Titolo Tesi

Federico

May 18, 2022

Contents

1 Introduction: fMRI, Autism, ML

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 a neurodevelopmental disorder and refers to a broad range of conditions manifesting as deficits in social communication and interaction, repetitive behaviours, and even speech difficulties. Early signs start appearing by the age of 2 or 3, but since there isn't a definitive medical test to diagnosite it, often it is discovered even after adolescence. An early diagnosis then, would provide the help and support they need, and, treating it at its early stage would hopefully lead to the regression of this condition. The main factors that bring to the developing of autism aren't very clear so far, and is becoming clearer that there isn't just a cause, but a concatenation and cohexistence of various of them, the most strongly suspected causes are genetic and environmental causes. 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%, Ambiental factors such as prenatal air pollution or certain pesticides are also suspected to be associeted with a increase of the risk to develop this disorder, even thouh the etiology is way more complex and it can't be assessed so accurately. Currently ASD is diagnosided by syntoms-based tests concerning behaviour assessment, From a neurological point of view, ASD, manifests in a reduced information processing owing to synaptic dysfunction that manifest 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 particularly this is age-dependent, over connectivity is usually associated to young children while under connectivity, with adolescences, and adults

In the last decades, different approaches to study this problem have been carried on, trying different diagnostic tool to investigate this matter, such as magnetic resonance imaging (MRI), functional-MRI (f-MRI), or electroencephalography (EEG). 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 in that specific brain area, because of the bigger demand of oxigen and other nutrients to that brain area. Blood vessels also increase their size 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's 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. During the study of a functional connectivity pattern, different traits are involved and characterise the signal in a strong and evident way, among these: sex and age of the patient are some of the most important both for structural and functional brain properties. Limiting our focus just on functional data, other factor conditioning the final outcome of a f-MRI data acquisition are the FIQ (Full intellective quotient) or the eye status at scan, about this, functional connectivity with open steady eyes is different from closed eyes, and furthermore, closed eyes patient may be fallen asleep during the scan acquisition and this would heavily marking and modify the functional patterns.

It appears clear that the distinction of asd patient among normal ones it's not a simple and straightforwand task. One of the most promising and powerful tool to tacke non-linear problems is machine learning with artificial neural network. ANN belong to artificial inteligence tools and make use of an alghorithm that learns from data, and modify its parameters with the aim of recognise distinctive properties from the data and make prediction or decision on new unseen data. An relevant aspect that makes machine learning and artificial neural networks so popular is the ability to deal with non-linear problems introducing several non linear non-linear functions during training acting on their inputs or on the sum of them. A restraint of machine learning though, is that to perform well, an ANN needs to be trained on a big dataset, and the bigger the dataset, the better its performances are, expecially for deep neural networks. To this end, particularly in medical field, 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. 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 wouldn't be possible to uniform them to a single common acquisition protocol. To tacke this problem and try to uniform inter-scan variability, in this work, two different aproaches are proposed: analytical harmonization and a deep learning aproach. Harmonization is a procedure that aims to remove 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 extract from data both disease and site information and use site information to remove the bias toward that site before making a prediction.

2 Dataset: ABIDE I & II

Data we are working with belong to the ABIDE dataset (Autism Brain Images Data Exchange): a project aiming to investigate autism using resting state fMRI scans acquired from different medical centers, and putting them together to collect as many data as possible.

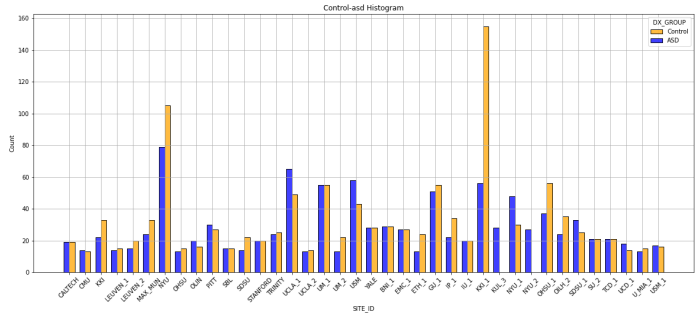
The whole ABIDE dataset was published in two releases: ABIDE I released in August 2012 and containing 1112 patient scans, and ABIDE II released in June 2016 containing 1114 scans. ABIDE I includes scans collected from 17 different sites, and the 1112 patients consist of 539 patients with ASD and 573 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.

Besides 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.

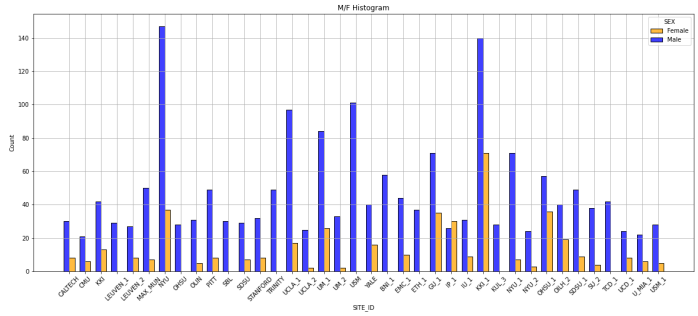
The vast majority of patients are males as shown in the histogram 1b, for a total amount of 1804 males and 422 females , while control/asd patient number, for each site is almost for each site as while the control/asd case per site is more or less balanced for almost every site, with the exception of KKI-1 that provided two times more controls than asd patients, and KUL-3 and NYU-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 1a

Patients in ABIDE dataset have ages ranging from 4 to > 50 years old, but as shown in figure 2a the vast majority of participant are younger than 40, precisely > 97 % of participant are under 40 y.o. so we can restrict our further analysys to patients belonging to this group. Intelligence quotient, whose distribution is shown in figure 2b, was not provided for every participant, in fact 171 out of 2226 values were not provided, and before proceding they were replaced by the mean of all the other values. The lack of a common acquisition protocol is also evident from the eye status at scan feature: as shown in figure 3b 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 3a.

For each patient in ABIDE both structural and functional images were provided.



(a)



(b)

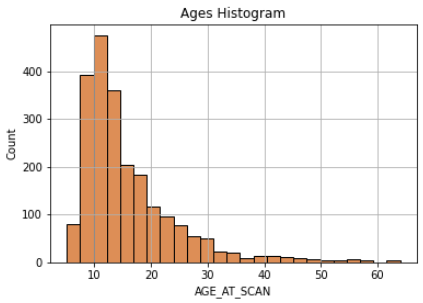
Figure 1

3 Image preprocessing

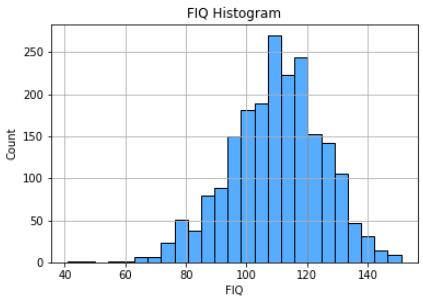
3.1 General overview

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

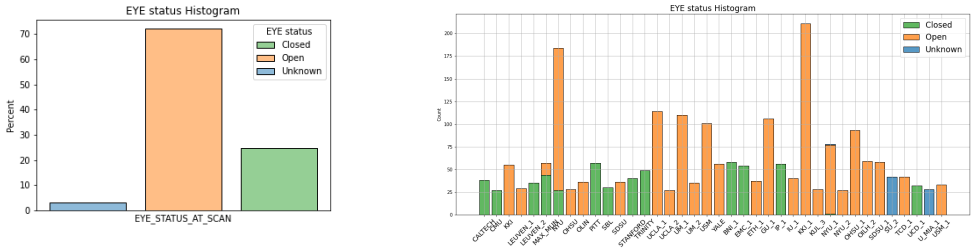


(a)



(b)

Figure 2



(a) (b)

Figure 3

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's 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 analyze 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 spaces is Tailarch and Monreal Neurological Institute's MNI-152, obtained from linearly coregistrating 152 T1 brain scans. A second registration step occurs within each patient to align EPI (functional) data to the structural image of that subject.

Looking at picture 4 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, it's possible to extract brain region information using an atlas. Atlases are in the same space as the template image (MNI space for example) and are brain images 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.

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 [**?**].

3.2 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 [1] for download. They consist of data from ABIDE I preprocessed by using

- Connectome Computation System (CCS)

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

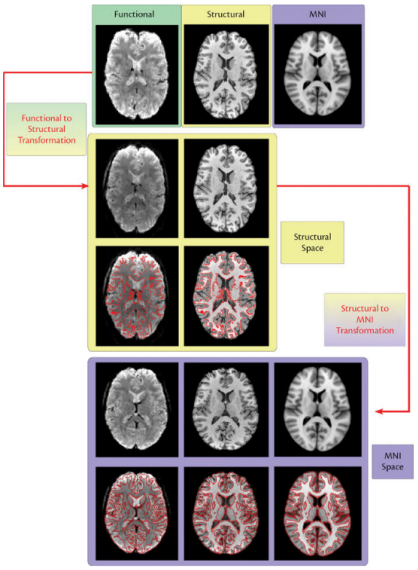


Figure 4: Registration steps: starting from the first row there are the acquired functional and structural images and the MNI template respectively The second row shows the functional image registered on the structural space The third row shows the same images as above, but overlaid with red boundaries extracted from the structural image The fourth row shown the final images both registered to the MNI 152 template

- 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.

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 speps 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. An atlas is a 3D standard brain template, each one including a different parcelization of the brain, which is obtained divding it into N labeled ROIs (Regions Of Interest) for anatomical and/or functional analysis. 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. According to previous studies, the atlas that gave best classification performances was HO, so this was the atlas we choose to employ for our work. The Harvard-Oxford 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 HO atlas employed by CPAC is provided by FSL library [2], 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 [3]. In this unified HO atlas (subcortical and cortical) the entire cerebral cortex area was removed because it's 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 it's named 3455; but there are no

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

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

information abut this ROI neither on Harvard-Oxford documentation nor on CPAC's/FSL's documentation, and because it's only made of 2 voxels it was excluded for our further analysis.

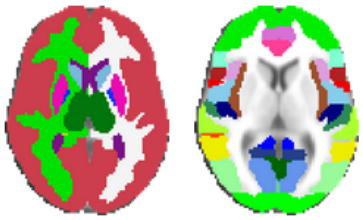


Figure 5: Harvard-Oxford subcortical and cortical atlas

As it's possible to notice from figure 6 timeseries extracted after our preprocessing pipeline do not exactly match those from ABIDE preprocessed, this is most likely due to two main reasons: the differeces 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.

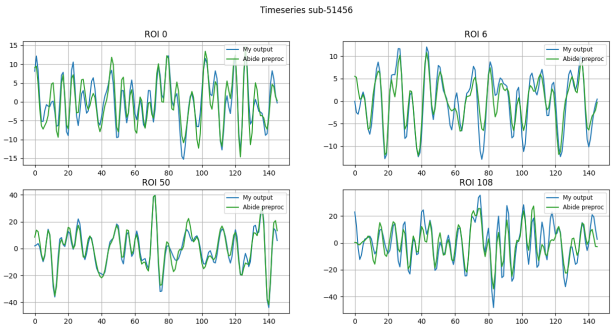


Figure 6: Comparison of 4 timeseries between ABIDE-preprocessed and our timeseries. Data show 4 randomly chosen ROIs: ROI 0, 6, 50, 108, belonging to patient 51456 from ABIDE I. It appears clear that the trend is the same, but there are some local differencies between the two plots due to differencies in

4 Data: Pearson and Wavelets

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

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

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.

4.1 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, it's 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.

$$Corr(x,y) = r_{xy} = \frac{Cov\,(x,y)}{\sigma_x \sigma_y} = \frac{E\left[(x-\mu_x)\,(y-\mu_y)\right]}{\sigma_x \sigma_y} \tag{2}$$

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

$$r_{xy} = \frac{\sum_{i=1}^{n}\left(x_i-\mu_{x_i}\right)\left(y_i-\mu_{y_i}\right)}{\sqrt{\sum_{i=1}^{n}\left(x_i-\mu_{x_i}\right)^2}\sqrt{\sum_{i=1}^{n}\left(y_i-\mu_{y_i}\right)^2}} \tag{3}$$

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 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 to test some hypotesis about correlations, we can run our test with the transformed variables which are normal distribuited with a known variance.

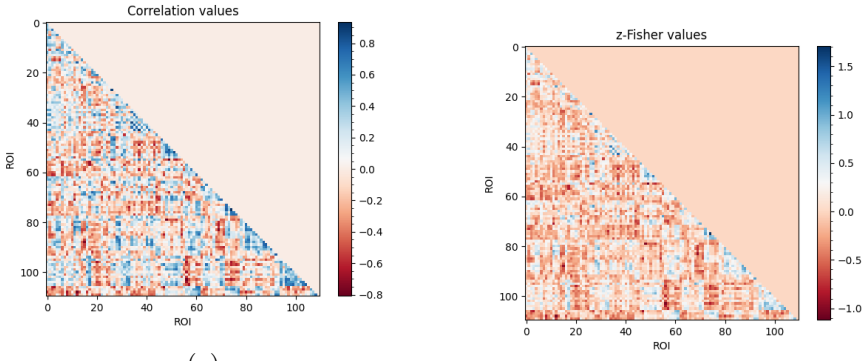
So this transformation allow us to obtaining a variable which is normally distribuited even when Pearson correlation coefficients follow a bivariate normal distribution or they are leaning toward the extremes.

$$z = \frac{1}{2} ln\left(\frac{1+r}{1-r}\right) = arctanh(r) \tag{4}$$

At the end of this analysis, we obtain correlation matrices like that shown as an example in fig 7, referred to patient 20243 from ABIDE I dataset.

4.2 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



(a)

Figure 7: Correlation and z-values matrix computed from timeseries extracted using Harvard-Oxford atlas, from patient 20243 belonging to ABIDE I dataset

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 cosine with different frequencies

$$f(t) = a_0 + \sum_{k=1}^{\infty}(a_k cos(2\pi kt/T) + b_k sin(2\pi kt/T)) \tag{5}$$

terms corresponding to the k-th frequency are called harmonics and are multiples of the fundamental frequency with k = 1.

The Fourier Transform is a natural extension of the Fourier series to aperiodic functions defined over the real axis, which doesn't allow a discrete superposition of sines and cosines terms, but it needs to be represented by a continue superposition. The FT transform takes a function from time or space domain and turn it into spatial or temporal frequency domain. It's a complex function because sines and cosines terms can be represented by a complex exponential.

$$\tilde{F}(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-i\omega t}dx \longrightarrow F(k) = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}} \tag{6}$$

To deal with discrete signal, for example a sequence x_n obtained from a discrete sampling of the continuous signal x(t), it's possible to make use of the Discrete Time Fourier Transform, wich allows us to write the $\tilde{F}(\omega) =$

On the other hand, the Discrete Fourier Transform, is useful to analyze 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.

$$\tilde{x_k} = \sum_{n=0}^{N-1} x_n \cdot e^{i2\pi kn/N} \tag{7}$$

where N in the total length of the timeseries an corresponds to the period of x_n as well; the frequencies at which samplings are computed are $\omega_k = 2\pi k/N$

One of the main drawback of FT though, is the lacking of spatial information, for example for a non stationary signal, it's possible that the signal contains a variable frequency component.

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 which provides information about frequencies and their spatial location by convolving the signal x(t) with a function w(t) called windowing function usually centered at zero which rapidly decays symmetrically.

$$\tilde{x}(F,T) = \int_{-\infty}^{\infty} x(t)w(t-T)e^{-2\pi iFt}dt \tag{8}$$

The most used functions w(t) are

1. Windowed Fourier Transform: $\psi_{ab}(x) = e^{ix/a}\phi(x-b)$ where $\phi(x)$ is a window function

2. wavelet transform: $\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\left(\frac{t-b}{a}\right)$ where a is a positive real number and defines the scale, while b is any real number and defines the shift parameters: it specifies the location of the wavelet in space or time.

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 it's a function local in both the temporal and frequency domain.

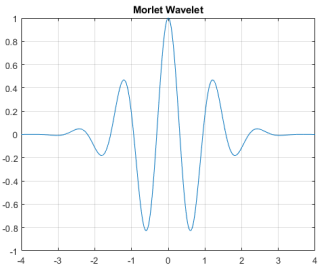
To be defined as so, a wavelet is required to satisfy two properties: to have zero mean (it must be oscillating) and unitary squared norm [**?**].

$$\int_{-\infty}^{\infty} \psi(x)dx = 0 \qquad \int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1 \tag{9}$$

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 9, such as Morlet, Meyer or Mexican hat. One of the most convenient for our analysis is the Morlet wavelet given by eq 10

$$\psi(t) = \pi^{-1/4}e^{iw_0t}e^{-t^2/2} \tag{10}$$



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 wavelets $\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\left(\frac{t-b}{a}\right)$. If the time-series is described by a function f, assumed to be real, in the equation below, the convolution of f and $\psi_{a,b}(t)$ is

$$W_{a,b}(f) = \int_{-\infty}^{+\infty} f(t) \cdot \psi^*_{a,b}(t)\, dt \tag{11}$$

It can be useful to visualize this as: given a and b, a wavelet is slided across the signal and for each time step a coefficient is extracted by multiplying the wavelet and the signal, and this operation is repeated for different values of a and b.

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 [4]

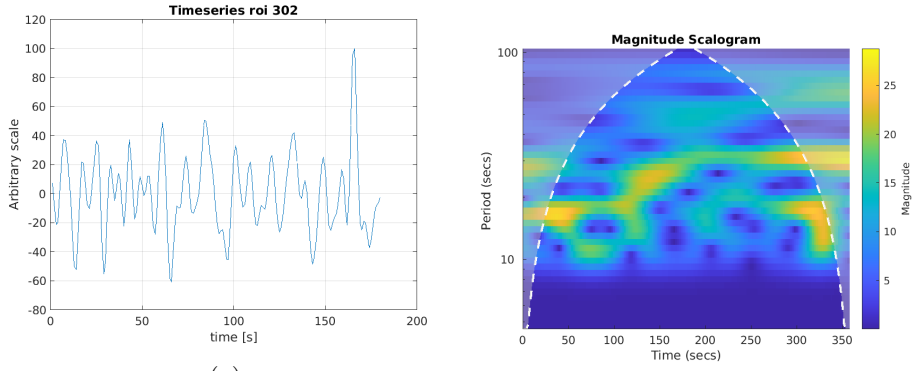
The difference between the continuous wavelet transform implemented on MATLAB and discrete wavelet transform lies in how finely stretching and shifting parameters are sampled: the CWT discretizes scale more finely than the discrete wavelet transform.

In the CWT, parameters are discretised based on a fractional power of two, by setting a = $2^{j/\nu}$ [**?**] [**?**] where ν, j are integers. 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 v, 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

Using CWT it's possible to obtain singal's information on a matrix: a time-frequency plot also called scalogram by convolving the signal, using different wavelets parameters a and b from a grid.

One problem that arise 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 a signal it extends itself outside the boundary of our data, and the convolution requires nonexistent values beyond the boundary. The confidence value obtained is then less accurate than other obtained for central values of time/space location. To overcome this problem it would be possible to accept this data information loss and truncate the values beyond boundaries; while an other aproach could be 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

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



(a)

Figure 8

outside the Cone of influence (COI).

Figure shows the timeserie belonging to ROI 308: left superior frontal gyrus of patient 51056 and its corresponding continuous wavelet transform.

In cwt figure, x axis corresponds to space or time points and on the y axis frequencies or periods from "a' parameter are shown. Values outside the COI (shown as a dotted white line) are shown with a lighter faded.

Continuous wavelet transform can only be used to analyze one signal at a time; to analize and compare two signals an analysis called wavelet coherence has to be carried on. From two CWTs it's possible to compute the Cross Wavelet Transform (XWT) which allow us to examine their cross-wavelet power and relative phases.

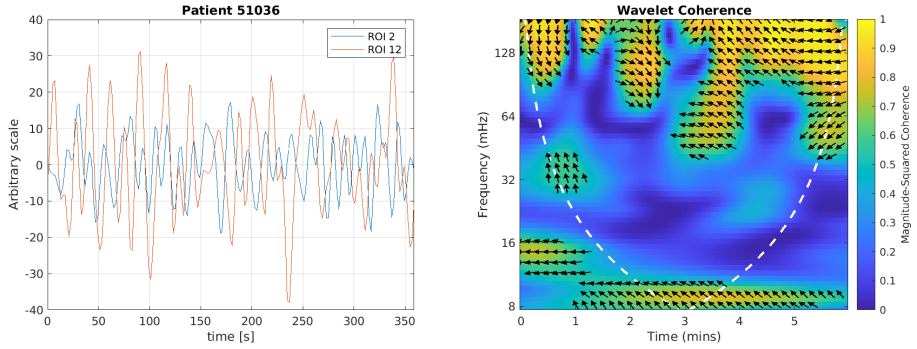
Using XWT is then possible to compute the Wavelet Coherence.

Denoting as $W^X(a,b)$ the continuous wavelet transform of a signal X, it's defined the *wavelet power spectrum* of a signal X(n) as $W^{XX}(a,b) = W^X(a,b)\left(W^X(a,b)\right)^*$ where the * represent the conjugate transpose. Similarly, the cross-wavelet spectrum of two time series X(n) and Y(n) is defined as $W^{XY}(a,b) = W^X(a,b)W^Y(a,b)$ whose module $|W^{XY}(a,b)|$ represents the amount of joint power between the two time series, and it's called *cross wavelet power* - From it it's possible to compute the the complex argument $\Delta\phi(a,b) = arctan\left(\frac{Im[W^{XY}(a,b)]}{Re[W^{XY}(a,b)]}\right)$ which represents the phase between X(n) and Y(n).

A squared cross-wavelet coherence $R^2(n,s)$ is defined as reported in equation 12. This coefficient ranges in the interval (0, 1) and represents the localized correlation coefficient between x and y in the time-frequency domain.

$$R^2(a,b) = \frac{|S(W^{XY}(a,b))|^2}{S(|W(a,b)^X|^2)S(|W(a,b)^Y|^2)} \tag{12}$$

To assess the level of significance over the noise, the statistical significance level of wavelet coherence has to be estimated using Monte Carlo methods generating a large (1000) samples of red noise for each time series, with the same 1-lag autoregressive coefficients (AR1 coefficient). For each pair of noise, a wavelet coherence matrix is calculated and stacked to obtain a $n \times m \times x \times 1000$ 3D noise matrix to extract the distribution of the entries. This null distribution of wavelet coherence coefficient is used to estimate



(a) (b)

Figure 9

the threshold to 5% significance level for the subsequent analysis.

To obtain a correlation-like coefficient from every two timeseries we are going to analyze, we use wavelet coherence coefficients from the matrix and their phase information to compute the time of in-phase coherence which can be seen as the percentage of time synchronicity between the two ROIs, defined as

$$c_{ij} = \frac{100}{N} \sum_{a,b}^{N} I\left\{R_{ij}^2(a,b) > a_{95}\right\} \cdot I\left\{-\frac{\pi}{4} < arg_{ij}(W^{XY}(a,b)) < \frac{\pi}{4}\right\} \quad (13)$$

Where 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 computer wc coefficient is regarded as significative.

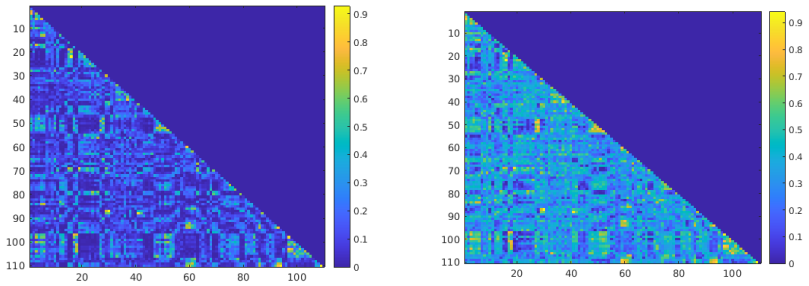
Similarily to this coefficient is the time of counter-phase coherence modifying from the formula above the phase condition into $I\left\{arg_{ij}(W^{XY}(a,b)) < -\frac{3\pi}{4} \ \vee \ arg_{ij}(W^{XY}(a,b)) > \frac{3\pi}{4}\right\}$

In short with this analysis, we are just considering coefficients with an high significance level (above 95%) and with a small, or big phase shift (between $-\frac{\pi}{4} and \frac{\pi}{4}$ and the opposite).

Wavelet coherence maps were calculated using MATLAB's Wavelet Toolbox, which employs the Morlet wavelet, and analyze the frequency range using 12 subscales per octave and 9 octave.

Figure 9 represents the wavelet coherence scalogram obtained from two ROIs of patient 51056.

From equation 13 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 50267 is shown in figure 10



(a) Coefficients of In-phase percentage

(b) Coefficients of Off-phase percentage

Figure 10: Correlation matrix created with Wavelet coefficients of in-phase and off-phase percentage. These come from patient 50267, ABIDE I

5 Machine learning

Machine learning is a branch of Artificial Intelligence (AI) that aims to learn from dataand gradually improve its accuracy. Algorithms are trained to make classifications or predictions and the bigger is the dataset to train the algorithm, the better accuracy and generalization can be achieved.

Machine leraning algorithms can be divided into two macro areas: supervised and unsupervised learning. From a dataset, containing features, an unsupervised algorithm's 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.

They usually aim to learn the probability distribution of the features in our data, and to do so, they try to find a new representation of the data, often simpler than the initial one. Simpler can be associated with

- lower dimensionality of data namely the reduction of the space where data lie of informally a reduction of the number of features in our data
- sparse representation of data transform the data in a representation containing zeros in most entries, to try to aligh input data along this new representation's axis. Usually resulting in an increasing of data dimensionality to avoid information lost deriving from setting most entries to zero
- Independent representaiton focus on the main sources of variance across data, trying to create statistically independent representation ofa initial data.

On the other hand, in supervised learning each input data is associated with a label, specifying the class that data belongs 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.

A sub-branch 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

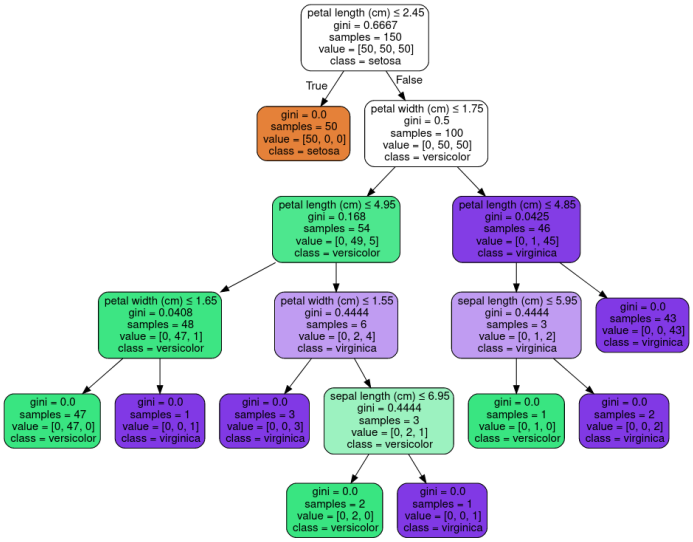


Figure 11

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.

5.1 Random forest

Random Forest classifier belongs to the category of *ensemble* classifiers, it combines multiple decision tree models to create a more poverful model, so to understand a random forest 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 construct trees so that the paartition are informative about the class labels. To practically construct a tree the algorithm

1. Randomly select m of the n total features from input data
2. Split the node by asking a question about the feature that would provide the best classes’ split
3. Iterare point (1) and (2) until a single data point is left

An example of a decision tree dealing with flower classification is shown in figure 11

Usually though, continue this iterative process untill it reaches a leaf, brings to the creation of a very 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 one class. An other strategy, which leads to the construction of a Random Forest, is the creation of a stronger model, more prone to generalize data without overfitting. 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 in a different way, so that we can reduce overfitting by averaging different results.

1. Randomly select n samples from the input dataset
2. Grow a decision tree from this sample
3. Putting all the predictions together it's possible to assign a class label by majority vote.

Randomness within trees of a single model is introduced with three strategies: 1. Randomly select some input data and buiding a tree with this subset of data 2. Randomly select a different subset of features at each split 3. A combination and implementation of both the previous strategies results in a model in which each tree has reduced predictive power, but when put all together, it results in an improved predictive power with a drastical reduction of overfitting.

Introducing this kind of randomness in a model results in reducing correlations between models and this brings to a reduced variance between outputs So the necessary steps to build a random forest are

5.2 Deep Learning: ANN

-Aggiungere: Dropout

Deep learning is a subset of machine learning whose algorithm try to emulate the learning process of the human brain. This kind of algorithms are called Artificial Neural Networks (ANN) and, such as a neuronal structure, they are made of several neurons organized into layers; each layer containsa certain amount of neurons each one linked to each others.

The simplest model of a neural network is the *perceptron* shown in figure , corresponging to a single neuron which 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 computed by applying a function to a linear combination of all the inputs $y = f(z) = f\left(\sum_i w_i x_i + b\right)$ This function that determines the outpu is called *activation function.*

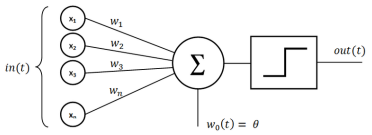


Figure 12: Perceptron

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 neutron acting like a perceptron, produces an output using the activation function which is the same for every neurons of the layer, but

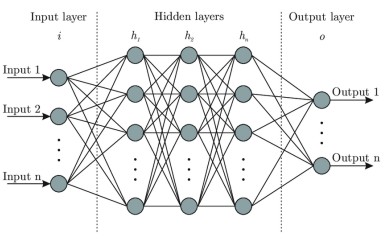


Figure 13: Artificial Neural Network

can differ from the activation function of other layers. Once collected each and every output from all the first layer neutrons, they are passed to the second layer and become the input to each neutron belonging to this layer. This is reiterated through all the layers up to the final output layer. As schematized in figure 13 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 neutrons 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.

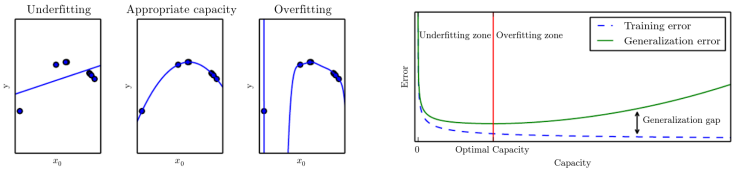
To fully define a neural network we just need some parameters called hyperparameters, the principals of which are:

- Number of layers
- Number of neutron 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



(a) (b)

Figure 14

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 14 the linear model is not able to describe the data's 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.

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 connections, during each training cycle. The percentage of neutron to drop, correspons to the probability p for a single neutron to be dropped and it is set by the user. A visual representation of what occurs is illustrated in figure 15 When discarding some neutrons in a layer, the remaining neutrons needs to rescale their weights toaccount the missing connection; doing so, every neutron 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 common adopted since its introduction in 2015. It is based on the ob-

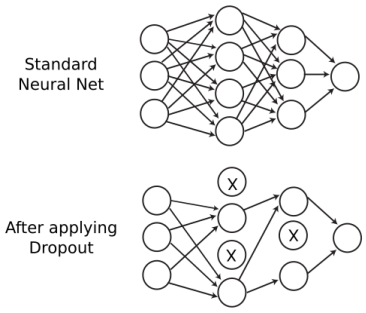


Figure 15

servation that a neural network works and performs better when its input are normalized, because this prevents the saturation of its neurons which occurs when because of an high input value, the neuron outputs value 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 scaled so that $x_i^l \to \tilde{x}_i^l = \frac{x_i^l - \mathbb{E}[x_i^l]}{\sqrt{Var(x_i^l)}}$

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, it's possible to actually train the model using the entire Train 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.

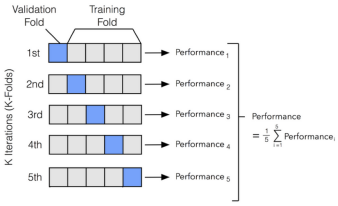
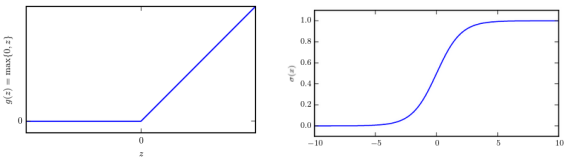


Figure 16: Schematic representation of a k-fold cross validation procedure

5.3 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



(a) ReLU (b) Sigmoid

Figure 17

- The ReLU functions $\phi(z) = max(0, z) = \begin{cases} 0 \ for \ z < 0 \\ z \ for \ z > 0 \end{cases}$ which returns the maximum 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 $f(x) = \frac{1}{1+e^{-x}}$ 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

5.4 Loss functions

DA RIVEDERE E SISTEMARE NOTAZIONE

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, we try to modify the network's weights to reduce this value searching for a minimum of this function.

This algorithm employed by a feedforward network for weights' fine tuning is called *backpropagation* (backward propagation of errors). 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 case of multiclass classification we call this function categorical crossentropy, while if we only have two classes, we are then 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:

$$H_p(q) = -\frac{1}{N} \sum_{j=1}^{N} y_j log(\hat{y}_j) + (1 - y_j) log(1 - \hat{y}_j) \tag{14}$$

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 $\{\theta_i\}_1^m$ $L(\theta_1, ..., \theta_m | y_i) = P(y_i | \theta_1, ... \theta_m)$.

For a subset of observed value we can write the likelihood as $L = P(\hat{y}_1, ..., \hat{y}_n | \theta_1, ... \theta_m)$ that in case we are dealing with independent variables is just the product of the single probabilities $L = \prod_{i=1}^{n} p(\hat{y}_i | \theta_1, ... \theta_m)$.

Our goal is to perform the maximization of this function during a logistic regression: A logistic regression is the estimate of the best model's parameters that models the discrete probability of two events:

In our practical problem 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 so in short D = $\{(\mathbf{x_i}, y_i)\}$.

Given an input data $\mathbf{x_i}$ and an activation function (sigmoid for example, which we recall is of the form $\sigma(z) = \frac{1}{1+e^{-z}}$) the probability of x_i to belong to the class $y_i = 1$ is given by

$$P(y_i = 1 | x_i, \mathbf{w}) = \sigma(\mathbf{x_i^T w}) = \frac{1}{1 + e^{-\mathbf{x_i^T w}}} \tag{15}$$

which in terms of a regression can be wrote as 1 - the probability to belong to the negative class namely

$P(y_i = 1) = 1 - P(y_i = 0)$

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 observing our data under our model

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

Since we are interested in maximize the likelihood, the best parameters are those that maximize the likelihood: $\begin{cases} \frac{\delta L}{\delta \theta_1}(\theta_1, ..., \theta_m) = 0 \\ ... \\ \frac{\delta L}{\delta \theta_1}(\theta_1, ..., \theta_m) = 0 \end{cases}$ To compute the derivative of a product is a nontrivial task, hence it's 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 $log(L) = \sum_{i=1}^m y_i log(\sigma(x_i^T \mathbf{w})) + (1 - y_i) log(1 - \sigma(x_i^T \mathbf{w}))$ which taken with a negative sign corresponds to the cross-entropy function.

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

$P(y_{i,m'} = 1|\mathbf{x_i}, \{\mathbf{w_k}_{k=0}^{M-1}\}) = \frac{e^{-\mathbf{x_i^T w_{m'}}}}{\sum_{m=0}^{M-1} e^{-\mathbf{x_i^T w_m}}}$

and the relative negative log-likelihood will be in the form:

$L = -\sum_{i=1}^n \sum_{m=0}^{M-1} y_{im} log(P(y_{im} = 1|x_i, w_m)) + (1 - y_{im}) log(1 - P(y_i m = 1|x_i, w_m))$ where there's a summation over the index i which indicate the i-th input data and over the index m, concerning all the possible M classes.

5.5 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 ..

5.6 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's 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

$$\begin{aligned} X^T X &= (USW^T)^T (USW^T) \\ &= WS^T U^T USW^T \\ &= WS^2 W^T \end{aligned} \tag{16}$$

where we used the definition of orthogonal matrix for U $U^T U$ = 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

$$\begin{aligned} Var[Z] &= \frac{1}{m-1} Z^T Z \\ &= \frac{1}{m-1} W^T X^T X W \\ &= \frac{1}{m-1} W^T W S^2 W^T W \\ &= \frac{1}{m-1} S^2. \end{aligned} \tag{17}$$

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.

$$\frac{\lambda_i}{\sum_{k=0}^{p} \lambda_k} \tag{18}$$

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.

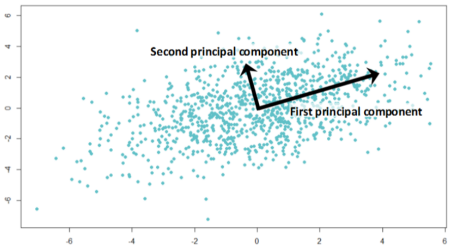


Figure 18

5.7 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

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{19}$$

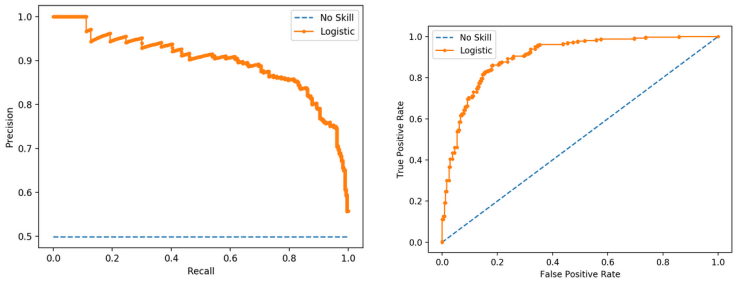
Even though accuracy may seem a good parameter, it isn't the most accurate one, when we're dealing with a unbalanced test dataset, if for example, in a binary classification with A and B classes, if just 2 out of 10 samples belongs to a different class, let's say A and B, and the model predicts every 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.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{20}$$

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 it's 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 thresolds. This is achieved by studying the precision-recall curve of which is reported an example in figure 19a. 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 thresolds' effect, it's 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



(a) (b)

Figure 19

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 19b

6 Harmonization

The issue of collecting a large dataset putting together data from multiple centers allow us to work on dataset of bigger dimension and facilitates the generalization and the robustness of the model, but, as a drowback, it introduces variability in our analysis such as differences due to scanner 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 called harmonization. Harmonization was tested on different datasets and proven to be an effective way to reduce scanner effects **??** **??** and on functional connectivity data showing to reduce scanner effect.

6.1 Harmonization - theory

The state-of-art procedure used in genomic is called ComBat (meaning of combat?), specifically adapted for functional connectivity to mitigate scanner effects, is based on a previous method initially proposed [reference to johnson, fortin] in 2007, for gene expression studies to compute batch effect corrections, and then implemented by Fortin et al **??** . ComBat technique is based on the simpler Location and scale adjustment methods, but it's an improved version of it: 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.. The model assumes that a feature can be modeled as a linear combination of the biological variables plus the site effect, and the erroes 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

$$y_{ijf} = \alpha_f + x_{ij}^T \beta_f + \gamma_{if} + \delta_{if}\epsilon_{ijf} \tag{21}$$

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

The location-and-scale-adjusted data y_{ijf}^* are given by $y_{ijf}^* = \frac{y_{ijf} - \hat{\alpha}_f - X \cdot \hat{\beta}_f - \hat{\gamma}_{jf}}{\hat{\delta}_{jf}} +$ $\hat{\alpha}_f + X \cdot hat\beta_{jf}$, where $\hat{\alpha}_f, \hat{\beta}_f, \hat{\gamma}_f, \hat{\delta}_{jf}$ are estimators of the corresponding parameters, based on the model.

ComBat uses empirical Bayes to improve the variance of the parameters estimate $\hat{\gamma}_{if}$ $\hat{\delta}_{if}^2$. Specifically it is assumed that that the site-effect parameters follow the normal distribution and the inverse gamma distribution respectively $\gamma_{if} \sim N(Y_i, \tau_i^2)$ $\delta_{if}^2 \sim \frac{\beta^\alpha}{\Gamma(\alpha)}(1/x)^{\alpha+1} \cdot e^{-b/x}$

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

1. 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 22

$$Z_{ijf} = \frac{y_{ijg} - \hat{\alpha}_f - X\hat{\beta}_f}{\hat{\sigma}_f} \tag{22}$$

2. Site effect parameters estimate using parametric empirical priors: assuming that standardized data follow the normal distribution $Z_{ijf} \sim N(\gamma_{jf}, \delta^2 jf)$ it is also assumed that that the site-effect parameters follow the normal distribution and the inverse gamma distribution respectively $\gamma_{jf} \sim N(\eta_j, \tau_j^2)$ $\delta_{jf}^2 \sim InverseGamma(\lambda_j, \theta_j) = \frac{\theta^\lambda}{\Gamma(\lambda)}(1/x)^{\lambda+1} \cdot e^{-\theta/x}$

   Where the hyperparameters $\eta_j, \tau_j^2, \lambda_j, \theta_j$ are empirically estimated from standardized data using the method of moments so we obrain γ_{jf} *and* δ_{jf}^2 and we use them for the third and last step

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

$$y_{ijf}^{ComBat} = \frac{\hat{\sigma}_f}{\delta_{jf}}(Z_{ijf} - \gamma_{jf}) + \hat{\alpha}_f + X\hat{\beta}_f \tag{23}$$

6.2 Harmonization - results

As explained before, 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 [5] package for Pyhton, developed by Pomponio et al. **??** which implemented and added some features to the previous NeuroComBat Python package [6].

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 6.1 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.

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 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. To avoid discarding these data, just for these two sites, we substituted NaN values with original unmodified data.

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

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

Figure 20 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 20b 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 20a 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.

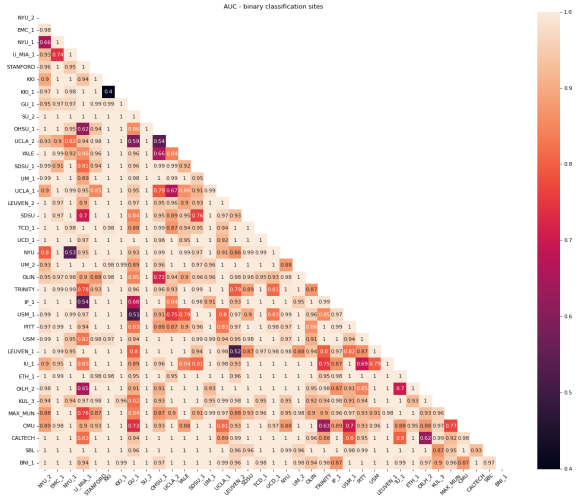
To have a visual representation of how much each feature is modified in respect of its sourced site, in fig 21 are shown two features among 5995: *feature 324* and *feature 2800* as a function of sites. For a clearer idea of what is shown, feature 324 represents the correlation between right and left 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, 22

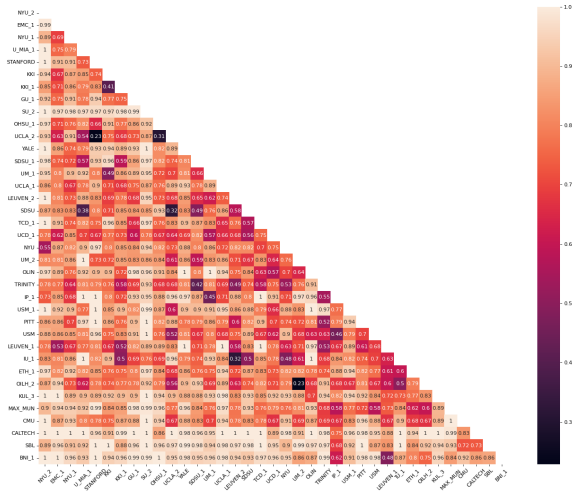
7 Domain-adversarial NN

An other approch is trying to train a neural network to recognise discriminative features, making train as much site-independent as possible. The genral 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

$$\min_{W,b,V,c} \left[\frac{1}{n} \sum_{1}^{n} L_y^i(Gy(G_f(x_i)), y_i | W, b, V, c) \right] \tag{24}$$

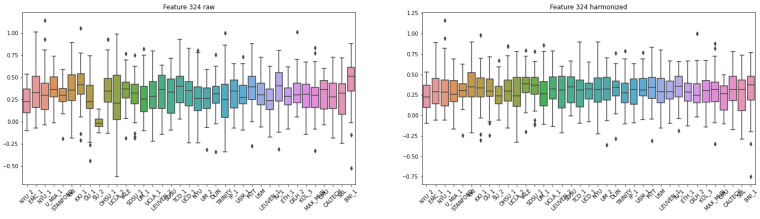


(a)

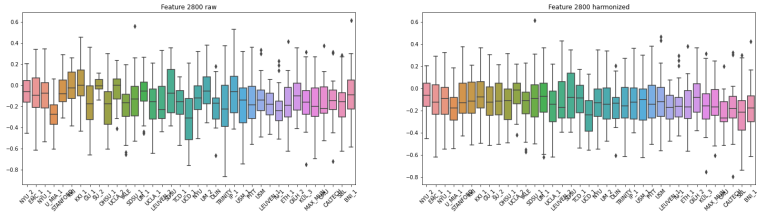


(b)

Figure 20: Binary classification

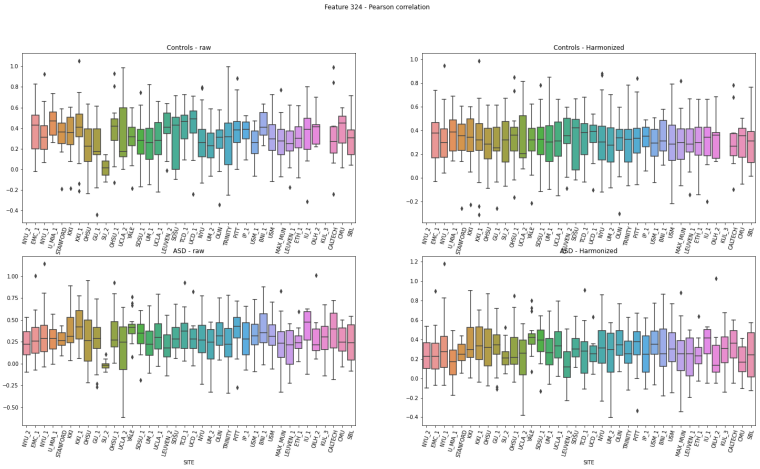


(a)

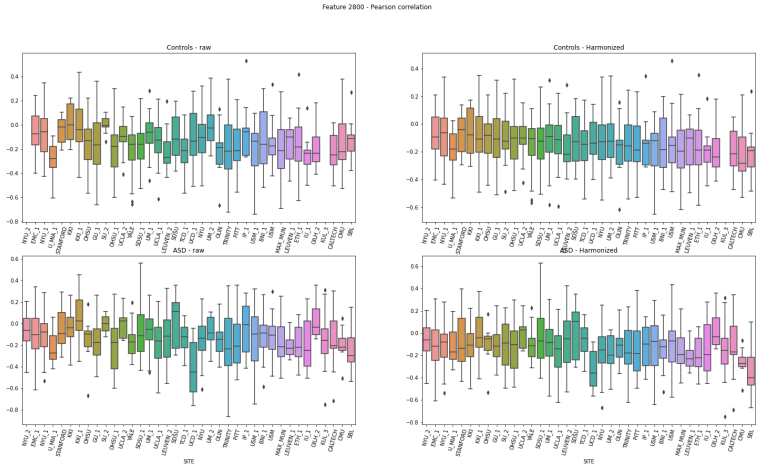


(b)

Figure 21: Features



(a)



(b)

Figure 22: Features

. To tackle the domain independence problem, an other term is added to the summation: a term coming from a domain classifier layer: $G_d : R^D \rightarrow [0, 1]$ 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

$$E(W, b, V, c, u, z) = \frac{1}{n} \sum_{i=1}^{n} \tag{25}$$

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

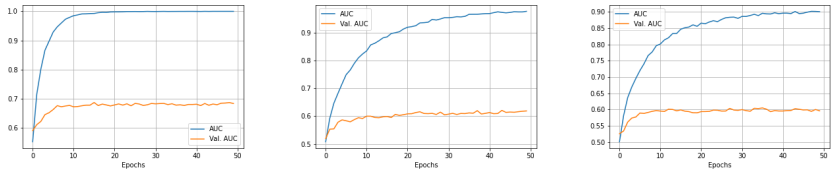
$$(\hat{\theta}_f, \hat{\theta}_y) = \underset{\theta_f, \theta_y}{argmin} E(\theta_f, \theta_y, \hat{\theta}_d) \qquad \hat{\theta}_d = \underset{\theta_d}{argmax} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d) \tag{26}$$

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.

8 Deep learning: implementation and results

In our work we employed a simple deep neural network architecture illustrated in figure .. with 2 hidden made up of 64, and 32 nodes respectively activated by a ReLU function and each layer separated from the next by a Batch-Normalization and a 0.3-Dropout layer. After the 32-node layer we used a single output layer with a single node activated by a sigmoid function, for classification. Since we are working in an underlying overfitting condition we choose this network structure after a grid search to find the minimal parameters' configuration which allows best classification scores.



(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 23

Overfitting is clear if we take a look at the learning curves in figure 23 which shows the training and validation AUC curves for two models: one of them (23c) even if too much simplified, shows the classical trend of an overfitting state, however, it performes a bit worse than the other two more complex models.

In our model, as loss function we used binary crossentropy and the neural network's weights were optimized via Adam optimizer with a learning rate of 10^{-4} and model's performancies were assessed with a 10 or 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: One way to implement a model able to predict category label without any influence of site's information is to train a model with two different losses as is done in the article **??** and combine them in order to create a loss in the form

$$L = L_1 - \lambda L_2 \tag{27}$$

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.

We also tried to find something with the same underlying idea, namely minimize the error on classification, while maximizing but with a slightly different implementation

Following a similar structure as the DNN, it can be divided into three main components: the first feature extraction branch comprising the DNN up to the 32-node layer; from this point the network is splitted into two branches: one categorical classificator, along the lines of the DNN consists of the last outpput layer with a sigmoiod activation function, and the other branch, 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 multi-output layer with N nodes, being N the

total number of sites input data belong to. This final layer is activated by a softmax function, and the loss function employed for site classification 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's sitelabel correctly, which would have low impact on the whole branch classifier making the adversarial strategy useless.

We mainly carried out three different kind of analysis using this DNNs, and compared them with the scores obtained from the adversarial network. Specifically we performed: 1) classification of raw (not harmonized) data, 2) classification on the whole harmonized dataset, creating the harmonization model on all control subjects and applying the model on all ASD subjects, and 3) classification with harmonization implemented inside the K-fold cross-validation, creating the model on controls and applying it to ASD, and 4) classification using the domain adversarial neural network.

For each one of them we implemented a K-fold cross validation scheme and reported our results as the mean AUC and the standard deviation of the scores of each k-fold.

Following this procedure we implement a k-fold cross validation using stratifiedKFold provided by sklearn library. 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 24

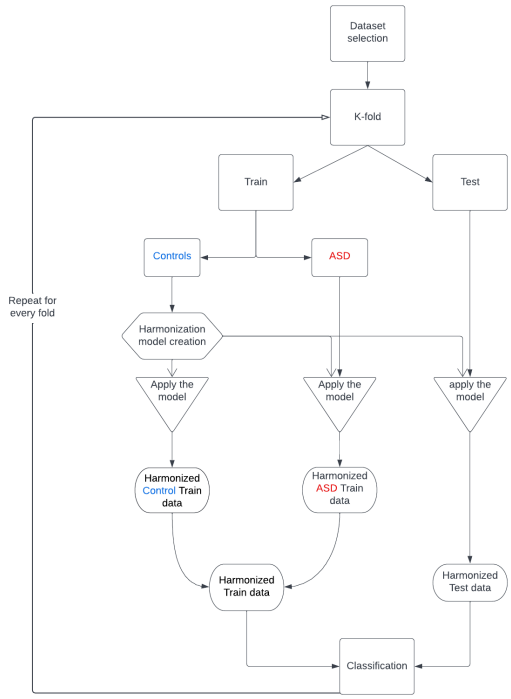


Figure 24: Flowchart: implementation of harmonization inside each fold

This is the best way to mantain train and test data as much separated as possible and avoid data leakage between train and test.

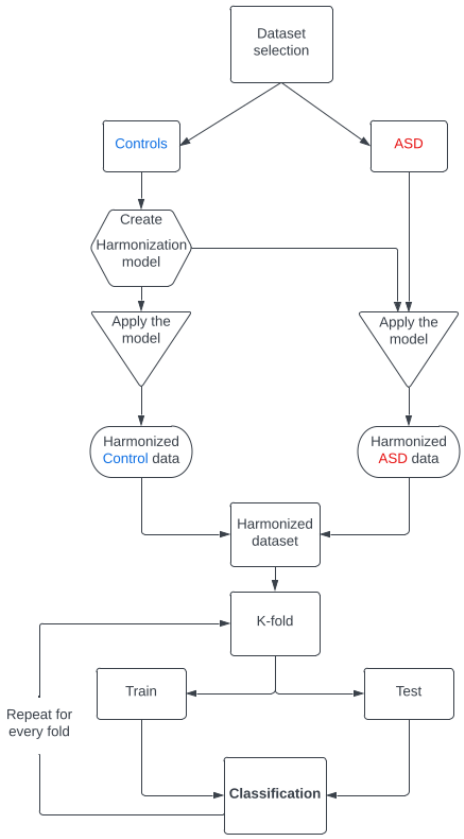


Figure 25: Flowchart: implementation of harmonization upstream

Analysis were carried out using different selection criterias and thresholds which brought us to different datasets combination:

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.

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 26

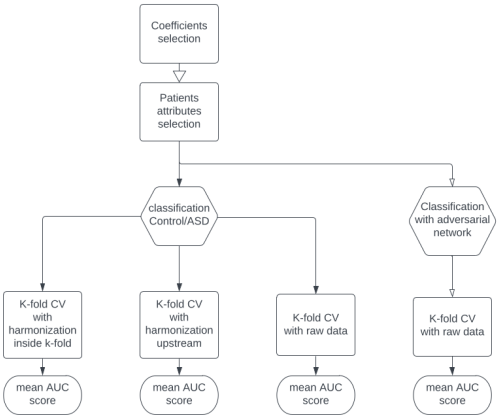


Figure 26: Flowchart of the analysis carried out in this work:

1. Select one type of coefficients: Pearson correlation, Wavelet coherence coefficients
2. Select the attributes of patients 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 8 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.

8.1 Results with pearson coefficients

Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I+II	70 ± 2	74 ± 2	70 ± 1	68 ± 1
AB I	70± 3	71 ±3	73± 5	69 ± 3
AB 2	63 ±5	68± 6	64± 4	66 ± 4
AB I + II - EYE open	70 ±4	71± 4	70 ±3	70 ± 2
AB I open EYE	73± 4	76± 6	75 ±3	73 ± 5
AB II EYE open	73 ±4	71± 4	71±5	67 ± 4
AB I + II AGE=all, EYE=all	70± 1	75 ±1	70 ±2	65 ± 1

Table 1: AUC score obtained from three different kind of analysis with pearson coefficients

8.2 Results with Wavelet coefficients

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

Three main analysis were carried out

1. In phase wavelet coefficients
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

Table 2: Win + Wout

Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I+II	65 2	71 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	63 4	72 6	67 3	67 4
AB II EYE open	66 2	69 3	67 2	68 2
AB I + II AGE=all, EYE=all	63 2	70 2	65 2	63 3

Table 3: Only Win

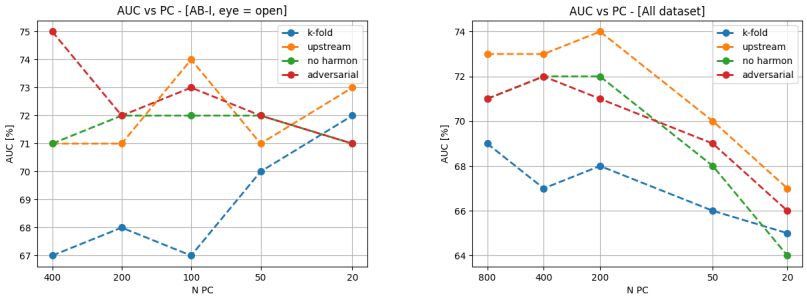
Dataset	Harmon K-fold	Harmon. upstream	No harmon.	Adv 264-8
AB I+II	65 2	71 2	62 2	64 1
AB I	66 3	73 3	65 3	66 2
AB 2	62 3	65 3	61 4	60 2
AB I + II - EYE open	64 4	70 3	66 3	65 3
AB I open EYE	64 5	68 5	64 3	64 4
AB II EYE open	66 2	68 3	66 4	65 5
AB I + II AGE=all, EYE=all	66 3	72 3	66 3	64 3

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				
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

Table 4: Win - Wout

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

Table 5: Explained variance [%] for different numbers of principal components, and for the two dataset used in the analysis



(a) (b)

Figure 27

8.3 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 5

9 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 28

SHAP (SHapley Additive exPlanation) is a technique based on game theory wich provides a method for model explanability. It's 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 29 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.

9.1 Shapley values - Theory

Game theory is a branch of mathematic related on the study of mathematical models to conceive social situations among competitive players [7] 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 [**?**]. 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

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

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 $\nu(S)$ = 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

$$\phi_i(\nu) = \frac{1}{N} \sum_{S \subseteq N \setminus \{i\}} \binom{N-1}{|S|}^{-1} (\nu(S \cup \{i\}0) - \nu(S)) \tag{28}$$

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

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 feature in our input data. For a single feature, its Shapley value is defined as the average marginal contribution of that feature's value 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.

To do so and simplify the workflow, they introduce a coalition vector x' = $\{0,1\}^F$: where F is the total number of features: z' is a vector of the same length of x, and is a binar vector made by zeros and ones where zero means that we withheld the corrisponding variable from x in our analysys, 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 it's 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

$$g(z') = \phi_0 + \sum_{i=0}^{N} \phi_i z'_i \tag{29}$$

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

1. Local accuracy:
$$g(x') = \phi_0 + \sum_{i} \phi_i x'_i = f(x) \tag{30}$$
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:
$$x'_i = 0 \Longrightarrow \phi_i = 0 \tag{31}$$

3. Consistency: if we have two different functions g' and g, indicating eith z'/i the setting where $z'_i = 0$, if
$$g'(z') - g'(z'/i) \geq g(z') - g(z'/i) \Longrightarrow \phi_i(f', x) \geq \phi_i(f, x) \tag{32}$$

In 1975 was demostrated that the only coefficients satisfying all of these properties are Shapley values. For this reason employing them into some pre-existing algorithms enhanced them and made them more reliable.

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, and different algorithms implemented in SHAP proposed a solution for this problem, depending on the model we are working on: example are Tree SHAP, a fast and exact algorithm to computeSHAP values for treed 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.

The problem with implementing an algorithm for the computation of shapley values is that we should evaluate the model for each possible feature permutation so that for a model with p features we should repeat this process p! times which in our case would bring to 5995! $\sim 10^{20046}$ in our case. To avoid this calculus Shapley employ by defining a background dataset with a subset of Train data we used to train the model, then during shapleyt computation it just fill in the missing features with features from this background. We then take the average of all the model outputs from this new artificial dataset

The idea behind algorithms like KernelSHAP comes from LIME algorithm [**?**], that avoid the computation of all the permutations by choosing

a background subset from input data and evaluate the permutation only between this subset. So if we want to evaluate k data samples, (each one containing p features), and we choose a background dataset of size n, the algorithm will perform evaluations only n*k times following this aproach:

1. Input a sample x, a model f, and a background data set x1,x2,...,xn . We want to interpret x.

2. Replace a subset of entries in x, by entries in xi, where xi is a data from x1,x2,...,xN. This way we create a synthetic or perturbating dataset from our original instance x

3. Evaluate the function f on the perturbed dataset.

4. Once we computed these n scores, each using a different x_i from the background dataset, we fit them to compute the feature importance values for our instance x.

DeepSHAP alghorithm works with them same principles of KernelSHAP, making use of a background dataset, 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 used in this algorithm.

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 []

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 backpropagation: 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 of a feature we plotted them using SHAP feature importance, based on the relation between feature importance and shap value: feature with high absolute shapley value are important. The importance value for a feature j is then calculated from $I_j = \frac{1}{n}\sum_{j=1}^{n} \|\phi_j^{(i)}\|$

Shap is implemented as a free Python package with MIT license, developed and mantained by Scott Lundberg. 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.

9.2 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.

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 of the same size of the test dataset we used, shaped $n_samples \times p_features$ We then have to input the test dataset

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 29

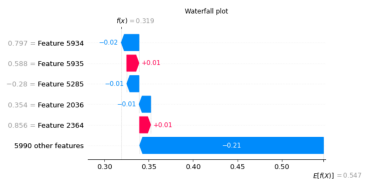
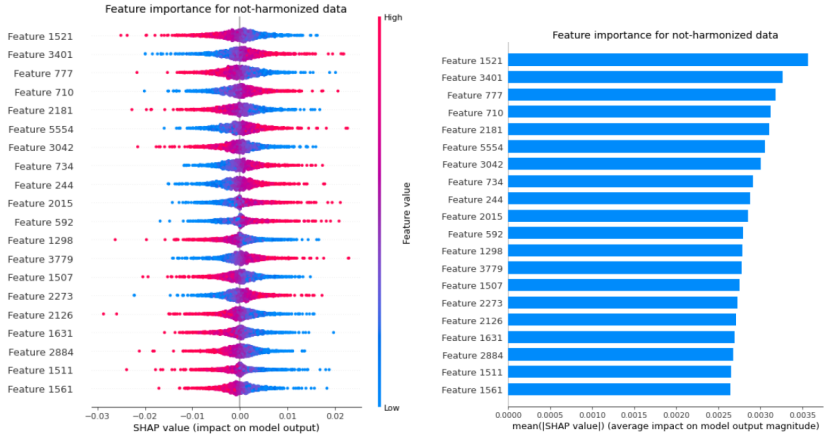


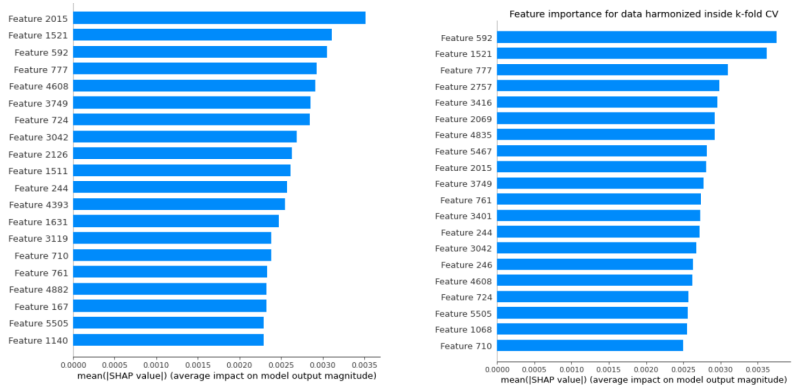
Figure 29

It is also possible to wisualize a bar plot of all the feature from the most to the least important. 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 6-40. Our goal is to check if before and after the



(a) (b)

Figure 30: Not harmonized



(a) upstream (b) kfold

Figure 31

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 **??** but this kind of plot is not the best fsuitable or readibility in our case because of the large amount of features and the small contribution of each feature to the final outcome. 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.

One feature seems to be persistend in all the three pipelines: feature 1521 representing the correlation between right middle temporal gyrus and left angular gyrus.

This bar plot is shown in figure **??**

A Principles of MRI imaging

Nuclear magnetic resonance imaging, is a widespread imaging technique It's widely employed in radiology to obtain anatomical images or study physiological processes, taking advantage of its flexible sensitivity to different tissues It's 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

$$\overrightarrow{\tau} = \overrightarrow{\mu} \times \overrightarrow{B} \tag{33}$$

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

$$\omega_0 = \gamma B_0 \tag{34}$$

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:

$$M_z(t) = M_0(1 - e^{-t/T_1}) \tag{35}$$

. 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 it's 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.

$$M_{x,y}(t) = M_{x,y}(0) \cdot e^{-t/T_2} \tag{36}$$

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 34 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 flox 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 $\phi_G(z,t)$

$$s(t) = \int_0^t dz\rho \ (9.14) \ pag \ 145 \tag{37}$$

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 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 32 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.

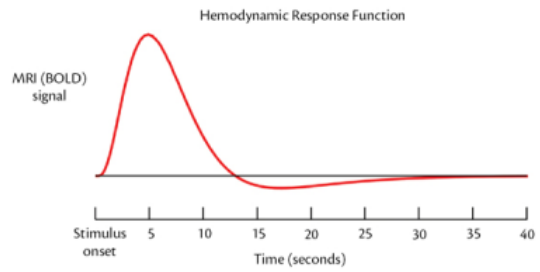


Figure 32

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 deoxyhemoglobin, 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.

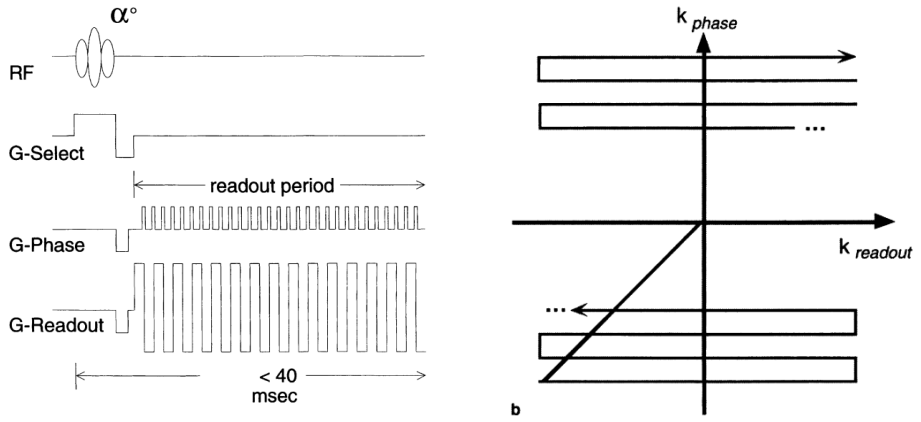
A.3.1 Gradient Echo sequences

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.3.2 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 33a 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 it's 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



(a) (b)

Figure 33

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

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 6: Classification scores using only out-of-phase Wavelet coefficients

References