [8] [1], even though for a better assessment, medical and biological informations should be taken into account as well .

is of great importance to make an early diagnosis to provide the help and support they need, because, hopefully, treating it in its early stage, could bring to a regression of this condition.

The main factors that bring to the developing of autism aren't very clear so far, and is acquiring more consensus between scientists and doctors that there isn't just a single cause, but a concatenation and cohexistence of various of them, the most strongly suspected of which are genetic and environmental, intended as environmental exposure to air pollution, during pregnancy, expecially during the first days of embrional development (day 20-24 of gestation).[11], that is associated with an increased risk of develop this disorder.

Genetic is assiciated with a rare gene's mutation, such as a deletion, duplication or inversion, and even though most of the mutation that increase the risk of developing autism are not been traced, it has been assessed that it has a percentage of inheritability around 90%, resulting from studies on omozygotes twins,[2].

From a neurological point of view, ASD, usually entails a reduced information processing due to synaptic dysfunction that manifests in a reduced or altered brain activity. [**?**] This

is confiermed by several but controversal studies where both hyper functional connectivity and hypo-FC were detected in asd patients. in particular it seems to have an age-dependent trend, for over connectivity is usually observed in young children while under connectivity in adolescences and adults.[15].

In the last decades, different approaches to study this problem have been carried out, to find a relation between different brain areas involved in this hyper or hypo connectivity Different diagnostic tool were proposed to investigate this matter, from different perspectives and at different scales, such as magnetic resonance imaging (MRI), functional-MRI (f-MRI), or electroencephalography (EEG). Magnetic resonance imaging was employed to study brain with structural images of it, while EEG and f-MRI aim to study temporal signals with two different temporal resolution, since with EEG is possible to study signal in scale time comparable with the neuronal activation time timescales $\sim \mu s$, while with f-MRI we look for a signal due to the hemodynamic response following a neuronal activation (timescales $\sim s$). Among them, f-MRI is what is drawing more interest, and in particular resting state f-MRI was proven to be suitable for examining functional connectivity among brain regions that different highlighted to be more affected by autism rather than local brain regions.

f-MRI is a diagnostic tool that investigate blood flow variations across the whole brain structure. The measure the blood flow is directly linked with the neuronal activity, an increase of neuronal activity leads to a boost in blood flow and an increase of vessel's size within a specific brain area, because of the bigger demand of oxigen and other nutrients to the neurons in that area.

The different neural connections are identified by measuring the BOLD fluctuations, (acronym for blood-oxigen-level dependent) from different areas of the brain, using the signal difference between oxy (diamagnetic) and deoxyhemoglobin (paramagnetic). Oxyhemoglobin appears then brighter than deoxyhemoglobin The spontaneous fluctuations in the BOLD fluctuations during resting state are considered a strong indicator for the assessment of the properties of the brain system.

Resting state f-MRI is an acquisition performed while the patient is in a relaxed state, and is not performing any active task, he is simply asked to stay still with eye close or open while fixating a cross. Following this setup the brain is at rest and its activity is not perturbed by active tasks such as moving an arm or passive actions like being exposed to different visual stimuli, which are common activities for task-based functional connectivity, a different acquisition setup to investigate BOLD fluctuations between subjects in a task-stimulated state and control states. Resting state setup revealed to be a powerful tool to investigate the intrinsec generated brain activity and study the altered functional connectivity networks in subjects with mental disorder

During the study of characteristic traits between control and ASD patients, different attributes are involved and characterise data in a strong and evident way, the most important of which are age, sex or FIQ (full intellective quotient). Age for example affects both structural and functional data, with an observed overgrown of the brain volume in toddlers and hyperconnectivity mentioned before, that are both traits that tend to decrease going in on age, and on control subjects, even though it is not well characterised, seem to show a decreasing of brain structure and functional connectivity in older ages [**?**] Sex is the other feature that gives characteristic traits to data, studies show that some brain area show an increased functional connectivity in females, and from a combined analysis of both age and

sex it appears that in men some brain structures show more pronounced aging effect than women [**?**] Other factor conditioning the final outcome of a f-MRI data acquisition are the FIQ (Full intellective quotient) or, limiting our focus just on functional data, the eye status at scan plays an important role, about this, functional connectivity with open steady eyes results to be different from closed eyes, with strong differences and higher connectivity in different brain areas between the two coohrts of subjects. [**?**] and furthermore, an other aspect to be take into account regarding closed eye patients, is that during the scan acquisition, cthey may fall asleep, and this would heavily modify the functional brain activity, resulting in a modified functional connectivity pattern.

It appears clear that the distinction of asd patient among normal ones is not a simple and straightforwand task because of all these confounding factors concorring to the characterization of acquired data.

To take on this challenge, one of the most promising tools that allow us to deal with complex, non-linear problems is the use of machine learning algorithm to study data extracted from fMRI such as pairwise correlations between different regions of the brain, or image of brain themselves from MRI scans to study with convolutional neural networks (CNN). Machine learning algorithm belong to artificial inteligence tools and make use of an alghorithm that learns from data, and modify its parameters with the aim of recognising distinctive properties from them and makes prediction, classification, of new unseen data.

A relevant aspect that makes machine learning and artificial neural networks so popular is as cited before, the ability to deal with non-linear problems, and it accomplish so by introducing several non linear functions during training, to get non-linear outputs from each input data or from a combination of them.

A restraint of machine learning though, is that to perform well, algorithms need to be trained on a big dataset, because the bigger the dataset, the more it is able to generalize information and the better are its performances on new data, and this is true expecially for deep neural networks.

To this end, particularly in medical field where data are not easily available from a single centre or are not in a sufficient amount to perform a large-scale analysis, several data from different acquisition centers need to be put together, and in the last years, several multicenter medical dataset such as the Human Connectome Project, the Alzheimer's Disease Neuroimaging Initiative (ADNI) or Autism Brain Imaging Data Exchange (ABIDE) were created to try to create a large collection of data to permorm significant statistical analysis. Unfortunately, one drawback of multicenter datasets is the unavoidable bias towards the site the data belongs to, resulting from hardware and scan's procedure differences, and it would be a really hard request to uniform all medical center across the world to uniform to a single common acquisition protocol. To tacke this problem and try to uniform inter-scan variability, we need an harmonization procedure to remove site-dependent information and try to uniform data. In this work, two different aproaches are proposed: analytical harmonization and a deep learning aproach.

Analytical harmonization is a procedure that modifies data with shifts and rescaling to remove only inter-site related effects in multi-site data, while trying to preserv all other information as biological-related features. The second aproach, the deep learning one, tries to make classification of Control/ASD data extracting from data both relevant information for Control/ASD classification and site-dependent information and use the latter to remove

the bias toward sites before making a prediction.

Chapter 2

fMRI

Functional MRI aims to measure brain activity by detecting changes in blood linked to a neuronal activation.

This kind of analysis is referred as blood-oxygen-level dependent (BOLD) imaging.

The process that from a neuronal stimulus leads to a measurable blood signal is governed by the hemodynamic response function (HRF). As shown in figure 2.1 after a neuronal stimulus on a timescale of $\approx$ 10-100 ms, the BOLD signal takes $\approx$ 6s to reack a peak and a total of 20-30 s to return to its zero baseline.

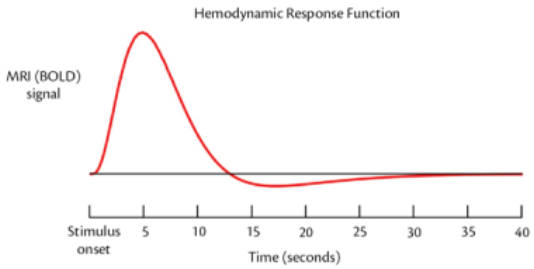


Figure 2.1: Evolution of BOLD signal of an adult brain during time. At the instant t=0 the neuronal stimulus occurs; after that, it takes around six seconds to reach a maximum of signal, and a total of more than twenty-five seconds to return to its baseline value

Signal comes from changes of blood-oxygen level of hemoglobin: an iron molecule carying oxygen throughout veins and capillaries. Hemoglobin can be either oxyhemoglobin or deoxy-hemoglobin, these two molecules differs in their magnetic susceptibility as oxyhemoglobin is diamagnetic while deoxyhemoglobin is paramagnetic. The presence of deoxy-hemoglobin causes local field inhomogeneities, leading to a reduction of signal from water molecules, because molecules go out of phase with one another more quickly, reducing the total magnetization.

This effect is characterized by T_2^* relaxation time so this is the tuning parameter related to this kind of acquisition; this gives the alternative name of BOLD-images as T_2^*-weighted images.

Three different types of tissues are present in the brain: cerebrospinal fluid (CSF), and gray and white matter.

Chapter 3

Machine learning

Machine learning is a branch of Artificial Intelligence (AI) that aims to learn from data and apply this knowledge to operate on other, new, data such as making predictions or classifying them, and by training training process, gradually improve its accuracy in the task it performs.

Algorithms are trained to make classifications or predictions and, as a rule of thumb, the bigger and homogeneous is the dataset to train the algorithm, the better accuracy and generalization (namely perform well on different datasets) can be achieved.

According to the analysis we aim to perform, and the type of dataset we are working on, machine leraning algorithms can be divided into two macro areas: supervised and unsupervised learning.

From a dataset, consisting on a collection of data, each one containing a certain number of features, an **unsupervised** algorithm goal is trying to learn the principal properties from the dataset's structure and to extract information from data without human labour to annotate and label each data. Analysis with unsupervised algorithms include clustering or dimensionality reduction. Clustering is maybe the most simple and intuitive example of an unsupervised learning: it is basically a classification process through which unlabeled data are reorganised and classified into subgoups according to some common properties or distance measures that arise from the analysis. In this way, data of each group, called cluster, share a certain degree of similarity in feature probability distribution. Dimensionality reduction is another common example of unsupervised learning process: it is performed with the task of finding an different representation of the data, with a lower dimensionality, and preserving as much information as possible about data. This can be accomplished either by compressing data into a lower-dimensional space, or by searching the main source of variance across the data and create a representation in such a way that the dimensions of the new representation are statistically independent.

On the other hand, when we are dealing with supervised learning each input data is associated with a label, specifying the class which data belong to, so the algorithm is to learn the common features for each class trying to predict the label given the corresponding input, or to classify them according to the right label. Generally speaking, given an input data x and an associated label y, they try estimating p(y—x). These algorithms are referred as supervised because of the human labour needed to label each input data.

A subset of Machine Learning is Deep Learning, the main difference lies on the algorithms' structure and the learning techniques, as an instance, traditional machine learning methods

consist on algorithms like Random Forest, Support Vector Machines (SVM), K-nearest neighbor (KNN) and several more models, while deep learning models include Artificial neural networks (ANN), Convolutional neural networks (CNN) and more. As we will see in furter paragraphs, they have a common structure similar to a neural structrure, made by different neurons connected to each other.

3.1 Random forest

Da rivedere un paio di frasi. Random Forest classifier belongs to the category of *ensemble* classifiers, and is a common machine learning algorithm. It combines multiple decision tree models to reduce overfitting of data and achieve a better generazability to create a more powerful model. Therefore, to properly understand a random forest algorithm, we need to start understanding how a Decision Tree classifier works.

A decision tree aims to learn distictive traits from input data by asking yes/no questions. Focusing on the valus of a single input feature, the classifier splits the dataset based on the value of that feature according to an if/else statement like "if feature_i > a ". Doing so each branch of a decision tree consists of a question which splits the dataset into two smaller sub datasets. Considering different features, the process is then recursively repeated untill it reaches an end point called leaves, corresponding to the ultimate partition, containing a single data point belonging to a single class. The goal of this tree is to construct branches so that the partition are informative about the class labels. In a practical way, given a dataset X made of different data samples, each one containing n features, the algorithm construct a tree following these steps: Starting from the top node called root, it searches among the n features the one which allows the best split between classes and split the dataset into two subsets, each one constituting a node. This split is performed according to a pre-defined objective function we want to maximize throughout the tree construction. Usually these kind of functions regard some kind of impurity measures such as the difference between the impurity of a parent node and the sum of the impurity of the child. The lower the impurity of the child, the bigger the Information Gain. This splitting process is repeated for each node in order to achieve the best information gain for each split, and the process is iterated and the tree is grown untill a single node is left. The important features according which a tree is forked at each node, can be visualized and if we plot the tree structure. This property is one of those which make decision trees so popular: they are easy to interpret and understand since their structure and the information of each node can be plotted to have a visual feedback of what is going on during the growing process. Other positive sides of decision trees are, their easy to use implementation which requires little data preparation and their rapidity since their complexity goes like $O(n_{features} n^2_{samples} log(n_{samples}))$ [**?**]

Three common impurity measures or splitting criteria are commoply used in decision trees: Gini impurity, entropy and classification error. As an example we report the Gini impurity measure. If a node contains a sub-dataset Q with a total number of data n, belonging to different classes k, the gini impurity measure is obtained by the simple relation

$$H(Q) = \sum_k p_k(1 - p_k) = 1 - \sum_k p_k^2 \tag{3.1}$$

Where p_k is the fraction of data in Q belonging to class k.

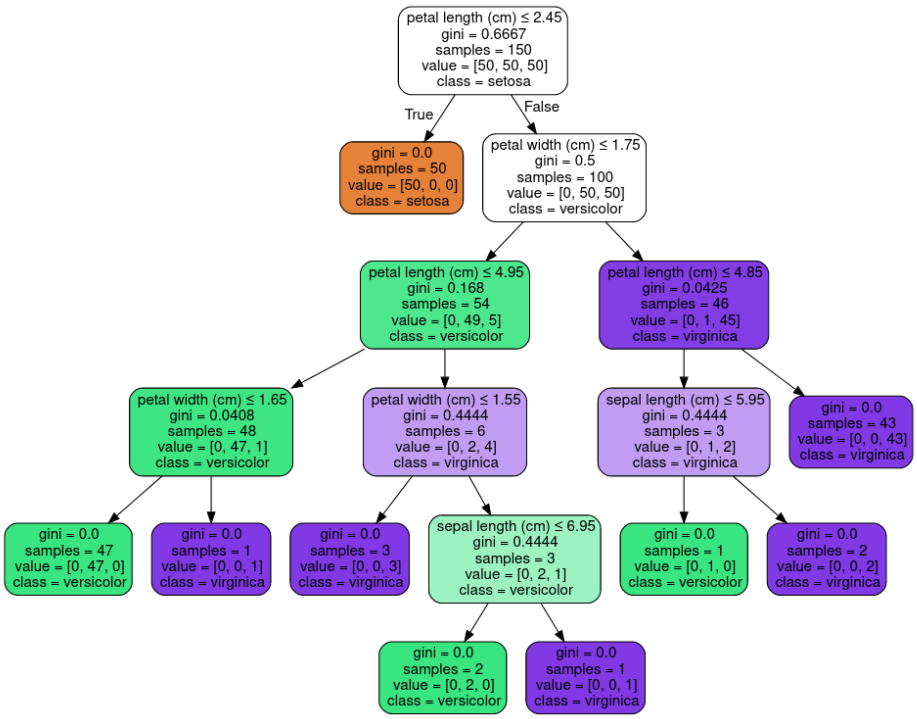


Figure 3.1: A graphical representation of a Decision Tree for a classification task of the Iris dataset consisting on three different classes of iris: Setosa, Versicolour, and Virginica. Each node of the tree shows the criterion used for its split, the gini values, the total number of samples in that node and the main class that data belong to

An example of a decision tree dealing with flower classification is shown in figure 3.1. We choose this figure because is easier to understand how a decision trees works with this example where each feature is labeled with an intuitive name and contains a physical quantity easily comparable with everyday life such as petal lenght expressed in centimeters.

Instead of looking at the whole tree, we may be interested in what feature contributed the most to the final output, to this end, a decision tree usually weigths feature with a number from 0 to 1 according to how much a feature contributed to the impurity decrease along the tree. [**?**] [**?**]

Usually though, one of the main drowbacks of decision trees is that the iterating process towards the reaching of a leaf, can bring to the creation of a over-complex model that overfits the training data, that is, the model accurately learned the training dataset but is not able to generalize well to a test dataset. To prevent overfitting, one possible strategy is to early stop the iteration towards the leaf, and leave a node with more than a single sample left. An other strategy, which leads to the construction of a Random Forest, is the creation of a stronger model, more prone to generalize data reducing overfitting.

Random Forest

A Random Forest is a collection of decision trees where each tree is slightly different from the others, each tree tends to overfit, but it does so in a different way, so that we can

reduce overfitting by averaging different results. Random forest gets its name because of the insertion of randomness during the construction of these different trees. The process whereby a Random Forest is built, given an input dataset X with N different samples, there are two main steps where randomicity plays an important role: when selecting the number of samples on to which built a tree n_trees, and when selecting the number of feature to use to grow each tree. In summary, a random forest, once selected the number of tree to be created:

1. Randomly select n≤N samples from the input dataset, this operation is called bootstrap.
2. Grow a decision tree from this bootstrap sample, but to each split, it randomly limit the number of feature available and compute the best split on this remaining subset of feature. This avoids correlations between trees resulting in better performances
3. Repeat steps 1) and 2) n_trees times.
4. Once all the trees are created, if we want to make a prediction, we put together all the trees, and use each one of them to make predictions. The final prediction of the random forest is assessed by majoring vote: the predicted class will then be the one predicted by the majority of classifiers.

Introducing this kind of randomness in a model results in a lower correlation between models and this brings to a reduced variance between outputs. Informations regarding features importance can be extracting from a random forest classifier by extracting them from each tree, and by averaging the impurity decrease from all the decision trees in the forest.

3.2 Deep Learning: ANN

We discussed in the previous paragraph one of the most important machine learning algorithm, but, as mentioned before, there's a subset of machine learning algorithm with a charactristic common structure, thanks to which they get their name as Deep learning algorithm. The structure of algorithms belonging to this family are inspired by a brain neuronal structure, and even if in a simplified way, they try to emulate the learning process of a brain. These algorithms own the adjective "deep "to their structure: they are organised in layer, each one containing several foundamental unit called neuron. Neurons between layers are connected to each other like synapsis transmit a signal between neurons in a biological brain. For this structure which reminds a neuronal brain structure,they are caller artificial neuronal networks

The foundamental unit which constiture a neuronal network is an artificial neuron, one of the most relevant example of artificial neurons is the *perceptron* shown in figure 3.2.

A perceptron is the foundamental unit of a supervised deep learning algorithm: it takes as input a data, for example an n-dimensional array $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ and returns an output: a real number

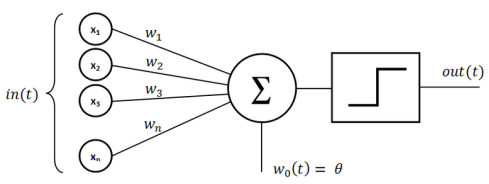


Figure 3.2: A schematic representation of a perceptron: this perceptron receives an input vector t, with n features $x_1, ...x_n$, each one weighted with a different weight $w_1, ...w_n$ and a offset w_0, and compute a linear com-

computed by applying a function to a linear combination of all the inputs $y = f(z) = f\left(\sum_i w_i x_i + b\right)$. The function that determines the output is called *activation function*, and can be either a simple step function, or a slightly more complex function we will briefly discuss in section **??**.

A deep neural network is a hyerarchical organization of neurons into layer, connected to each other. Input data are passed to the first input layer where each neuron acting like a perceptron, produces an output using the activation function which is set the same for every neurons of the layer, but can differ from the activation function of other layers. Once collected each and every output from all the first layer neurons, they are passed to the second layer and become the input to each neuron belonging to this layer. This is reiterated through all the layers up to the final output layer. As schematized in figure 3.3 an artificial neural network is mainly composed of three parts: an input layer, come middle layers also called *hidden layer* and a final layer. Each neuron of a layer is linked to all the neurons of the previous and next layer. The connection between two neurons is weighted, according to an initial setup of the network during which weights are randomly set.

When reach the end of this process, we obtain the output: a numerical value related to every single input data $\mathbf{x}$.

Using matrix formalism the entire input-output process can be written as

$\mathbf{y} = \sum_j^n w_{ij} x_j + b_i$ where $\mathbf{x} \in \Re^n$ $\mathbf{y} \in \Re^m$ being m the output dimension given by the number of neurons in the output layer.

During the training process, given an input, the algorithm calculate an output (prediction) and this process is referred as *forward propagation*. This prediction is then compared with the actual value this output should be, and during the *backpropagation* the algorithm modifies its weights in order to minimize the difference between the actual and the predicted value.

The goal of a machine/deep learning algorithm is to learn from data using a train dataset, and generalize it in order to perform well on an unseen dataset called Test dataset. Performing well means to produce a low error on the Test dataset after minimizing the error on train dataset.

To fully define a neural network we just need some parameters called hyperparameters, the principals of which are:

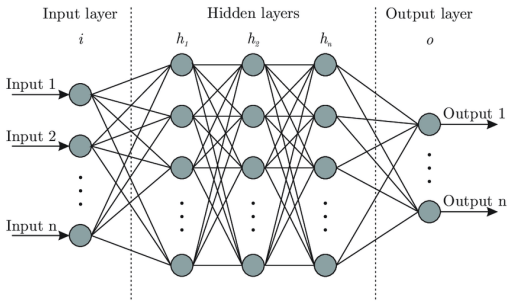


Figure 3.3: Schematic representation of an artificial neural network containing an input layer, three hidden layers and an output layer: a single vector contained n features (input 1, ... input n) is given as input to the first Input layer i. All the inputs are linked to every neurons of the first hidden layer, which would compute a linear combination of its inputs. All the hidden layers are linked to the other hidden layers, and at the end, n different outputs are returned.

- Number of layers
- Number of neuron for each layer
- Activation function for each layer
- Number of epochs (or iterations)
- Learning rate

One of the most important concept in machine learning when characterising a network is its Capacity. It refers to the level of complexity that a model is able to learn. This is strictly linked to one of the critical issue in machine leraning: the concept of underfitting and overfitting.

Underfitting occurs when the model isn't able to learn the required amount of informations during train; this usually happens for shallow network, when the model has a small number of parameters in regard to the number needed to explain the input features, having low parameters results in poor performances because the model isn't able to learn the underlying structure of a complex dataset.

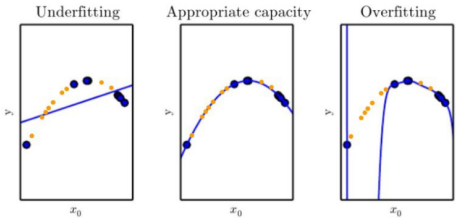
On the contrary, overfitting occurs when the model has too many parameters compared to those required to learn the input features. What happens then is that the model memorizes all the data it sees during train but it is not able to generalise them. As a result, the model performs well on the Train dataset and has low performances in an unseen dataset, called Test datset.

As an example in a two-dimensional space, if we aim to fit a dataset consisting of some points sampled from a quadratic curve, using three different functions: a linear function, a quadratic function and a polynomial function with grade equal to the number of data points. As shown in figure 3.4 the linear model is not able to describe the data distribution while the polynomial function has too many parameters, and it is perfectly able to fit our distribution, but it would perform very poorly to describe a dataset sampled from the same quadratic distribution, furthermore, if the number of parameters (the grade of the polynomial in this case) is greater or equal to the number of data points, we obtain that an infinite number of different curves are suitable to fit our data, and finding the best one, which performs well on the test dataset is an hard task. We should therefore pay attention when choosing the number of parameters on a model to avoid under- or more likely over- fitting.

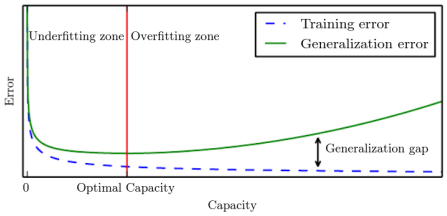
Regularization strategies Some regularization procedures are often implemented to avoid overfitting, the strategy is to build a model with a capacity slightly higher than the necessary in order to perform well on the training dataset, and then, to avoid overfitting, implement some regularization techniques to achieve good generalization performances.

Some of the most popular are Dropout and Batch-Normalization

- Dropout is a strategy which allows us to obtain the same effect of training train different model with our input data and take the average output at the end without paying high computational costs. It is usually applied to hidden layers' neurons and accomplish this task by randomly dropping a certain fraction of hidden neurons and their



(a) Underfitting, overfitting and appropriate fit in a 2D dataset: three models with different capacities are trained on blue data, if the model is too simple, in this case if the fit function has a few free parameters, it is not able to properly fit data, on the other hand, if it has too much parameters in respect to how much we would need to fit our data, it perfectly match train data distribution, passing through each data point, but performs very poorly on the test dataset (orange points). Whit an appropriate number of parameters, (middle figure) the model is able to fit train data, and generalizes well to Test data.



(b)

Figure 3.4: General, qualitative trend of a loss function vs. capacity of model: the more capacity it has, the more is able to reduce the error on train set (blue dotted line) with a tendence to overfit train data. Beyond a certain capacity the error on the test dataset (red line) increases because overfitting of training data results in a lower ability to generalize to new data.

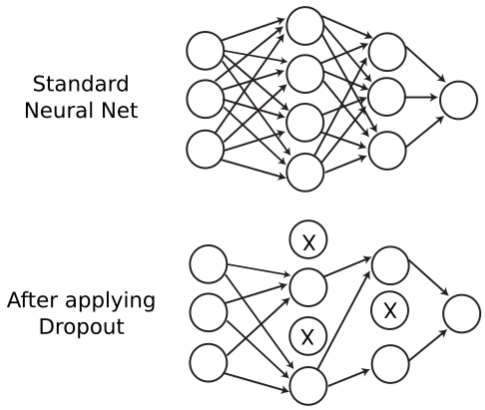


Figure 3.5: Schematic representation of a 0.5 dropout procedure: in a standard neural network, each neuron is linked to all the others, but if we implement a 0.5 dropout, for each iteration, neurons on hidden layers have a 50% probability to be dropped and excluded from the computation of outputs

connections, during each training cycle. The percentage of neuron to drop, correspons to the probability p for a single neuron to be dropped and it is set by the user. A visual representation of what occurs is illustrated in figure 3.5. When discarding some neurons in a layer, the remaining neurons needs to rescale their weights to account the missing connection; doing so, every neuron cannot rely on the the input of all the preceding neurons and the network is forced to learn more robust patterns from the data.

- BatchNormalizzation is a regularization scheme that has been commonly adopted since its introduction in 2015. It is based on the observation that a neural network works and performs better when its input are normalized because this prevents the saturation of its neurons. A neuron can in fact saturate and settle to a certain value because of an high input, this causes the neuron outputs value to be always close to the asymptotic end of its activation function, resulting in a biased and less accurate prediction. What is essentially done is then a simple scaling of each neuron's input: for a layer l with d neurons its input $\mathbf{x} = \{x_1^l, x_2^l, ..., x_d^l\}$ is normalized by removing the mean value across all the input data, and divide for their variance $x_i^l \rightarrow \tilde{x}_i^l = \frac{x_i^l - \mathbb{E}[x_i^l]}{\sqrt{Var(x_i^l)}}$

Cross validation procedures To monitor the evolution of the model's performances during train is a common practise to split the train dataset into two subsets: one that will be the actual train dataset, and a smaller one, called validation dataset used to make constant checkups on how well the model is learning with the train set, by making constant test of its performances.

The validation dataset is used for the fine tuning of hyperparameters and it typically consists of 10-30% of the whole train dataset. When assessed the best hyperparameters combination for our model, is possible to actually train the model using the entire Train

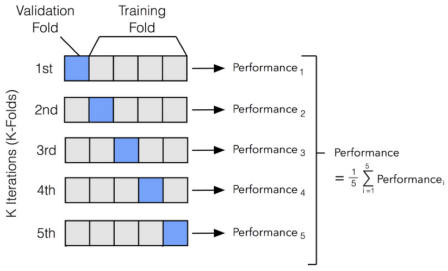


Figure 3.6: Schematic representation of a k-fold cross validation procedure with k = 5: Dataset is partitioned into 5 subsets, four of which are used for training and the remaining one for the testing dataset. during each iteration a model is trained over 4 different folds and tested on the remaining one, untill each fold has been used at least once as part of training and once as test.

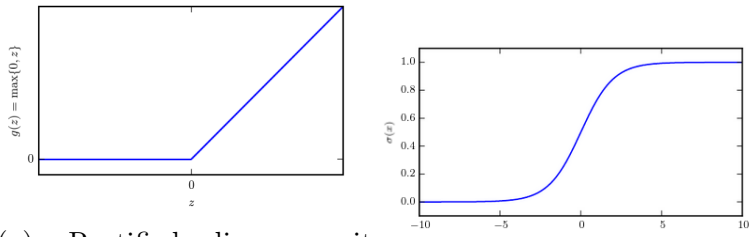
dataset. Even if we manage to use the entire Train dataset for the training process, However, when dealing with small datasets, dividing it into Train and Test can be a non trivial issue because of the shortage of data for a proper train procedure.

To work this out, a prodecure called k-fold cross validation can be implemented, at the cost of increasing computational costs. It consists in the creation of k different partition of the main whole dataset (before splitting it into train + test). From these partitions, k-1 are used as train, and the last is used as Test. Moving forward every combination of partitions is used so that each different partition is used as test, and the others as train. Then for each run k-th there are two subsets (a train and a test) from the original dataset. This way, for each iteration, the Train dataset is different, and the model is tested on a different dataset each time. In practise what is usually done, is not a sequential partitioning the dataset, to avoid creating folder containing all the same label if the dataset was ordered, but randomically picking data in order to create subsets containing the same proportion between classes as the main dataset. After that, we can imagine this process like the creation of k different models, each one trained on a different partition(k-1 folds) and tested on the remaining k-th fold, and we take as a result the average score across all these models.

3.3 Activation functions

The activation function of a neuron, and consequently of a layer defines the output of each neuron belonging to that layer, there are different activation function, linear or non-linear. A linear activation function outputs a value $f(x) = w^T x + b$ where w^T indicates the transpose of the weigths vector. In practise, however, it would not be very useful to introduce a linear activation function since we aim to introduce non-linearity in our model to tackle more complex models. There are several non-linear function available and, depending on what data we are working on, or on what kind of classification task we are performing, some of the most popular are

- The ReLU functions $\phi(z) = max(0, z) = \begin{cases} 0 \ for \ z < 0 \\ z \ for \ z > 0 \end{cases}$ which returns the maximum



(a) Rectified linear unit (ReLU) function: it returns zero if its argument is negative and returns the argument itself if it is positive.

(b) Sigmoid function: returns a value between 0 and 1, according to equation 3.2

Figure 3.7

value between the input and zero, it essentially puts to zero all negative inputs and leaves the positives untouched. A modified version of the ReLU function called Leaky ReLU was introduced defined as $\phi_{leaky}(z) = \begin{cases} \alpha z \ for \ z < 0 \\ z \ for \ z > 0 \end{cases}$ where α is a coefficient usually < 1, typically of the order of 10^{-2}

- The sigmoid function, or logistic function is defined as

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

and outputs a real number between 0 and 1. For this reason is a common choice when we have to predict a probability, for example in a binary classification problem it can be interpreted as the probability that output belongs to the class 0 or 1.

- The softmax function for multiclass classification, gives the probability of the input belonging to each (mutually exclusive) class. The softmax can't be applied independently to each output z_i, since it depends on all elements of $\mathbf{z}$ The probability of belonging to the i-th class over a total of M classes is $p(i = i|z) = \phi(z) = \frac{e_i^z}{\sum_{j=1}^{M} e^{z_j}}$ being M the total number of classes

3.4 Loss functions

In order to train a network and improve its performances we need to compare the predicted (output) value with the actual value and compute some sort or error, or distance between these two values, and try to gradually minimize it.

The function to minimize is denoted as the *loss function* Computing the loss, returns a value of how much the prediction is far from the actual label, and then, the network's weights are modified in order to reduce this value and searching for a minimum of this function.

and the main technique to do so, is the likelihooh maximization

The most common loss function is the cross-entropy loss, it relies on the concept of cross-entropy between two distributions $\hat{y}$ *and* y and is defined as $H(y, \hat{y}) = -\sum_i^n y_i \cdot log(\hat{y}_i)$ Where

the sum is intended over all the n possible values a variable y can assume (all the n possible classes in a classifiction problem). If we only have two classes, we are performing a binary classification and the loss becomes, (if we explicit the sum for two $i = \{0, 1\}$ and take the average value across all the data samples) we obtain the binary crossentropy:

$$J = -\frac{1}{N}\sum_{j=1}^{N} y_j log(\hat{y}_j) + (1 - y_j) log(1 - \hat{y}_j) \tag{3.3}$$

The estimation of the minimum of this function can be seen as the maximization of the likelihood: Given a probabilistic model depending on m different parameters $\theta_1 ... \theta_m$, the likelihood is the probability of observing a value $\hat{y}_i$ given a set of parameters $\{\theta_i\}_1^m$

$$L(\theta_1, ..., \theta_m | \hat{y}_i) = P(\hat{y}_i | \theta_1, ... \theta_m). \tag{3.4}$$

For a subset of observed value we can write the likelihood as $L = P(\hat{y}_1, ..., \hat{y}_n | \theta_1, ... \theta_m)$.

This probability, if we are dealing with independent variables, can be rewritten as the product of the single probabilities, of observing each sample $\hat{y}_i$ given out model with parameters $\{\theta_i\}_1^m$:

$$L = \prod_{i=1}^{n} p(\hat{y}_i | \theta_1, ... \theta_m). \tag{3.5}$$

In our practical problem if we aim to maximize this function, the parameters to be optimized $\{\theta_i\}_1^n$ are the weights of the DNN model $\{w_i\}_1^n$ and our data is a set of N datapoints $\{x_1 ... x_N\}$ each one associate with a label $\{y_1 ... y_N\}$ which can be either 0 or 1 which can be indicated as $D = \{(\mathbf{x_i}, y_i)\}$.

From now on we will denote our parameters to optimize as $\mathbf{w} = w1, .., wm$ Given an input data $\mathbf{x_i}$ and an activation function (sigmoid for example, which we recall is of the form $\sigma(z) = \frac{1}{1+e^{-z}}$) we model the probability of x_i to belong to the class $y_i = 1$ as:

$$P(y_i = 1 | x_i, \mathbf{w}) = \sigma(\mathbf{x_i} \cdot \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{x_i} \cdot \mathbf{w}}} \tag{3.6}$$

And since we are dealing with a binary classification, where y_i can be either 0 or 1, the probability to belong to one class is 1 minus the probability to belong to the other class

$$P(y_i = 1) = 1 - P(y_i = 0) \tag{3.7}$$

Putting this all together for a set of data $D = \{(\mathbf{x_i}, y_i)\}_1^N$ substituting this result in the log-likelihood function we obtain the likelihood of my observations $\hat{y}_i = \sigma(x_i | w)$, from equation 3.5 becomes

$$L = P(D|\mathbf{w}) = \prod_{i=1}^{N} \sigma(\mathbf{x_i w})^{y_i} (1 - \sigma(\mathbf{x_i w}))^{1 - y_i} \tag{3.8}$$

The best parameters are those that maximize the likelihood: $\begin{cases} \frac{\partial L}{\partial w_1}(w_1, ..., w_m) = 0 \\ ... \\ \frac{\partial L}{\partial w_1}(w_1, ..., w_m) = 0 \end{cases}$ To compute the derivative of a product is a nontrivial task, hence is much simpler to compute

the logarithm (since the logarithm is a monotonic function, so compute the maximum of a function is the same as computing the maximum of the logarithm of that function).

We obtain then

$$log(L) = \sum_{i=1}^{N} y_i log(\sigma(x_i \mathbf{w})) + (1 - y_i) log(1 - \sigma(x_i \mathbf{w})) \tag{3.9}$$

which taken with a negative sign, and averaged over all the N data samples corresponds to the cross-entropy function in equation 3.3.

This result obtained for binary classification can be generalized to perform a multi-class classification called Softmax regression. In a multi-class classification, labels y_i become binary vectors of dimension M, with all but one entry equal to zero, and the only entry equal to 1 specifies the class.

In this case, given a model with parameters $\mathbf{w_k}_{k=0}^{M-1}$ the probability for an input $\mathbf{x_i}$ to belong to class m' is given by the softmax function

$$P(y_{i,m'} = 1 | \mathbf{x_i}, \{\mathbf{w_k}\}_{k=1}^{M}) = \frac{e^{-\mathbf{x_i^T w_{m'}}}}{\sum_{m=1}^{M} e^{-\mathbf{x_i^T w_m}}} \tag{3.10}$$

and the relative negative log-likelihood called the **categorical crossentropy** will be in the form:

$$L = -\sum_{i=1}^{n} \sum_{m=0}^{M-1} y_{im} log(P(y_{im} = 1 | x_i, w_m)) + (1 - y_{im}) log(1 - P(y_i m = 1 | x_i, w_m)) \tag{3.11}$$

where there's a summation over the index i which indicate the i-th sample data and over the index m, concerning all the possible M classes.

3.5 Gradient descent and Backpropagation

To perform the minimization of our loss function, even if it'd be theoretically possible to find a minimum by means of an analythical way, in practise, in all the neural network applocation, the number of weigths is so big that a numerical methods must be employed, the most popular of which is the *gradient descent*.

Let's denote the loss function as $J(w) = \frac{1}{n} \sum_{i=1}^{N} E(y_i, \hat{y}_i)$ Given a generic loss function $J(\mathbf{w})$ where $\mathbf{w}$ is a vector of weights, the minimum of this function corresponding to the vector w_0 can be found by following these simple steps:

1. Choose a random initial guess for w_0 and start iterating
2. At iteration i+1 we have a weights vector w_{i+1} given by the formula $w_{i+1} = w_i$ - $\eta \nabla$ J(w_i)

where the ∇ indicate the gradient of the cost function with respect to w_i components, and η is the learning rate of our gradient algorithm, it specifies how big each step will be during the descent toward the minimum of the cost function.

A drawback of this gradient descent algorithm is that before upgrading the weiths we need to compute all the gradients for each data in input data namely, after the whole dataset is been seen, resulting in a huge computational cost.

An optimized version of this algorithm calle dStchastic Gradient Descent (SDG) is then often used because it has some advantages such as decrease computational cost and, as the name suggests, introduce stochasticity resulting in less chance for this algorithm to get stuck in a local minimum. It works by approximating the gradient of the cost function calculated with all the input data, with a gradient computed using only a small subset of input data called minibach.

If we have N input data, we can create subsets containing m elements, and obtain N/m subsets. Therefore, the gradient is computed on a mini-batch and weights are updated and this process is repeated for each mini batch. Once the gradient was computed over all the mini batches, this is called an epoch, which is one of the hyperparameters to be set when choosing a model and a training strategy.

Even though SGD already performs quite well, it can be further tuned introducing the concept of momentum. Momentum is represented by a parameters $0 \leq \gamma \leq 1$ and it takes track of the discending direction by running an average over all the preceding encountered gradients, and helps the algorithm speeding up the descending process if in a certain direction the gradient is persistent. But in addition we can take into account even the steepness all over the dimension, and to do so we must introduce second order momenta in our algorithm, also called uncentered variance.

To accomplish this, we would ideally need to to calculate the hessian matrix but this comes at the cost of increasing computational cost. Recently intriduced algorithm can accomplish this task by approximate the calculus of the second momenta. Doing so, we keep track of the curvature and take big leaps in steepest direction and small steps in flatter ones, allowing us to adaptively change the descending step size according to the shape of our multi-dimensional curve. One of the most popular among these algorithms is Adam: it accomplish this computational task by making use of two different optimization algorithms: AdaGrad and RMSProp. and adapts the learning rate taking into account both the first and the second moment, leading it to perform better and quicker in findind the minima than simple SGD with momentum algorithms.

After the computation of the loss value, the next step is the process called backpropagation, through which network's weights are updated. Backpropagation can be summarized in 3 steps:

1. calculate all the activation of each neuron in a layer
2. calculate the error of the output layer L, which for each neuron will be $\Delta_j^L = \partial E / \partial z_j^L$ being z the output of this neuron
3. Backpropagate the error exploiting the chain rule of derivatives to compute the error of every neuron in the previous layer l $\Delta_j^l = \sum_k \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$
4. Compute the gradient of the function E with respect to model's weights and modify them using equation ..

3.6 Dimensionality reduction: PCA

PCA is a method to perform dimensionality reduction, it is an unsupervised learning algorithm which aims to reduce the dimensionality of input data, preserving as much information as possible, It does so attempting to learn a new representation of data with lower dimensionality than the initial one and whose element have no linear correlation with each other. It performs then an orthogonal transformation of data, in order to find the direction along which data variance is biggest.

Concretely, if we consider a set of n input data vectors lying in a space $\Re^p$, we can represent them as a matrix $X \in \Re^{n \times p}$ where n is the total number of input data and p is the dimensionality of each data (the number of features in each data vector).

We can suppose without losing generability that feature distribution across our data have zero mean, so that for each feature x_i with i = 1, ..., p, $\mu_i = \mathbb{E}[x_i] = 0$.

PCA computes the covariance matrix $\Sigma \in \Re^{p \times p}$ given by $\Sigma(X) = \frac{1}{n-1} X^T \cdot X$, where each entry can be written as $\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^{n} (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$, and if we are in the hypotesis of zero mean $\mu_j = \mu_k = 0$, we obtain $\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^{n} (x_j^{(i)} x_k^{(i)})$.

Since the covariance of a variable with itself is its variance (Cov(a,a)=Var(a)), in the main diagonal we actually have the variances of each initial variable. And since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

We are interested in finding a new representation performing a linear transformation that reduces the covariance between different features.

The eigenvectors of the principal components represent the direction of the maximum variance and the corresponding eigenvalues, defines their magniture.

To find them, PCA makes use of Singular Value Decomposition (SVD): a linear algebra analysis which is basically a factorization of a n x p matrix X that (in a case X is a real matrix), to rewrite it as X = USW^T where U and V are respectively m x m and n x n orthogonal matrices whose columns are called left- and right-singular vectors of X, and S is a m x n diagonal matrix. Diagonal values of S: $S_{ii} = s_i$ are unically determined by X and are called singular values of X.

According this analysis, we can rewrite

$$\begin{aligned} X^T X &= (USW^T)^T (USW^T) \\ &= WS^T U^T USW^T \\ &= WS^2 W^T \end{aligned} \tag{3.12}$$

where we used the definition of orthogonal matrix for U $U^T U = \mathrm{I}$.

We therefore rewrite the covariance matrix $Var[X] = \Sigma$ as $\Sigma = \frac{1}{n-1} WS^2 W^T$.

Here the singular values of X: λ_i are related to eigenvalues of the covariance matrix by the relation $\lambda_i = s_i^2/(n-1)$

Using this results if we compute the new data matrix Z = X^T W, its covariance matrix

$$\begin{aligned} Var[Z] &= \frac{1}{m-1} Z^T Z \\ &= \frac{1}{m-1} W^T X^T X W \\ &= \frac{1}{m-1} W^T W S^2 W^T W \\ &= \frac{1}{m-1} S^2. \end{aligned} \tag{3.13}$$

This shows that when we project the data x to z using the linear transformation W, the resulting transformation has a diagonal covariance matrix which implies that the individual elements of z are mutually uncorrelated.

To reduce the dimensionality of our data, we need to extract the first $\tilde{p}$ eigenvalues from the covariance matrix, ordered in a descending order, and collect the corresponding projection matrix $W_{\tilde{p}}$ made of the corresponding vectors of the righ-singular vectors W.

And the projection of our data from p-dimensional space into $\tilde{p}$-dimensional space is $\tilde{Z} = X\tilde{W}_p$

An important parameters to quantify "how well" PCA is able to explain is the *variance explained ratio*, given by the ratio of an eigenvalues and the sum of all the eigenvalues.

$$\frac{\lambda_i}{\sum_{k=0}^{p} \lambda_k} \tag{3.14}$$

The singular vector with the largest corresponding singular value is the first principal component, and with decreasing order, the other mutually orthogonal vectors are the next principal components. In figure **??** is reported a graphic example of two PC extracted from a set of two-dimensional data, along the direction where variance is greater, there's the first principal component and the second component is in the orthogonal direction.

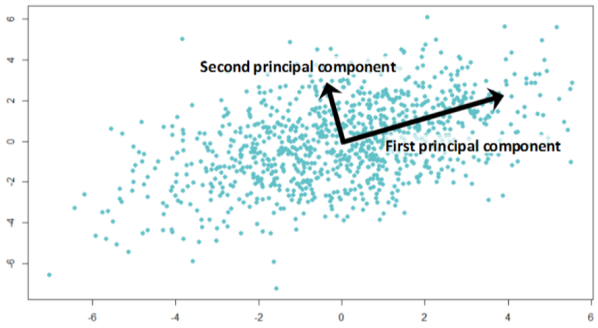


Figure 3.8: Example of the first two principal components in a 2D dataset

3.7 How to assess network's performances

Once the model has been trained, it is finally applied to the test set to assess the performances of the model to an previously unseen dataset. A comomon metric for model performances'

evaluation is Accurcay: it is simply defined as the ratio between the total number of correct prediction and the total number of predictions (total number of the test's data), so, using T and F for true and false, and P and N for positive and negative, accuracy is so defined

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.15}$$

Even though accuracy may seem a good parameter, it isn't the most accurate one, when we're dealing with a unbalanced test dataset, if for example, in a binary classification with A and B classes, if just 2 out of 10 samples belongs to a different class, let's say A and B, and the model predicts every data belonging to the class B, we will get a score of 80 % accuracy, which is not a thruthful result because the model isn't making any distinction between the two classes. There are then other ways for quality assessment that overcome this problem. One of this is based on the introduction of two quantities: precision and recall. Precision is the ratio between true positive and total positive cases (TP + FP), which is a measure of how many positive predicted cases were actually positive, while recall measures the percentage of actual positives that were correctly classified, or in other words it represents the true positive rate.

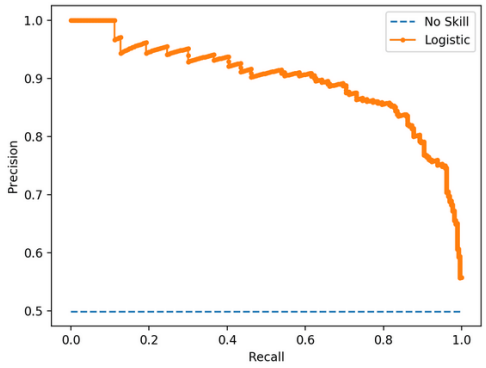
$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{3.16}$$

Unfortunatelly precision and recall are linked in a sense that often enhancing one decreases the other and vice-versa. There's therefore a trade off between recall and precision. If we want for instance a model to score more true positive, risking to have more false positives, we can set a threshold. To reach less the x% positive missing means set the recall to (100-x)%, this operation is called setting the operating point and is done by setting a threshold. This threshold though is not always estblished at the beginning, and operating point that would be the best suited for our classification goals is not clear a priori. One way to summarize the information a precision-recall curve can provide us, is to compute the area under the curve, known as "average precition "What is done is study the model under all the possible thresholds. This is achieved by studying the precision-recall curve of which is reported an example in figure 3.9a. The closer a curve lies on the upper right corner (high precision and high recall), the more correctly the model is working.

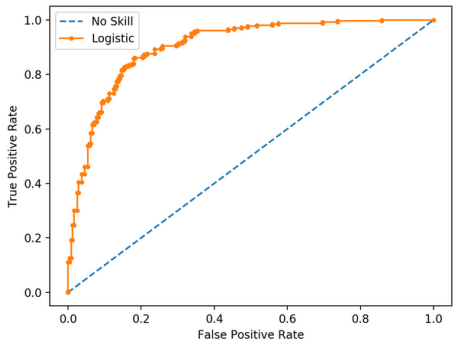
Besides the precision-recall curve, a similar curve is usually employed to study different thresholds' effect, is called the receiver operating characteristics curve, usually referred as ROC curve. The ROC curve is build from the true positive rate (recall) and the false positive rate (FPR) and shows the evolution of TPR vs FPR. The ideal curve would be close to the top left (high tpr and low fpr) and the less accurate is the model, the more this curve tend to lay down to the bisector line.

To summarize the model's performances with a single number using ROC's curve information, we compute the Area Under the Curve AUC. The reference value for an AUC is 0.5 which is obtained when a model is just randomly predicting and it corresponds to a curve laying on the bisector. The AUC can be explained as the probability that randomly picked point from the positive class, will have an higher score (according to the model) than a randomly picked point from the negative class so that the percentage value of AUC is an estimate of the probability that the model is able to distinguish between the two classes.

An example of ROC curve is shown in figure 3.9b



(a) Example of a precision-recall curve plotted for different thresholds values



(b) Example of a ROC curve values of True positive and False positive rates for different thresholds values

Figure 3.9

Chapter 4

Explainable AI (XAI) with SHAP

As methods to learn pattern from data becomes more complex, they become harder to interpret, deep learning represent an example of a technique to search for nonlinear relations between data, but introducing nonlinearity, makes an outcome difficult to interpret and it ends up considering a machine learning model as a black box without any hint of its inner precesses that occurred when a result is output. But if a model perform well, or bad on a dataset, we would like to know why and what feature influenced the most the outcome of our model.

Figure 4.1

SHAP (SHapley Additive exPlanation) is a technique based on game theory wich provides a method for model explanability. is a powerful tool to get rid of the "black box "idea of a machine learning model and try to understand its deep mechanism and the reasons why a model gave a certain output, or a certain prediction related to an input sample like a vector of feature or an image for example. As is shown in figure 12.1 just as a visual example, a common machine learning model acts like a black box that returns an output value after some non-linear unknown calculations on some input values; with an explanatory model, we are able to quantify the contribution of each feature of our input data and assess what and why are the most important features and how much they contributed to the final outcome.

4.1 Shapley values - Theory

Game theory is a branch of mathematic related on the study of mathematical models to conceive social situations among competitive players [1] It have had a great development in the XX century expecially during and after the Second world war thanks to the contribution of mathematicians like John Von Neumann, John Nash and Lloyd Shapley. Game theory can be mainly divided into two main classes namely cooperative and non-coperative game. Non-cooperative game concern competition between individual player and the main task is to find a good strategy for each player and a big contribution came from Nash with the concept of Nash equilibrium [6]. Cooperative games concern competition between groups of player forming coalitions; one of the the main tasks is to find a way to divide the total utility among the member of a coalition in an equally way proportionally of how much they contributed to the final score. In 1951 Lloyd Shapley in 1951 Introduced a way to compute the exact amount of payoff for each player making use of what were named after him: Shapley values [**?**][**?**]. Shapley introduced its values for coalition games:

A coalition games involves N players, and different subsets called coalitions created from these N players. Each subset of players S gain a payoff at the end of the game. A function ν maps every subset to its payoff $\nu(S) = \text{payoff(S)} \in \Re$. Based on which player had more influence in this final score, we ask how to split this payoff in a fair way between each player of the subset, where "fair "is to be intended as to each, proportional to his own contribution. A solution for this problem comes from **shapley values** $\phi_i(\nu)$. They are specific for each player i $\in$ N in a coalition S $\in$ N and represents the marginal contribution of that player to the final score, where marginal contribution is defined as the difference on the score of the coalition when player i joins the coalition. In other words they are the difference between the coalition's score with player i and the group's score without player i $\nu(S \cup \{i\}) - \nu(S)$.

It represents the incremental benefit of including player i in a the subset and average over all subsets that include player i.

The mathematical formulation Shapley introduced for his value is

$$\begin{aligned} \phi_i(\nu) &= \sum_{S \subseteq N \setminus \{i\}} \binom{N-1}{|S|}^{-1} [\nu(S \cup \{i\}0) - \nu(S)] \\ &= \sum_{S \subseteq N \setminus \{i\}} \frac{S!(N-S-1)!}{N!} [\nu(S \cup \{i\}0) - \nu(S)] \end{aligned} \tag{4.1}$$

And they exactly represent the amount of reward for each player i.

4.2 SHAP

The idea behind SHAP is to use these Shapley values to exaplain every single feature's contribution in our machine learning model, treating it as a cooperative game

To apply the concept of Shapley values to a machine learning model we just adapt some terms we used for game theory: the payout is the model's prediction and the players are the

[1] https://plato.stanford.edu/entries/game-theory/

feature in our input data. For a single feature, its Shapley value is defined as the average marginal contributison of the value of that feature across all possible coalitions.

Shap can be regarded as a local explainatory method, and before it, several algorithms belonging to this cathegory were implemented as an attempt to create an explainatory technique for machine learning models. In general, local explainatory model's objective can be defined as the attempt to explain an output f(x) after a single instance x that in our case can be a vector of features. Local methods differs from global methods, because the latter provide us a global explanation of the model, across all the instances, they are able to attribute an importance to each feature to determine which contributed the most to the output of the model. One of the key point of SHAP is its flexibilityu, being both a local and global explainatory model.

Being a local model, SHAP share with other algorithms some implementation features: given a single input vector x, to simplify the workflow, they introduce a coalition vector x' = $\{0,1\}^F$: where F is the total number of features: x' is a binary vector of the same length of x, where binary means made just by zeros and ones: zero means that in our analysis we are going to withheld the corrisponding variable from x in, and the one that we are including it.

To map the x' vector to the corresponding feature values vector a mapping function $h_x(x')$ = x is introduced to returns the actual features' value from the corresponding input vector x. Our goal is to find a simplified model g(x') to work with, because is simpler to work with a binary vector x' than with the actual vector x. We want that this simplified model, when applied on the coalition vector x', is able to approximate our output f(x): $g(z') \approx f(h_x(z'))$ if z' $\approx$ x'. The first important condition on this function g is that it has to be a linear function

$$g(z') = \phi_0 + \sum_{i=0}^{N} \phi_i z'_i \tag{4.2}$$

And several methods before SHAP were created that satisfy this condition such as LIME [12] or DeepLIFT [14] but they lacked additional properties that SHAP, proposed and implemented by Lundberg [**?**] has:

1. Local accuracy:
$$g(x') = \phi_0 + \sum_i \phi_i x'_i = f(x) \tag{4.3}$$
when x = $h_x(x')$ and not just an approximation $f(x) \approx g(x')$. Here ϕ_0 represents the coefficient corresponding to a vector x' with all entries equal to zeros.

2. Missingness:
$$x'_i = 0 \Longrightarrow \phi_i = 0 \tag{4.4}$$

3. Consistency: if we have two different functions g' and g, indicating eith z'/i the setting where $z'_i = 0$, if
$$g'(z') - g'(z'/i) \geq g(z') - g(z'/i) \Longrightarrow \phi_i(f', x) \geq \phi_i(f, x) \tag{4.5}$$

In 1975 was demostrated that the only coefficients satisfying all of these properties are Shapley values, and methods which employ different coefficients usually violate local accuracy and/or consistency. For these reasons SHAP employs Shapley values into pre-existing

algorithms and enhances them providing an unified aproach to assess feature importance for different models without any violation of the axsioms above.

To compute the Shapley value for a feature we have to evaluate the model on all the possible subsets S we can create with our data S$\subseteq F \setminus \{i\}$. That in a problem with an high number of features would result in a huge computational work. To bypass this problem, we can choose not to calculate the exact shapley value but just an approximation of it, depending on the model we are working on, SHAP implemented different algorithms such as Tree SHAP, a fast and exact algorithm to compute SHAP values for trees and ensembles of trees, KernelSHAP which is a model agnostic explanation method, or DeepSHAP which is optimized to work faster on deep models by making use of the knowlegde of the structure of a neural network. Each one of these algorithms is built upon previous algorithms like LIME or DeepLIFT

LIME is a local, model-agnostic, interpretability model, made to explain the prediction of ant classifier in faithful, but only local way, meaning that it can accurately explain a single prediction but it is not able to generalize to many. Since Kernel SHAP is built upon LIME algorithm, it pick up an important strategy from it, to avoid the computation of all the permutaiton KernelSHAP adopts an aproximation methods based on the introduction of a perturbed dataset: starting from a single instance x, it randomly creates a perturbed data by randomly setting to 0 or 1 the corresponding entries on several coalition vectors x', doing so it creates a perturbed dataset made of different x' vectors.

The next step is to make prediction for all of these perturbed vector. Since x' and x are related through the mapping function h, for each x'we can retrieve the corresponding values from x and input this vector to our model and then uses the pre-trained model to make predictions on this perturbed dataset.

As reported before, 1 means that we are including the corresponding feature in this analysis while 0 is associated to a lack of the corresponding feature. Since not all the machine learning models can easily deal with lacking features, to simulate the lacking of a feature, when we have a 0 in the x' vector, the algorithm replaces the corresponding feature's value with the most uninformative value. This is chosen using a set of data from training called *background* datset, of arbitrary size, from it, features are averaged, and the this value is replaced in our vector to simulate the absence of that feature.

Using this synthetic dataset, the model is evaluated on each vector and these scores are collected, and these outputs are subsequently used to accomplish a minimization task to finally find the shapley coefficients. Minimization is performed in a fit, but before it, each output value is weighted according to the distance of the synthetic vector with the original space vector by using coefficients π_x defined as

$$\pi_x = \frac{p-1}{\binom{p}{|z'|} \cdot |z'|(p-|z'|)} \tag{4.6}$$

Where we indicated as p the total number of features in the original feature vector x anc with —z'— the number of element in that subset, and cosequently (p-—z'—) is the number of features not included in the subset.

To perform a fit, the loss function to minimize is in the form $L(f, g, \pi_x) = \sum_{z \in Z}[f(h(z)) - g(z')]^2\ \pi_x$ where g(z') is the simplified function expressed in equation 4.2, given by a combination of coefficients ϕ_i that are the parameters of the fit.

In a nutshell, the main steps of KernelSHAP algorithm are

1. Given a feature vector $\mathbf{x} = [x_1, x2, ..., x_p]$, where $x_1, .., x_p$ are the features (numbers), a model f and a background set $X_{bckg} = \{\mathbf{x1}, ..., \mathbf{xn}\}$, we want to explain $\mathbf{x}$

2. replace a subset of entries of x with values from the background dataset:

3. create n different copies of the original input fector x, all slightly different from each other because of these replacements

4. Evaluate the model on each copy

5. Use these outputs to fit simple model and find shapley coefficients

As mentioned above, KernelSHAP is a model-agnostic algorithm, in the sense that can easily used with every kind of model, but SHAP includes some model-specifics algorithms that make use of a previous general knowledge of a model's structure to optimize the explainer's performances and speed up the process. DeepSHAP, is one of these model-specific algorithms and it is what we are going to use in our analysis.

DeepSHAP alghorithm works with them same principles of KernelSHAP, making use of a background dataset to simulate missing values, but it is also optimized to perform better on deep models: it is in this sense, an enhanced version of the DeepLIFT algorithm because of the introduction of the background dataset and because of the use of Shapley values rather than DeepLIFT values.

DeepLIFT aims to compute the difference between the output of our model with our data and a reference value computed as the output of our model with some reference data. The choice of the reference data depends on what kind of data we are working on: images, or genomic data for example. for genomic the most common way to produce reference output is to shuffle some inputs, evaluate the model on them and average across all the scores. It is also shaped to perform on deep learning algorithm computing its DeepLIFT value during the backpropagation process. DeepLIFT attributes to each feature x_i a value $C_{\Delta x_i, \Delta y}$ that represent the effect of that input being set to a reference value rather than its actual value. This reference value is chosen from the backgound dataset and represents an uninformative value for that feature

In a nutshell, let t represent the output of some inner neuron and let x1, x2, ... , x_n be some preceding neurons necessary to compute t, if we label t_0 the reference output, we can compute the value $\Delta t = t - t_0$ and use them to define the DeepLIFT values $C_{\Delta x_i \Delta t}$ so that $\sum_{i=1}^{n} C_{\Delta x_i \Delta t} = \Delta t$. For a given input neuron x with difference from reference Δx and a target neuron t, is defined a multiplier $m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x}$ which represents the contribution of Δx to Δt.

For a more detailed explanation of how DeepLIFT works we refer to its presentation article [14]

DeepSHAP combines shap values for smaller components of a network to compute values for the whole network, it does so recursively passing Deep LIFT multipliers during back-propagation: we can indeed re-writing multiplier values in terms of shap values, for a target neuron f3 we have $m_{x_j f_3} = \frac{\phi_i(f_3, x)}{x_j - E[x_j]}$ obtaining $\phi_i(f_3, y) \approx m_{y_i, f_3}(y_i - E[y_i])$

To express the significance we can rely on the relation between feature importance in a qualitative way and shap value: feature with high absolute shapley value are important. The absolute importance value for a feature f is calculated as the mean of the magnitude of all the shapley value for that feature, so the mean is performed across all the samples we used to calculate these values

$$I_f = \frac{1}{n} \sum_{i=1}^{n} \|\phi_i^{(f)}\| \tag{4.7}$$

Shap is implemented as a free Python package with MIT license, developed and mantained by Scott Lundberg [2]. This package includes different classes such as TreeExplainer tuned to perform rapidly on tree and forest-like models, linearExplainer which deals with linear model and is able to compute the exact shapley values and not just an aproximation, and the algorithm we are using DeepExplainer, suitable to explain deep learning models and to compute an approximate value of shapley values.

[2] https://github.com/slundberg/shap

MATERIALS AND METHODS

Chapter 5

Dataset: ABIDE I & II

Data we are going to work on belong to the ABIDE dataset (Autism Brain Images Data Exchange): a project founded with the aim of investigate autism using MRI structural and resting state fMRI scans, to address the problem with these two different aproaches, collecting data acquired over the years from different medical centers, and putting them together in a single dataset.

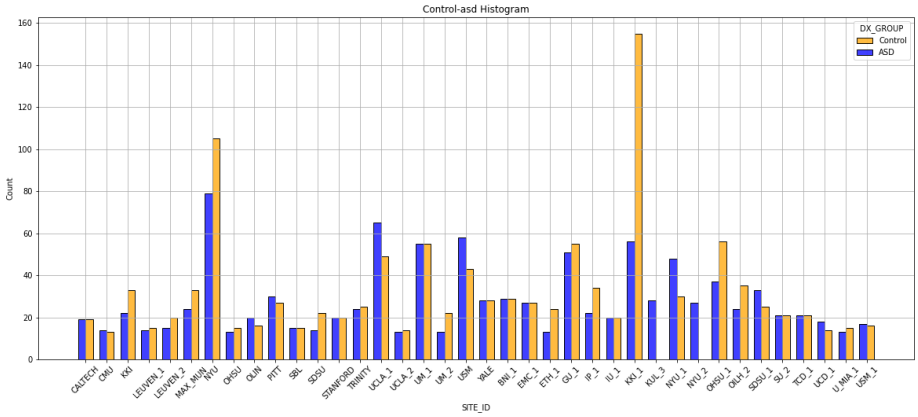
The whole ABIDE dataset was published in two releases: ABIDE I released in August 2012 and containing 1112 patient scans, and ABIDE II released in June 2016 containing 1114 scans. ABIDE I includes scans collected from 17 different sites, and the 1112 patients consist of 539 patients with ASD and 573 typical control patients. ABIDE II includes scans collected from 19 different sites, and the 1114 patients consist of 593 patients with ASD and 521 typical control patients. Not every site belonging to ABIDE II is different from those of ABIDE I, but, even though some medical centers are the same, the scanner type, or acquisition pipeline and parameters may have been changed during the time interval between the two releases, so in the following analysys, they are regarded as different acquisition sites. For each site autism was diagnosed either by gold standard diagnostic instruments, clinical judgment or a combination of clinical gold standard procedures.

In addition to scan images, ABIDE provides every information related to each patient, as age, sex, intelligence quotient (FIQ), eye status during the scan (open or closed), and every additional information provided by patients.

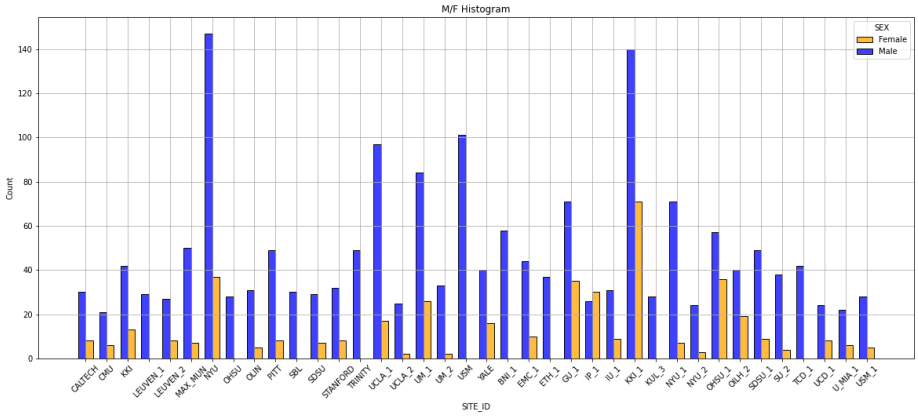
The vast majority of patients are males as shown in the histogram 5.1b, for a total amount of 1804 males and 422 females , while control/asd patient number, for each site is almost for each site as while the control/asd case per site is more or less balanced for almost every site, with the exception of KKI-1 that provided two times more controls than asd patients, and KUL-3 (Katholieke Universiteit Leuven) and NYU-2 (NYU Langone Medical Centre, Sample 2) that only provided ASD cases. For a visual comparison the number of controls/asd for each site is displayed on the histogram in figure 5.1a

Patients in ABIDE dataset have ages ranging from 4 to > 50 years old, but as shown in figure 5.3a the vast majority of participant are younger than 40, precisely > 97 % of participant are under 40 y.o., also, the majority of sites provide only young patients in a restricted age range, but as can be seen from figure **??** there are some sites that acquired patients with a wide age rande.

In our analysis we can restrict to patients belonging to an age range between 4 and 50.



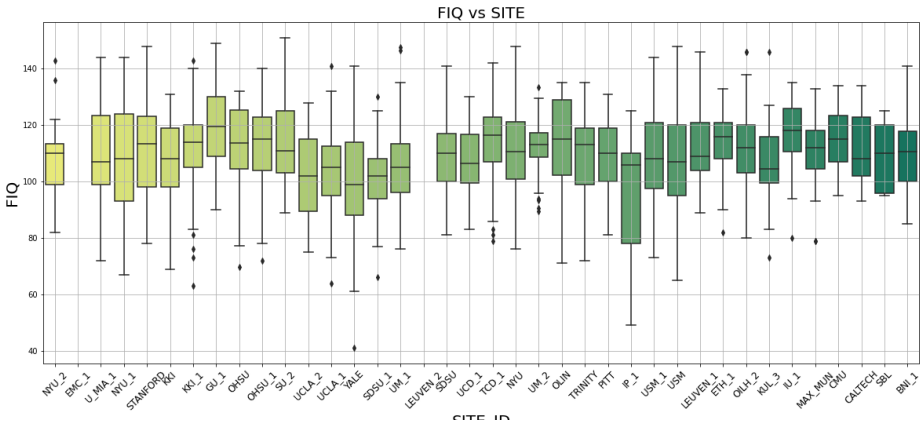
(a) Histogram of control and ASD subject per site from the whole ABIDE I & II dataset



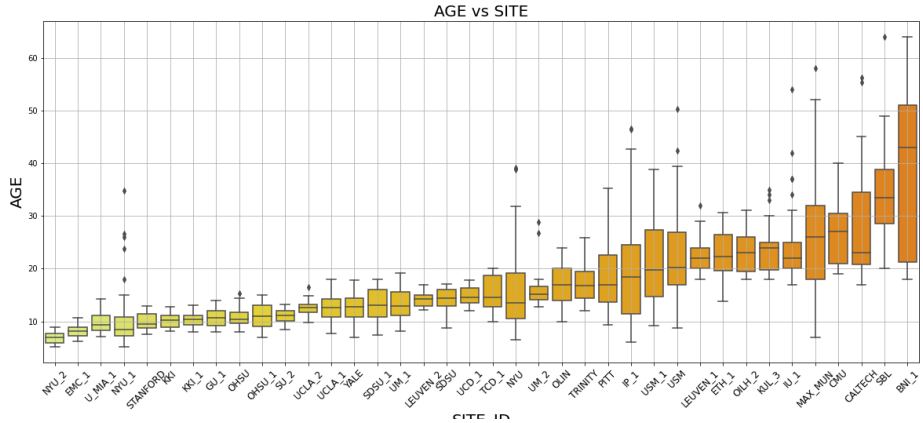
(b) Histogram of male and females subjects per site from the whole ABIDE I & II dataset

Figure 5.1

Intelligence quotient, whose distribution is shown in figure 5.3b, was not provided for every participant, in fact 171 patients out of 2226, coming from sites UM1 and EMC1 (figure **??**) lack this information. In our further analysis, before proceding these values were replaced by the average value of all the other provided values. The lack of a common acquisition protocol is also evident from the eye status at scan feature: as shown in figure 5.4b each site acquired scans either with open eyes or with closed, without a common method, and sometimes this information is not even specified. In its entirely, the whole dataset consists of more than 70% of patient acquired with open eyes, as shown in figure 5.4a.

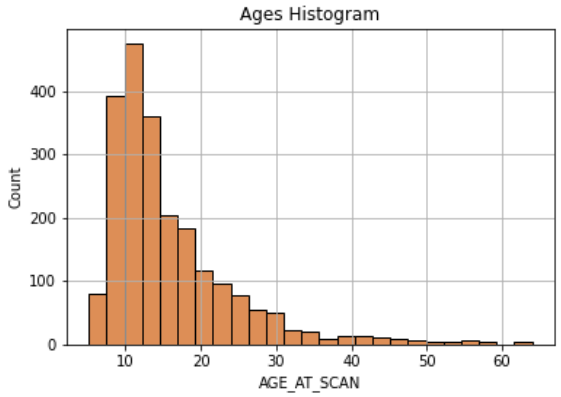


(a) Boxplot of patient's FIQ (together controls and ASDs) per site

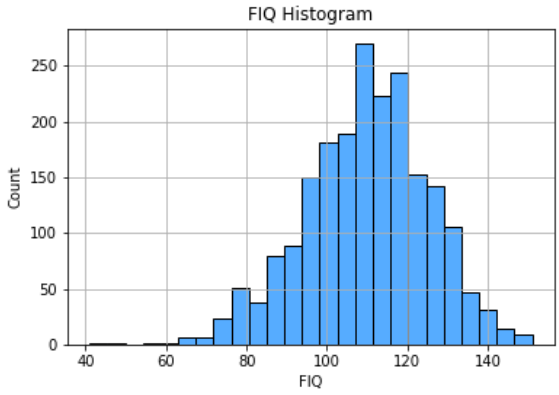


(b) Boxplot of age of patients per site

Figure 5.2

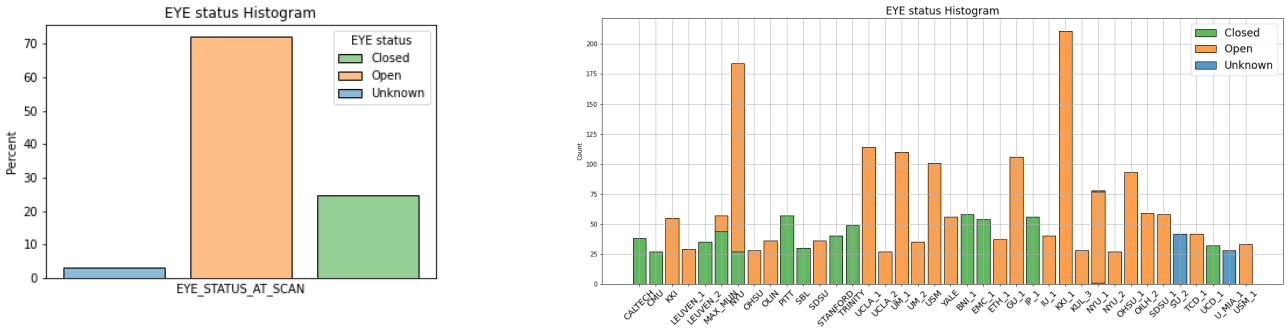


(a) Age distribution of the whole ABIDE I & II dataset



(b) FIQ distribution of the whole ABIDE I & II dataset

Figure 5.3



(a) (b)

Figure 5.4: Histograms of patients with open, closed and unknown eye status in the whole ABIDE I & II dataset (fig 5.4a) and per site (fig 5.4b)

Chapter 6

Image preprocessing

6.1 Common preprocessing steps

For a better undertanding of how MRI and f-MRI and signal detection work, we refer to appendix.. (talk briefly about echo time and TR ?)

During an MRI and f-MRI scan session, data are usually acquired slice by slice, and the thinner is the slice, the more spatial resolution we can accomplish. But there's a trade-off between spatial and temporal resolution: decrease the slice's thickness would lead to a better space resolution but at the cost of increasing repetition time to mantain the same Signal to Noise ratio (SNR). This is because the signal is proportional to the number of hydrogen nuclei, which is proportional to the slice volume.

In order to obtain a strong BOLD signal, echo time plays a significant role as well: to obtain the maximum strenght signal it has to be set $T_E = T_2^*$ so in the case of BOLD signal it should be around 30 ms. Images acquired using a shorter echo time have a weaker BOLD signal because of the lack of signal to detect. [**?**]

Acquired data suffer from different source of noise and artifacts deriving from both hardware and physiologigal sources. As an instance, breath rate can affect BOLD signal because of the induced local motion of the brain's vessels, or the change in blood oxygenation and pressure

One of the main artifact intrinsit to the signal nature is distortion and field inhomogeneities, deriving from making the scan sensitive to the BOLD signal which intrinsecally is the detection of a signal loss due to field distortion. This could be corrected by employing small coils inside the scan to smooth magnetic field differences, this process called shimming still left some artifacts, therefore common data preprocessing pipelines use extra acquisitions to create a field map of the remaining field inhomogeneitie and a shift and stretching voxel to correct for them.

Head motion is due to the physical movement of the patient inside the scanner. It results in a misalinement from one acquired volume to the next. Physiological noise is caused by the cardiac cycle and the breathing of the patient. They respectively provoke pulsatile motion of arteries, CSF and tissues and small head motion. As a second effect, the variable amount of air in the chest affect the magnetic field B_0. To correct for head motion, motion correction steps are performed at the beginning. It works by spatially applying transformations as rotation or translation volume by volume, aiming to overlap every acquired slice to a chosen

reference volume, like the first or that in the middle.

For EPI data, slice timing correction is usually performed as well. It aims to correct artifacts deriving from the sequentiality of acquisition for each slice of the brain is acquired at different time. The entire time elapsed to acquire all the slices is called repetition-time, and it usually is from 1 to 3 seconds. Slice timing correction uses interpolation in time to shift the BOLD timeseries of each voxel, in order to align them to a reference starting time. The use of interpolation, though can lead to a slight loss of high frequencies information.

A further common step is spatial smoothing of both structural and functional data. It operates calculating a weighted average of each pixel over neighbored voxels. To this end, a gaussian kernel with a chosen FWHM is applied to create the weights. Spatial smoothing is useful to avoid abrupt changes of signal between two neighbouring voxels.

Band-pass temporal filtering is commonly applyed to BOLD data, aiming to reduce artifacts from hardware like the slow changing in the baseline of the BOLD signal over time. A low pass filter removes high frequencies above a cut off frequency, it is commonly applied in processing resting state fmri data because the physiological signal is driven by low frequences oscillation while high ones are associated to noise.

A common further step is nuisance regression, which aims to reduce the structured noise: it works computing timecourses called nuisance regressors, These signals include motion and BOLD signal fluctuation from white matter or cerebro spinal fluid. From them, the variance is computed and this value is removed from the data using multi linear regression analysis.

When we need to run a group analysis, meaning analyse and compare different patient's images, one of the most relevant step is *registration*: structural (T1) data are aligned over a standard coordinates system space to universally describe location of the different brain parts, to make sure that the same voxel coordinate corresponds to the same brain area for all the subjects. The most common template, provided by packages like FSL is Tailarch and Monreal Neurological Institute's MNI-152 that replaced the previous Tailarch and Tournoux template and was adopted by the International Consortium of Brain Mapping (ICBM) as the international standard; it is also called ICBM152 for this reason.

To understand why MNI152 differs from the Tailarch and Tournoux we have to spend a few words about the latter. [1] The Teilarch and Tournoux atlas was published in 1988 and introduced some innovative aspects to tackle the problem of the great variability in brain anatomy between people which limited the accuracy of statistical analysis. they introduced a common coordinate system to identify different brain locations, based on some anatomical landmarks, and introduced a spatial transformations to match different brains. They choose two anatomical areas: the anterior commissure and the posterior commissure, which are relative invariant between different brains and conjuncted them with an axis. An horizontal plane passing through this axis was chosen such that it was perpendicular with the interemishpere axis. This way they created a 3D coordinate system called the Tailarch coordinate system. Afterwards, to match different brains they described a set of spatial transformation, one for each different brain quadrant, to transform a brain to match another's principal anatomical structures.

From this template, a first MNI template was created, called MNI305, created by manual scaling 241 brains to the Tailarch template and averaging them to obtain new template, and

[1] https://doi.org/10.1038/nrn756

then transform 305 additional scans to this new template and averaging them to create a second average and final template (MNI305).

From this template, MNI152 was finally created and published in 2001, by registering 152 T1 scans to the MNI305 template and averaging them.

This template is used for the structural registration of a patient's brain. When we work with functional images though, a second registration step occurs within each patient to align EPI (functional) data to the structural registered image of that subject.

Looking at picture 6.1 we can see an example of step by step functional and structural image registration to a MNI152 template: first the functional image is registered to the structural and next, they are both registered to MNI template.

Once the structural and functional images were registered to a standard space, is possible to extract brain region information using an atlas.

Atlases are in the same space as the template image (MNI or Tailarch space for example) and consist of a 3D standard brain template where different brain areas are marked with different color intensities to associate each area to a label. This division into areas and the subsequent labeling, is called parcellization.

There are different atlases, each one including a different parcelization of the brain, which is obtained by divding it into N labeled ROIs (Regions Of Interest) to focus on anatomical and/or functional regions, according to the study we are carrying out.

One of the most popular atlas is Harvard Oxford atlas obtained by manually labelling 37 MRI scans, aligning them to the standard MNI template and finally averaging each transformed label [**?**].

6.2 Implementation: CPAC, preprocessing pipeline, atlases

ABIDE I data are available in a preprocessed format, the Neuro Bureau Preprocessing Initiative took care of data preprocessing and shared its results making them publicly available. Four preprocessing software and aproaches were employed each from a different group and every one publicly available [2] for download. They consist of data from ABIDE I preprocessed by using

- Connectome Computation System (CCS)
- Configurable Pipeline for the Analysis of Connectomes (CPAC)
- Data Preprocess Assistant for Resting-State f-MRI (DPARSF)
- Neuroimaging Analysis Kit (NIAK)

The preprocessing steps implemented by the different softwares are similar, they differ on their foundational software (Python, MATLAB..) the algorithm implementation and their parameters and preprossessing steps' order.

[2] http://preprocessed-connectomes-project.org/abide/cpac.html

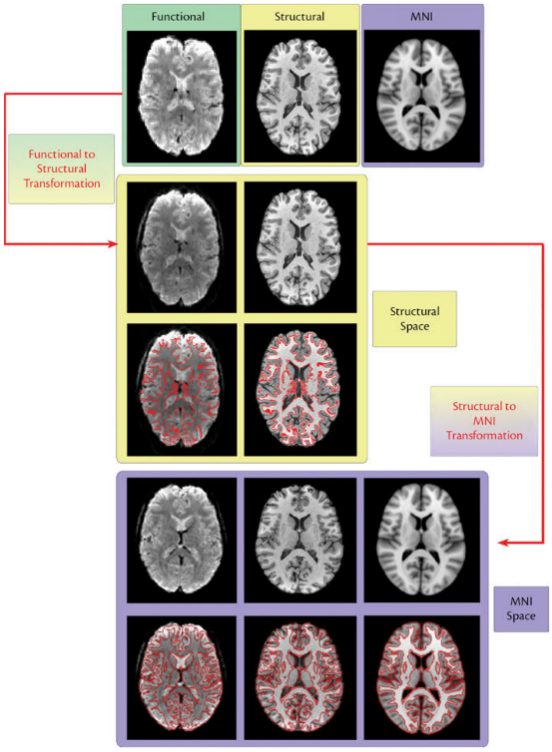


Figure 6.1: Registration steps: starting from the first row there are the acquired functional and structural images and the MNI template respectively The second row shows the functional image registered on the structural space The third row shows the same images as above, but overlaid with red boundaries extracted from the structural image The fourth row shown the final images both registered to the MNI 152 template

But since only ABIDE I dataset is avaible preprocessed, we need to repeat the preprocessing procedures in order to obtain a dataset including both ABIDE I and II preprocessed with the same pipeline

In our work, we choose to preprocess data using CPAC (Configurable Pipeline for the Analysis of Connectomes): a configurable, open source pipeline, based on Nipype platform. CPAC was run on a Docker container installed on a computer and our hardware and software setup consisted on:

- 16-core Intel i7-5960X processor and 64 Gb RAM
- Ubuntu 20.04 operating System
- CPAC 1.8.1 installed on Docker v. 20.10.11

We were allowed to run 3 patients in parallel, reserving 4 cores for each partecipant and up to 12 Gb memory to each patient, necessary to save intermediate-steps outputs.

We choose to employ CPAC because, basing on machine learning classification results, [**?**] CPAC prove to be the most efficient preprocessing pipeline to preprocess ABIDE dataset. CPAC employs tools like AFNI, FSL and ANTS to perform image correction of structural MRI and rs f-MRI. Data were preprocessed following the same steps as the pipeline employed to create the ABIDE-preprocessed dataset. This pipeline includes both antomical and functional preprocessing. Anatomical pipeline steps consist in:

- Skull removal using AFNI's 3dSkullStrip
- Tissue segmentation using FSL-FAST, to separate gray matter, white matter and CSF, uging a thresholding probability map whose threshold's values were set the same as ABIDE-preprocessed pipeline values
- Registration to a standard template using ANTS, with a spatial resolution of 2mm

Functional pipeline consists of the following steps

- Slice timing correction using AFNI-3dTshift
- Motion estimate and correction using AFNI-3dvolreg
- Distortion correction using PhaseDiff and AFNI 3dQWarp
- Create a brain-only mask of the functional data using AFNI 3DAUTOMASK

At the end of functional preprocessing steps, timeseries are extracted from each patient, making use of different atlases such as Automated Anatomical Labeled (AAL), Harvard-Oxford (HO), CC200, CC400, DesikanKilliany. As explained in section 6.1, different atlases differ in numbers and types of ROIs For example the DesikanKlein atlas employed in c-pac consists of 94 ROIs of which 32 cortical regions each side, 3 ROIs belonging to cerebellar vermis and 29 subcortical regions. The Automated Anatomical Labeled created was created in 2002 [**?**] and consisted of 90 total ROIs, 45 each emisphere, since then, different modified

and improved version were released, the last of which AAL3 consisting of 166 ROIs. However, the AAL employed in CPAC is an intermediate version and consists on 116 ROIs[3].

CC200 and CC400 are atlases created for functional parcellization, consisting of 200 and 392 ROIs respectively [4].

According to previous studies [**?**] the atlas that gave best classification performances was Harvard-Oxford, so this was the atlas we choose to employ for our work. The H-O atlas consists of a subcortical and a cortical atlas, with a total of 117 ROIs of which 21 subcortical and 96 corticals (48 for each emishpere). The H-O atlas employed by CPAC is provided by FSL library [5], and as includes both subcortical and cortical atlases, even though there's only 111 ROIs, namely 14 subcortical and 97 corticals. The lacking ROI in the subcortical areas are L/R cerebral white matter, L/R cerebral cortex, L/R lateral ventricle and the brain stem [6]. In this unified HO atlas (subcortical and cortical) the entire cerebral cortex area was removed because is replaced by the finer parcellization of the cortical areas from the cortical atlas. The extra cortical ROI in this H-O atlas is present in the 83th position and is named 3455; but there are no information abut this ROI neither on Harvard-Oxford documentation nor on CPAC's/FSL's documentation, and because is only made of 2 voxels it was excluded for our further analysis.

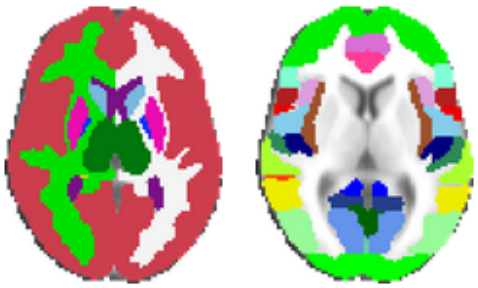


Figure 6.2: Horizontal section of Harvard-Oxford subcortical (sx) and cortical (dx) atlased

As is possible to notice from figure 6.3 timeseries extracted after our preprocessing pipeline do not exactly match those from ABIDE preprocessed, even if it appears clear that the trend is the same, but there are some local differencies between the two plots. This is most likely due to two main reasons: the differences in software version, and the lacking of a detailed step-by-step pipeline legend showing the value of all the sub parameters employed during the CPAC analysis pipeline. Abide preproccessed data were obtained using one of the first version of CPAC; nowadays, after more than 7 years, CPAC and the libraries it relies on were upgraded several times and this may have slightly affected the final outcome of the process. We found though the old pipeline configuration file employed on abide preprocessed [7], but since it belongs to a previous version of CPAC, it lacked lots of sub-parameters and

[3]http://preprocessed-connectomes-project.org/abide/Pipelines.html

[4]http://ccraddock.github.io/cluster_roi/atlases.html

[5]https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/

[6]Subcortical ROIs https://neurovault.org/images/1700/ Cortical ROIs https://neurovault.org/images/1705/

[7]https://github.com/preprocessed-connectomes-project/abide

sub-settings added during these years. For this reason, in our pipeline configuration file, parameters in common between our file and abide preprocessed' were set the same as abide's, and for all the others we choose to left the CPAC's default values.

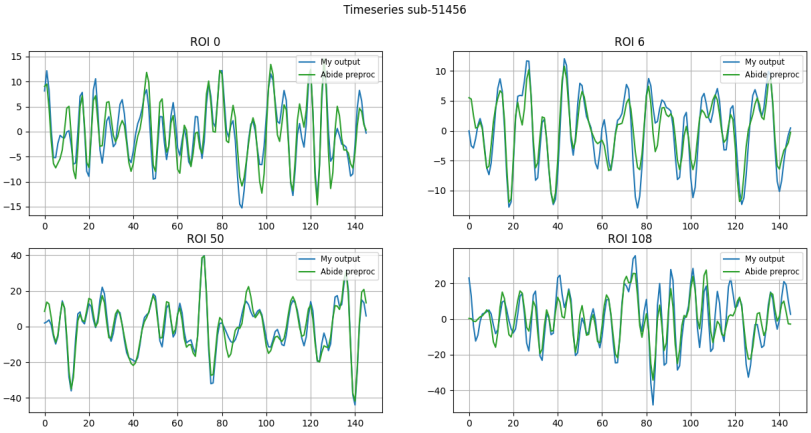


Figure 6.3: Comparison of 4 timeseries between ABIDE-preprocessed and our timeseries. Data show 4 randomly chosen ROIs: ROI 0, 6, 50, 108, belonging to patient 51456 from ABIDE I. Corresponding to ROI 10, 26, 1901, 4702

Chapter 7

Connectivity measures

Once all 110 timeseries were extracted from each patient, we need to create a correlation matrix, comparing each timeseries with all the others. The total number of combination that is possible to obtain from n timeseries is given by

$$N_{comb} = \frac{n \cdot (n-1)}{2} \tag{7.1}$$

So in the case of HO atlas with 110 ROIs we obtain 5995 combination each one expressed by a correlation coefficient computed either as Pearson coefficients or resulting from same wavelet analysis.

7.1 Correlation and Z-Fisher transform

Pearson correlation often simply called correlation coefficient is the measure of a linear relation lying between two sets of data x_1 and x_2, is defined as the covariance of (x1, x2) over the product of the standard deviation of the two sets and has the important properties to be scale-invariant in magnitude.

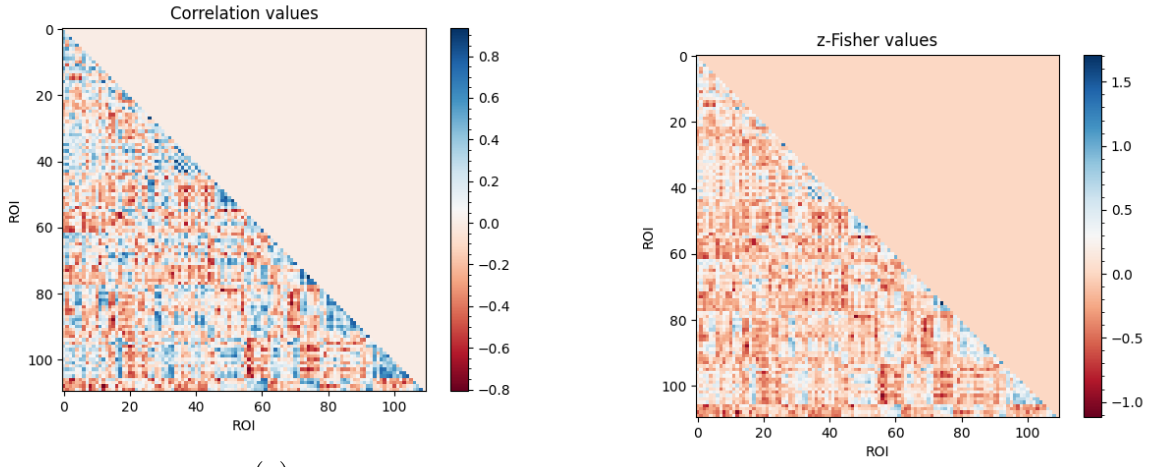
$$Corr(x,y) = r_{xy} = \frac{Cov\,(x,y)}{\sigma_x \sigma_y} = \frac{E\left[(x-\mu_x)\,(y-\mu_y)\right]}{\sigma_x \sigma_y} \tag{7.2}$$

This correlation coefficient assumes values between -1 and +1 where the extremes correspond to exact anti-correlation or correlation respectively, so that, if a linear relation lies between the two sets, a high absolute value indicates that the two series tend to be simultaneously greater or lower than their respective means

Given two series x and y, of length n correlation can be easily computed by

$$r_{xy} = \frac{\sum_{i=1}^{n} (x_i - \mu_{x_i})\,(y_i - \mu_{y_i})}{\sqrt{\sum_{i=1}^{n} (x_i - \mu_{x_i})^2}\sqrt{\sum_{i=1}^{n} (y_i - \mu_{y_i})^2}} \tag{7.3}$$

For each patient the Pearson correlation coefficient was computed for each timeseries' pair. Before further analysis, a common way to proceed is to transform each coefficient with Fisher z-transformation. The reason behind this common operation appears clear when



(a)

Figure 7.1: Correlation and z-values matrix computed from timeseries extracted using Harvard-Oxford atlas, from patient 20243 belonging to ABIDE I dataset

we are dealing with high correlate variables, when Pearson correlation distribution results in an highly skewed distribution, Fisher's transform sought to transform it into a normal distribution of which the standard error is approximately constant equal to $\sigma = \sqrt{N-3}$ where N is the total number of points, and it does not depends on the values of correlation.

With this property, Fisher's transform is important also when we want to test some hypotesis about correlations, we can run our test with the transformed variables which are normal distribuited with a known variance.

Thus, this transformation allows us to obtain a variable which is normally distribuited even when Pearson correlation coefficients follow a bivariate normal distribution or they are leaning toward the extremes. In our data, this difference in distributions is not pronounced, as is possible to notice from figure **??** becuse on average we are not working with highly correlated or uncorrelated variables, but we are dealing with a bivariate distribution, then performing this transformation is a common practise to work with more normally-distributed variables.

$$z = \frac{1}{2} ln \left(\frac{1+r}{1-r} \right) = arctanh(r) \tag{7.4}$$

At the end of this analysis, we obtain correlation matrices like that shown as an example in fig 7.1, referred to patient 20243 from ABIDE I dataset.

7.2 Wavelet analysis

A different aproach to compute a correlation coefficient is by making use of wavelet analysis.[**?**] [**?**] Wavelet is a mathematical tool for analyzing time series or images, and provides a comprehensive way for investigating the bivariate relationship between timeseries both in time (or space) and frequency domain. To understand wavelet transformation it could be helpful

to compare it with Fourier analysis. Fourier analysis allows us to expand a periodic function f(t) into series, ideally infinite sums of weighted sines and cosines with different frequencies

$$f(t) = a_0 + \sum_{k=1}^{\infty}(a_k cos(2\pi kt/T) + b_k sin(2\pi kt/T)) \tag{7.5}$$

In equation **??** terms corresponding to the k-th frequency are called harmonics and are multiples of the fundamental frequency corresponding to k = 1. The Fourier Transform is a natural extension of the Fourier series to aperiodic functions defined over the real axis, which don't allow a discrete superposition of sines and cosines terms, and for this reason they need to be represented by a continue superposition. The FT transform takes a function from time or space domain and turns it into spatial or temporal frequency domain. is a complex function because sines and cosines terms can be represented by a complex exponential.

$$\tilde{F}(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t}dx \tag{7.6}$$

To deal with discrete signal, for example a sequence x_n obtained from a discrete sampling of the continuous signal x(t), is possible to make use of the Discrete Time Fourier Transform, wich allows us to write the $\tilde{F}(\omega)$ as in equation 7.7 after the right arrow.

The Discrete Fourier Transform, is useful to analyse discrete periodic signal, it acts on a function defined on a finite domain returning a sequence of samples of the discrete fourier transform sampled at an interval equals to the reciprocal of the duration of the input sequence.

$$\tilde{F}(k) = \tilde{x}_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}} \tag{7.7}$$

where N is the total number of points of the timeseries x(t). The frequencies at which samplings are computed are $\omega_k = 2\pi k/N$. The series we obtain from relation 7.7: $\tilde{x}_k$ is periodic and its period it equal to N.

One of the main drawback of FT though, is the lacking of spatial information, for example for a non stationary signal, is possible that the signal contains a variable frequency component in different space or time locations.

A time-frequency analysis provides for this lacking computing both frequency and spatial information from a signal, allowing us to analyse non stationary signals and obtain informations both on time (or space) and frequency domain. Since we are working on signal defined over a time domain, from now on we would refer only to "time "and "frequency ", but we keep in mind that all the relation are true for spatial and spatial frequency domain as well.

The process through which we exctract informations about frequencies (F) and time location (T) is a convolution of the signal x(t) with a function w(t) called windowing function usually centered at zero which rapidly decays symmetrically.

$$\tilde{x}(F,T) = \int_{-\infty}^{\infty} x(t)w(t-T)e^{-2\pi iFt}dt \tag{7.8}$$

The convolution term $w(t-T)e^{-2\pi iFt}$ changes according to the type of analysis we are performing. Parameters F and T are often simply called a and b, where b is a time translation

parameter, and it acts the the same way through all the type of analysis, by simply shifting our convolution function throughout the entire timeseries x(t). On the other hand a is the parameter that differenciate the most different kinds of analysis: it is the frequency parameter and the way it is introduced in a convolution determines the developing of the transform. The two main types of time-frequency analysis are

1. Windowed Fourier Transform (WFT): $\psi_{ab}(t) = e^{it/a}\phi(t-b)$ where $\phi(t)$ is a window function of constant width and the parameter a acts as frequency modulation (freq $\sim$ 1/a): the lower is a, the greater the number of oscillation inside the window $\phi(t)$.

2. Wavelet Transform (WT): $\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})$ where ψ is a function called *mother wavelet*; a is a positive real number and defines the dimension of ψ: with a>1 we obtain a dilation and a<1 corresponds to a contraction. b is any real number (positives and negavies) and defines the location of the wavelet in time.

In Wavelet Transform, the equivalent to the window function in WFT is a wavelet function ψ.

A wavelet, literally "small wave "is a wave function limited both in space and period: begins at zero, grows and decrease in a limited time period and returns to zero. So that is a function local in both the temporal and frequency domain.

To be defined as so, a wavelet is required to satisfy two properties: to have zero mean (it must be oscillating) and unitary squared norm [**?**].

$$\int_{-\infty}^{\infty}\psi(x)dx = 0 \qquad\qquad \int_{-\infty}^{\infty}|\psi(x)|^2dx = 1 \tag{7.9}$$

It is defined over the space L^2 of Lebesgue measurable functions that are both absolutely integrable and square integrable. In this space the two properties can be satisfied. A wavelet transform is then formally a mapping from $L^2(\Re) \to L^2(\Re^2)$.

There are different wavelets that satisfy properties 7.9, such as Haar (fig 7.2a), Meyer (fig 7.2c) [**?**], Mexican hat (fig 7.2b) or Morlet (fig 7.3) [1].

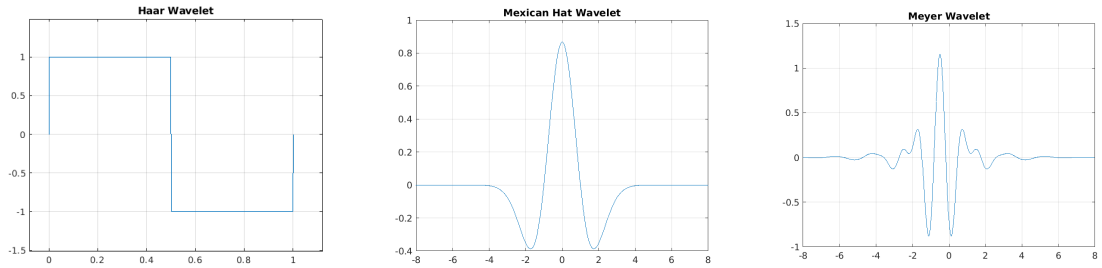
What we are going to use in our analysis is Morlet wavelet, defined by equation 7.10 and shown in figure 7.3. With this wavelet the trade-off between time and frequency resolution can be controlled by the choice of ω_0.[5] We choose to run our analysis with a value of $\omega_0 = 6$ since it provides a good balance between the two resolution [3] and besides, this value gives the best ratio between Fourier Period and the scale parameter a during a Transform, equal to $\lambda = 1.03$. This way results in frequency domain are more interpretable since the scale parameters a used for the Transform is almost equal to the Fourier period.

$$\psi(t) = \pi^{-1/4}e^{iw_0t}e^{-t^2/2} \tag{7.10}$$

Two main type of analysis can be performed using wavelets: continuous wavelet transform (CWT) and discrete wavelet transform (DWT) and the CWT is what we employed for our analysis.

Continuous wavelet transform decomposes a time series in time-frequency domain by successively convolving the timeseries with several scaled and translated version of the mother

[1] https://it.mathworks.com/help/wavelet/gs/introduction-to-the-wavelet-families.html



(a) (b) (c)

Figure 7.2: Visual comparison between three different types of mother Wavelets, Haar wavelet (fig 7.2a) is a simple square wave function, mexican hat (fig 7.2b) is a Wavelet belonging to gaussian function family and represents the negative normalized second derivative of a gaussian function; and Meyer wavelet (fig 7.2c) a wavelet with applications in fields like adaptive filters

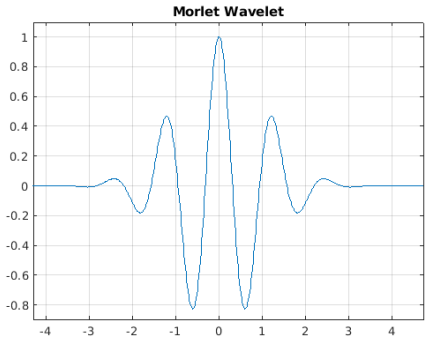


Figure 7.3: Morlet Wavelet

wavelet ψ: $\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})$. If the timeseries is described by a function f, assumed to be real in the equation below, the convolution of f and $\psi_{a,b}(t)$ is

$$W_{a,b}(f) = \int_{-\infty}^{+\infty} f(t) \cdot \psi^*_{a,b}(t)\, dt \tag{7.11}$$

Where the $*$ indicates the complex conjugate. This integral is performed for different values of a and b. It can be useful to visualize this integral as: set a value for a, a wavelet centred in b, is slided across the signal by changing the value of b and for each of these values, a coefficient is extracted by integrating the product between the wavelet and the signal; in this way, coefficients are function of frequency (or scale) and time. This operation is repeated for different values of a and b, and allows us to assemble a matrix called *scalogram*, which is basically a plot of these coefficients in time-frequency domain.

To computationally implement CWT, a discretized version was implemented on MATLAB tools Wavelet toolbox, but it does not be confused with the Discrete wavelet transform [2]

The difference between the continuous wavelet transform implemented on MATLAB and

[2]https://it.mathworks.com/help/wavelet/gs/continuous-and-discrete-wavelet-transforms.html

discrete wavelet transform lies in how finely stretching and shifting parameters are sampled: the CWT discretizes scale more finely than the discrete wavelet transform.

In the CWT, parameters are discretized based on a fractional power of two, by setting a = $2^{j/\nu}$ [3] [**?**] [**?**] where ν, j are integers. The parameter ν is often referred to as the number of "voices per octave" because increasing the scale by an octave (namely to double the frequency) requires ν intermediate steps, for example from $f = f_0 2^{\nu/\nu}$ *to* $f = f_0 2^{2\nu/\nu}$, ν steps are required. The larger the value of ν, the finer the discretization of the scale parameter. In the DWT, the scale parameter is always discretized to integer powers of 2, 2^j, j=1,2,3,..., so that the number of voices per octave is always 1. Since it employs a more rough discretization, DTW is usually used for denoising and compression of signals and images.

In our analysis we are going to use DTW implemented in matlab with the default parameters of 12 voices per octave.

One problem that arises from working on a timeseries, which is a signal with a finite support, is that when a wavelet is located at the beginning of at the end of the signal, the wavelet extends itself outside the boundary of the signal, and the convolution would require nonexistent values beyond the boundary. The confidence value obtained is then lower than other obtained for central values of time location. To overcome this problem it would be possible to accept this data information loss and truncate the values beyond boundaries; whereas an other aproach could be to artificially extend data using methods such as zero padding which assumes that the signal is zero outside its original support, or symmetrization which extend the signal symmetrically outside the boundaries or smooth padding which recovers a signal by extrapolating values from the first derivative values or from the signal itself. Symmetrization method is the one employed for the subsequent analysis. Areas of the scalogram affected by these edge effects are indicated as "outside the Cone of influence (COI) ".

Figure 7.4 shows the timeserie belonging to ROI 2201: Right Angular Gyrus of patient 51056 from ABIDE I, and its corresponding Continuous Wavelet Transform. x axis corresponds to time points and on the y axis frequencies or periods derived from the parameter a can be represented.Just as an example, we choose to display the y axis as periods. The COI is shown as a dotted white line and values outside the COI, where edge effect becomes effective are shown with a lighter faded. The color is a visual representation of the magnitude of each wavelet coefficient.

Wavelet coherence

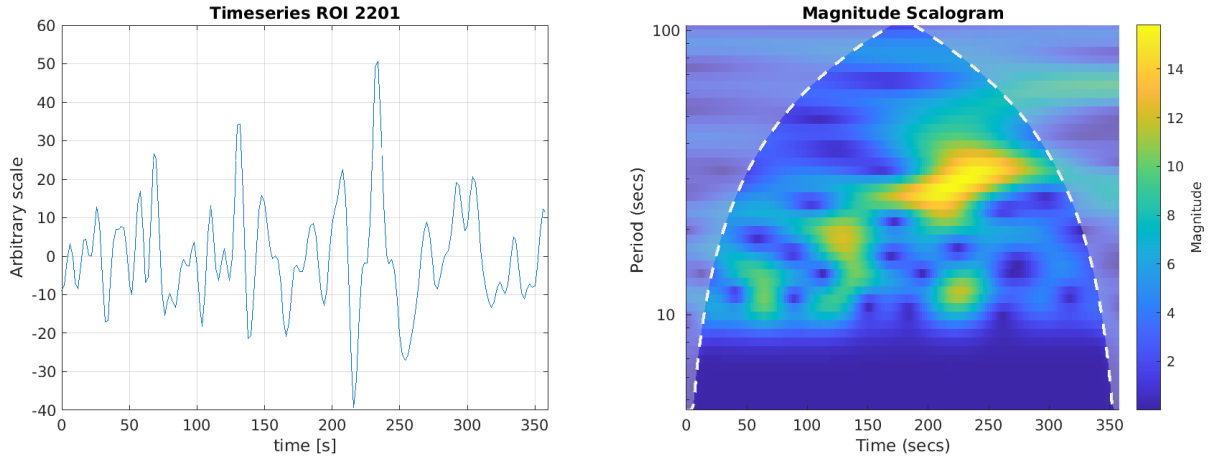
Continuous wavelet transform can only be used to analyse one signal at a time; if our goal is to analize and compare two signals using wavelet transform, the analysis to perform is called wavelet coherence.

From two Continuous Wavelet Transforms is possible to compute the Cross Wavelet Transform (XWT) which allow us to examine their cross-wavelet power and relative phases.

Using XWT is then possible to compute the Wavelet Coherence.

Denoting as $W^X(a,b)$ the continuous wavelet transform of a signal X, is defined the

[3] This way, the discretized wavelet can be written as $\frac{1}{2^{j/\nu}}\psi\left(\frac{t-b}{2^{j/\nu}}\right)$



(a) (b)

Figure 7.4: Timeseries (fig 7.4a) of ROI 2201: Right Angular Gyrus of patient 51056 from ABIDE I, and its corresponding Continuous Wavelet Transform (fig 7.4b). x axis corresponds to time points and y axis the periods derived from the scale parameter a of the wavelet convolving function. The COI is shown as a dotted white line and values outside the COI, are shown with a lighter faded. The color represents the magnitude of each wavelet coefficient.

wavelet power spectrum of a signal X(n) as

$$W^{XX}(a,b) = W^X(a,b)\left[W^X(a,b)\right]^* \tag{7.12}$$

where the * represent the conjugate transpose. Similarly, the *wavelet cross spectrum* of two time series X and Y is defined as

$$W^{XY}(a,b) = W^X(a,b)[W^Y(a,b)]^* \tag{7.13}$$

whose module $|W^{XY}(a,b)|$ represents the amount of joint power between the two time series, and is called *cross wavelet power*. From the cross wavelet spectrum (eq 7.13), is possible to compute the the complex argument

$$\Delta\phi(a,b) = arctan\left(\frac{Im\left[W^{XY}(a,b)\right]}{Re\left[W^{XY}(a,b)\right]}\right) \tag{7.14}$$

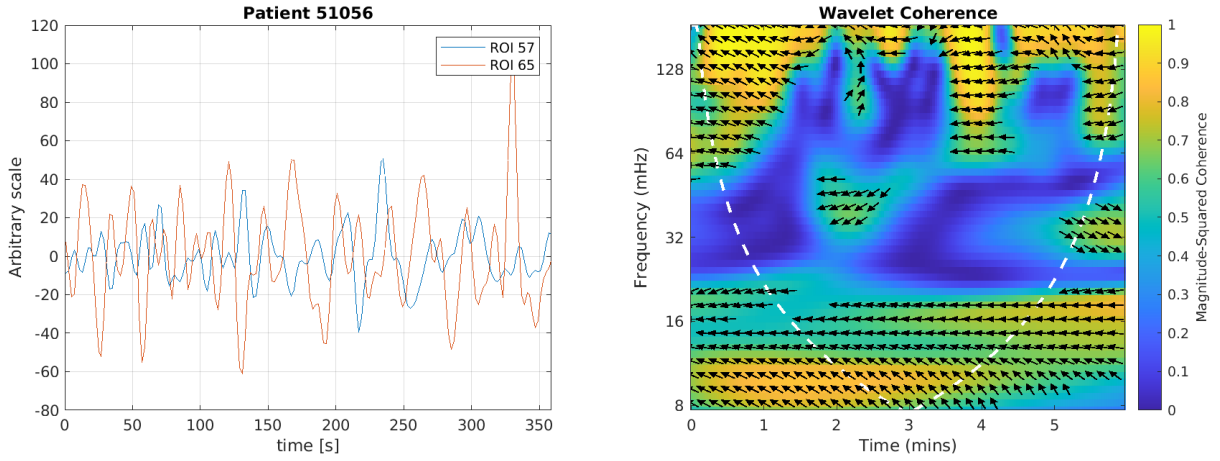
which represents the relative phase between X and Y, for each given value of the parameters a and b, and is defined over the interval $[-\pi,\pi]$.

The wavelet coherence $R^2(a,b)$ is finally defined as reported in equation 7.15. This coefficient ranges in the interval (0, 1) and represents the localized correlation coefficient between X and Y in the time-frequency domain.

$$R^2(a,b) = \frac{|S(W^{XY}(a,b))|^2}{S(|W^X(a,b)|^2)S(|W^Y(a,b)|^2)} \tag{7.15}$$

Where S is a smoothing operator both in frequency and time defined as [**?**] [3]

$$S = S_{scale}S_{time}(W) \quad S_{time} = W \cdot c_1^{\frac{-t^2}{2a^2}} \quad S_{scale}(W) = W \cdot c_2\Pi(0.6)$$



(a) (b)

Figure 7.5: Plot of two timeseries from patient 51056 in figure 7.5a and their relative Wavelet Coherence scalogram. On the x axis the timepoints and on the y axis the frequencies of this time-frequency decomposition. The color represents the magnitude of the cross-power coefficients and the relative phase shift is represented as arrows pointing to the right is phase shift is zero ad to the left is is 180 degrees. Starting from 0 degrees arrows rotate counterclockwise.

where c_1, c_2 are normalization constants, Π is a boxcar (rectangle) function and 0.6 is an empirically determined factor for Morlet Wavelet [**?**].

An example of Wavelet coherence scalogram is shown in figure 7.5b. It represents the wavelet coherence computed from two timeseries (fig 7.5a)of patient 51056: ROI 57 corresponds to Right Angular Gyrus and ROI 65 to Right lateral occipital cortex. The color represents the magniture of the cross-power, and phase information is represented as oriented arrows, pointing towards an imaginary 360 degrees circle, where the zero phase shift is represented by an arrow pointing right and a 180 degrees, by a left arrow.

Our goal is to extract from this Wavelet coherence matrix, a single value, interpretable as a correlation coefficient, just like we did with Pearson coefficients. We can accoplish so, by extracting two types of information from this scalogram: the magnitude of the correlation and relative phase between the two signal.

We can extract the magnitude information of each entry of the scalogram, but before doing so, we need to assess the level of significance of this coherence matrix over the noise, this way we can estimate the statistical significance level of our values, and only take the significative ones.

The theoretical procedure [3] [**?**] is to generate, using Monte Carlo methods, a large (1000) samples of wavelet coherence matrices using red noise timeseries. These red noise samples should be generated, for each time series, with the same 1-lag autoregressive coefficients (AR1 coefficient) as the two timeseries we are computing wavelet coherence on, and then, for each pair of red-noise timeseries, we should compute wavelet coherence matrix. However, since AR1 coefficients have little impact on the significance level [3], we choose to generate a single sample of 1000 pairs of red noise and for each pair we computed the wavelet coherence

matrix.

For each pair of noise, the corresponding wavelet coherence matrix is stacked to obtain a $A \times B \times x \times 1000$ 3D noise matrix to extract the distribution of the entries. This null distribution of wavelet coherence coefficient is used to estimate the threshold to 5% significance level for the subsequent analysis. We refer to the value corresponding to this 95-th percentile as a_{95}.

As a second step we extract the relative phase information of each entry of the matrix, and combining these two informations together, we can estimate the time of in-phase (or out of phase) coherence which can be seen as the percentage of time synchronicity (or antisynchronicity) between the two timeseries. [**?**] This time of in-phase coefficient is defined as:

$$c_{ij} = \frac{100}{N} \sum_{a,b}^{N} I\left\{R_{ij}^2(a,b) > a_{95}\right\} \cdot I\left\{-\frac{\pi}{4} < arg(W^{XY}(a,b))_{ij} < \frac{\pi}{4}\right\} \tag{7.16}$$

Where the indices i and j refers to the timeseries, N is the total number of points inside the cone of influence (COI) $I\{...\}$ is either 0 or 1 depending on if the condition inside is satisfied; a_{95} is the threshold value above which the computed Wavelet coherence coefficient is regarded as significative.

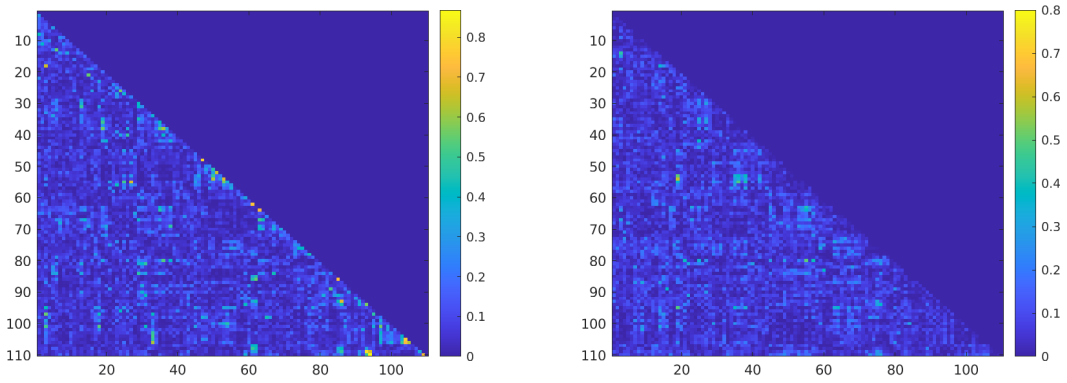
Similarily to this coefficient is the time of counter-phase coherence modifying from the formula above the phase condition into

$$I\left\{arg(W^{XY}(a,b))_{ij} < -\frac{3\pi}{4} \ \vee \ arg(W^{XY}(a,b))_{ij} > \frac{3\pi}{4}\right\}$$

In short with this analysis, we are just considering coefficients with an high significance level (above 95%) and with a small $\in [-\pi/4, \pi/4]$, or big phase shift ($< -\frac{3\pi}{4} \vee > \frac{3\pi}{4}$).

Wavelet coherence maps were calculated using MATLAB's Wavelet Toolbox, which employs the Morlet wavelet, for the wavelet decomposition, as mentioned before, and decomposes the frequency range using 12 subscales per octave and 9 octave.

From equation 7.16 both coefficients in-phase and anti-phase matrix coefficients were computed. The correlation coefficients matrix for each subject, were then flattened and used as input to the neural networks. An example of correlation matrix created with in-phase and out-of-phase coefficients from data of patient 51056 is shown in figure 7.6



(a) Coefficients of In-phase percentage (b) Coefficients of Off-phase percentage

Figure 7.6: Correlation matrix created with Wavelet coefficients of in-phase and off-phase percentage. Patient 51056, ABIDE I

Chapter 8

Harmonization

Since nw, a big percentage of neuroimaging studies have been carried out with limited data acquired from a single centre, to minimize varibility in data due to the different instrumentations employed. In recent years, however, there's the tendency to create shared datasets, by pooling together data acquired from different centers. A positive side of creating a large dataset putting together data from multiple centers is the possibility to work on dataset of bigger dimension, resulting in obtaining more statistically accurate analysis, and facilitates the generalization and the robustness of the model. As a drowback, however, it introduces variability in our analysis such as differences due to scanner models, acquisition parameters, generally known as scanner effects. A level out procedure becomes necessary to avoid the model to learn these differences deriving from inomogenity of data because of differences in data acquisition procedures. This procedure is generally referred as harmonization. Harmonization was tested on different datasets and has proven to be an effective way to reduce scanner effects in different kind of datasets such as genes' microarray data, diffusion tensor imaging or structural MRI. **??** **??**.

8.1 Harmonization - theory

The state-of-art procedure used in genomic is called ComBat (named after Combating Batch effects), used for structural MRI but aplicable to any kind of imaging data, is used to mitigate scanner effects. It is based on a previous method initially proposed in 2007, for gene expression studies to compute batch effect corrections, later implemented by Fortin et al **??** for the harmonization of cortical MRI volumes, and finally in its current improved version, developed by Pomponio et al. [9] in 2019. In this paragraph we are going to explain how ComBat technique works and after that, we'll see how it was modified and improved in the latter implementation by Pomponio et al.

ComBat technique is based on simpler Location and scale adjustment family methods, but it represents an improved version of them: it uses the empirical Bayes to improve the estimation of the site parameters. It aims to reduce inter-site variability while preserving biological variability such as differences due to sex, age, FIQ (Full intellective quotient), ICV (Intra Cranical Volume) and so on. The model assumes that a feature can be modeled as a linear combination of the biological variables plus the site effect, and the errors introduced

with site effect can be modeled as both a multiplicative and an additive term and that it can be standardized by adjusting the mean and the variance across the batches.

Let y_{ijf} be the numeric value of the feature f for the patient i acquired with the scan (or equivalently, from the site) j, so that i = 1, 2, ..., K indexes the scanner, j = 1, 2, ..., N_i indexes the total number of subjects acquired for each scanner i, and f ranges from 1 to F being F the total number of features. Besides, let's assume that this value y, can be written as

$$y_{ijf} = \alpha_f + x_{ij}^T \beta_f + \gamma_{if} + \delta_{if}\epsilon_{ijf} \tag{8.1}$$

where:

- α_f is the mean value for the feature f,
- x_{ij} is the entry of the matrix X created with the covariates of interest such as age, sex,
- β_f is the vector of regression coefficients corresponding to X for the feature f,
- γ_{if} and δ_{if} represent the additive and multiplicative terms for site-i effects related to feature f respectively,
- ϵ_{ijf} is a error term which is assumed to follow a normal distribution with zero mean and variance σ_f^2.

The final location-and-scale-adjusted data y_{ijf}^* are given by

$$y_{ijf}^* = \frac{y_{ijf} - \hat{\alpha}_f - X \cdot \hat{\beta}_f - \hat{\gamma}_{jf}}{\hat{\delta}_{jf}} + \hat{\alpha}_f + X \cdot \hat{\beta}_{jf} \tag{8.2}$$

where $\hat{\alpha}_f, \hat{\beta}_f, \hat{\gamma}_f, \hat{\delta}_{jf}$ are estimators of the corresponding parameters, based on the model.

One of the most disadvantages of using a simple location and scale batch adjustment is that it requires a large batch size for the implementation because is not robust to outliers in small sample sizes.

ComBat uses empirical Bayes to provide a more robust adjustment for the parameters $\hat{\gamma}_{if}$ $\hat{\delta}_{if}^2$, making the model able to deal with small-sized dataset as well.

The process ComBat employs to estimate feature-dependent parameters, and to adjust data for batch effect can be summarized in 3 steps:

1. **Standardization**: Data are standardized feature-wise, so that every feature has similar overall mean and variance. least square regression, is then performed to determine parameters $\hat{\alpha}_f, \hat{\beta}_f, \hat{\gamma}_{jf}$ and subsequently, $\hat{\sigma}_f^2 = \frac{1}{n}\sum_{ji}\left(y_{ijf} - \hat{\alpha}_f - X\hat{\beta}_f - \hat{\gamma}_j f\right)^2$ being n the total number of patients.

   The standardized data are then calculated by equation 8.3

$$Z_{ijf} = \frac{y_{ijf} - \hat{\alpha}_f - X\hat{\beta}_f}{\hat{\sigma}_f} \tag{8.3}$$

2. **Empirical Batch parameters estimate**: We assume that standardized data follow a normal distribution $Z_{ijf} \sim N(\gamma_{jf}, \delta^2_{jf})$ and we seek for a proper estimation of parameters $\gamma_{jf}, \delta^2_{jf}$. it is also assumed that these site-effect parameters follow the normal distribution and the inverse gamma distribution respectively

$$\gamma_{jf} \sim N(\eta_j, \tau_j^2) \qquad \delta^2_{jf} \sim \text{Inverse Gamma}(\lambda_j, \theta_j) = \frac{\theta^\lambda}{\Gamma(\lambda)}(1/x)^{\lambda+1} \cdot e^{-\theta/x}$$

And these hyperparameters $\eta_j, \tau_j^2, \lambda_j, \theta_j$ are empirically estimated from standardized data Z_{ijf} by using the method of moments. we then obtain improved estimation of parameters γ^*_{jf} and δ^*_{jf} and we use them for the third and last step, where we adjust our data using them.

3. **Adjust the data**: After calculated the site-effect parameters we are finaly able to adjust our initial data using the relation

$$y^{ComBat}_{ijf} = \frac{\hat{\sigma}_f}{\delta^*_{jf}}(Z_{ijf} - \gamma^*_{jf}) + \hat{\alpha}_f + X\hat{\beta}_f \tag{8.4}$$

which is just an equivalent way to write equation 8.2, using parameters estimated in the previous passage.

So far, this was the main idea behind ComBat technique, but we said before that the last implementation of this technique by Pomponio et al. [9] is an improved version of it. They differn in the modelling of the biologically covariates, which in the formulas above are expressed by the terms $\hat{\alpha}_f + X\hat{\beta}_f$ which is a simple linear model. Pomponio substituted this linear model with a Generalized Additive Model (GAM). In this model covariates such as sex, age, FIQ, are represented by terms $x_{ij},\ z_{ij},\ w_{ij}$ which allow a better parametric modelling to deal with non-linear trends such as the trend of cortical thickness in relation to age. This way, the terms $\hat{\alpha}_f + X\hat{\beta}_f$ in equation 8.2 are substituted by a linear combination of function F of these covariates

$$\hat{\alpha}_f + X\hat{\beta}_f \ \longrightarrow\ F_f(x_{ij}, z_{ij}, w_{if}...) = a_f + g_f(x_{ij}) + h_f(z_{ij}) + p_f(w_{ij}) + ... \tag{8.5}$$

Where functions g_f, h_f, p_f can be either linear or non-linear functions of our covariates, according to how we want to model these covariates, or in other words, according to wether we want to specify some kind of non-linear relationship between covariates and our data.

Chapter 9

Domain-adversarial NN

An other approch is trying to train a neural network to recognise discriminative features, making train as much site-independent as possible. The general idea underlying this network comes from a reviewed and readapted version of the solution of the problem tackled in Ganin - Ustonova article Their work takes place when we possess two different datased let's call them a Source and a Target dataset, containing features following two different distributions. Moreover, source dataset is labeled while Target is unlabeled, so their goal was to create a network able to distinguish relevant features from the labeled dataset and make them domain independent. We illustrate the main aspects of this problem using a shallow neural network with a single hidden layer consisting of D nodes. Let's suppose the network takes as input an m-dimensional features vector; the hidden layer is then a function $G_f : R^m \to R^D$ of the type $G_f(x; \mathbf{W}, \mathbf{b}) = f(W \cdot x + b)$, where $\mathbf{W}$ and $\mathbf{b}$ are the matrix and vector parameters to be optimized. Similarly the output will be a function $G_y : R^D \to [0, 1]$ of the type $G_y(G_f(x; W, b); V, c) = f'(V \cdot G_f(x) + c)$. Where V and c are parameters to be optimized: a vector and a scalar respectively. A usual train carried on only on the (labeled) Source domain, will therefore bring to the minimization of the quantity

$$\min_{W,b,V,c} \left[\frac{1}{n} \sum_{1}^{n} L_y^i(Gy(G_f(x_i)), y_i | W, b, V, c) \right] \tag{9.1}$$

. To tackle the domain independence problem, an other term is added to the summation: a term coming from a domain classifier layer: $G_d : R^D \to [0, 1]$ of the type $G_d(G_f(x; W, b); u, z) = f'(V \cdot G_f(x) + c)$ that learns the vector and scalar parameters u and z, by trying to classify the domain. With this addition, the complete quantity to be optimized becomes

$$E(W, b, V, c, u, z) = \frac{1}{n} \sum_{i=1}^{n} \tag{9.2}$$

So the purpose of this kind of training is to find a saddle point by finding minimizing parameters for feature loss and maximizing those related to the site loss, namely

$$(\hat{\theta}_f, \hat{\theta}_y) = \underset{\theta_f, \theta_y}{argmin} E(\theta_f, \theta_y, \hat{\theta}_d) \qquad \hat{\theta}_d = \underset{\theta_d}{argmax} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d) \tag{9.3}$$

This could be achieved by training a network in an adversarial way, trying to minimize the category label loss and maximize site label loss. this is achieved by adding a domain

classifier branch just after a main feature extractor branch. The domain classifier branch is characterized by a *gradient reversal layer* at the top of it. This layer has no parameters and don't act during forward propagation, but, during backpropagation, it inverst all the weight from the subsequent layer before passing them to the preceding one. So as shown in figure -insert figure- the network is forked after the feature extractor branch, on one side, he starts the label predictor branch, and on the other, there's the domain classifier branch with the gradient reversal layer at the beginning. The setup of this work, differs from Ganin's because we don't have a labeled Source and an unlabeled Target domain, but they are both labeled, so there's no need to keep them separated and also, in our case, the Target domain is not just made of two domains, but of as many sites our data belong to.

In this work, the label classifier branch ends with a sigmoid function wich return only one class label $y_i \in [0, 1]$ while the domain classifier branch ends with a multi-node layer of a number equal to the total sites' number. An adversarial loss is also implemented, aimed to aviod that if, for instance, the domain classifier, correctly classify one input, the computed error would be low, and it would have a low contribution during the back propagation, and leaving the weights almost unmodified after the backpropagation, and therefore leaning towards that site.

Chapter 10

Harmonization - results

As explained in section 8.1, ComBat simultaneously models and estimates biological and non biological terms, and algebraically removes the estimated addivite and multiplicative site-effect terms.

Harmonization procedure was tested on the whole ABIDE I + II dataset to have a visual representation of how features are modified after the harmonization is applied.

To test harmonization procedure and evaluate its performances in making sites' feature uniform, we used NeuroHarmonize [1] package for Pyhton, developed by Pomponio et al. [9] which implemented and added some features to the previous NeuroComBat Python package [2].

To use Neuroharmonize package we need to input the feature data to harmonize and the corresponding covariate information with site, age, sex, and other biological quantities we desire to take into account. For our data these information were provided by a csv file on ABIDE website so that for each patient are collected as much information as they could. A regression model is thus created with parameters estimated with the procedure explained in section 8.1 and it can be applied to this input data or to other new data provided that they belong to the same sites we used to create our model.

We tested harmonization procedure on 1494 male patients coming from ABIDE I and ABIDE II of which 722 controls and 772 cases, choosing as covariate to preserve the Age, and the FIQ (Full intellective quotient) to mantain important biolgical trends in the data and avoid overcorrection.

The central idea is to create the model on control subjects only, to create a model without the influence of informations related to ASD patients whose feature might follow a different distribution than control patients. Then once created the model on control subject, it is applied to both control and ASD subjects, so that site-related information are eliminated from our data.

Under this general way to proceed, we used a Random Forest Regression to test the site classification performances before and after the harmonization procedure. In this analysis, we split the model into a Train and Test subsets, using the Test subjects, we created a model with only controls as explained before, The model was then applied to the case subjects belonging to train, and to both case and controls subjects of the Test set.

[1] https://github.com/rpomponio/neuroHarmonize

[2] https://github.com/Jfortin1/ComBatHarmonization

A random forest regressor was trained on this dataset attempting to make binary classification of the site, for each pair of sites.

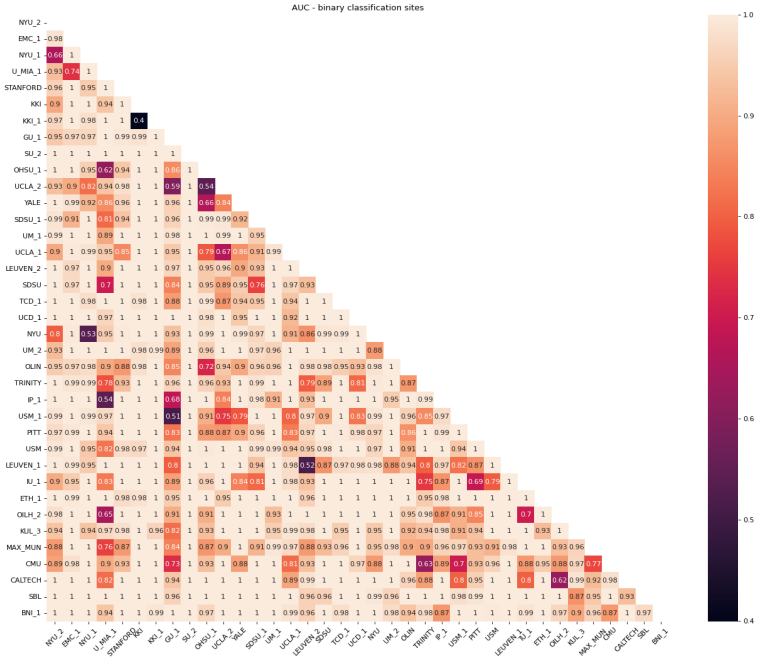
This aproach, allows train and test dataset to be still independent, because harmonization model was only created on Train subjects and once learnt the parameters, is applied to the Test dataset, so the data we use to train the random forest, are kept separated from the test set, so that there's no information leakage of the test data into the training data.

Note: As can be seen from figure 5.1a two sites namely `NYU_2` and `KUL_3` provided only ASD patients, and, since we are estimating the mean and the variance for each feature across all sites, when we train a model only on control cases, and we apply it on patients belonging to new sites, we don't have information to scale and modify those data so the model would output NaN values. In our analysis we excluded these sites since we can't perform a proper harmonization of them, however, just in the following figures to have a visual feedback of how harmonization works, we substituted NaN values with original unmodified data.

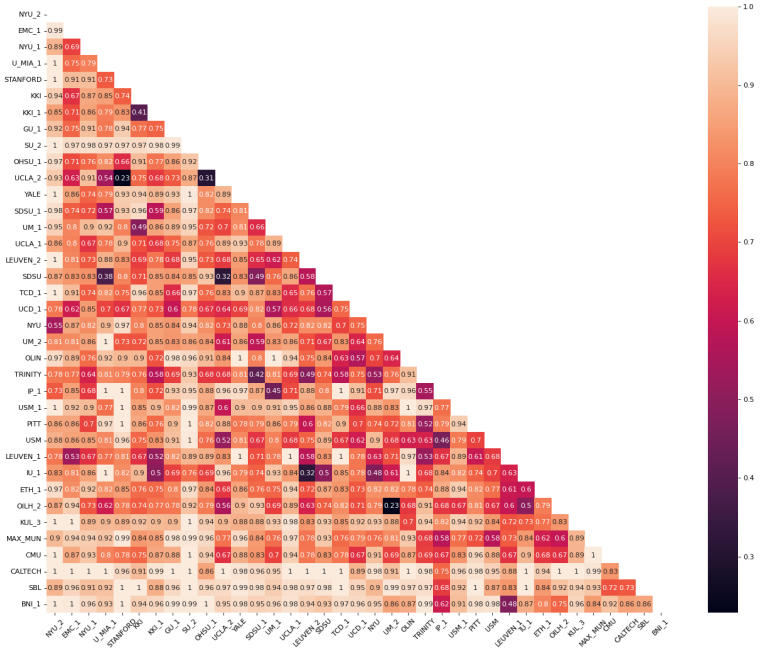
Figure 10.1 shows the result of classification on the dataset described above. Scores are reported in terms of AUC for the classification of each pair of sites, and sites along the x and y axes are ordered by incresging average age of patients coming from that site. Specifically, fig 10.1b shows the AUC score of the model trained with the raw, not harmonized dataset, it is clear from this binary classification that almost every site is well distinguishable from the others by the value of AUC, almost uniformly near to 1. On the contrary, in fig 10.1a values are lower, meaning that sites are not so distinguishable anymore, however, a certain distinctive features are still present for sites presenting high difference in mean age. This is a consequence of choosing age as covariate, which preserves differences in age-dependent features.

To have a visual representation of how much each feature is modified in respect of its sourced site, in fig 10.2 are shown two features among 5995: *feature 324* and *feature 2800* as a function of sites. For a clearer idea of what is shown, feature 324 represents the correlation between right and left inferior frontal gyrus, and feature 2800 that between left precuneous cortex and the left inferior frontal gyrus Their values are plotted as a box along the y axis and sites are indicated along the x-axis, it is clear how harmonization is a powerful tool to level up feature and remove site-dependent information from them.

If we split this plot into control and ASD patients to see the effect of harmonization separately on these subsets, 10.3

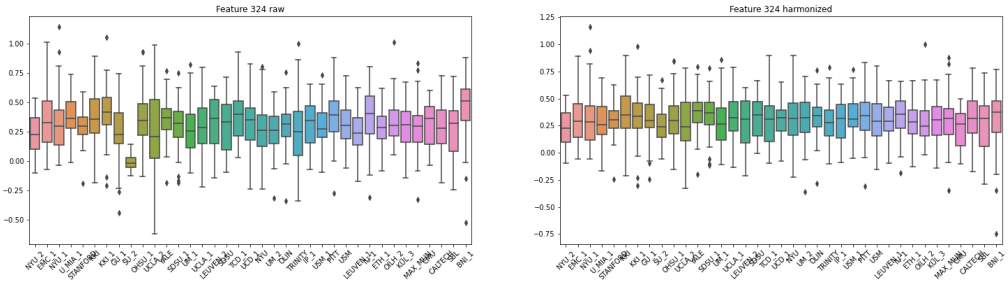


(a) Heatmap with AUC score of a binary classification site vs site with raw data

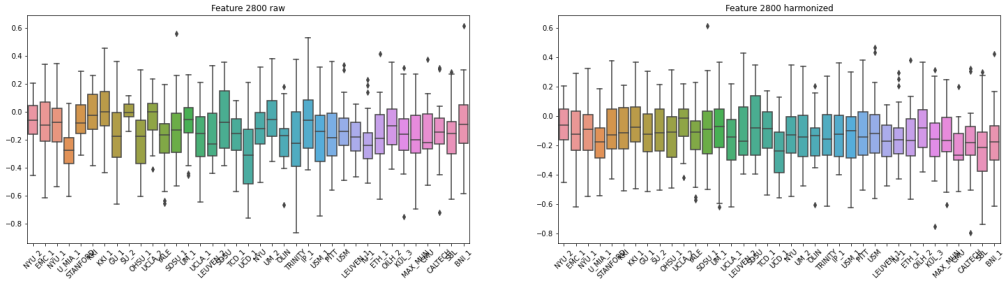


(b) Heatmap with AUC score of a binary classification site vs site with raw data

Figure 10.1: Comparison of two heatmaps with AUC score of binary classification site vs site with raw (fig 10.1a) and harmonized data (fig 10.1b) classified using a Random Forest Classifier

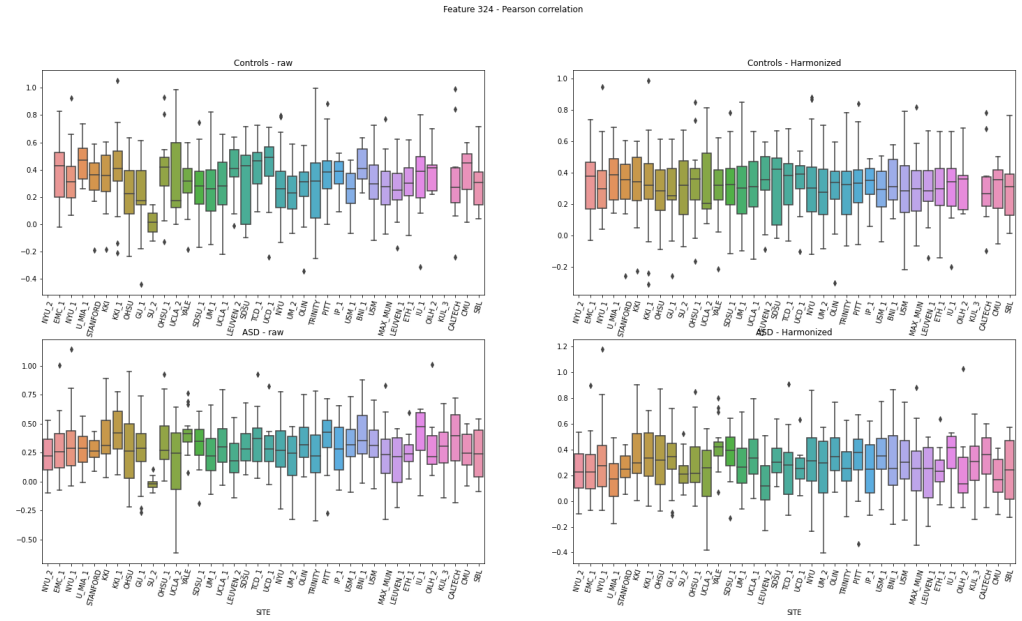


(a) Feature 324

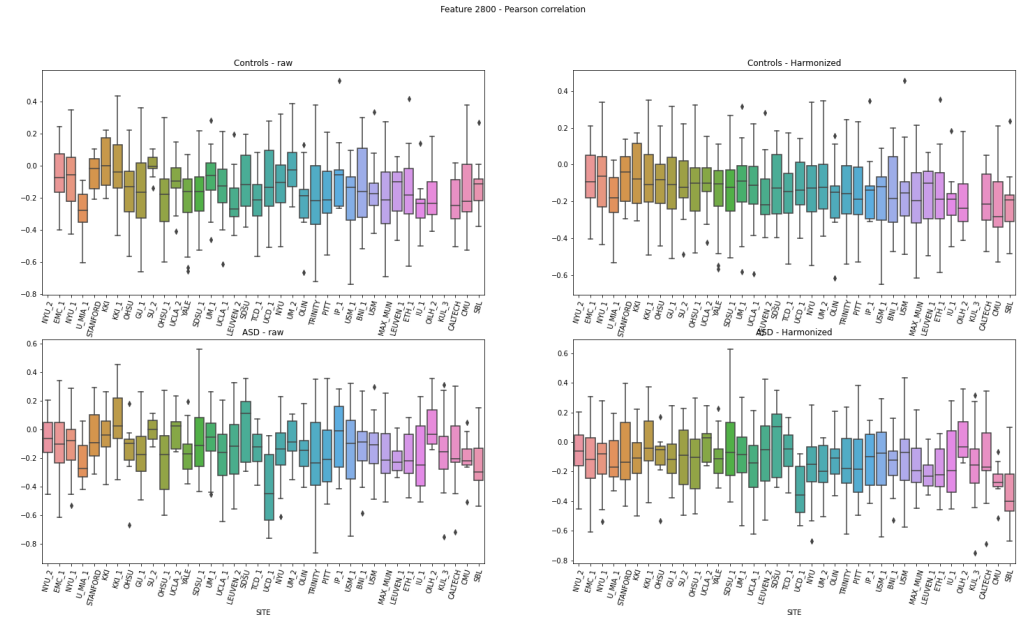


(b)

Figure 10.2: Pearson-based connectivity coefficients per site for features 324 (fig 10.2a) and 2800 (fig 10.2b) before and after harmonization

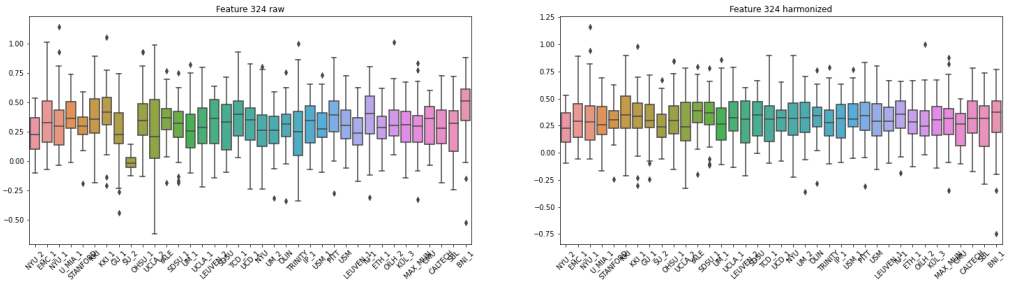


(a)

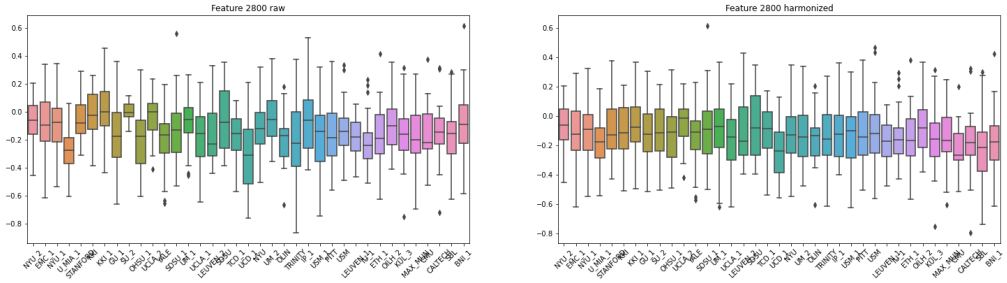


(b)

Figure 10.3: Pearson-correlation coefficients features 324 and 2800 before and after harmonization, separated plot for controls and ASD



(a)



(b)

Figure 10.4: Wavelet-based connectivity coefficients of in-phase percentage, for features 324 (fig 10.4a) and 2800 (fig 10.4b) before and after harmonization

Chapter 11

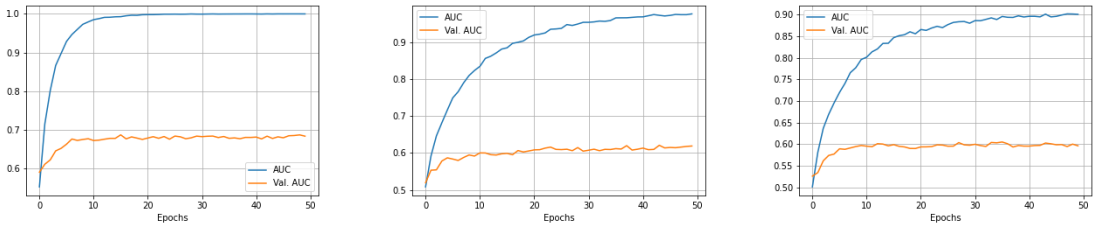
Deep learning: implementation and results

11.1 Model's structure

In this work we performed a classification task, using a shallow neural network built using keras library for Python. Since we are working in an unavoidable overfitting condition, we tried to build a network with as less feature as possible, but still preserving good classification performances. We then started the research of our newtork with a network comprising only 3 layers and we searched the minimum number of neurons to employ in each layer. We started with a trivial network made by just 3 neuron in the first layer, 2 nodes in the second and at last a single-output layer. Even if this was just an attempt to put a lower limit to the number of neurons, we noted that even with this configuration, the network tends to overfit our data. This is clear if we take a look at the learning curves in figure 11.1 which shows the training and validation AUC curves for two models: the first one of them (11.1c) even if with the simplified structure just mentioned, shows the classical trend of an overfitting state, however, it performes a bit worse than the other two more complex models. We continued adding neurons to each layer and tried different configurations. With a configuration of 8-8-1 performances were slightly higher, so we set the second layer to 8 neurons and tried changing the number of the neurons in the first layer, performances seemed to increase as the number of neurons increased untill they reached a stable value. Different attempts and the relasted classification scores are reported in table **??** in appendix. We decided to pick the configuration which gave the minimum number of neurons, beyond which performances were stable, and the addition of more neurons to create a more complex network was unjustified.

In our work we employed a the neural network schematized in figure 11.2. This network consisted of 2 hidden layers made up of 264, and 8 neurons respectively, both activated by a ReLU function and separated by a Batch-Normalization and a Dropout layer with a dropout chance of 30%. After the 8-node layer we added a second Batch-Normalization layer, with a second dropout layer once again wich 30% dropout. After them we put a single output layer with a single neuron for the classification output. This last layer is activated by a sigmoid function, which outputs a real number between 0 and 1.

Since we are still working in an underlying overfitting condition we set the subsequent hyperparameters after a grid search on learning rate and number of epochs, setting a vali-



(a) AUC learning curve corresponding to a model with a structure 64-8-1 nodes trained on the entire not-harmonized dataset

(b) AUC learning curve corresponding to a model with a structure 8-8-1 nodes trained on the entire not-harmonized dataset

(c) AUC learning curve corresponding to a model with a structure 3-2-1 nodes trained on the entire not-harmonized dataset

Figure 11.1

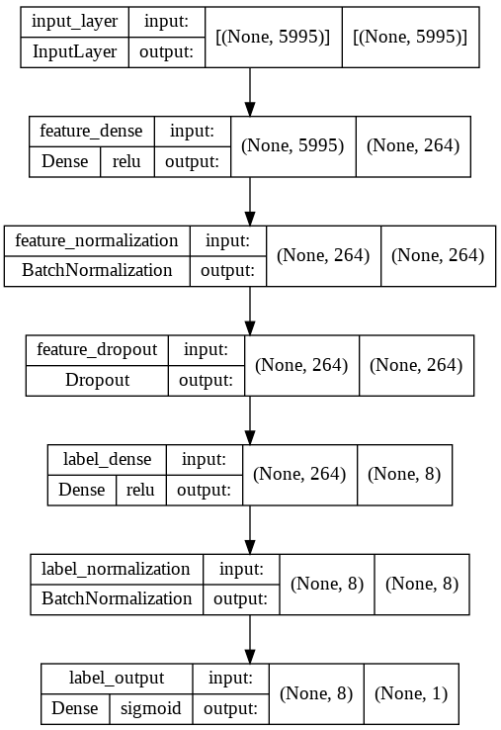


Figure 11.2: Schematic structure of the deep neural network employed in this work for Control/ASD classification, with a scheme 264-8-1. Each box contains the layer's name, layer's task (Input, Dense, BatchNormalization etc..), activation function, input shape (number of neurons) and output shape.

dation set of 25% of Train data, and studying the evolution of the learning curves, of which some exaples are illustrated in figure **??** to find the minimal parameters' configuration which allows high classification scores.

In our model, as loss function to minimize we used binary crossentropy and the neural network's weights were optimized via Adam optimizer with a learning rate of 10^{-4}. Adam optimizer was chosen over Stochastic Gradient Descent optimizer since Adam reached similar

performances with a lower error in respect of SGD. Model's classification performances were assessed with a 5 k-fold Cross validation, collecting the AUC score for each fold and then computing mean and std deviation of these results.

The other neural network employed was the site-adversarial neural network: a model able to earn from data both class and site information, and avoid some sort of bias due to site, when performing class classification. We tried two different aproaches to this task, but later we focused only on one of them. The first attempt to implement a model able to predict category label without any influence of site's information is a model with two different branches, one for the output of Control/ASD prediction, and one for the output of the predicted site. With these two informations, a loss is created by combining two different loss functions, one for the class and the other for site: the idea is similrar to that proposed in the article **??**. This new loss is in the form

$$L = L_1 - \lambda L_2 \tag{11.1}$$

Where L_1 is in our case the loss for the binary classification Control/ASD that we want to minimize (binary crossentropy), while L_2 is the loss for site classification (categorical crossentropy) we aim to maximize in order to avoid to learn any information somehow linked to site, and λ is a parameter to set empirically, to control the contribution of the site's loss to the overall loss. This way, the minimum of the total loss is reached with parameters that minimize the category classification loss and maximize the site classification loss. Performances of this kind of model are strictly linked to the value of the parameter λ, in a sense that if a certain value gave an optimal performance on a certain dataset, on an other dataset, for example one created with some more constraints on patients, the value for which one had the best scores needed to be changed. And also, since this kind of model gave slightly worse performances than simple deep model, we looked for an other strategy and carried out the analysis with this second type of model.

We tried to build something with the same underlying idea, namely minimize the error on classification of asd/controls, while maximizing the loss related to the site classification. Our task is in fact to avoid the model to learn some site-related pattern from data and we accomplished so with a model whose inner weights are updated moving towards a minimization of the gradient for classification, and in the opposite direction for site classification, as explained in section 9.

Following a similar structure of the model before mentioned, with a composition of the two losses, the structure of this model can be divided into three main components, two of which are exactly the same of the deep neural model explained at the beginning of this section. The three components that make up this model are: a first feature extraction branch comprising th input layer, the 264-nodes and the 8-node layer, between which we put a Batch-normalization and a 0.3 dropout layer, as we did in the first DNN. From this point on the network is splitted into two branches: one categorical classificator, similarily to the DNN consists of a last layer with a sigmoiod activation function for Control/ASD classification, and the other branch, the third component of this network, the site classifier consisting of a gradient reversal layer as the first layer followed by a layer with 16 nodes, and after a batch Batch-Normalization layer, the last layer: a multi-output layer with N nodes, being N the total number of sites input data belong to. The structure of this network is illustrated in figure 11.3

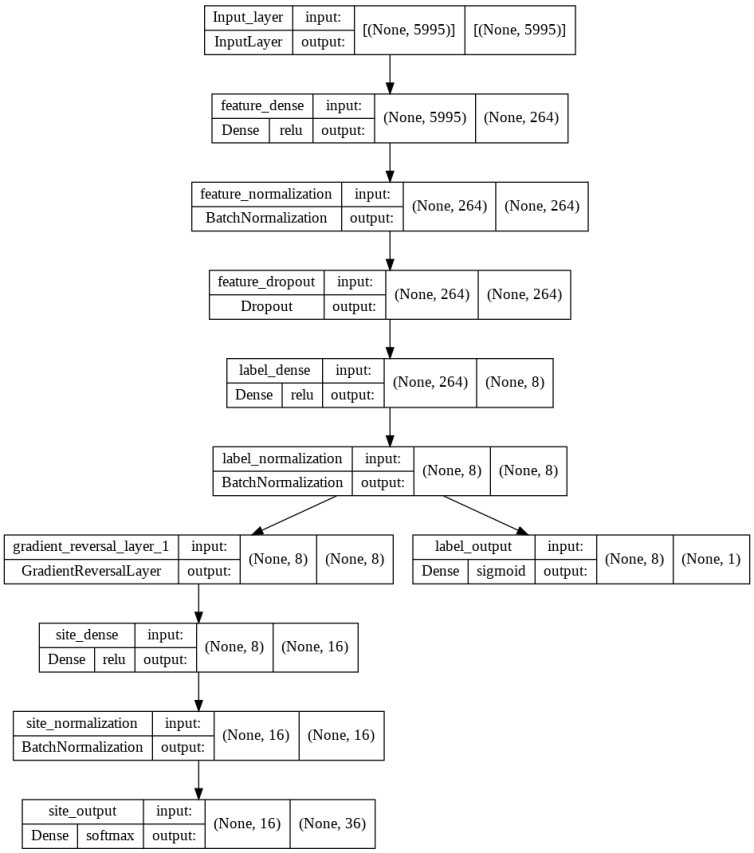


Figure 11.3: Structure of the adversarial neural network with two outputs: a single-neuron output for binary classification controls/ASD and a multi-class classification for site classification. Each box contains the layer's name, layer's task (Input, Dense, BatchNormalization etc..), activation function, input shape (number of neurons) and output shape.

The final layer of the site-classification branch is activated by a softmax function, and the loss function employed is the categorical crossentropy implemented using a custom function to deal with a possible low crossentropy score when by chance the site classifier, categorize one or more input data site label correctly, which would have low impact on the whole branch classifier making the adversarial strategy less effective.

Categorical crossentropy is the loss which allows to deal with multi label output as explained in 3.4.

Note: To work with this loss we need to encode each site numerically before before use its label into our model because functions like these can't deal with string variables such as site's names, so we have to associate each one of them with a number: site_a = 0, site_b = 1 and so on, but if we stop to this point and input integer numbers $\{0, ..., s\}$ where s is the total number of sites, to our model, it ends up considering sites with higher number as "larger "than that and produces not-optimal results. To avoid this a common strategy is to transform these variables according to a technique called One-hot-encoding. This common aproach converts each value to a vector of len s, and containing all zeros but in the entry corresponding to that site, where it puts

1, so thast each site label becomes a binary vector, with just a single 1 in position corresponding to that specific site.

We tested both these different adversarial models (the one with loss combination and the one with gradient reversal) and we noticed that they had more or less similar performances, but we choose to carry out the next analysis with this second model, the gradient reversal one, because, even if, like the model with loss compositions, in the gradient reversal layer, a parameter λ controls the weights updating during the backpropagation, results obtained with this model are more stable in respect to the the value of this parameter. So changing the value of this parameter does not affect significantly the performances of this model.

11.2 Analysis workflow

We mainly carried out three different kind of analysis using the DNN explained before, and compared them with the scores obtained from the adversarial network.

For each one of these analysis, we implemented a K-fold cross validation scheme to report our results. We determined and reported model performances as the mean AUC value and the standard deviation of the scores for each k-fold. A schematic

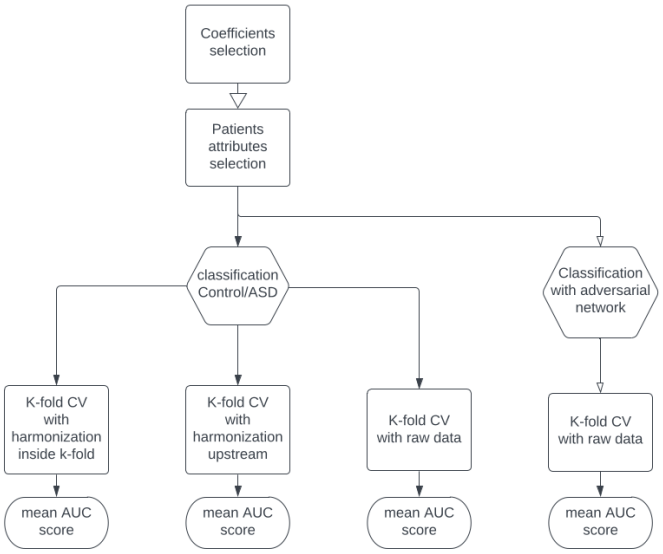


Figure 11.4: Flowchart of the analysis carried out in this work:

1. Select one type of coefficient: Pearson correlation, Wavelet coherence coefficients
2. Select the desired attributes of patients in order to make cuts on dataset, for example use the whole ABIDE I + II dataset or just ABIDE I, limit analysis to a certain age range, sex, or eye status open or close.
3. Use these data + labels to make classification following the three pipelines described in section 11 and the fourth analysis pipeline using the adversarial neural network
4. For each of these branches, implement a K-Fold cross validation and return the results in terms of mean AUC and std deviation of the values across every fold.

Since we are not working with a huge amount of data (< 1600) samples, we choose a 5-fold CV, we made this choice because with a 10 k-fold CV, the error on the mean AUC score was high, so we choose to hold a greater number of data in the Test to reduce variability among them and hence reduce the standard deviation of the model's score. K-fold CV was implemented using the stratifiedKFold class provided by sklearn library for Python

The data we are using to perform classification are the Pearson correlation (Fisher-transformed) data, or the correlation values coming from the analysis with wavelets.

In detailes the four types of analysis we carried out are:

1. Classification of asd/Controls using raw (not harmonized) data

2. Classification using the whole harmonized dataset, creating the harmonization model on all the control subjects and applying the model on all ASD subjects, and then split this dataset into a train and test subsets for each k-fold. The flowchart to schematise this implementation is showh in figure 11.6.

3. Classification with harmonization implemented inside the K-fold cross-validation, creating the model on controls and applying it to ASD: following this procedure data were splitted into a train and a test dataset, therefore using train data we computed the harmonization model only on control patients and then apply the model to asd patient and both Control and ASD Test data. The whole flowchart is shown in figure 11.5

4. Classification using the domain adversarial neural network.

Harmonizing data inside a k-fold is the best way to mantain Train and test data as much separated as possible and avoid data leakage between train and test. It is a common source of error to introduce some kind of data leakage between the Train and the Test dataset, when we harmonize on the entire dataset, the harmonization model permorms a fit using all the data, so each data is modified according to information obtained from all the other data, so in data belonging to the train dataset there's already information about the Test dataset, and this leads to a misleading increase of model's performances. Some articles perform a Leave One Site Out Cross validation. In our case we can't do that if we want to mantain Train and Test independent because if we compute the harmonization model only on the train dataset, we wouldn't have information to apply it to the test dataset since its data belong to a site not included in the model.

Analysis were carried out using different selection criterias and thresholds on the dataset which brought us to different datasets combination. We excluded from out analysis data coming from sites NYU_2 and ... because as noticed in section **??** they didn't provide control data but only ASD patients, and without them, we can't perform harmonization on this sites because we would need to create the model only on Control data. We left with a total number of 1501 perason correlation data and The first classification attempt was using the whole dataset from ABIDE 1 and 2, consisting of only male patients aged between 5 and 40 years old, without any further constraint on eye status at scan. Then, with these same constraint we tried to classify separately ABIDE 1 and ABIDE 2 datasets. The second analysis was performed on a dataset with the same constraint as above, but with an additional condition on eye status choosing to select only patient who kept eyes open throughout the entire scan time. The latter analysis were carried out with these same constraints, but separately on ABIDE 1 and ABIDE 2 dataset. So in whort we have 6 different analysis shown in table 11.1

A further analysys concerns the feature's dimensionality reduction, that is the reduction of features from the input dataset using PCA For each one of the analysis above, PCA was implemented to restrain the effect of overfitting.

From the initial input data consisting of 5995 features, we tried to classify using n principal components The workflow of the entire analysis can be schematized in the workflow diagram 11.4

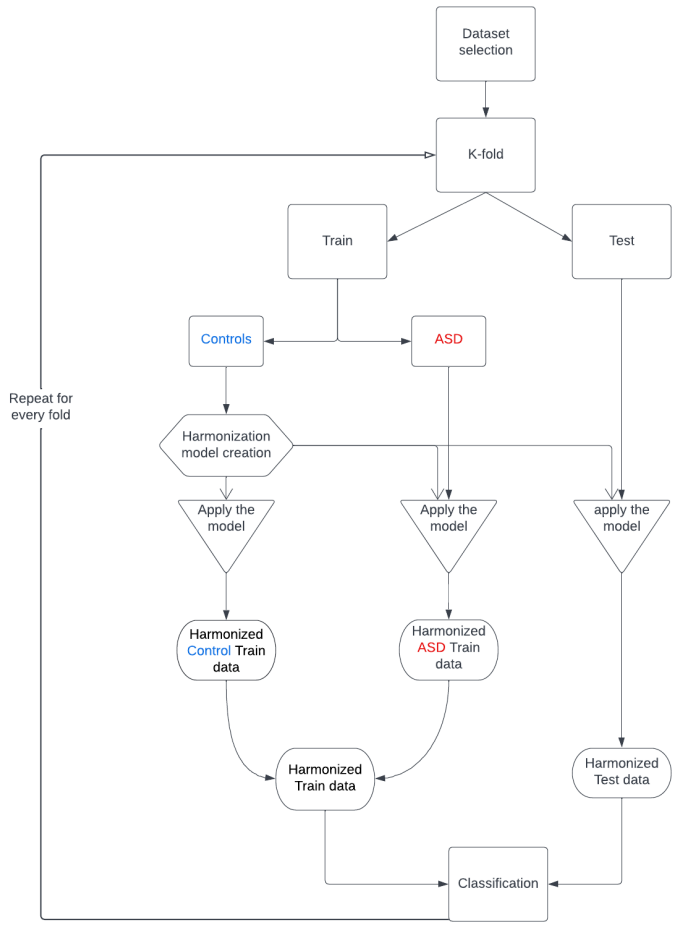


Figure 11.5: Flowchart: implementation of harmonization inside each fold

Dataset	Tot	Controls	ASD
AB I+II eye = all, sex = males, age(5, 50)	1470	737	733
AB I + II eye = open	1026	514	512
AB I	841	426	415
AB I eye=open	568	281	287
AB II	629	311	318
AB II eye = open	458	233	225

Table 11.1: Total number of data and control/ASD amount for each subset and thresholds

11.3 Results with Pearson coefficients

With Pearson correlation coefficients normalized with Fisher transform we obtained the following AUC scores, listed in table 11.2 with our DNN models, and we compared some of these results with a Random Forest classifies, to asses whether is convenient to use a deep model instead of a common machine learning algorithm such as Random Forest.

Analysis with Random Forest were performed on the whole dataset: ABIDE I+II, and on the dataset which gave best classification performances: ABIDE I with open eye, following the three main pipelines of: Hamonization inside K-fold, Harmonization upstream, and Raw

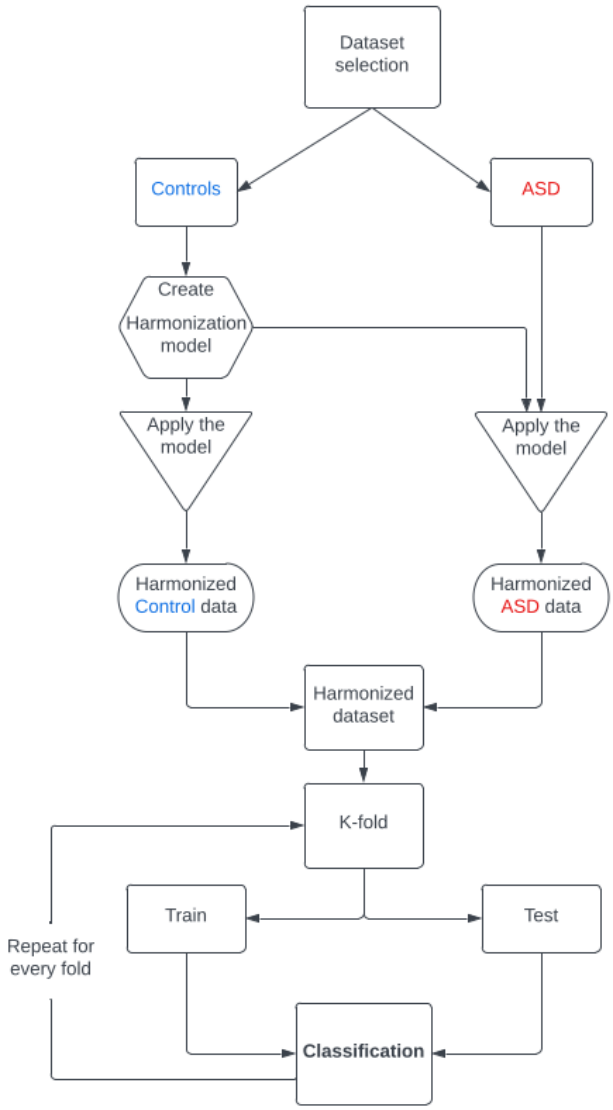


Figure 11.6: Flowchart: implementation of harmonization upstream

Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I+II	71±1	74±2	73±3	69±2
AB I	71±3	74±2	72±3	71±4
AB 2	63 ±5	68± 6	64± 4	66 ± 4
AB I ± II - EYE open	72±3	75±4	72±3	71±3
AB I open EYE	73± 1	76±2	72±1	72±4
AB II EYE open	66±3	70±6	69±6	68±6

Table 11.2: AUC score obtained from three different kind of analysis with Pearson coefficients

data (non-harmonized). Comparison results between a DNN model and RF results are listed on table 11.3

AB I + II		
	DNN	RF
K-fold	71±1	66+2
Upstream	74±2	72+1
Non-harmon	73±3	65+1

AB I eye = open		
	DNN	RF
K-fold	73± 1	70+3
Upstream	76±2	71+2
Non-harmon	72±1	68+3

Table 11.3: Classification scores comparison between the DNN model and a Random Forest classifier, on Pearcon correlation coefficients, for dataset: ABIDE I + II and ABIDE I with open eyes

11.4 Results with Wavelet coefficients

Following the same path as with Pearson coefficients, we tried to run classification on Wavelet coefficients

Four main analysis were carried out

1. In phase wavelet coefficients, reported in table 11.6
2. In phase and counter phase coefficient stacked together in a single array long twice In-phase coefficients only
3. Coefficients resulting from the subtraction of the two: In phase - Counter phase: denoted as W_in - w_out, reported in appendix, in table B.2
4. Counter phase wavelet coefficients, reported in appendix in table B.1

Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I±II	65± 2	74± 2	66± 2	67 ±2
AB I	67± 3	73 ±2	68± 3	68± 2
AB 2	62 ±4	68± 4	63± 4	62± 4
AB I ± II - EYE open	65 ±4	71 ±3	66 ±3	66± 3
AB I open EYE	69± 4	74± 6	67± 3	67 ±4
AB II EYE open	66 ±2	69 ±3	67± 2	68 ±2

Table 11.4: Classification scores: AUC of joined W_in and W_out coefficients

These results were compared with a Random Forest classifier obtaining the following scores: (table 11.5)

ABIDE I + II		
	DNN	RF
K-fold	65± 2	60+2
Upstream	74± 2	74+3
Non-harmon	66± 2	59+3

AB I eye = open		
	DNN	RF
K-fold	69± 4	62+5
Upstream	74± 6	69+3
Non-harmon	67± 3	61+5

Table 11.5: Classification scores comparison between the DNN model and a Random Forest classifier, on Wavelet coherence coefficients W_in and W_out, for dataset: ABIDE I + II and ABIDE I with open eyes

Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I±II	65± 2	71± 2	62± 2	64± 1
AB I	66± 3	73 ±3	65± 3	66± 2
AB 2	62 ±3	65 ±3	61 ±4	60 ±2
AB I ± II - EYE open	64± 4	70 ±3	66± 3	65 ±3
AB I open EYE	64 ±5	68 ±5	64 ±3	64 ±4
AB II EYE open	66 ±2	68± 3	66± 4	65 ±5

Table 11.6: Classification scores: AUC, only w_in coefficients

Explained variance		
N PC	All dataset [%]	AB-I eye=open [%]
800	93	–
400	78	98
200	62	77
100	48	58
50	36	41
20	24	26

Table 11.7: Explained variance [%] for different numbers of principal components, and for the two dataset used in the analysis

11.5 Results with PCA

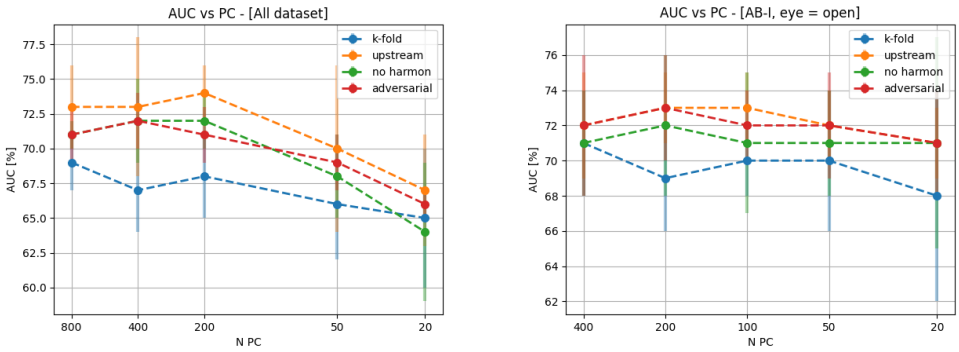
When applying PCA to a dataset we seek to explain as much variance as possible, for this reason we start our pca analysis extracting a number of principal components that explain more then 90% of variance, then we reduced this number halving if. The maximum number of principal components is limited by the number of data we can use. If we have a dataset of $n\_samples$ samples, each with $n\_features$ features, the maximum number of PC is given by $N\_pc = min\{n\_samples, n\_features\}$. This is because in a $n - feature$-dimensional space, our points lie in a $n\_samples$-dimensional hyperplane and all the variance can be explained inside this subspace. In our case $n\_samples$ is always $<$ $n\_features$, so the number of principal components will be limited by the number of data in the Train dataset.

We choose to run PCA analysis using coefficients that gave the best classification results in the previous analysis, Pearson correlation coefficients. We started our analysis choosing the proper number of PC to explain $> 90\%$ of variance and then gradually reduced them as reported in table 11.7

Results obtained from this analysis are reported in table 11.8 and shown in figure 11.7

PC	Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adversarial
800	AB I+II	69±2	73±3	71±1	71±2
400	AB I+II	67±3	73±5	72±3	73±3
	AB I EYE open	71±3	72±3	71±3	72±4
200	AB I+II	68±3	74±2	72±2	71±2
	AB I EYE open	69±3	73±2	72±4	73±3
100	AB I+II	67 ±2	72±2	69±1	70±2
	AB I - EYE open	70±2	73±2	71±4	72±2
50	AB I+II	66±4	70±6	68±3	69±2
	AB I - EYE open	70±4	72±3	69±3	72±3
20	AB I+II	65±5	67±4	64±5	66±1
	AB I - EYE open	68±6	71±2	71±6	71±3

Table 11.8: Classification resluts: AUC obtained during all the four analysis, with different number of Principal Components



(a) Classification on the entire dataset ABIDE I+II

(b) Classification on dataset ABIDE I - only open eyes

Figure 11.7: AUC scores obtained from Controls/ASD classification of data using different numbers of principal components during dimensionaliti reduction with PCA. REsults obtained with different analysis are marked with different colors: classification with data harmonization implemented inside the k-fold cross validation procedure are marked **blue**, classification of data harmonized upstream **orange**, classification of raw data (not harmonized) **green** and AUC scores of controls/ASD classification obtained with the adversarial network **red**.

Chapter 12

SHAP - Implementation and results

To instantiate the desired class, DeepExplainer in our case, we need the trained model and a fraction of the training data as background of arbitrary size N_{bkg} keeping in mind that the bigger is this background subset, the more accurate is the computing of shapley values, but the more computationally expansive this process will be, in particular it would occupy a vast amount of RAM memory and eventually run out of it; by the way, since the error on them $\sim 1/\sqrt{N_{bkg}}$, a background dataset made of 100 samples is a good compomise to have a relatively small error. We choose though, a background size of 500 samples for the entire dataset and a size of 100 samples when working on ABIDE I only open eyes, since we have a reduced number of train data to collect background data from.

Once the explainer is created on our model, we need to input as many test dataset as we want to explain, and for each one of them, the explainer will output an array containing a shap coefficient for all the n_features of each test data. Once inputed a batch of test data, we obtain a matrix of the same size of the test dataset we used, shaped n_samples x n_features. We choose to input all the test dataset for each fold to obtain a more accurate shap value for each feature.

Since we are presenting our results following a k-fold cross validation scheme, we had to implement this shap process inside the CV. To do so and collect the final result we computed the shap values for all the test samples and we stacked them together. At the end of the CV, shapley values are computed for the whole dataset, and we can proceed to visualize the results We can visualize the contribution of each feature on pushing or pulling the predicted output from the baseline output of our model.

Positive shapley values, representing positive contribution to our output are colored red and negative ones are blue. An example is shown in figure 12.1. Even if this is just an example taken from previous analysis on our data, and it appears clear that there is not a feature whose contribution is way greater than the others. Each feature makes a small contribution to the final outcome, but what we are looking, is for feature that, even if contributed poorly, are recurring through all the analysis.

To visualize the total amount of "contribution "for each feature, it is possible to visualize a bar plot of all the feature from the most to the least important. Each bar represents a feature importance, computed from equation 4.7. We choose to run this analysis only on Pearson correlation coefficients since they gave the best classification results and on the whole dataset consisting on ABIDE I + ABIDE II, only males and with an age range of 5-40. Our goal is

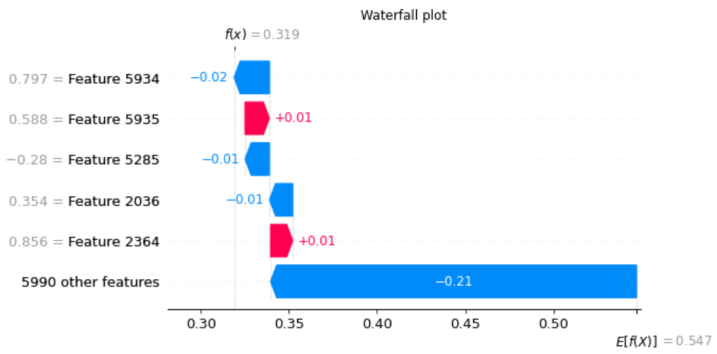


Figure 12.1: Example of a waterfall plot obtained with our data. For each row is displayes the feature's name and value and its contribution to the final output: negative contribution are colored blue and positive ones red. It is also marked the reference output in the bottom right corner, and the output of this instance on the top left.

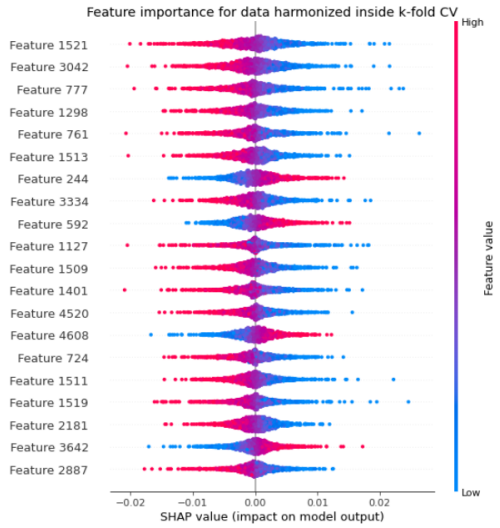
to check if before and after the harmonization procedure, the most important features still remain the same or if they change depending on the kind of analysis we are carrying out.

The plot of feature's importance can be either the module of shap values or shap values themselves (positive or negative), an example of the latter is figure 12.2: this plot shows for each instance (for each sample from the test data), the contribution of one particular feature to the final output related to that sample, and is represented as a single colored spot, the color indicated the strength of the contribution of that feature to the final outcome, and the position in respect to the vertical line, indicates the sign of that contribution: on the right we found those that gave a positive contribution (pushed the output towards values greater than the reference output), and on the left are placed those that had a negative contribution. This kind of plot, though, is not the best suitable for readibility in our case because of the large amount of features and the small contribution of each feature to the final outcome, in respect to other feature, or in other words there isn't a standing out feature, or group of feature that have a contribution way greater than the others, but in fact feature's importance has a quite smoothed descending trend. An example to visualize this trend is showed in figure 12.3 and thus, for a better readability we prefer reporting the results in terms of absolute shapley value, where we just consider the module of the shapley values, regardless its positive or negative contribution to the output, just to visualize the absolute contribution of that feature to a all the output.

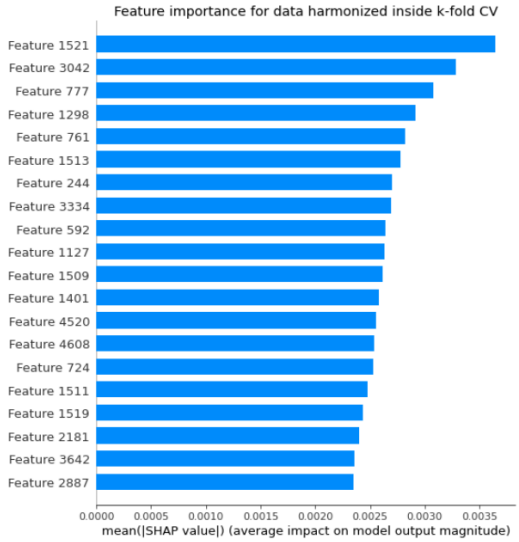
This bar plot is shown in figure 12.3

Feature importance with DNN We can see from this bar plots of the first 20 most important features that some feature seem to be persistent through all the four pipelines; if we limit our analysis on these first twenty we find that: feature 592 and 4521 appear, even if in different order of importance, in all our four analysis; features 244 592 777 1513 1521 3042 3334 3401 appear in 3 out of 4 analysis, Let's see what they represent:

- Feature 592: correlation between Right Middle Temporal Gyrus (anterior division) and Left Superior Temporal Gyrus (anterior division)



(a) Plot of dotted shapley values: for each feature each instance (each test sample) is represented by a dot, the position indicates a positive or negative contribution to the output, while the color represents the module of that contribution

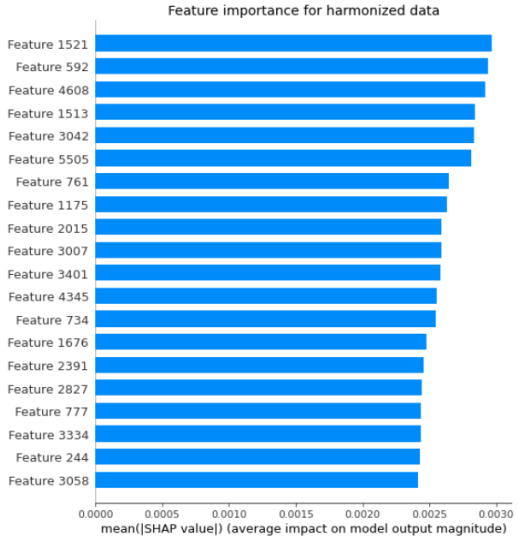


(b) Bar plot of shapley values: for each feature is represented the mean absolute value of all the shapl values computed for each instance

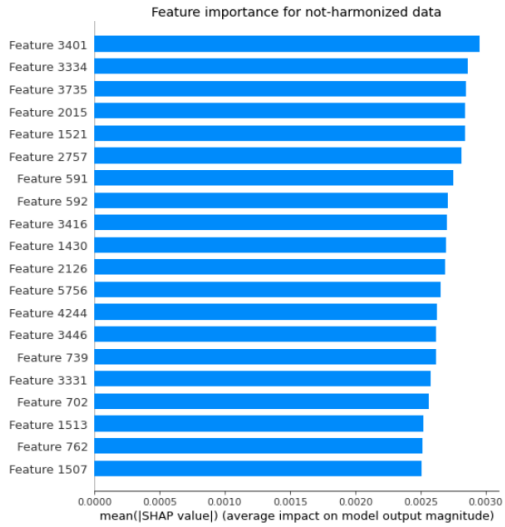
Figure 12.2: Two different plots of shap values for the first 20 important features during the K-fold harmonization peocedure

- Feature 1521: correlation between Left Angular Gyrus and Right Middle Temporal Gyrus (posterior division)
- Feature 244: correlation between Right Inferior Frontal Gyrus (pars triangularis) and Right Accumbens
- Feature 777: correlation between Left Middle Temporal Gyrus (temporooccipital part) and Right Middle Temporal Gyrus (posterior division)
- Feature 1513: correlation between Left Angular Gyrus and Right Temporal Pole
- Feature 3042: correlation between Right Frontal Orbital Cortex and Left Middle Temporal Gyrus (temporooccipital part)
- Feature 3334: correlation between Right Parahippocampal Gyrus (posterior division) and Right Accumbens
- Feature 3401: correlation between Right Parahippocampal Gyrus (posterior division) and Right Parahippocampal Gyrus (anterior division)

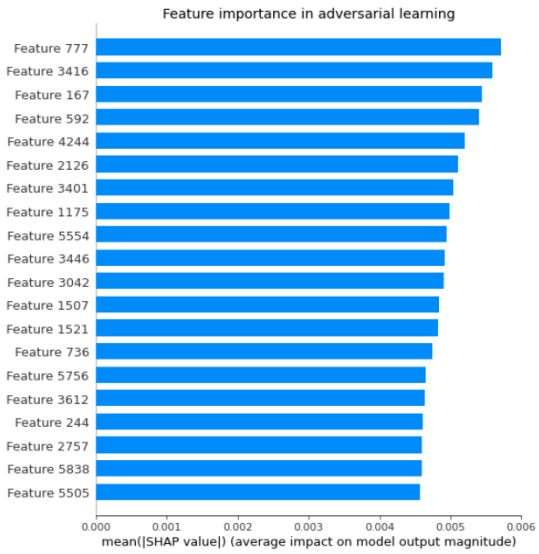
Feature importance with Random Forest Random forest: common to the three analysis are: 592, 1521, 1513, 3334 shown in figure B.1



(a) upstream

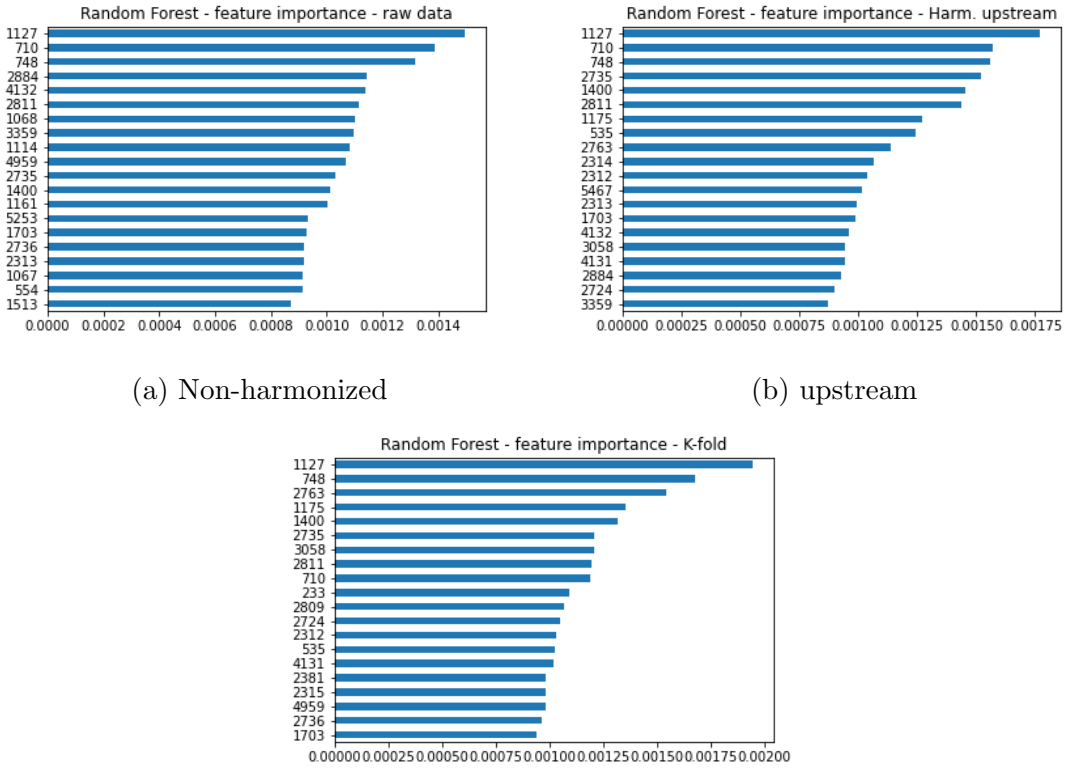


(b) Raw data



(c) adversarial

Figure 12.3



(a) Non-harmonized

(b) upstream

(c) kfold

Figure 12.4: Feature importance obtained with a Random Forest classifier: results on the entire dataset.

- Feature 592: correlation between Right Middle Temporal Gyrus (anterior division) and Left Superior Temporal Gyrus (anterior division)
- Feature 1521: correlation between Left Angular Gyrus and Right Middle Temporal Gyrus (posterior division)
- Feature 1513: correlation between Left Angular Gyrus and Right Temporal Pole
- Feature 3334: correlation between Right Parahippocampal Gyrus (posterior division) and Right Accumbens

Cross-analysis on feature importance Since we want to exclude from our analysis the procedure of upstream harmonization because it creates a bias in classification results, from now on, when we refers to harmonized data, we are referring to data with harmonization procedure implemented inside kfold.

To assess what are the most relevant recurrent feature between these different types of analysis, we choose to limit our analysis to the 1% important features among 5995, we collected the first 60 important features and we limited our analysis to only 5 classification procedures.

1. Classification with harmonization implemented inside k-fold using the DNN
2. Classification with harmonization implemented inside k-fold using the Random Forest classifier
3. Classification with raw data using the DNN
4. Classification with raw data using the Random Forest classifier
5. Classification using the adversarial neural network

Firstly we checked if there are common features among the first 60 important features of each method between these five classification. But we did not find anything in common. If we exclude, though, from our analysis the adversarial learning classification we find that features 1513, 3058, 1067 are common between the remaining classification methods. They represent:

- Feature 1513: correlation between Left Angular Gyrus (nan) and Right Temporal Pole (nan)
- Feature 3058: correlation between Right Frontal Orbital Cortex (nan) and Left Angular Gyrus (nan)
- Feature 1067: correlation between Right Postcentral Gyrus (nan) and Right Superior Temporal Gyrus (posterior division)

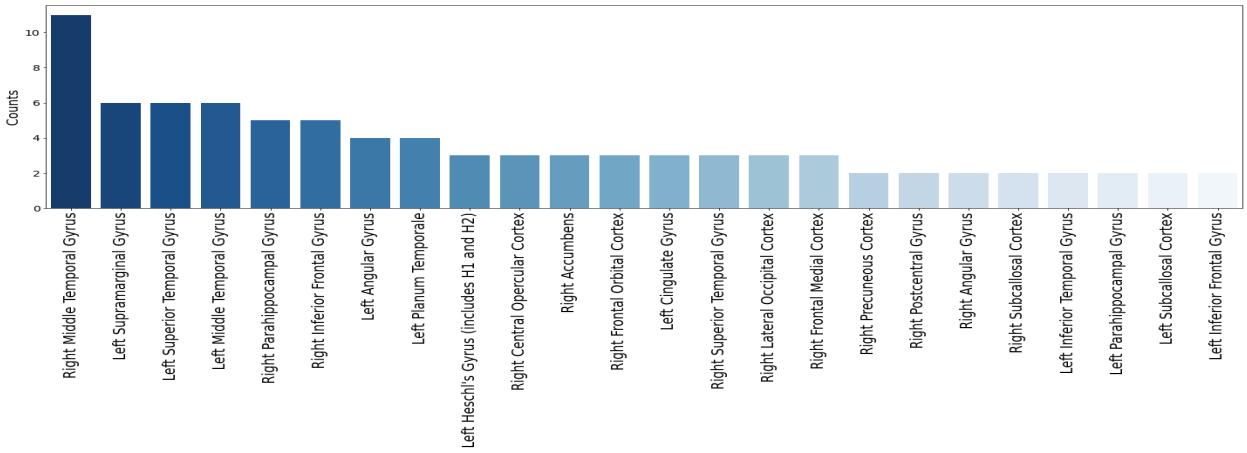
Since each feature is representative of a correlation between two brain areas, and each brain areas is involved in more than 100 features, we asked if there are some brain areas which are particularly discriminative and recurrent between the first 60 features of these classification pipelines. We plotted an histogram to represent the number of occurrences for each brain areas among the first 60 features for each classification procedure, and then we looked for common features between two classification procedure at a time and we created an histogram to number how many times a certain brain areas is involved in these common features throughout all the pairs of classification methods we are focusing on.

Let's give a look at the histograms of the most recurrent brain areas between all the 60 important features for each classification procedure: figure 12.5

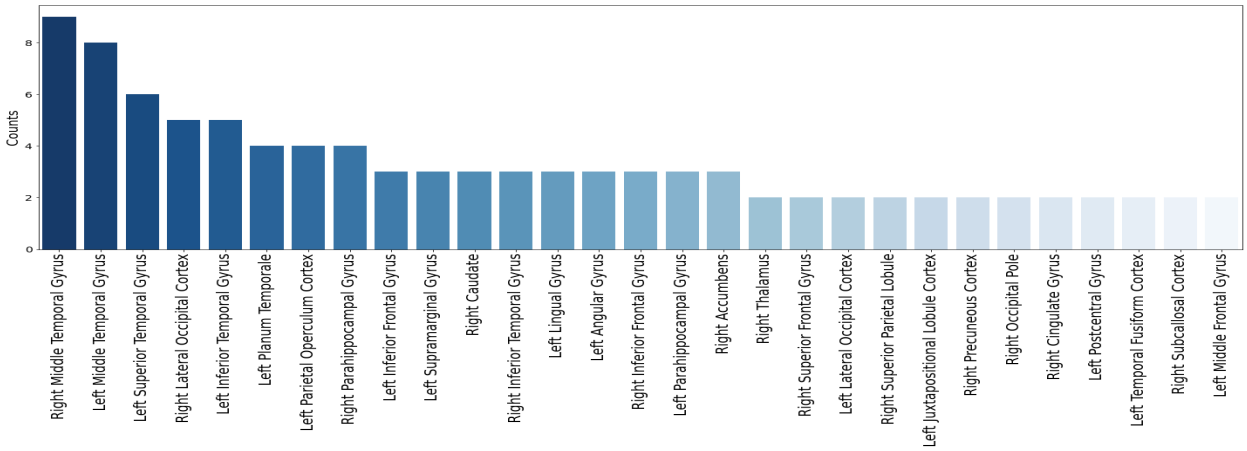
Now, we run analysis on subgroups of the previous classification methods and we report the number of common important features among the first 60 most relevant as a number and percentage.

- Focusing on random forest classifier, we compared important features between 2 and 4 and found that 33 features out of 60 are common (55%).
- Focusing on DNN we compared pipelines 1 and 3 finding that 22 features (37 %) out of 60 are common.
- Comparing the same analysis pipeline with DNN and Random Forest we find that with harmonized data: procedure 2 and 1 there are 13 common features (22 %)
- Comparing results on raw data obtained with DNN 3 and Random Forest 4 we obtain 7 common features (12%)
- Comparing the adversarial results 5 with the harmonized data with DNN 1 we obtain 21 common features (35%)
- Comparing the adversarial results 5 with DNN classification of raw data 3 we obtain 28 common features (47%)

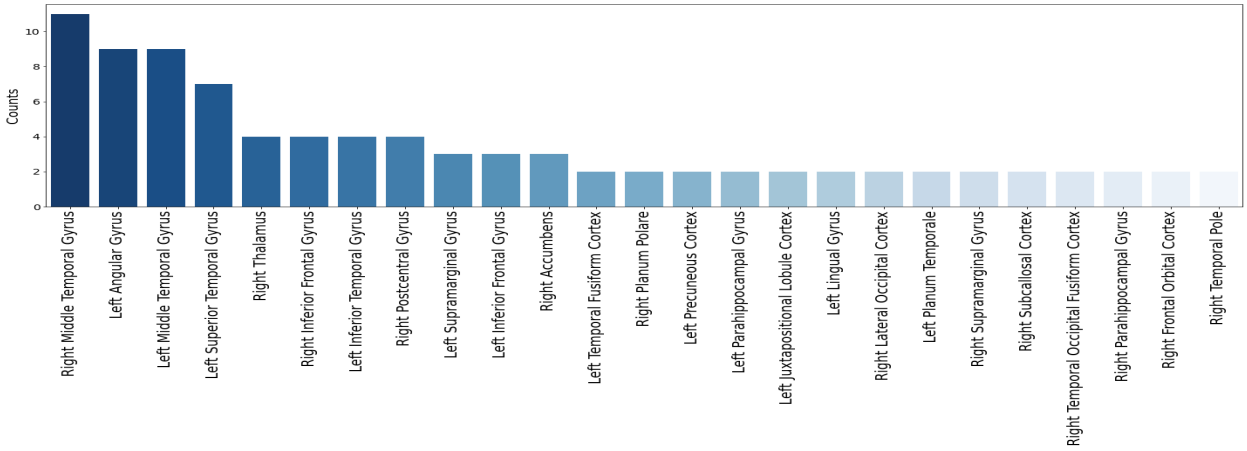
Since these common features are the most important we are willing to study if there's any brain areas significantly involved more than others. To this end, as mentioned above, we computed in figure 12.6 the overall histogram of the most recurrent brain areas between these common feature.



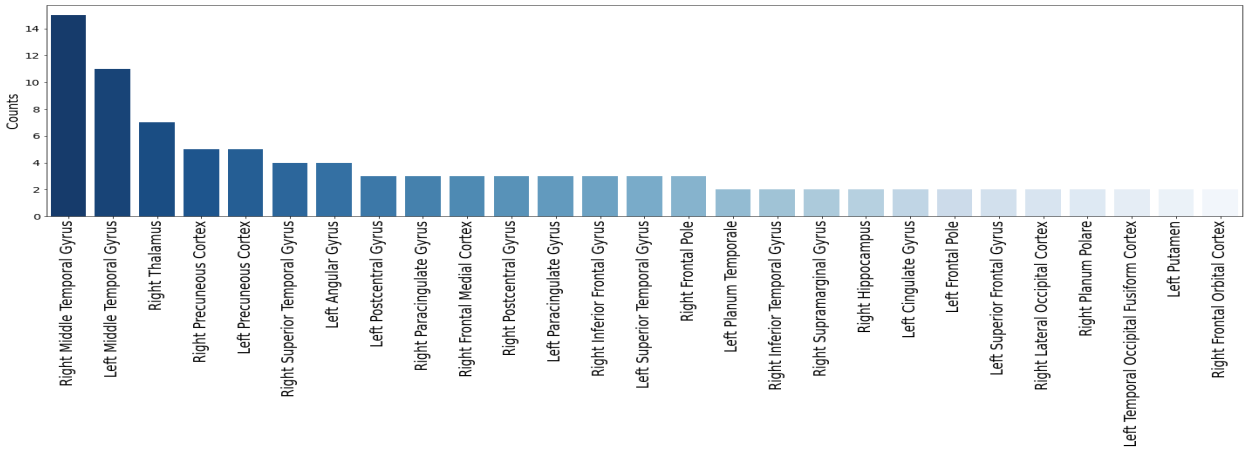
(a) Important areas from Raw data classified with DNN



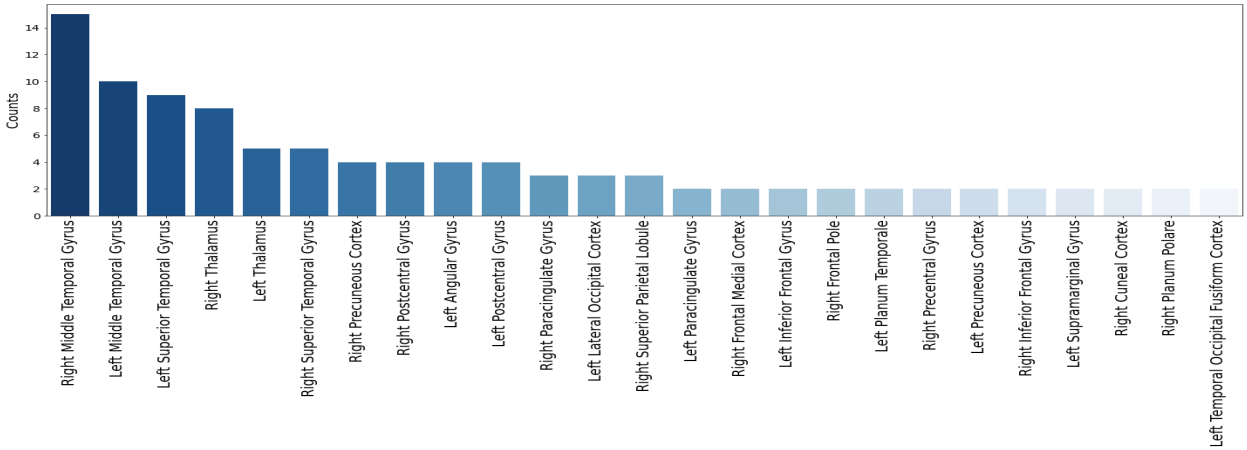
(b) Important areas from Raw data classified with Adversarial network



(c) Important areas from harmonized data classified with DNN



(d) Important areas from harmonized data classified with Random Forest



(e) Important areas from Raw data classified with Random Forest

Figure 12.5: Histograms of important brain areas extracted from the first 60 important features of different classification procedures

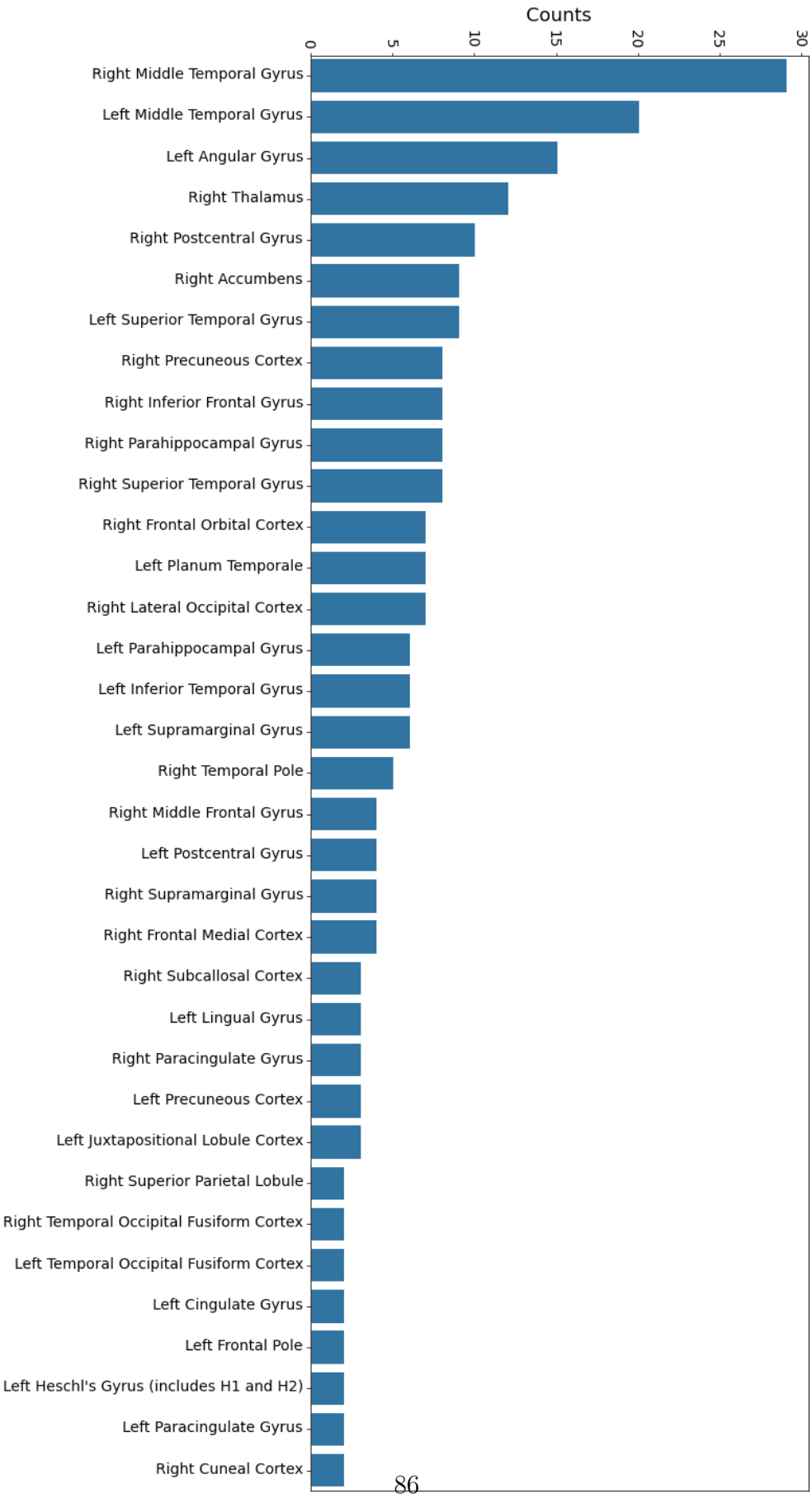


Figure 12.6: Histogram of the most important brain areas throughout all the analysis

Appendix A

Principles of MRI imaging

Nuclear magnetic resonance imaging, is a widespread imaging technique is widely employed in radiology to obtain anatomical images or study physiological processes, taking advantage of its flexible sensitivity to different tissues is based on the interaction of a nuclear spin of any tissue's molecule, with an external magnetic field $\mathbf{B}_0$. The principal molecule we are referring to is water, and subsequently, hydrogen is the dominant nucleus in MRI. When a nucleus undergoes an external magnetic field, the interaction would make the angular moment $\overrightarrow{\mu}$ align with the magnetic field, but, since protons have a thermal energhy associated with the temperature T, they are not static, so they start precessing around the symmetry axes given by the direction of the field $\overrightarrow{B_0}$. The force applied to a spinning proton under a magnetic field is given by

$$\overrightarrow{\tau} = \overrightarrow{\mu} \times \overrightarrow{B} \tag{A.1}$$

The rotation frequency ω_0 of the spinning proton is given by the Larmor equation

$$\omega_0 = \gamma B_0 \tag{A.2}$$

where γ is a constant related to the proton, in wates it has a value of $\gamma = 2.7 \cdot 10^8 \frac{rad \cdot s}{T}$ and B_0 is the external magnetic field. For a typical scanner with a magnetic field of 2-3 T, Larmor frequency assumes values of $\sim MHz$ just below radio waves. proton spins can assume two values: up and down in relation to the external field direction ; after a body underwent a magnetic field, its total magnetization is given by the net difference between spins up and down This net magnetization causes a flux which can be detected by an external coil only if it is perturbated from its equilibrium state. This is achieved by using another external magnetic field under the form of a radiofrequency impulse, with a frequence resonting with the Larmor frequence of preprocessing. This flips the spin in a direction usually perpendicular to the original B_0 one, which is supposed to be along $\hat{z}$ axis. Thus protons will start rotating in the x-y plane, still continuing preceding around their symmetry axis. Doing so, they gradually lose their initial energy and tend to realign along z-axis. This process is called relaxation and it follows an exponential decay with a time constant T_1 called longitudinal relaxation (longitudinal in relation to the original $B_0\hat{z}$ direction), or spin-lattice relaxation, referring to the loss of thermal energy through interaxtion with surroading lattice. The total magnetization along z-axis during time, after the rf impulse, will then regrow along z-axis following:

$$M_z(t) = M_0(1 - e^{-t/T_1}) \tag{A.3}$$

. An other effect to be take into account is the interaction of spin with a the local field where the external field and the neighbor's one coexist. This spin-spin interaction causes the dephasing of spin. This brings to two different relaxation velocities along z-axis and x-y plane called transversal relaxation which occurs without energy exchange, and is characterized by a time constant T_2. The two relaxation processes are described by Bloch equations and once integrated bring to the equation of the evolution of magnetization over time.

$$M_{x,y}(t) = M_{x,y}(0) \cdot e^{-t/T_2} \tag{A.4}$$

In a real system, however, there's and additional dephasing source, coming from external field inhomogeneities. This effect is often taken into account by the introduction of a different decay time T_2' which along with T_2 bring to a overall time costant given by $\frac{1}{T_{2*}} = \frac{1}{T_2} + \frac{1}{T_2'}$. Sometimes, though, this additional T_2' is so small compared to T_2, that dominates over it, resulting in a rapid loss of information. This can be avoided employing a specific rf pulse sequence called spin-echo method.

A.1 spin-echo method

The spin echo method employes two rf pulses the first with an angle of $\pi/2$ and the second of π.

1. The first radio-frequency pulse is applied with an angle of $\pi/2$ and this produces the spin to flip over a transversal plane and they gradually start dephasing because they undergo small field nonuniformities different from point to point, so they start fanning out, some move faster and some are delayed in relation to the average magnetization vector

2. The second rf impulse is sent, with an angle of π to the transversal magnetization axis y'. This ensures that all the accumulated phases flip and become negative,

3. Late and early phases continue to accumulate delay, but now this delay pushes them towards the main magnetization vector,and the phase will therefore return zero. The elapsed time up to this moment is called *Time to echo* T_E.

A.2 Image acquisition and k-space

For imaging acquisition, both effects must be taken into account. Moreover, it is necessary to relate a signal with a spatial position. In order to do so, in addition to the initial static magnetic field B_0 there's another field with lower intensity than B_0 which is not uniform in intensity, it follows a spatial gradient so that the total magnetic field along $\hat{z}$-axis is given by the sum of this two contributes, and the signal contains space-varying frequency components according to equation A.2 which can be rewritten as $\omega(z) = \gamma B(z)$ being x the spatial coordinate and with B_z now given by the relation $B_z(z,t) = B_0 + z \cdot G(t)$. The physical process underlying signal acquisition is Faraday induction, according to which an electrical potential is related to the variation of a magnetic flux over time $fem = -\frac{d\Phi}{dt}$ being Φ the

varying flux through the receiving coil. The varying flux is obtained just after the application of the rf pulse, while tipped spins are precessing and realigning to z-axis, and all this process is called free induction decay (FID). This coil detect only the signal deriving from transversal plane because the one along longitudinal axes would be saturated from the strong magnetic field of the main B_0 coil. Changes in acquisition sequences and rf pulses allow us to emphasise one of the three fundamental parameters T_1, T_2, ρ where ρ is the proton concentration inside the tissue.

An image acquired with a spin echo sequence can be T1, T2 or proton density weighted

The acquired signal is examined by inverse Fourier transform, exploiting the differences from Larmor frequency which cause a phase displacement along z-axis $\phi_G(z,t)$

$$s(t) = \int_0^t dz\rho \ (9.14) \ pag \ 145 \tag{A.5}$$

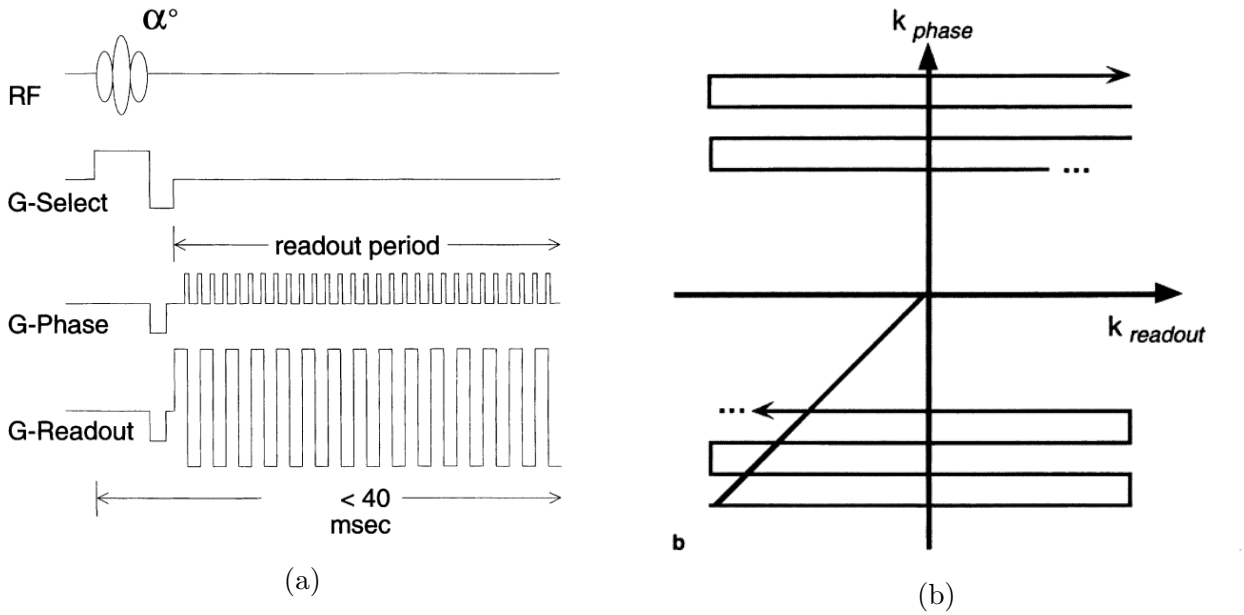
Data are acquired under the form of a matrix called k-space. K-space is a coordinate system used to store spatial frequencies information before applying the inverse fourier transform. Low spatial frequencies, corresponding to large object across the whole real image, are encoded at the center of the matrix and high spatial frequencies corresponding to small objects and finer detailes, are encoded in the peripheries.

The construction of k-space is done step by step in relation to the gradient applied time by time, producing a trajectory on the k-space. Each spatial line of k-space is acquired after TR, so the signal has to be recreated each time. Starting from k = 0, the center of k-space, after a preencoding gradient, we move toward negative position of both k-phase and k-frequency; subsequently a frequency encoding gradient (also called readout gradient) is applied and the signal is collected along the frequency encoding direction. The signal is then reformed and the process is repeated, from k = 0, to a new k-phase position and an other line of k-frequency is acquired.

A.3 Gradient Echo sequences

DA RIVEDERE QUESTA E EPI

The sequence employed to acquire functional imaging is the Echo Planar Imging (EPI) which is a particular sequence from the family of gradient echo sequences. Gradient echo sequences differs from spin echo because they allow a faster acquisition time because they use an excitation flip angle less than 90 deg, this allows a minor time interval to make spins back to their longituginal component as they are not completely flipped on the transverse plane. The transverse component just created decay and dephase according to T2*. If a gradient is applied, they dephase faster, and it is then reversed to make them rephase. In an gradient echo sequence, the echo peak lies on T2* decay curve while in a spin echo, it lies on T2 decay curve; and thanks to whis it is susceptible to any uniformity variation among which that due to hemoglobin variation. Since TR are short there is some transversal magnetization left after the acquisition, in *spoiled* Gradient Echo, the residual magnetization is wiped out. Ultra fast gradient echo start with a 180 degrees pulse, to invert longitudinal magnetization, in order to increase T1 weighting, and allk-space lines are acquired after this impulse. This first pulse is followed by minor pulses, and k-space is acquired after each echo.



(a) (b)

Figure A.1

A.4 EPI

During an EPI session, a single rf inpulse of 90 degrees is applied and the whole k-space is acquired in a single acquisition: as shown in figure A.1a after the initial RF pulse, together with the gradient for slice selection (G-Select), the acquisition of k-space information starts from the left bottom as in figure **??** and each line along k-frequency corresponging to the readout gradient (G-Readout) is acquired. The period of the G-Readout is the echo time T_E, and is modulated such that is sensible to T_2^* setting $T_E \approx T_2^*$. During the acquisition of the whole k-space each line is shifted from the other along k-phase direction by applying brief pulses of the phase encoding gradient (G-Select) called blip, creating a snake-like path during the acquisition of the k-space matrix.

Pixel-to-pixel frequency difference in the phase-encoding direction emphasizes chemical shift artifact. EPI generates gradient echoes under the FID curve created by the RF flip. As the curve decays according to T2*, gradient EPI sequences willl be T2* weighted. Sensitivity to T2*, of course, introduces sensitivity to artifacts caused by changes in magnetic susceptibility (eg, air/tissue interfaces) and imperfect magnet shim

Appendix B

Other classification results

Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I±II	60±1	71±3	63±2	60±1
AB I	61±2	69±3	62±2	61±2
AB 2	56±3	69±4	60±4	59±4
AB I ± II - EYE open	63±2	68±3	64±2	63±3
AB I open EYE	63±5	74±3	71±4	72±2
AB II EYE open	63±5	70±3	63±3	62±3
AB I ± II AGE=all, EYE=all	65±3	71±2	66±2	63±2

Table B.1: Classification scores using only out-of-phase Wavelet coefficients

Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I±II	67± 3	70± 3	66 ±3	66 ±3
AB I	70± 3	72 ±2	68 ±3	68± 3
AB 2	61± 4	65± 4	64 ±5	65 ±4
AB I ± II - EYE open	70± 1	73± 2	71 ±1	68 ±2
AB I open EYE	71± 4	74 ±3	71 ±3	68 ±2
AB II EYE open	62 ±4	66 ±3	68 ±3	64± 5

Table B.2: Classifiaction scores using (W_in - W_out) wavelet coefficients coefficients

B.1 Feature importance results on ABIDE I open eye dataset

The same tests we did to assess feature importance on ABIDE I + II dataset, we carried out on ABIDE I with only patient with open eye. Here in figure B.1 we report the results obtained from shap, related to the first twenty most relevant features. We can notice at a first glance that there are less feature common to all the four analysis: There are no common features among them that we can find across all these four analysis, if we limit our study on the first twenty. We have to search among the first thirty features to find something in common, which is still a good procedure since we are dealing with 5995 features and the first 40 are just the 0.7% of them.

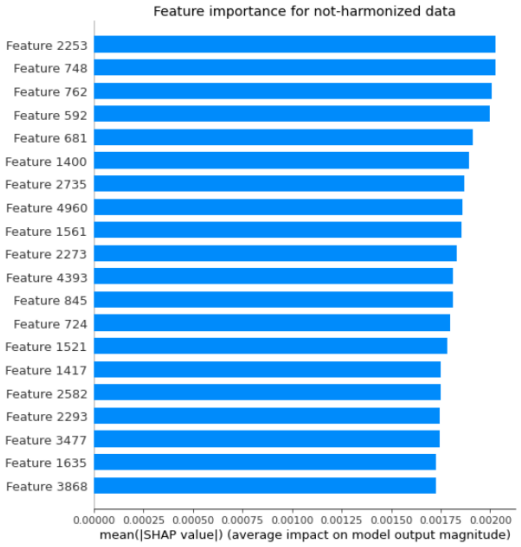
Structure	k-fold	upstream	no harmonization
3-2-1	60±2	63±3	60±4
8-8-1	64±1	65±4	64±5
64-8-1	69±1	72±2	68±1
64-32-8-1	68±1	72±2	69±1
128-8-1	70±2	71±3	69±2
128-64-1	70±1	72±3	69±2
264-8-1	70±2	73±3	70±2
512-8-1	71±2	73±2	70±2
1024-8-1	70±1	74±2	71±2
1024-32-1	70±2	74±2	71±3

Table B.3: Different model structures and related performances for the three main analysis we carried out during this entire work. Hilighted the structure we choose to employ through all our analysis

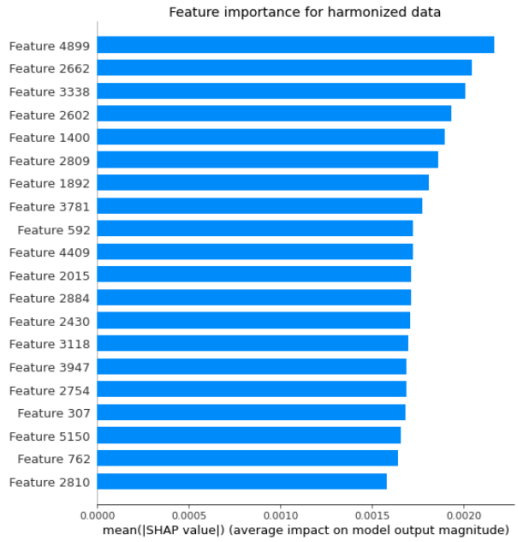
We find that, among the first 30 features, feature 1400, 762, 748, 3042 are common to the four analysis. If we limit again our study to the first 20 features, we can see that are common to three different analysis these additional features: 592 3477 3868

- Feature 1400: correlation between Left Supramarginal Gyrus (posterior division) and Right Inferior Frontal Gyrus (pars triangularis)
- Feature 762: correlation between Left Middle Temporal Gyrus (temporooccipital part) and Left Middle Frontal Gyrus
- Feature 748: correlation between Left Middle Temporal Gyrus (temporooccipital part) and Right Thalamus
- Feature 3042: correlation between Right Frontal Orbital Cortex and Left Middle Temporal Gyrus (temporooccipital part)
- Feature 592: correlation between Right Middle Temporal Gyrus (anterior division) and Left Superior Temporal Gyrus (anterior division)
- Feature 3477: correlation between Left Parahippocampal Gyrus (posterior division) and Right Precuneous Cortex
- Feature 3868: correlation between Right Temporal Fusiform Cortex (posterior division) and Right Inferior Temporal Gyrus (anterior division)

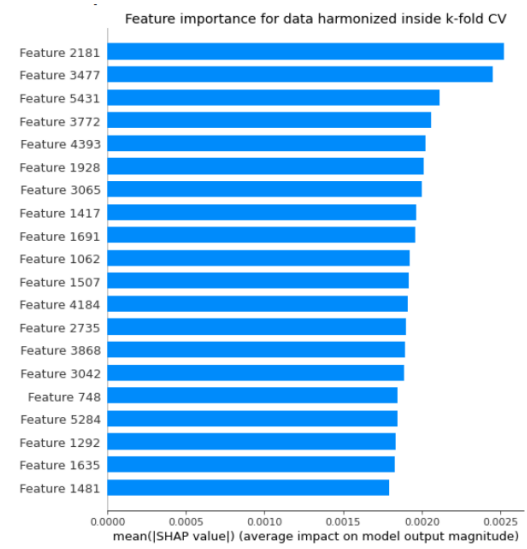
With a random forest we obtain the following feature importances, searching among the first 40 features, since as we can notice from figures **??** there are not common features between the first twenty. We find that features common to the 3 pipelines are: 3042, 1417, 748, 1400, 762



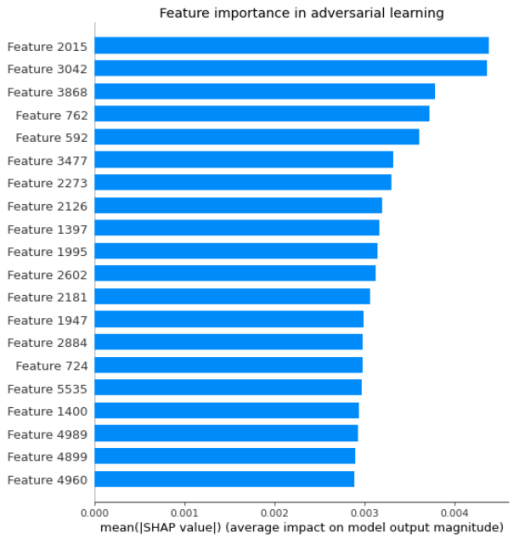
(a) Non-harmonized



(b) upstream

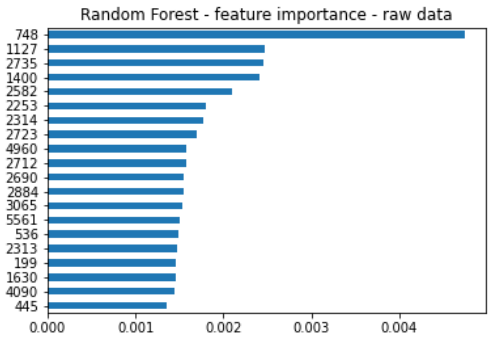


(c) kfold

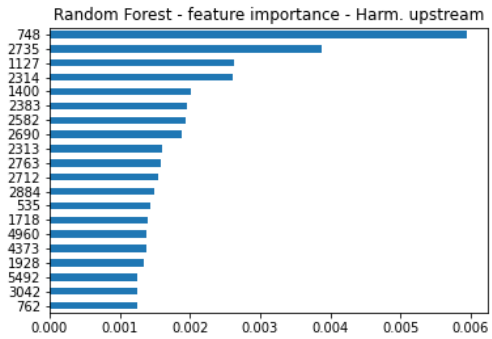


(d) adversarial

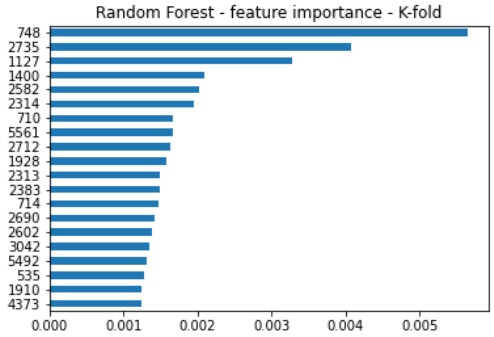
Figure B.1: Shap results on the dataset that gave the best classification performances: ABIDE I only open eyes.



(a) Non-harmonized



(b) upstream



(c) kfold

Figure B.2: Feature importance obtained from a Random Forest classifier: results on ABIDE I only open eyes.

- Feature 3042: correlation between Right Frontal Orbital Cortex and Left Middle Temporal Gyrus (temporooccipital part)
- Feature 1417: correlation between Left Supramarginal Gyrus (posterior division) and Left Middle Temporal Gyrus (temporooccipital part)
- Feature 748: correlation between Left Middle Temporal Gyrus (temporooccipital part) and Right Thalamus
- Feature 1400: correlation between Left Supramarginal Gyrus (posterior division) and Right Inferior Frontal Gyrus (pars triangularis)
- Feature 762: correlation between Left Middle Temporal Gyrus (temporooccipital part) and Left Middle Frontal Gyrus

Bibliography

[1] Ann Le Couteur, Gyles Haden, Donna Hammal, and Helen McConachie. Diagnosing autism spectrum disorders in pre-school children using two standardised assessment instruments: The adi-r and the ados. *Journal of Autism and Developmental Disorders*, 38:362–372, 2 2008.

[2] C. M. Freitag. The genetics of autistic disorders and its clinical relevance: A review of the literature, 1 2007.

[3] A Grinsted, J C Moore, and S Jevrejeva. Application of the cross wavelet transform and wavelet coherence to geophysical time series nonlinear processes in geophysics application of the cross wavelet transform and wavelet coherence to geophysical time series, 2004.

[4] SAMUEL B. GUZE. Diagnostic and statistical manual of mental disorders, 4th ed. (dsm-iv). *American Journal of Psychiatry*, 152:1228–1228, 8 1995.

[5] Karsten Müller, Gabriele Lohmann, Jane Neumann, Maren Grigutsch, Toralf Mildner, and D. Yves Von Cramon. Investigating the wavelet coherence phase of the bold signal. *Journal of Magnetic Resonance Imaging*, 20:145–152, 7 2004.

[6] John F. Nash. Equilibrium points in ¡i¿n¡/i¿ -person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1 1950.

[7] World Health Organization. The icd-10 classification of mental and behavioural disorders : diagnostic criteria for research, 1993.

[8] Sally Ozonoff, Beth L. Goodlin-Jones, and Marjorie Solomon. Evidence-based assessment of autism spectrum disorders in children and adolescents, 2005.

[9] Raymond Pomponio, Guray Erus, Mohamad Habes, Jimit Doshi, Dhivya Srinivasan, Elizabeth Mamourian, Vishnu Bashyam, Ilya M. Nasrallah, Theodore D. Satterthwaite, Yong Fan, Lenore J. Launer, Colin L. Masters, Paul Maruff, Chuanjun Zhuo, Henry Völzke, Sterling C. Johnson, Jurgen Fripp, Nikolaos Koutsouleris, Daniel H. Wolf, Raquel Gur, Ruben Gur, John Morris, Marilyn S. Albert, Hans J. Grabe, Susan M. Resnick, R. Nick Bryan, David A. Wolk, Russell T. Shinohara, Haochang Shou, and Christos Davatzikos. Harmonization of large mri datasets for the analysis of brain imaging patterns throughout the lifespan. *NeuroImage*, 208, 3 2020.

[10] Isabelle Rapin and Roberto F. Tuchman. Autism: Definition, neurobiology, screening, diagnosis. *Pediatric Clinics of North America*, 55:1129–1146, 10 2008.

[11] Helen V. Ratajczak. Theoretical aspects of autism: Causes—a review. *Journal of Immunotoxicology*, 8:68–79, 3 2011.

[12] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. volume 13-17-August-2016, pages 1135–1144. Association for Computing Machinery, 8 2016.

[13] Diana L Robins, Deborah Fein, and Marianne Barton. Modified checklist for autism in toddlers, revised, with follow-up (m-chat-r/f) tm, 2009.

[14] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. 4 2017.

[15] Kaustubh Supekar, Lucina Q. Uddin, Amirah Khouzam, Jennifer Phillips, William D. Gaillard, Lauren E. Kenworthy, Benjamin E. Yerys, Chandan J. Vaidya, and Vinod Menon. Brain hyperconnectivity in children with autism and its links to social deficits. *Cell Reports*, 5:738–747, 11 2013.