POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Computational neuroscience between machine learning and topology

Supervisors

Prof. Francesco VACCARINO

Prof. Robert LEECH

Dr. Marco GUERRA

Dr. František VÁŠA

Candidate

Marina D'AMATO

Academic Year 2020-2021

Acknowledgements

I would like to express my gratitude to Professor Francesco Vaccarino for giving me the opportunity to work on this project and for having guided me during the development of the thesis, providing me with valuable advice and stimuli to overcome difficult moments. I thank Marco for being a mentor, a point of reference, for the time he dedicated to me, for the constant presence, the availability, the kindness, the words of comfort, because his help and his teachings have enriched me and have allowed to give a new shape to this project.

A heartfelt thanks to King's College who virtually welcomed me, to Professor Robert Leech who allowed me to work on a project so close to my interests and allowed me to explore an extremely fascinating field, giving me the freedom to follow the direction I felt more appropriate, guaranteeing me full confidence. To Frantisek, who has always believed in me and recognized my work, who managed to overcome the difficulties of distance by offering me availability and professionalism, who motivated and supported me, offered me advice from a professional and human point of view.

I thank my family for their constant support and presence, for making the kilometers that separated us imperceptible. I thank my mom, for making these years in Turin possible, for giving me the freedom to choose, grow, make mistakes and improve, unconditionally believing in me and in my abilities. Thank you for sharing my anxieties and taking charge of my fears, lightening the burden I carried with me. I thank her for always being there, when I called her too much and too little, for the desperation that she has always preceded and followed every exam. Without her all this would not have been possible and I will always be grateful to her for giving me the freedom to make any choice, even the ones that took me away from home. To my brother Vittorio, who has always been there taking care of me in many different ways. To Tino and Virag, who have always given me good advices for the future and believed in me.

Thanks to Caterina and Francesca, my friends of a lifetime for always being

a certainty and a fixed point, in the present and in the future. Thank you for the words, for the advice, for always walking with me, for the sincerity, the frankness, for being life companions, because every happiness with them is amplified and every difficulty is less scary.

Thanks to Chiara and Walter that despite of the time and the kilometers deeply know and understand every part of me, for the silent certainty of always finding them by my side and never losing them, for being so special and unique and close to my heart.

Thanks to Giulio, because he has made Turin my home and my place of the heart, for his complicity, for having lightened these years by sharing this path with me, for having been a breath of lightheartedness, for having listened to me, understood and supported me and for giving me some of the most beautiful and unforgettable moments, from the apartment in corso Marche to the house in via Fabbriche. To Irene, for taking care of us, for always replying instantly to every message forgiving my biblical response times, for always finding the words to cheer me up. To Luca, with whom I feel connected in many different ways - and that should be enough to make him imagine a cinematic scene of this moment - for the laughs, for the playlists, for the unspoken words and for the endless days and nights in the study room.

Thanks to my two buddies of the heart, who have been the most beautiful and precious gift of the last university years. To Adele, for her purity and simplicity, for giving me the opportunity to enter her world, for having become so important in a very short time, for being an inspiration, a point of reference, for being so unconsciously fantastic. To Andrea, who will persist in calling me a colleague forever but entered my heart from the first (unwanted) hug at the Off Topic, for always encouraging me to improve myself, for giving me points of views different from mine, for making me reflect, for allowing me to open up.

Finally, I would like to express my love and gratitude to Costantino, for having given me serenity and lightness, for having illuminated my days, even the darkest ones, for keeping me in balance, for helping me to believe in myself, for the sincere advice and honesty, to always give me the strength to face any difficulty and the freedom to make any choice while always remaining by my side. Thank you for never leaving me alone, for always finding the right words, for being such a good and immense person. Thank you because, with you by my side, I am no longer afraid.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The field of computational neuroscience and neuroimaging is showing a great interest in the application of statistical and mathematical techniques to represent and study complex brain structures. Neural data is complicated and, as the field of brain connectomics has developed, new techniques to represent and analyze the human connectome to obtain a description of the brain's structural and functional connections emerged.

There exist different imaging techniques to acquire measurements of brain structure and activity, such as electroencephalography, magnetoencephalography, calcium imaging or functional magnetic resonance imaging. One of the main challenges in neuroscience consists in understanding the global brain organization and the correlation that exists among different brain measurements. Network theory and analysis is often used to address these kinds of problems, as the functional or structural connectivity between each pair of brain regions can be expressed, in matrix form, in terms of the correlation between the time series data or the morphometric feature vectors of the two regions in question. Connectivity matrices can be investigated to analyze network-level properties of the brain and can reveal biomarkers of subjects' clinical traits.

In this work, we use structural data to construct *morphometric similarity matrices* based on inter-regional similarity of multiple morphometric features measured using multimodal MRI. Capturing the correlation structure associated to different brain regions in matrix form pave the way for many studies in terms of types of correlations studied and kind of analysis performed: all these choices can considerably affect the pipelines used to observe these data and the obtained results.

Here, we first investigate strengths and shortcomings of different measures of correlation used to construct the morphometric similarity matrices and then we inspect and compare two different techniques to analyze them.

In Chapter 2 we present the morphometric similarity matrices formulation and different ways to measure the similarity coefficients, including the tangent space parametrization that is able to deal properly with the geometrical properties of positive definite matrices such as the correlation ones. Then, we propose different dimensionality reduction techniques, both data-driven and hypothesis-driven.

After the construction of these different kinds of morphometric similarity matrices, we investigate the application of predictive models on one side and of topological data analysis on the other side. The former refers to the use of machine learning classification and regression models to predict a discrete (diagnosis) and a continuous (age) label respectively.

In Chapter 3 we explain the mathematical background behind the main Machine Learning classification and regression algorithms that are used as a first step of our analysis. In Chapter 4 we present Topological Data Analysis and persistent homology. Topological Data Analysis is a branch of applied mathematics that has been recently spread in neuroscience for its ability to infer robust features and extract meaningful information from complex datasets. The goal of these techniques is to study the shape of topological spaces and the properties of these spaces that remain invariant during time. Topological features such as cavities and holes can have interesting meanings in neuroscience: for instance, if we consider structural data a hole could indicate axonal dropout, while for functional data cavities can give insights on regions that display correlated activities.

Finally, in Chapter 5 we show the results that we obtained on two different datasets of structural connectivity data.

Chapter 2

Morphometric Similarity Networks

The brain is a complex interconnected network made up of billions of neuronal elements. One of the most important neuroscientific problems consists in understanding the pattern of links among the units of a nervous system in terms of connectivity.

Brain connectivity can be distinguished into *structural connectivity* and *functional connectivity*: structural or anatomical connectivity refers to the set of physical connections between neuronal elements; on the other hand, functional connectivity denotes a statistical concept that captures the dependency between functional signals from different brain regions.

In this work we refer to a technique for cortical network mapping that consists in the creation and analysis of the so-called *morphometric similarity networks*.

2.1 Theory background

Deriving structural connectomes from human neuroimaging is a challenging problem which still does not have a unique solution. Some standard approaches used for this purpose are: tractography from diffusion-weighted imaging (DWI) and structural covariance networks.

Diffusion-weighted imaging Diffusion-weighted imaging is a type of MR imaging that aims at reconstructing the trajectory of axonal tracts from the principal directions of the diffusion of water molecules. This technique is useful to recognize localized patterns of anatomical connectivity. The drawbacks consist in the fact that:

- it is difficult to use it to model connectivity among all brain regions because long-distance projections are systematically under-recovered
- statistical analysis is compromised by head movement
- there is a large number of false-positive connections

Structural covariance networks Structural covariance networks (SCN), proposed for the first time in [1], are networks in which nodes represent brain regions and edges represent morphological correlations between them. These networks are constructed from inter-regional correlations computed from a set of individual images and are based on the phenomenon of *structural covariance* between brain regions.

It is known that the structure of cortical regions vary deeply among different individuals. Although these differences could depend on factors that affect each person and each region independently, a structural covariance among different brain regions has been recognized. This means that differences among subjects in the structure of a certain cortical region often covary with differences in other cortical regions.

To construct structural covariance networks, a single morphometric feature is considered at a time - such as cortical thickness. Measures of this feature at each region in multiple images are collected and then the covariance between regional estimates of the feature, for each pair of regions, is computed.

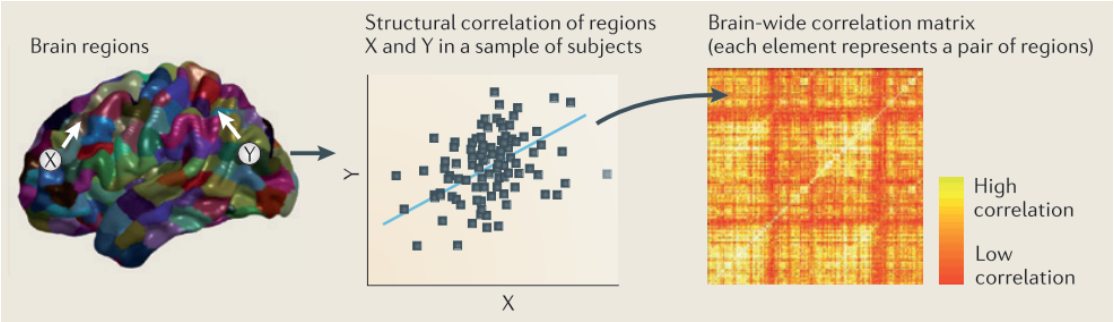


Figure 2.1: Measuring structural covariance among brain regions [1]

Morphometric similarity networks Morphometric similarity networks (MSN), first introduced in [18] and then used in [14] and [19], are conceptually similar to structural covariance networks, but they introduce a powerful and novelty strategy: on one hand we have SCN that estimate the correlation among brain regions of a single macro-structural morphometric feature measured in multiple images; on the other hand, MSN estimate the correlation among brain regions of *multiple* macro- and micro-structural morphometric features in a single individual.

We now explain the pipeline followed to construct these networks.

1. First, different morphometric features are extracted from MRI and DWI data for each subject. Then, the data are mapped to the same cortical parcellation template, which in this case is constituted by 308 contiguous regions of approximately equal area according to the Desikan-Killiany atlas.

   In this case, we can see the table of 10 morphometric features computed for each region: fractional anisotropy (FA), mean diffusivity (MD), magnetization transfer (MT), gray matter volume (GM), surface area (SA), cortical thickness (CT), intrinsic curvature (IC), mean curvature (MC), curved index (CI), folding index (FI).

   This means that we obtain a matrix $nF \times nR$ for each subject, where nF represents the number of morphometric features and nR the number of brain regions.

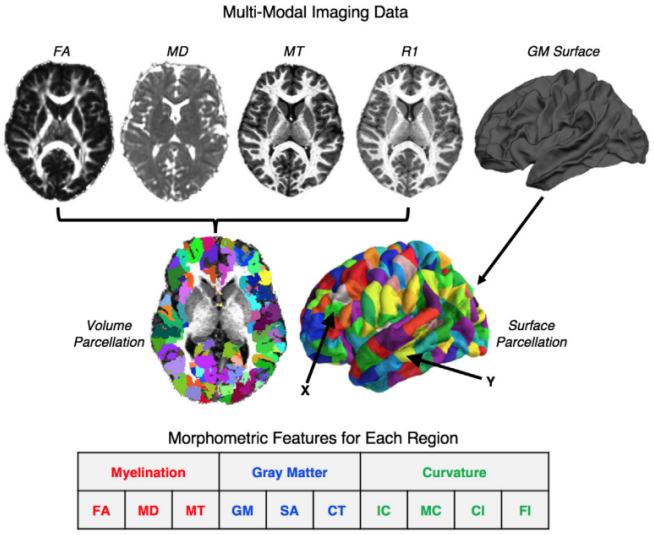


Figure 2.2: For each subject, computation of many different MRI parameters that are mapped to the same atlas [18]

2. The next step is based on the construction of the *morphometric similarity matrices*. We compute the morphometric similarity between each possible pair of regions by considering the *correlation* between the morphometric feature vectors associated to each region. In this case, in the end we obtain a 308 × 308 matrix, for each subject.

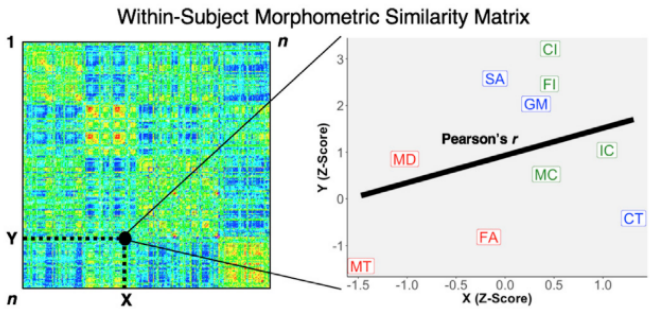


Figure 2.3: The morphometric similarity between each possible pair of regions is computed considering the correlation between their morphometric feature vectors [18]

3. To obtain the morphometric similarity networks, the matrices are thresholded to construct an adjacency matrix in which only the strongest connections survive. We can see in Figure 2.5 that we can visualize the results in matrix format, in anatomical space or in a topological representation where the closest nodes are the ones that are connected.

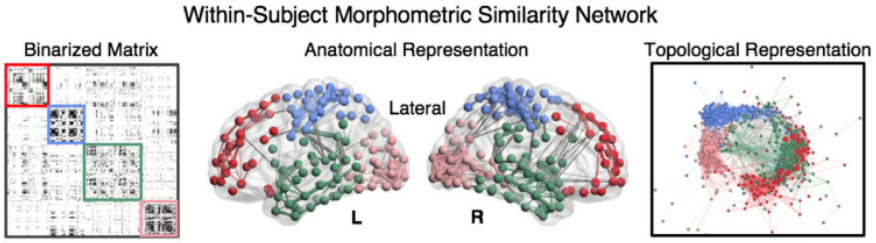


Figure 2.4: Thresholding of the morphometric similarity matrices to obtain morphometric similarity networks [18]

2.2 Types of morphometric similarity matrices

Computing the correlations structure associated with multiple morphometric measurements gives rise to a connectivity matrix that encodes the relationships among the features of different brain regions. To construct the morphometric similarity matrices different notions of *proximity* or *similarity* are considered and compared.

2.2.1 Normalization

The first step that needs to be performed before constructing morphometric similarity matrices is normalization whose aim is to put all the features on the same scale. We consider parametric and non-parametric methods of normalization.

Z-score

Z-score is a measure that indicates how many standard deviations away a data point is from the sample's mean. The process that leads from a raw score, which represents an observed data point, to a standard score, that is the z-score, is called *standardization*.

Standardization is a feature scaling technique that consists in centering the values around the mean with a unit standard deviation. As a result, the features have zero mean and standard deviation equal to 1.

The formula to convert a raw score x into a standard score is the following one:

$$z = \frac{x - \mu}{\sigma} \tag{2.1}$$

where μ and σ are the mean and the standard deviation of the population respectively.

Of course, when the population mean and standard deviation are unknown, we can use their estimates:

$$Z = \frac{x - \bar{x}}{S} \tag{2.2}$$

where $\bar{x}$ and S are the mean and standard deviation of the sample.

Median/MAD

A non-parametric alternative to the Z-score is the median/MAD normalization which substitutes the median to the mean and the median absolute deviation to the standard deviation.

The median is also a measure of central tendency but it is very insensitive to outliers. As discussed in [11], we can consider a measure of robustness which is the *breakdown point* that measures the proportion of incorrect observations (i.e., set to infinity) an estimator can handle before giving as a result a false value (i.e., infinity or null). For instance, given n observations and the sample mean, we can infer that the breakdown point of the mean is 0 since it takes just one arbitrarily large value to get an arbitrarily large result for the mean. On the other side, the median is the estimator with the highest breakdown point (0.5).

The same reasonings can be done for the median absolute deviation (MAD), which is also immune to the sample size. The median absolute deviation is defined

as the median of the absolute deviations from the data's median and is computed as follows:

$$MAD = median(|x - \tilde{x}|) \tag{2.3}$$

where $\tilde{x} = median(X)$.

We can use this non-parametric equivalent of the Z-score to normalize our data in the following way:

$$Z_{non-par} = \frac{x - Md(x)}{MAD(x)} \tag{2.4}$$

where $Md()$ corresponds to the median and $MAD()$ to the median absolute deviation.

2.2.2 Correlation

After the normalization of the data, the second step is to estimate the correlation for each pair of normalized morphometric feature vectors. Also in this case, we first consider two alternatives: a parametric and a non-parametric method of correlation.

Correlation measures how two or more variables are related to one another. The degree of correlation can be computed through several correlation coefficients that quantify the *strength* and the *direction* of the relationship. The values of this measure range from −1 to 1, where:

- a correlation coefficient of 1 means that there is a *positive correlation* between the two variables, so for every positive increase in one variable, there is a positive increase of a fixed proportion in the other;
- a correlation coefficient of −1 means that there is a *negative correlation* between the two variables, so for every positive increase in one variable, there is a negative decrease of a fixed proportion in the other;
- a correlation coefficient of 0 means that the two variables are not correlated at all.

Pearson's r

The most common correlation coefficient is Pearson's r that evaluates the linear relationship between two variables. It is computed by taking the ratio of the covariance of the two variables, normalized by the square root of their variances.

$$\rho_{X,Y} = corr(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{2.5}$$

where E is the expected value, cov is the covariance and μ_i and σ_i represent the mean and the standard deviation respectively.

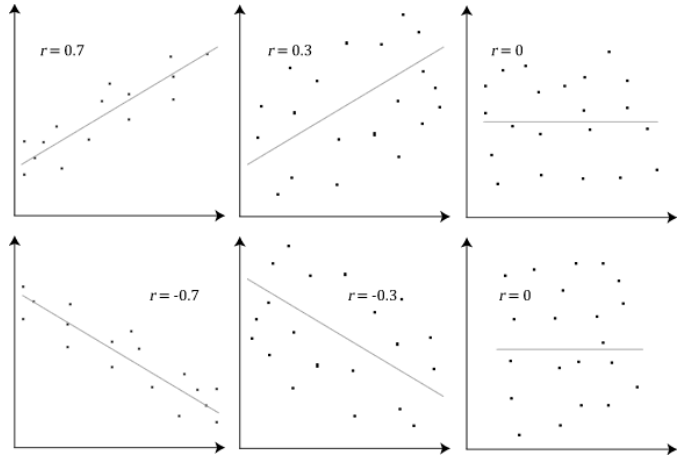


Figure 2.5: Pearson Correlation Coefficient and associated scatterplots

Definition In statistics, **ranking** is the transformation in which numerical or ordinal values are replaced by their rank when the data are sorted. This means that a number, called rank, is assigned to values in a list in ascending or descending order.

Spearman's ρ Spearman's rank correlation coefficient is a measure of rank correlation which is the statistical dependence between the rankings of two variables. On the one hand, Pearson's r assesses linear relationships; on the other hand, Spearman's ρ assesses monotonic relationships, i.e. relationships in which the variables change together but not necessarily at a constant rate. This means that if we have that when one variables increases, the other one does too but at a different rate, Spearman's coefficient equals 1, while Pearson's coefficient is positive but less than 1.

To compute Spearman's ρ, given a sample of size n, the n raw scores X_i, Y_i are first converted to ranks rg_{X_i}, rg_{Y_i} and then we have that the rank coefficient is equal to:

$$r_s = \rho_{rg_X, rg_Y} = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X} \sigma_{rg_Y}} \tag{2.6}$$

where ρ is Pearson's correlation coefficient applied to rank variables, cov is the covariance and σ_{rg_X} and σ_{rg_Y} are the standard deviations of the rank variables.

If all n ranks are distinct integers, we can use the alternative formula:

$$r_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{2.7}$$

where $d_i = rg(X_i) - rg(Y_i)$ is the difference between the ranks of each observation

and n is the number of observations.

Partial correlation

As an alternative to correlation, we can also consider *partial correlation*, which avoids indirect effects in the correlation structure.

Formally, the partial correlation between X and Y given a set of controlling variables $Z = \{Z_1, ..., Z_n\}$ is the correlation between the residuals (deviations of the dependent variable observations from the fitted regressor function) resulting from the linear regression of X with Z and of Y with Z.

In plain words, it measures the direct connectivity between two variables by estimating their correlation after regressing out the effects of all the other variables. In this way we avoid misleading results that, for instance, are obtained computing the correlation coefficient among two variables in presence of other variables, called *confounding*, that are related to the ones of interest.

Solving the linear regression problems explained before is not the only way to derive the partial correlation among variables. We can obtain all partial correlations between any two variables X_i and X_j of a set V of cardinality n by means of the inverse covariance matrix, which is also known as *precision matrix*.

Given the covariance matrix $\mathbf{\Sigma}$, we derive the precision matrix $\mathbf{\Omega} = \omega_{ij} = \mathbf{\Sigma}^{-1}$ and we compute the partial correlation as follows:

$$\rho_{X_i, X_j \cdot V \setminus \{X_i, X_j\}} = -\frac{\omega_{ij}}{\sqrt{\omega_{ii}\omega_{jj}}} \tag{2.8}$$

If all of the variables are normally distributed, we can infer that two variables are conditionally independent given the other ones if the partial correlation coefficient equals 0.

The drawback of this solution is that inverting matrices is not only computationally expensive but also an unstable process which requires for shrunk estimates. When the dimension p of the covariance matrix is bigger than the number of observations available n, then the matrix is not invertible; when the ratio p/n is less than one, the matrix is invertible but numerically ill-conditioned, increasing the estimation error.

The solution to this problem is a statistical technique called *shrinkage* that consists in "shrinking" the extreme values in a sample towards a central value.

Ledoit-Wolf shrinkage estimator Given a matrix $\boldsymbol{X}$ of dimensions $p \times n$ where n is the number of independent and identically distributed observations $\{\boldsymbol{x_i}\}_{i=1}^{n}$ which are $p-$dimensional random variables with zero mean and covariance

$\mathbf{\Sigma}$, the goal is to find an estimator $\hat{\Sigma}$ which minimizes the mean squared error:

$$E\left\{||\hat{\mathbf{\Sigma}} - \mathbf{\Sigma}||_F^2\right\} \tag{2.9}$$

where $||\cdot||_F$ is the Frobenius norm.

The classical estimator of $\mathbf{\Sigma}$ is the sample covariance matrix $\hat{\boldsymbol{S}}$ defined as:

$$\hat{\boldsymbol{S}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x_i}\boldsymbol{x_i}^T \tag{2.10}$$

but, as already seen, this estimator is usually ill-conditioned for large p.

A better conditioned estimate for the covariance matrix is:

$$\hat{\boldsymbol{F}} = \frac{Tr(\hat{\boldsymbol{S}})}{p}\boldsymbol{I} \tag{2.11}$$

where Tr stands for the trace of the matrix.

This estimate has a low variance but an increased bias. A trade-off between bias and variance is reached by shrinkage of $\hat{\boldsymbol{S}}$ towards $\hat{\boldsymbol{F}}$, which gives as a results the following class of estimators:

$$\hat{\mathbf{\Sigma}} = (1-\hat{\rho})\hat{\boldsymbol{S}} + \hat{\rho}\hat{\boldsymbol{F}} \tag{2.12}$$

where the matrix $\hat{\boldsymbol{F}}$ is called *shrinkage target* and the parameter $\hat{\rho}$, who ranges between 0 and 1, represents the *shrinkage coefficient*.

The goal is to find the shrinkage coefficient that minimizes the mean squared error (2.9). Ledoit-Wolf [10] approximates the optimal shrinkage coefficient solving the optimization problem and obtaining as a results:

$$\hat{\rho} = \frac{\beta^2}{\delta^2} \tag{2.13}$$

where $\beta^2 = E\left\{||\hat{\boldsymbol{S}} - \mathbf{\Sigma}||_F^2\right\}$ and $\delta^2 = E\left\{||\hat{\boldsymbol{S}} - \mu\boldsymbol{I}||_F^2\right\}$, $\mu =< \Sigma, I >$.

Tangent space

By construction, covariance matrices are symmetric and positive definite, which means that all their eigenvalues are positive. One issue is that their geometry is non-Euclidean, since they lie on a non-linear surface (*manifold*) called *positive semidefinite cone*. This means that we can't apply mathematical operations like subtractions if we want to preserve geometry and positive definiteness: for instance, the difference of two positive definite matrices does not correspond to the positive definite covariance matrix of a signal.

The solution that allows to apply mathematical operations on these matrices and to treat them correctly is based on the idea that we should follow the structure of the non-linear surface in which they lie. For this reason, using Riemannian geometry and the tangent space parametrization is a simple way to deal with positive definite matrices. This kind of parametrization has been already used in the neuroscience field to construct connectivity matrices, as seen in [23], [5], [16]. The theory background of this part is based on [6] and [15].

Using the Euclidean metric to measure the distance between two positive definite matrices means taking the upper triangle of the two matrices $\boldsymbol{\Sigma_i}$ and $\boldsymbol{\Sigma_j}$, vectorizing them and computing the Euclidean distance. If we consider Riemannian geometry, we need to define a new metric, called Riemannian distance.

The Riemannian distance between two positive definite matrices is the length of a geodesic curve which is the shortest differentiable path connecting these matrices represented as points on the manifold. The distance is computed as follows:

$$\delta(\boldsymbol{\Sigma_i}, \boldsymbol{\Sigma_j}) = ||log_m(\boldsymbol{\Sigma_i}^{-\frac{1}{2}} \boldsymbol{\Sigma_j} \boldsymbol{\Sigma_i}^{\frac{1}{2}})||_F \tag{2.14}$$

where log_m is the matrix logarithm.

We denote the space of all $n \times n$ symmetric positive definite matrices as $PD(n)$ and, for visualization purpose, we consider $PD(2)$. We take a matrix $\boldsymbol{A} \in PD(2)$ that is in the form:

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ b & c \end{bmatrix}, \quad ac - b^2 > 0, \quad a > 0 \tag{2.15}$$

We can consider the matrix as a point $(a, b, c) \in \mathbb{R}^3$ and we can visualize in Figure 2.6 the space with two labeled points p_0 and p_1.

The straight line is the geodesic in $\mathbb{R}^{n^2}$ and it does not remain contained within $PD(2)$. On the other hand, the geodesic γ computed when we consider the space as Riemannian, lies completely within $PD(2)$.

Given this, we can project the matrices on a tangent space which approximates the local structure of the manifold. Since this space is Euclidean, we can apply mathematical operations in this space preserving the geometry of the matrices. Moreover, many machine learning algorithms can not be directly used in a Riemannian manifold, but if we project the data in the tangent space we deal with Euclidean objects on which we can apply classification and regression algorithms.

In order to project the data, we need a reference point in the manifold, which is the point where the tangent plane touches the manifold. This point should be

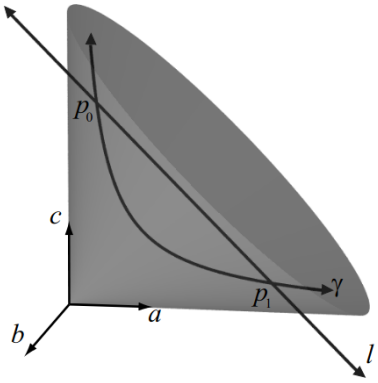


Figure 2.6: The space of $PD(2)$ with the geodesic and straight line between two points [6]

close to the projected matrices, but of course it is reasonable to think that each matrix could have a different reference point: in this case, the matrices would be projected to different tangent planes. For this reason, we need to find a symmetric positive definite matrix that stands for a group reference, in the sense that it is close to all the covariance matrices to guarantee their projections on the same plane. To compute the representative matrix of the group $\boldsymbol{\Sigma}_\star$ the Fréchet mean is used which defines the mean as the point that minimizes the expected value of the sum-of-squared distance function.

After computing $\boldsymbol{\Sigma}_\star$, the procedure to transform the matrix in the tangent-space parametrization consists in whitening the matrix. Given a covariance matrix $\boldsymbol{\Sigma_i}$,

1. Compute the whitened matrix $\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{i}} = {\boldsymbol{\Sigma}_\star}^{-\frac{1}{2}} \boldsymbol{\Sigma_i} {\boldsymbol{\Sigma}_\star}^{-\frac{1}{2}}$. If we apply the eigendecomposition on $\boldsymbol{\Sigma}_\star$ we can rewrite the matrix as follows:

$$\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{i}} = \boldsymbol{V} \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{V}^{\boldsymbol{T}} \boldsymbol{\Sigma_i} \boldsymbol{V} \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{V}^{\boldsymbol{T}} \tag{2.16}$$

   where $\boldsymbol{V}$ is the matrix that contains the eigenvectors of $\boldsymbol{\Sigma}_\star$ alongs its columns and $\boldsymbol{\Lambda}$ is a diagonal matrix that contains the corresponding eigenvalues.

2. Compute the matrix logarithm:

$$logm(\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{i}}) = \tilde{\boldsymbol{V}} log(\tilde{\boldsymbol{\Lambda}}_{\boldsymbol{i}}) \tilde{\boldsymbol{V}}^{\boldsymbol{T}} \tag{2.17}$$

   where $\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{i}} = \tilde{\boldsymbol{V}} \tilde{\boldsymbol{\Lambda}}_{\boldsymbol{i}} \tilde{\boldsymbol{V}}^{\boldsymbol{T}}$.

Also in this case, the Ledoit-Wolf shrinkage estimator is used at the beginning to compute the covariance matrices before projecting them into the tangent space. This ensures that we deal with well-conditioned matrices.

2.3 Dimensionality reduction

2.3.1 Principal Component Analysis

The main goal of PCA is to reduce the dimensionality of data while preserving as much variability as possible, that means preserving the information in the data. It is an unsupervised learning algorithm which applies a linear transformation to map m-dimensional input features into k-dimensional latent factors called *principal components*.

The key properties of the principal components are the following ones:

- they **maximize the variance** of the dataset under certain constraints
- they are orthogonal, so **uncorrelated**, with each other

Suppose we have a dataset $\boldsymbol{X}$ with n instances and p predictors. The first assumption is that each of the variables in $\boldsymbol{X}$ is centered to have mean zero. Given an instance of the dataset $\boldsymbol{x}_i$, we obtain the following linear combination of the sample feature values:

$$z_{i,1} = \phi_{1,1}x_{i,1} + ... + \phi_{p,1}x_{i,p} \tag{2.18}$$

In the equation above:

- the vector ϕ_1 is called *loading vector* and its components define the direction in the feature space along which the data vary the most;
- $z_{1,1}, ..., z_{n,1}$ are the *principal component scores* and they are obtained when we project the n data points $x_i, ...x_n$ in the direction defined by the loading vector.

The linear combination is normalized in such a way that the sum of squares of the loadings is equal to 1 in order not to have an arbitrarily large variance.

The first principal component is obtained solving the following optimization problem:

$$\max_{\phi_{1,1},...,\phi_{p,1}} \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p}\phi_{j,1}x_{i,j})^2 \tag{2.19}$$

$$s.t. \sum_{j=1}^{p}\phi_{j,1}^2 = 1 \tag{2.20}$$

The objective function represents the maximization of the sample variance of the principal component vector: in this way we preserve most of the information.

In fact, it has been obtained by replacing the expression of the linear combination in the equation of the sample variance which is

$$\frac{1}{n}\sum_{i=1}^{n} z_{i,1}^2 \tag{2.21}$$

The second principal component is obtained following a similar optimization problem since the goal is to maximize the variance among all the linear combinations that are uncorrelated with the previous one. The constraint of the second principal component to be uncorrelated with the first one means that the two components have to be orthogonal.

The same procedure is applied for the further components: each time we maximize the variance but we force the new component to be orthogonal to the previous ones.

The variance of the projected data can be rewritten in a new form to obtain a new formulation of the problem:

$$\frac{1}{n}\sum_{i=1}^{n} z_{i,1}^2 = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{z_1}^T \boldsymbol{x_i}\boldsymbol{x_i}^T \boldsymbol{z_1} = \boldsymbol{z_1}^T \boldsymbol{\Sigma_x} \boldsymbol{z_1} \tag{2.22}$$

$$s.t. \|\boldsymbol{z_1}\| = 1 \tag{2.23}$$

where $\boldsymbol{\Sigma}_x$ is the empirical covariance matrix.

In this way, we can see that the principal components are the eigenvectors of the Covariance Matrix that correspond to the largest n eigenvalues.

We can also express the PCA optimization problem as a problem of minimization of the *reconstruction error*. Given a compression matrix $\boldsymbol{W}$ that performs a mapping in a lower dimensional space and a reconstruction matrix $\boldsymbol{U}$ able to recover the original vector $\boldsymbol{x}$ from his compressed version $\tilde{\boldsymbol{x}} = \boldsymbol{U}\boldsymbol{y}$ where $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$, we obtain the following formulation:

$$\min_{\boldsymbol{W},\boldsymbol{U}} \sum_{i=1}^{n} \|\boldsymbol{x_i} - \boldsymbol{W}\boldsymbol{x_i}\|^2 \tag{2.24}$$

Proportion of variance explained The goal of PCA is to preserve as much variability as possible of the original data. To do so, we should choose a right number of principal components, in such a way that they preserve a proportion of variance higher than a certain threshold.

We can define the **total variance** explained as:

$$\sum_{j=1}^{p} Var(\boldsymbol{X_j}) = \sum_{j=1}^{p} \frac{1}{n}\sum_{i=1}^{n} x_{i,j}^2 \tag{2.25}$$

and the **variance explained by the m-th principal component** as:

$$Var(\boldsymbol{Z_m}) = \frac{1}{n}\sum_{i=1}^{n} z_{i,m}^2 \tag{2.26}$$

Therefore, the **proportion of variance explained** of the m-th principal component can be computed as follows:

$$PVE_m = \frac{\sum_{i=1}^{n} z_{i,m}^2}{\sum_{j=1}^{p}\sum_{i=1}^{n} x_{i,j}^2} \tag{2.27}$$

It is a quantity between 0 and 1 which gives us an idea of the informativeness of the new feature space.

2.3.2 Linear Discriminant Analysis

Linear Discriminant Analysis is a supervised algorithm that creates linear combinations of the original features by maximing the *separability* between classes. It is the extension of the *Fisher's discriminant analysis* on a situation of any number of classes, while the Fisher's LDA is applied when we work with two classes.

Suppose we have two classes and d-dimensional samples $\boldsymbol{x_1}, ..., \boldsymbol{x_n}$ where n_1 samples come from the first class and n_2 samples from the second class. The goal is to find a separating line on which to project the data points: the direction of this line is given by the unit vector $\boldsymbol{v}$ and the projection of a generic point $\boldsymbol{x}_i$ on the line is given by $\boldsymbol{v^T x_i}$.

To study separability between the two classes, we can define the means of the projection of each class as:

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{1}} = \frac{1}{n}\sum_{x_i \epsilon C_1}^{n_1} \boldsymbol{v^T x_i} = \boldsymbol{v^T \mu_1} \tag{2.28}$$

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{2}} = \boldsymbol{v^T \mu_2} \tag{2.29}$$

where $\boldsymbol{\mu_1}$ and $\boldsymbol{\mu_2}$ are the means of classes 1 and 2 respectively. In this way we can define the separation between the projections as $|\tilde{\boldsymbol{\mu}}_{\boldsymbol{1}} - \tilde{\boldsymbol{\mu}}_{\boldsymbol{2}}|$.

However, this measure is not yet useful for our purpose since it misses an important information: the variance of the classes. For this reason, we need a normalization factor proportional to the variance to measure also the spread of data around the mean. We define the *scatter* for all the data and for each class as:

$$\boldsymbol{s} = \sum_{i=1}^{n} (\boldsymbol{x_i} - \boldsymbol{\mu_x})^2 \tag{2.30}$$

$$\tilde{\boldsymbol{s}}_{\mathbf{1}}^{\mathbf{2}} = \sum_{\boldsymbol{x_i} \epsilon C_1}^{n_1} (\boldsymbol{v}^{\boldsymbol{T}} \boldsymbol{x_i} - \tilde{\boldsymbol{\mu}}_{\mathbf{1}})^2 \tag{2.31}$$

$$\tilde{\boldsymbol{s}}_{\mathbf{2}}^{\mathbf{2}} = \sum_{\boldsymbol{x_i} \epsilon C_2}^{n_2} (\boldsymbol{v}^{\boldsymbol{T}} \boldsymbol{x_i} - \tilde{\boldsymbol{\mu}}_{\mathbf{2}})^2 \tag{2.32}$$

Normalizing by both the two scatters we obtain that we want to find the projection line with direction defined by $\boldsymbol{v}$ which maximizes:

$$J(\boldsymbol{v}) = \frac{(\tilde{\boldsymbol{\mu}}_{\mathbf{1}} - \tilde{\boldsymbol{\mu}}_{\mathbf{2}})^2}{\tilde{\boldsymbol{s}}_{\mathbf{1}}^{\mathbf{2}} + \tilde{\boldsymbol{s}}_{\mathbf{2}}^{\mathbf{2}}} \tag{2.33}$$

Expressing the objective function in terms of $\boldsymbol{v}$, after calculus manipolation, we obtain:

$$J(\boldsymbol{v}) = \frac{\boldsymbol{v}^{\boldsymbol{T}} \boldsymbol{S}_{\boldsymbol{B}} \boldsymbol{v}}{\boldsymbol{v}^{\boldsymbol{T}} \boldsymbol{S}_{\boldsymbol{W}} \boldsymbol{v}} \tag{2.34}$$

where $\boldsymbol{S}_{\boldsymbol{W}} = \boldsymbol{S1} + \boldsymbol{S2}$ which are the *separate class scatter matrices* that measure the scatter of original samples before projection:

$$\boldsymbol{S_j} = \sum_{\boldsymbol{x_i} \epsilon C_j} (\boldsymbol{x_i} - \boldsymbol{\mu_1})(\boldsymbol{x_1} - \boldsymbol{\mu_1})^T \tag{2.35}$$

and $\boldsymbol{S}_{\boldsymbol{B}} = (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^T$ measures the separation between the means of the two classes before projection.

We can write the objective function as an eigenvalue problem and immediately find the solution which is represented by:

$$\boldsymbol{v} = \boldsymbol{S}_{\boldsymbol{W}}^{-1}(\boldsymbol{\mu_1} - \boldsymbol{\mu_2}) \tag{2.36}$$

Chapter 3

Machine learning for classification and regression

In this chapter, we will consider the theory that underlies machine learning methods for addressing both classification and regression tasks. We will later use these techniques to address problems in the computational neurology field. The background references for the next sections are [13] and [20].

3.1 Learning and generalization

The discipline of Machine Learning exploits several computational methods to reach the goal of improving performance at some task using experience. In this context, the *learner* is a computer program that is capable of learning from *experience*, that refers to the past data available to him. The data used for learning is given as a collection of *instances* or data points, each of them characterized by a set of qualitative or quantitative measurements called *features*.

Machine Learning techniques are used to solve several kinds of tasks of different nature, such as:

- **Classification**: the task consists in assigning a *categorical label* to each instance;
- **Regression**: it consists in predicting a *real value* for each item;
- **Clustering**: it consists in identifying similarities in the data to partition the items into homogeneous subsets;
- **Dimensionality reduction**: transform the data into a lower-dimensional representation which aims at preserving the main information and properties of the original one.

The goal of learning is to improve performance: to quantify the improvement, different metrics are used according to the specific task, as discussed in Section 3.2.

We can divide Machine Learning algorithms into three main areas according to the types of data available to the learner:

- **Supervised learning**: the learner is given a collection of tuples which consists of the instances and the associated *labels*, that are the categories or real values assigned to each input example. The goal is to learn the relationships between the instances and the labels to make predictions for the unseen points. This area is associated with both classification and regression problems;
- **Unsupervised learning**: the learner is given only the data without labels and makes predictions for the unseen points by learning some properties of the distribution. Typical problems are the ones associated with clustering and dimensionality reduction tasks.
- **Reinforcement learning**: the goal is to instruct an *agent* that interacts with an *environment* and, according to its actions, receives a reward or a penalty. The final object is to maximize the reward.

In the context of Machine Learning, the goal of learning goes hand in hand with that of *generalization*. This means that the learner should be able to understand the relationships existing in the input data to make accurate predictions also on instances never seen before.

We know that, for a given task, the learner has access to a set of data called *training data*: $S = ((\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_n}, y_n))$ which is a finite sequence of pairs in $X \times Y$, where X refers to the instance space and Y to the space of possible labels: for instance, for a classification task Y is a discrete set; for a regression task, $Y \subseteq \mathbb{R}$. Since the input points are represented by a set of measurements, we can imagine the input space to be a subspace of all $d-$dimensional real-valued vectors $\mathbb{R}^d$, where each $\boldsymbol{x} \in \mathbb{R}^d$ is a feature vector.

The learner's output is a target function called *prediction rule* or *hypothesis* $h : X \to Y$. The goal is to find a good approximation of this function among the *hypothesis space*, which is the space of candidate functions explored during the learning algorithm.

The first assumption that needs to be made is that the input instances are independently and identically distributed (i.i.d.) according to some unknown probability distribution D as well as the labels and that there exists a correct labeling function $f : X \to Y$ such that $y_i = f(x_i) \quad \forall i$, that is what the learner tries to achieve.

We define a *loss function* as a function that measures the difference between the predicted and the true labels, so it quantifies the goodness of the prediction of

a specific hypothesis.

Definition Given a function h, a loss function L and a joint probability distribution D over $X \times Y$, the **generalization error** or **risk** of h is defined as the probability that the classifier does not predict the correct label on a random data point or, in other terms, as the expectation of the loss function:

$$R(h) = \mathbb{P}_{x \sim D}[h(x) \neq f(x)] = \mathbb{E}[L(h(x), f(x))] \tag{3.1}$$

where the expectation is taken with respect to the distribution D.

The generalization error of a hypothesis is not directly accessible to the learner, since it does not know the distribution D and the correct labeling function f. The learner has only access to a specific subset of the input domain, which is the training data that can be used to compute an approximation of the risk.

Definition Given a hypothesis h, a labeling function f and a sample $S = (x_1, ..., x_n)$, the **empirical error** or **empirical risk** is defined as:

$$R_{emp}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), f(x_i)) \tag{3.2}$$

The learning paradigm that is based on the idea that the learning algorithm should choose a hypothesis which minimizes the empirical risk is called *Empirical Risk Minimization* (ERM).

The error of an ERM algorithm can be decomposed into two components: the first component is called *approximation error* or **bias** and depends on wrong assumptions and a bad prior knowledge in the learning algorithm; the second component is called *estimation error* or **variance** and depends on sensitivity to small fluctuations in the training set. The objective is to find a trade-off between choosing a more complex or less complex hypothesis.

In fact, we have that algorithms with high bias produce simpler models that can lead to the problem of *underfitting*: in this case, the model is not able to recognize important regularities in the data; on the other hand, high-variance models tend to be too complex and capture also noise in the training data: in this case, the model performs well on the training data, but can't generalize on unseen instances so it has a low empirical error but a high generalization error and this problem is called *overfitting*.

3.2 Training and performance evaluation

Hold-out
To reduce the problem of overfitting, a standard procedure is to divide the dataset into two disjoint partitions: the training set, used during the learning phase, and the test set, which is used after the training to get an estimate of the true generalization error. In this way, one can detect the point in which the generalization error diverges from the training error computed during the learning process. Usually the split is performed using 80% or 75% of data for training and the remaining part for testing. To perform the split in a better way, the data can be shuffled first and the split can be stratified to preserve the distribution of the data between the two sets. This technique is called **hold-out** but it still suffers from issues of high variance, since we don't know which partition of the data is chosen for testing and the results can be very sensitive to changes in the partition.

Cross validation
To reduce the variance of the estimator, a valid alternative is to use (Stratified) k–fold Cross Validation. It is a resampling method which consists in repeatedly drawing samples from a training set. It is based on the following procedure:

1. the training set is randomly divided into K parts called *folds* of the same size;
2. for each $k = 1, ..., K$, $k - 1$ folds are used to train the model which is then evaluated on the remaining k;
3. the operation is repeated for each k and each time the test subset (that in this case is called *validation set*) changes;
4. in the end, the results are averaged.

We can see a representation of the procedure in Figure 3.1. The Stratified K–fold, in addition, preserves the distribution of data for each fold. This technique is very effective and it reduces bias and variance since every data point gets to be in the validation set but also in the training set.

Hyperparameters tuning
Every Machine Learning algorithm is characterized by a certain number of *hyperparameters* which are parameters whose value is specified as input before the learning process begins. Changing these values deeply affect the performance of the models: for this reason, a hyperparameter tuning phase is needed to evaluate how the model behaves when different configurations of the hyperparameters are chosen.

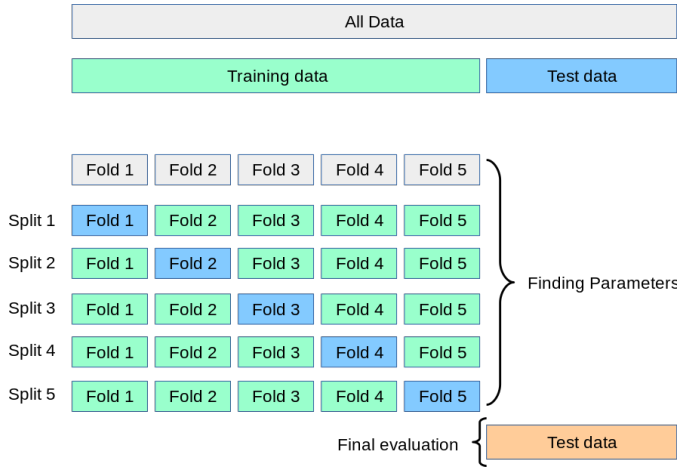


Figure 3.1: Cross-validation technique

Usually, to perform hyperparameter tuning we declare a *grid* and for each hyperparameter we specify a set of alternative values. In this way, we perform a *grid search* which is simply an exhaustive evaluation of all the possible combination of values for each hyperparameter.

Also in this case, the weakness of dividing the dataset only into two partitions emerges. In fact, the basic idea would be to instantiate a model with a given set of hyperparameters and evaluate it to the test data. But, in this way, the results would be biased because the hyperparameters would be tailored on that specific partition. For this reason, we can perform the grid search in combination with the $k-$fold cross validation, to see how the configuration affects the results for all the $k-$folds and obtain an average of the results.

Metrics

After all the preliminary phases described in the previous sections, we are ready to evaluate the goodness of our estimators and to find the one that best suits the analysis. The goodness of each model is evaluated through different metrics according to the task.

Before the explanation of the different metrics, we first give the following definitions related to the classification task in the binary case:

- True positive (**TP**): #elements that belong to the positive class and for which the prediction is right
- True negative (**TN**): #elements that belong to the negative class and for which the prediction is right

- False positive (**FP**): #elements that belong to the negative class but are assigned to the positive one
- False negative (**FN**): #elements that belong to the positive class but are assigned to the negative one

The metrics that we consider for classification are the following ones:

- **Accuracy**: it is the proportion of correct predictions over the total number of data classified.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

  When the dataset is imbalanced, accuracy scores may be misleading since we could have a high result but the predictions of the minority class could be mostly wrong. Also, there are some cases in which a mistake in the prediction has a different gravity according to the class, so other metrics should be preferred.

- **Recall**: proportion of actual positives that were identified correctly

$$recall = \frac{TP}{TP + FN} \tag{3.4}$$

- **Precision**: proportion of positive identifications that were actually correct

$$precision = \frac{TP}{TP + FP} \tag{3.5}$$

- **F1-Score**: harmonic average of the previous two

$$f1 = 2\frac{rp}{r + p} \tag{3.6}$$

For what concerns the regression task, we consider the following loss functions:

- **Mean squared error**: average squared difference between the estimated values and the actual value

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.7}$$

- **Mean absolute error**: it measures the average magnitude of errors in a set of predictions

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(|y_i - \hat{y}_i|) \tag{3.8}$$

  where n is the number of predictions

3.3 Manage dataset imbalancing

One important problem in Machine Learning happens when we deal with a dataset that is not balanced in terms of the cardinality of the classes. When the classes are not balanced, we inject a bias in the model since the algorithms tend to categorize new samples mostly into the majority class. Moreover, the accuracy results may be very high but the scores would me misleading since the prediction of instances belonging to the minority class would be probably low. Besides, models trained on unbalanced datasets often have poor results when they have to generalize.

There are different techniques to address this problem, such as:

- **undersampling**: randomly delete some of the observations from the majority class in order to match the cardinality of the minority class;
- **oversampling**: oversample the minority class to get the same numbers of observations as the majority class; one of the main algorithm of oversampling is SMOTE [4] that performs data augmentation by synthesizing new data from the existing ones.

When we deal with an imbalanced dataset, we want our test set to preserve the realness of data.

- Artificially inflating the minority class or reducing the majority one will lead to performance metrics that are unrealistic, bearing no real relation to the real world problem we are trying to solve.
- We would never want to artificially balance the test set; its class frequencies should be in-line with what one would see "in the wild".
- If we apply oversampling techniques it may happen that we incur in duplications in the test set of data belonging to the train set and this could lead to misleading results.

So both the techniques should be applied only on the train set, in such a way that while the model learns to classify data it works with a balanced dataset, but

then the test set still reflects the imbalancing situation.

Undersampling One of the most common and simplest strategies to handle imbalanced data is to *undersample* the majority class, as shown in Figure 3.2: it means that we randomly removes samples from the majority class in such a way that it becomes equal in size to the minority class. The drawback of this method is that the classifiers will have less data at disposal and we may lose useful instances and information.

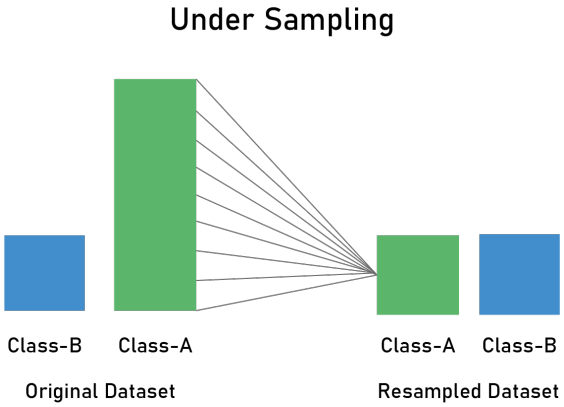


Figure 3.2: Undersampling technique

Oversampling In a similar way as before, random oversampling could be a solution to overcome the problem of imbalancing dataset. Random oversampling consists in duplicating the instances of the minority class in the training dataset, but we can understand that this implies that the model works with balanced data but it does not have any additional information.

An alternative to random oversampling consists in synthesizing new samples from the minority class through a data augmentation technique such as SMOTE. The SMOTE techniques consists in taking a sample from the minority class and its $k-$nearest neighbors. Then it takes the difference between the sample and its neighbors and multiplies it by a number between 0 and 1. Finally, it adds the result to the original sample. In this way it selects a random point along the line segment between two specific features. A visual representation in shown in Figure 3.3.

The formula behind the algorithm is the following one:

$$x_{new} = x_i + \lambda(x_j - x_i) \tag{3.9}$$

where $\lambda \in [0, 1]$.

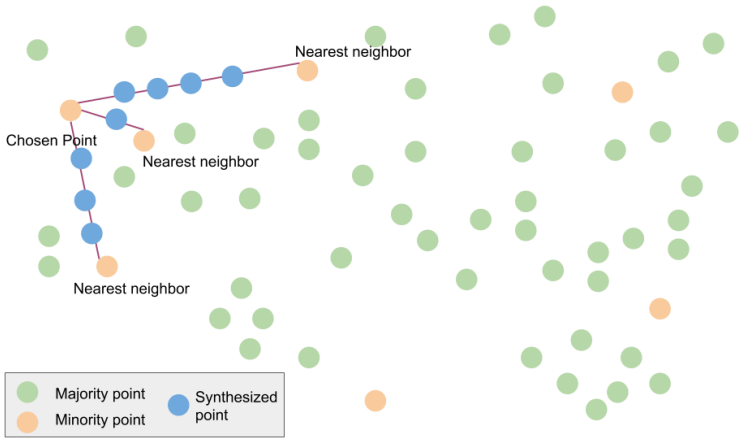


Figure 3.3: Oversampling technique with SMOTE

3.4 SVM

Support Vector Machine is a supervised algorithm that aims at finding the best hyperplane that separates the training data and maximizes the margin, which is the distance between the hyperplane and the closest points from any class called **support vectors**.

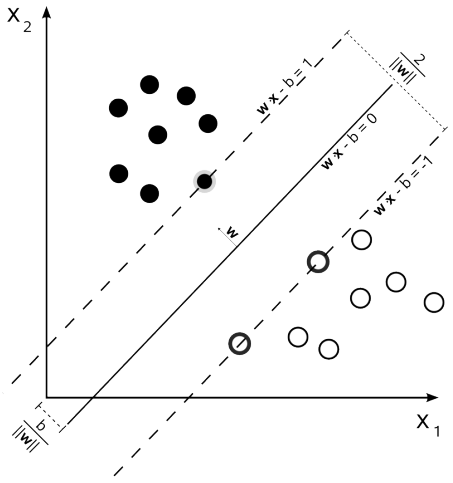


Figure 3.4: SVM algorithm

The hard margin problem

Suppose we have a set of n points $S = (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_n}, y_n)$, where each $\boldsymbol{x_i} \in \mathbb{R}^d$

and $y_i \in \{\pm 1\}$ and assume that the two classes are linearly separable. Consider an hyperplane defined by the equation

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0 \tag{3.10}$$

where $\boldsymbol{w}$ is the vector perpendicular to the hyperplane.

We can define the following inequalities for all the points that belongs to the positive (right of the hyperplane) and negative (left of the hyperplane) class respectively:

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \geq 1 \quad \forall y_i = +1 \tag{3.11}$$

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b \leq -1 \quad \forall y_i = -1 \tag{3.12}$$

Combining the two inequalities we obtain:

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i}) + b - 1 \geq 0 \tag{3.13}$$

As we can see in Figure 3.4, the hyperplane sits exactly in the middle of the two parallel hyperplanes that separate the two classes of data and the points that lie on these hyperplanes satisfy the following equations:

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b = 1 \tag{3.14}$$

for the support vectors of the positive class $\boldsymbol{x}_+$

$$\boldsymbol{w} \cdot \boldsymbol{x_i} + b = -1 \tag{3.15}$$

for the support vectors of the negative class $\boldsymbol{x}_-$.

We define the distance between the two hyperplanes, so twice the margin, as:

$$(\boldsymbol{x}_+ - \boldsymbol{x}_-)\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \frac{(1-b)-(-1-b)}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|} \tag{3.16}$$

In this way we can write the following optimization problem:

$$\max \frac{1}{\|\boldsymbol{w}\|} = \min \|\boldsymbol{w}\| \tag{3.17}$$

We can reformulate the equation as:

$$\min \frac{\|\boldsymbol{w}\|^2}{2} \tag{3.18}$$

$$\text{s.t.} \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x_i}) + b - 1 \geq 0 \tag{3.19}$$

where the constraint forces the hyperplane to correctly classify all the data points. Solving this problem through the Lagrange multiplier technique, we obtain a solution that is the weighted linear combination of only the points from the two classes that lie on the margin, so the support vectors. In this sense the classifier needs only remember this points, so it scales well.

The main drawback of this procedure is that in most real situations data are not linearly separable and it is very rare to get all the points correctly classified. For this reason, it is necessary to find a trade-off between the two objectives of the algorithm: **maximize the margin** and **minimize the misclassification rate**. This trade-off is carried out through the *Soft Margin formulation* of the algorithm that relaxes hard constraints and introduces the hyperparameter C to reach this goal.

The soft margin problem

The Soft-margin SVM is a relaxation of the Hard-SVM and it can be applied even if the data are not linearly separable. The idea behind this variation is that we allow the constraint of the previous optimization problem to be violated for some of the examples in the training set.

To do so, we introduce nonnegative slack variables $\xi_i, ..., \xi_n$ that represent the permission to make mistakes and measure how much the previous constraint is violated.

We can rewrite the optimization problem as:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \tag{3.20}$$

$$s.t. \forall i, y_i(<\boldsymbol{w}, \boldsymbol{x_i}> + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{3.21}$$

C is a regularization term that represents the cost of misclassification. When C is small, more mistakes are allowed and misclassifications are given less importance, so it corresponds to a larger margin; when C is large, the cost of misclassification is higher and the margin is smaller and the model depends more on the data leading to overfitting. Small values of C lead to high bias and low variance, while large values lead to low bias but high variance.

The slack variables ξ_i correspond to the Hinge loss:

$$l^{hinge}((\boldsymbol{w}, b), (\boldsymbol{x}, y)) = \max\{0, 1 - y(<\boldsymbol{w}, \boldsymbol{x}> + b)\}. \tag{3.22}$$

In this way we can reformulate the soft margin problem as follows:

$$\min \|\boldsymbol{w}\|^2 + CL_S^{hinge}(\boldsymbol{w}, b) \tag{3.23}$$

where:

$$L_S^{hinge}(w) = \frac{1}{m}\sum_{i=1}^{m}\max\{0, 1 - y_i<\boldsymbol{w}, \boldsymbol{x_i}>\} \tag{3.24}$$

is the average hinge loss over the training sample.

SVM with RBF kernel

SVM can be used in linear and non-linear classification tasks. The last situation is handled by means of the *kernel trick*. The idea behind this is that if data are non-linearly separable, they can be mapped in a higher dimensional space where there exists a hyperplane that linearly separates them. Kernel functions are very useful for this mapping since they allow us to not compute exactly the transformation of our data: in fact, we just need to know how to perform inner products in the feature space and we can do it by replacing them with kernel functions.

Given some domain set X, it is possible to choose a mapping $\phi : X \to \mathbb{R}^n$ and create the image sequence $\hat{S} = (\phi(\boldsymbol{x_1}), y_1), ..., (\phi(\boldsymbol{x_m}), y_m)$. However, computing linear separators over very high dimensional data may be computationally expensive. The solution to this problem is the kernel trick, which allows to use kernel to compute the inner products between the images of all pairs of data in the feature space (which is the only thing SVM needs).

The kernel function is defined as follows:

$$K(\boldsymbol{x}, \boldsymbol{x'}) = <\phi(\boldsymbol{x}), \phi(\boldsymbol{x'})> \tag{3.25}$$

According to the representer theorem, we can write $\boldsymbol{w}$ as:

$$\boldsymbol{w} = \sum_{j=1}^{m}\alpha_j y_i \phi(\boldsymbol{x_j}) \tag{3.26}$$

As a consequence:

$$\|\boldsymbol{w}\|^2 = <\sum_j \alpha_j \phi(\boldsymbol{x_j}), \sum_j \alpha_j \phi(\boldsymbol{x_j})> \tag{3.27}$$

$$= \sum_{i,j=1}^{n}\alpha_i\alpha_j < \phi(\boldsymbol{x_i}), \phi(\boldsymbol{x_j})> \tag{3.28}$$

Support vectors correspond to samples that have an alpha value that is greater than zero, so they are the only data useful. We can solve this problem by replacing the inner product with a kernel function.

A symmetric function $K : X \times X \to \mathbb{R}$, in order to be a kernel, must obey the *Mercer's theorem.* according to which the Gram matrix, which is G such that

$$G_{i,j} = K(\boldsymbol{x_i}, \boldsymbol{x_j}) \tag{3.29}$$

needs to be *positive semidefinite.*

There are different kinds of kernels and one of the most popular ones is the **Radial Basis Function (RBF kernel)**. It is defined as follows:

$$K(\boldsymbol{x}, \boldsymbol{x'}) = e^{-\gamma \|\boldsymbol{x} - \boldsymbol{x'}\|^2} \tag{3.30}$$

3.5 Decision Tree

The Decision Tree algorithm involves stratifying or segmenting the predictor space into a number of simple regions. It is a predictor, $h : X \to Y$, that chooses the label associated with an instance $\boldsymbol{x}$ by traveling from a root node of a tree to a leaf. At each node, the successor child is chosen on the basis of a splitting of the input space according to the best feature following a *greedy approach.*

The algorithm works as follows:

- The predictor space is divided into J distinct and non-overlapping regions $R1, ..., R_j$ which are usually represented as *boxes* ;
- For every observation that falls into the region R_j, the same prediction is made: for a classification task, it is the most commonly occuring class of the observations that lie in that region; for a regression task, it is the mean of the response values for the observations in R_j.

The tree-building process is:

- **top-down**: it starts from the root and recursively splits the predictor space;
- **greedy**: each node is split according to a local optimal decision without the certainty of obtaining a global optimum

Among all the possible splits, one chooses the one that maximizes the so called *gain* that quantifies the improvement due to that split according to different measures. The most important measures used to evaluate the goodness of a split for a classification problem are:

- **Gini index**: a measure of total variance across the K classes. Gini index is a measure of *purity* and a small value indicates that a node contains predominantly observations from a single class. The formula is:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{3.31}$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class.

- **Cross-entropy**: similar to the Gini index, but defined as follows:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk}) \tag{3.32}$$

For what concerns a regression problem, we aim at minimizing the mean squared error:

$$MSE = \frac{1}{N_m} \sum_{i \in D_m} (y^{(i)} - \hat{y}_m)^2 \tag{3.33}$$

where N_m is the number of samples in the $m-$th region, D_m is the corresponding subset in the $m-$th region, $y^{(i)}$ is the target true value and $\hat{y}_m$ is the predicted target value.

3.6 Bagging

Ensembles learning tecniques are widely used in Machine Learning and consists in the combination of prediction of different learning algorithms trained on many bootstrapped samples, as can be seen in Figure 3.5. Bagging is a technique to reduce

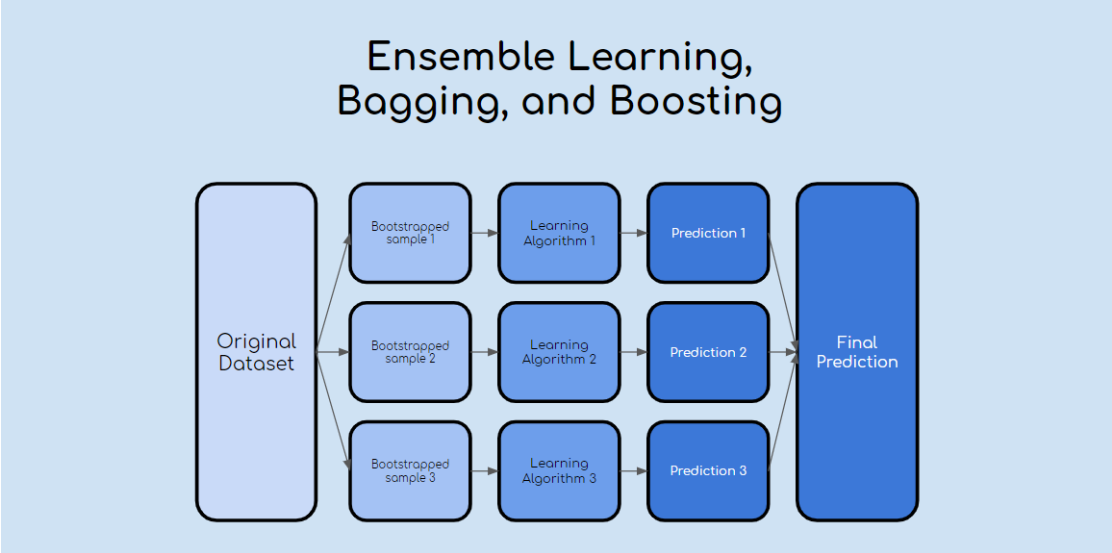


Figure 3.5: Examples of ensemble learning techniques

the variance of a statistical learning method. Since deeper trees are characterized by low bias and high variance, this method is greatly used in this context.

The idea is related to a theoretical result according to which *averaging a set of observations reduces variance*. Of course we don't have access to multiple training sets, so bagging is the solution of the problem which consists in *sampling with replacement* to obtain different datasets from the original training set.

The result of bagging consists in B different bootstrapped training sets. Then the model is trained on the b-th set to obtain a prediction $\hat{f}^{(*b)}(x)$ for the point x. In the end, all the predictions are averaged.

In bagging regression trees, the final prediction is obtained by averaging all the single results from each training set:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} f^{*b}(x) \tag{3.34}$$

For what concerns bagging classification trees, the final prediction is the result of the majority vote of the trees of the forest.

On average, each bagged tree makes use of around two-thirds of the observations. Since we sample with replacement we can have duplicate observations in each bootstrapped set, then each tree will be trained on average on 63.2% of training data.

3.7 Random Forest and Extra Trees

Random forests are ensemble learning methods that make use of a set of uncorrelated decisions trees to construct models that are more robust and less prone to overfitting. They are based on the *bagging method*, but they apply another improvement in order to decorrelate trees. In fact, in addition to taking the random subset of data, random forests also apply a random selection of the features rather than using all the features for each tree.

Tipically, among p predictors, a random selection of $m = \sqrt{p}$ is made.

So, when building decision trees, each tree is built from a sampling with replacement and the choice of the best split is made according to a subset of predictors. These sources of randomness reduce the variance of the model.

An alternative algorithm that increases the randomness even more is the Extra Trees. The differences with respect to Random Forest are:

- it does not use bootstrap samples but it uses the whole dataset

- when performing a split, it does not choose the best split according to a gain measure, but for each feature it chooses a random value for the split.

3.8 Boosting & AdaBoost

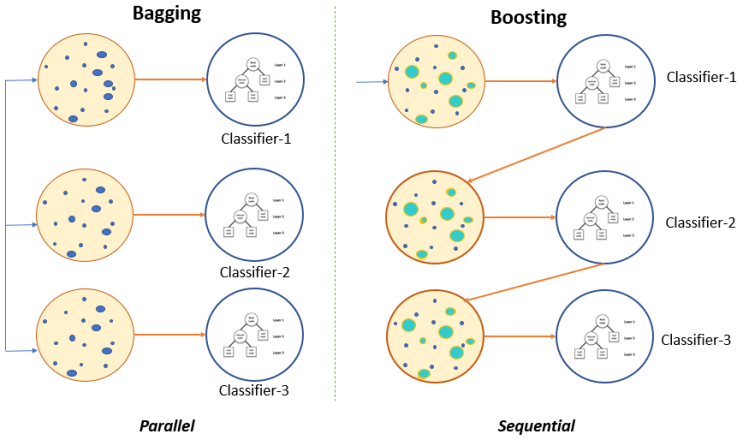


Figure 3.6: Comparison of bagging and boosting algorithms

Boosting is a technique very similar to the bagging method, with the main difference that it does not work in parallel but it is a sequential method (Figure 3.6). Its goal is to reduce not only variance but also bias in order to manage the bias-complexity tradeoff which consists in finding a tradeoff between the approximation and the estimation error. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor.

AdaBoost is one of the most important boosting technique whose goal is to combine multiple weak classifiers to build one strong classifier. It takes as input a training set of exemples $S = (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_m}, y_m)$ where $y_i = f(\boldsymbol{x_i})$ and proceeds for T consecutive rounds.

For each round $t = 1, ..., T$ it defines a distribution $D^{(t)}$ over the set of data. Then it passes the distribution and the set of data to the weak classifier which returns a weak hypothesis whose error ϵ_t is at most $\frac{1}{2} - \gamma$.

Then, it assigns a weight to this hypothesis equals to $w_t = \frac{1}{2} log(\frac{1}{\epsilon_t} - 1)$. This weight is used to update the distribution during the next round, in such a way that data points on which the hypothesis is wrong will get a higher probability mass while examples on which it is right will get a lower probability mass. In this way it forces the attention of the weak learner on those samples that make the classification difficult.

3.9 Logistic Regression

Logistic regression is a Generalized Linear Model used for classification tasks which squashes the regression line into a S-shaped curve between 0 and 1. To do so, a special non-linear function called *sigmoid* is applied to the output value of a linear regression model.

The sigmoid function is defined as follows:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{3.35}$$

Defining p as the conditional probability that x belongs to class 1 having seen the data x, logistic regression applies the following equation to assign a label to a sample:

$$p(X) = P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{3.36}$$

The goal of the learning is to estimate the parameters β_0 and β_1 in such a way that the likelihood of predicting the given class on the seen samples is maximized. We transform the equation above by applying the *logit function* to $p(X)$:

$$log\frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X \tag{3.37}$$

Then we perform maximum likelihood estimation to obtain an estimation of the parameters:

$$L(\beta_0, \beta) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i)) \tag{3.38}$$

Also in the cost function a new parameter called C is introduced: this is the inverse of regularization, thus meaning that smaller values imply stronger regularization.

3.10 SGD Classifier

The Stochastic Gradient Descent classifier is a linear classifier, such as SVM or Logistic Regression, trained via the Stochastic Gradient Descent optimization method which aims at minimizing the risk function using a gradient descent procedure.

Gradient descent

Gradient descent is an iterative algorithm that starts from an arbitrary point on a function and moves along its slope until it reaches its minimum.

The gradient of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ at $\boldsymbol{w}$ is denoted with $\nabla f(\boldsymbol{w})$ and is the vector of partial derivatives of f with respect to each independent variable:

$$\nabla f(\boldsymbol{w}) = \left(\frac{\delta f(\boldsymbol{w})}{\delta w_1}, ..., \frac{\delta f(\boldsymbol{w})}{\delta w_d} \right) \tag{3.39}$$

The value of the gradient defines the direction and rate of the *fastest increase*: when a point p is stationary, the gradient in that point equals 0; otherwise, the direction and the magnitude of the gradient vector gives the direction in which the function increases most quickly from p and the rate at which it happens. For this reason, since we are interested in a minimization problem, we want to move in the opposite direction which is the one of the *fastest decrease* given by the negative gradient.

The idea of the algorithm is:

1. Start with a random initial value of $\boldsymbol{w}$
2. At each iteration, take a step in the direction of the fastest decrease
3. Perform the update step according to the rule:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \nabla f(\boldsymbol{w}^t) \tag{3.40}$$

   where η is a parameter called *learning rate* which affects the convergence rate of the algorithm since it determines how large the moving step is.

Subgradients

If we deal with functions that are not differentiable, we can still use the gradient descent algorithm by generalizing it through subgradients.

For a convex function f, the gradient at $\boldsymbol{w}$ defines the slope of a tangent that lies below the function, that is

$$\forall \boldsymbol{u}, \quad f(\boldsymbol{u}) \geq f(\boldsymbol{w}) + < \boldsymbol{u} - \boldsymbol{w}, \nabla f(\boldsymbol{w}) > \tag{3.41}$$

On the other side, a *subgradient* of f at $\boldsymbol{w}$ is a vector $\boldsymbol{v}$ such that

$$\forall \boldsymbol{u}, \quad f(\boldsymbol{u}) \geq f(\boldsymbol{w}) + < \boldsymbol{u} - \boldsymbol{w}, \boldsymbol{v} > \tag{3.42}$$

In plain words, a subgradient of a convex function at a point w is the slope of any line that touches the function at w and lies below the function.

In the left-side of Figure 3.7 we can see the tangent line of f at $\boldsymbol{w}$; in the right-side several subgradients of a non-differentiable convex function are shown.

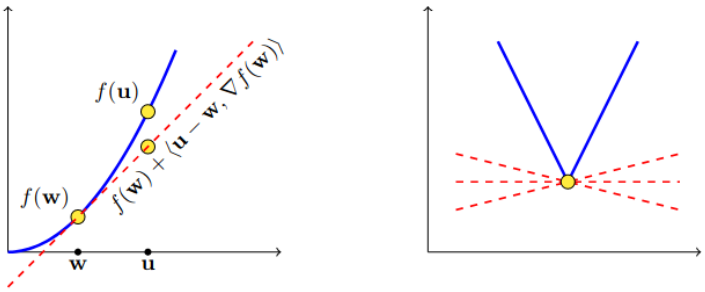


Figure 3.7: Tangent line of a function and sub-gradients

Algorithm
The goal of a learning algorithm is the minimization of the risk function $L_D(\boldsymbol{w})$ where D is the probability distribution from which the data are sampled, which is unknown. Being D unknown, also the gradient of the risk function is. To solve this problem, the Stochastic Gradient Descent algorithm allows to take an update step along a random direction only requiring that the expected value of this random vector at each iteration corresponds to a subgradient of the function at the current vector.

To approximate the gradient of the risk function, instead of computing the gradient all of the data points, it estimates it each sample at a time, selected randomly after shuffling the data. For each training example, the algorithm updates the model parameter according to the same rule as before which depends on the learning rate.

3.11 Linear Regression Models

Linear Regression
Linear Regression is the simplest regression algorithm that models the relationships between a scalar response and one (*simple linear regression*) or more (*multiple linear regression*) explanatory variables using linear predictor functions.

In a simple linear regression problem, the matrix form of the model would be:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{3.43}$$

where:

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \boldsymbol{X} = \begin{pmatrix} \mathbf{x}_1^\mathsf{T} \\ \mathbf{x}_2^\mathsf{T} \\ \vdots \\ \mathbf{x}_n^\mathsf{T} \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \tag{3.44}$$

- $\boldsymbol{y}$ is a vector of *response variables* or dependent variables;
- $\boldsymbol{X}$ is a matrix of row vectors $\boldsymbol{x_i}$ or column vectors $\boldsymbol{X_j}$ called *regressors* or independent variables. Usually the first column is a constant and the corresponding element of $\boldsymbol{\beta}$ is called *intercept*;
- $\boldsymbol{\beta}$ is the vector of *regression coefficients*;
- $\boldsymbol{\epsilon}$ is the vector of *error terms* which captures all the factors that may affect the dependent variable other than the regressors (natural variability, measurement errors, other sources of uncertainty).

The learning phase requires the estimation of the regression coefficients to minimize the error term $\boldsymbol{\epsilon} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}$. A standard method to estimate the parameters is the *Ordinary Least Squares* which aims at minimizing the residual sum of squares between the observed targets and the predicted ones. The idea is to find the coefficients $\boldsymbol{\beta}$ such that:

$$\hat{\boldsymbol{\beta}} = \text{argmin}_\beta \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 \tag{3.45}$$

This minimization problem gives as a result the corresponding estimator:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X^T X})^{-1} \boldsymbol{X^T y} \tag{3.46}$$

One strong assumption behind the Ordinary Least Squares method is that the features are independent, but we can have situations where the predictors are correlated not just to the response variables, but also to each others: this phenomenon is called *multicollinearity*. Multicollinearity can have as a consequence inaccurate estimations of the regression coefficients or the increase of their standard errors. In order to correct this phenomenon, we can use regularization techniques such as the *Ridge* and *Lasso* regression models.

Ridge

Ridge regression uses *L2 regularization* that adds an L2 penalty to the optimization problem, which is equal to the square of the magnitude of the coefficients:

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2 + \alpha ||\boldsymbol{\beta}^2|| \tag{3.47}$$

where α is a parameter that controls the amount of shrinkage: when $\alpha = 0$, the ridge regression is equal to the ordinary least squares regression; when $\alpha = \infty$. all coefficients tend towards to zero. The larger the value of α, the higher is the strength of the penalty term. A tuning of this hyperparameter is necessary to reach a good trade-off between bias and variance.

We can also apply Ridge to classification problems, by converting the binary labels to $\{-1, 1\}$ and then proceeds as a regression task. The predicted class corresponds to the sign of the regressor's prediction.

The limitation of this method is that it only minimizes the coefficients but it never excludes coefficients, even if they are close to zero, so it does not decrease the complexity of the model.

Lasso

On the other side, Lasso regression performs L1 regularization, so it adds a penalty equal to the absolute value of the magnitude of the coefficients:

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2 + \lambda ||\boldsymbol{\beta}|| \tag{3.48}$$

This method allows coefficient to be excluded when they are equal zero, so it can reduce the number of features and result in sparse models.

The parameter λ controls the strength of the penalty: if $\lambda = 0$, we have the result obtained with the Ordinary Least Squares method; as λ increases, more and more coefficients are set to zero and eliminated from the model.

There are different alternatives to fit the coefficients of the algorithm: the standard one is coordinate descent which consists in minimizing along coordinate directions to find the minimum; an alternative is to use the Least Angle Regression (LARS) algorithm which leads to the exact solution.

The Lasso Lars first finds the predictor most correlated with the response variable and increases its coefficient in the direction of the sign of its correlation with the response variable. It also take the residuals $\boldsymbol{r} = \boldsymbol{y} - \hat{\boldsymbol{y}}$. Then it stops when finds another predictor that has as much correlation with $\boldsymbol{r}$ as the previous one. It includes this predictor and continues until all predictors are in the model.

3.12 Multilayer Perceptron

The Multilayer Perceptron is a supervised learning algorithm that relies on an underlying neural network to perform the tasks of classification or regression. This class of algorithms has a structure that is inspired by the biological neural networks of human brain.

As in the biological neural network we have neurons connected by synapses that allow the transmission of electrical signals, in a similar way the structure of an artificial neural network is based on a collection of processing units called *perceptrons* connected by edges through which they can transmit signals (weights).

In simple words, the perceptron receives some inputs, performs a weighted linear combination multiplying them by some weights, then passes the result into an activation function to produce an output.

In the Multilayer Perceptron, multiple neurons are arranged into networks with at least three layers: an input layer, one or more hidden layers and an output layer, as shown in Figure 3.8. Moreover, the architecture is fully connected: it means that each neuron of a given layer is connected with weighted edges to all the neurons of the previous and, if exist, following layers.

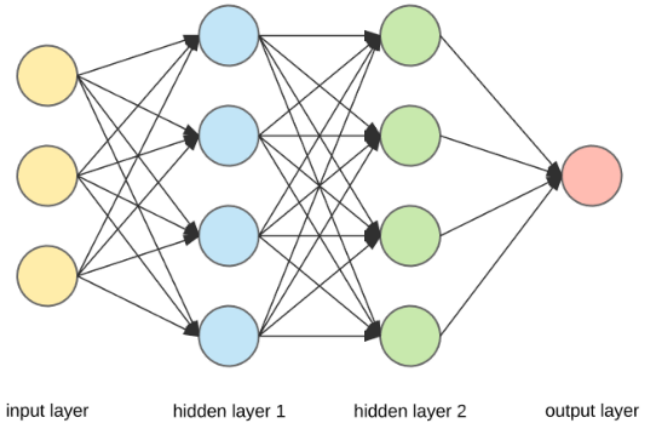


Figure 3.8: Example of a simple Artificial Neural Network

In the nervous system, neurons receive information by the dendrites; in the perceptron, we have the input layer which performs the same function of the dendrites and receive the data: in this way, we have one neuron for each input feature. All the layers after the input are called *hidden layers* because they are not directly explosed to the input; we can have different number of hidden layers according to the complexity and the depth of the network. The final layer is the output layer which gives as a result a value or a vector of values according to the given task.

The output of each neuron, which is sent to the next layer, is computed as a linear combination of all the input signals it receives plus a bias.

For instance, given the i^{th} perceptron, its output y_i is given by:

$$y_i = \phi_i \left(\sum_j (\omega_{ij} x_j) + b_i \right) \tag{3.49}$$

where:

- $\phi_i(\cdot)$ is the activation function of the i^{th} perceptron;
- ω_{ij} is the weight of the input x_j for the i^{th} perceptron;
- b_i is a bias

The weighted linear combination is given as argument of the activation function which acts as a threshold. The goal is to have a threshold that makes sure that the input is a signal and not noise and that is invariant for small variations such as translations (that's why we add a bias). The representation of a perceptron is shown in Figure 3.9.

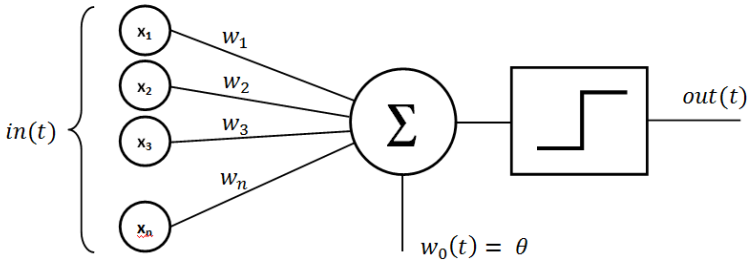


Figure 3.9: Representation of a neuron in an artificial neural network

The activation function is any nonlinear function that normalizes the output between $(0, 1)$ or $(-1, 1)$. For instance, the step function is used in such a way that if the input is above a certain threshold (such as 0.5) the activation function gives 1, otherwise it gives 0 as the output of the i^{th} neuron.

We can see some examples of activation functions in Figure 3.10:

- **Sigmoid function**, defined as:

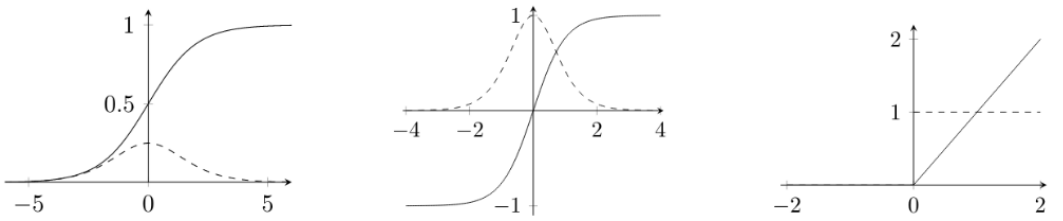
$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.50}$$

- **Hyperbolic tangent function**, defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.51}$$

- **Rectified Linear Unit (ReLU) function**, defined as:

$$ReLU(x) = \max(0, x) \tag{3.52}$$



(A) Sigmoid (B) Hyperbolic Tangent (C) Rectified Linear Unit

Figure 3.10: Examples of activation functions

The training procedure of a neural network consists of several passes called *epochs* and each epoch is divided into two phases: feed-forward and back-propagation.

- **Feed-forward**: the network is fed with the training data which proceeds, layer after layer, activating neurons until an output value is produced for each data point. At the end, the output of the network is compared with the ground truth and the loss is computed;
- **Back-propagation**: the error computed in the previous phase is propagated back to the hidden layers to calculate the gradient of the loss function with respect to all the weights and biases at each layer. Then, the parameters are modified according to the update rule provided by the optimization technique (such as Stochastic Gradient Descent).

For what concerns the classification task, the training consists in miniziming the *Cross-Entropy* loss function, defined as follows:

$$L = -\sum_{i=1}^{K} y_k \log(\hat{y_k}) \tag{3.53}$$

where $\boldsymbol{y}$ is the ground truth vector, $\hat{\boldsymbol{y}}$ is the estimate and K is the total number of classes.

Moreover, the last layer of a classification neural network is a *softmax output layer* which normalizes the output to a probability distribution over all the possible classes: this means that the values of the output vector, after applying the Softmax function, will be in the range $(0, 1)$ and will sum up to 1 so that they can be interpreted as probabilities.

On the other hand, regression tasks are solved by minimizing the *Mean Squared Error*:

$$\frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y_k})^2 \tag{3.54}$$

and the output layer is linear, which means that the activation function is the identity.

Chapter 4

Topological Data Analysis

In this chapter we will consider the theory behind Topological Data Analysis and persistent homology, an important technique that will be used in this work to inspect the neurological data and extract meaningful information. The background explained in the next sections takes as references [3] and [8].

4.1 Graph theory

The brain connectome and the organization and interactions among the neurons can be modeled exploiting network science and graph theory.

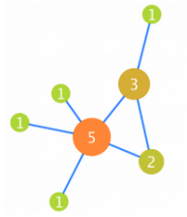
The mathematical concept that is used in network science to model the interconnections among a network is the one of *graph*.

Definition A **simple graph** is a pair $G = (V, E)$ where V is a set of *vertices* or *nodes* which represent the units of the network and $E \subseteq V \times V$ is a set of *edges* or *links* which represent the pairwise connections between units. Each link $e = (i, j)$ is an ordered pair of nodes i and j in V.

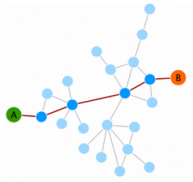
In some contexts it is useful to associate to each edge a positive scalar value called *weight*. In such case the graph is called **weighted** and it is described by a triplet $G = (V, E, W)$ where $W \subseteq \mathbb{R}_+^{V \times V}$ is the weight matrix whose elements W_{ij} are greater than zero if and only if there is an edge between nodes i and j.

A graph G is referred to as **undirected** if the weight (*weighted graph*) or adjacency matrix (*unweighted graph*) is symmetric, thus meaning that if an edge exists then also the one in the reverse direction does and, if the graph is weighted, the two links have the same weight.

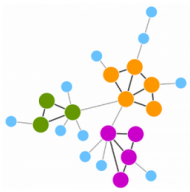
There are many measures used to describe graph topology.



Node degree For an unweighted and undirected graph, the degree of a node is the number of edges that connect it with the other nodes. If the graph is directed, we can distinguish between the out-degree and the in-degree, which are associated to the number of out-links (links going from the node i to the others) and in-links (links going inside the node i) respectively. When the graph is also weighted, the out-degree and in-degree represents the sum of the weights of the out-links and in-links respectively.
The degree distribution is the fraction of nodes having degree k as a function of k. Degree distribution identifies scale-free networks which are networks that have a small number of nodes with high degree, called *hubs*, and most of the nodes with low degrees.



Path length Path length is the minimum number of edges that must be traversed to go from one node to another. If the graph is weighted, it corresponds to the minimum sum of edge weights. Efficiency is inversely related to path length. The shortest path is the path of minimum length between any two nodes and is called *geodesic path*.



Clustering The clustering coefficient quantifies the number of connections between the nearest neighbours of a node as a proportion of the maximum number of possible connections. Clusters represent group of nodes that are more connected with themselves than they are with the others.

Modularity It is a measure of how well groups have been partitioned into clusters or communities. It compares the relationships in a cluster compared to what would be expected for a random number of connections.

One of the drawbacks of using graph theory to describe neural network is the fact that we can derive some local properties such as the node degree or global properties such as the average path length but these measures do not give a clear idea about the entire structure of the network. For this reason, algebraic topology can be used to obtain better insights and quantitative information about both the local and global properties of a graph.

4.2 Topological spaces

From a mathematical point of view, a graph is a discrete object where vertices are abstract elements and edges represent pairwise connections between them. On the other hand, in terms of topology, a graph can be considered as a 1-dimensional geometric object, where vertices are points and edges are curves that connect them. To switch from graph to topological spaces we need to exchange between discrete and continuous models of reality.

Topology is designed to capture the notion of continuity and to generalize it to more general spaces.

Definition A topological space is a pair (X, T) where X is a set and T is a collection of subsets of X such that:

- $\varnothing \in T$ and $X \in T$;
- for every (infinite) collection $\{O_\alpha\}_{\alpha \in A} \subset T$, we have $\bigcup_{\alpha \in A} O_\alpha \in T$;
- for every finite collection $\{O_i\}_{1 \leqslant i \leqslant n} \subset T$, we have $\bigcap_{1 \leqslant i \leqslant n} O_i \in T$.

The set T is called a **topology** on X and the elements of T are called **open sets**. The three conditions mean that the empty set and the set X itself are open sets, that an infinite union of open sets is an open set and a finite intersection of open sets is an open set.

4.3 Simplicial complexes

The main data structures used to represent topological spaces are the ones called *simplicial complexes* which decompose the spaces into simpler and smaller pieces in order to describe and deal with them in a easier way. A simplicial complex is a generalization of the concept of graph and it is a collection of elements called *simplices*.

Definition Let $u_0, u_1, ..., u_k$ be points in $\mathbb{R}^d$. Consider a linear combination of the points $x = \sum_{i=0}^{k} \lambda_i u_i$ with each $\lambda_i \in \mathbb{R}$. An **affine combination** is a linear combination where $\sum_{i=0}^{k} \lambda_i = 1$. A set of points is affinely independent if any two affine combinations of these points are the same if and only if their coefficients are. A **convex combination** is an affine combination with $\lambda_i \geq 0, \forall i$. The *convex hull* of n points is the set of all the convex combinations of the points. A **k-simplex** is the convex hull of $k+1$ affinely independent points and the convex hulls of the subsets of the points are called **faces**.

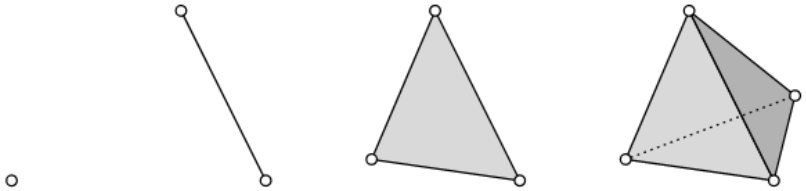


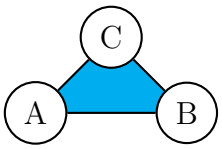
Figure 4.1: Examples of simplices

It is easy to visualize simplices in lower dimensions (Figure 4.1): a 0–simplex is a single vertex, a 1–simplex is an edge between two vertices, a 2–simplex is a triangle, a 3–simplex is a tetrahedron.

Definition A **simplicial complex** X is a finite set of simplices K such that:

- $\forall \sigma \in K$ and every non-empty $\tau \subset \sigma$ we have that $\tau \in K$: this means that every face of a simplex in X is still part of X;
- the intersection $\sigma \cap \sigma_0$ of any two simplices $\sigma, \sigma_0 \in K$ is either empty or a face of both.

The dimension of the simplicial complex is defined as the maximal dimension of any of its simplices. The dimension of a simplex is equal to the cardinality of the generators of its convex hull, minus 1.



$$K = \{[A], [B], [C], [A, B], [A, C], [B, C], [A, B, C]\}$$

From this perspective we can see that a graph is a one-dimensional simplicial complex whose vertices are the 0–simplices and edges are the 1–simplices.

Simplicial complexes can be considered not only as geometric objects, but it is also possible to give a combinatorial description that facilitates effective computations without constructing them in the Euclidean space. The combinatorial structure of a simplicial complex is obtained by replacing each simplex in a simplicial complex by the set of its vertices in order to discard the geometry of the simplices.

Definition An **abstract simplicial complex** with vertex set V is a collection X of finite subsets of V such that for every $\sigma \in K$ and every non-empty subset $\tau \subset \sigma$ we have that $\tau \in X$.

So an abstract simplicial complex is closed under restriction, meaning that any face of any simplex in X is also a simplex.

It is useful to consider, for a simplicial complex, the set of simplices with dimension at most k.

Definition Let X be a simplicial complex and $k \in \mathbb{N} \cup \{0\}$. The k–skeleton of X is the simplicial complex $\Gamma = \{\sigma \in X : \dim(\sigma) \geq k\}$.

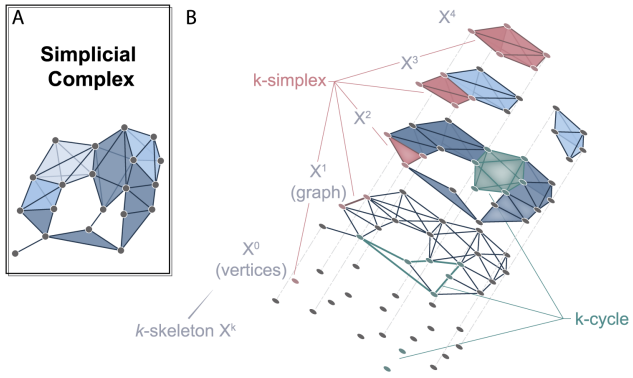


Figure 4.2: Examples of a simplicial complex and the associated k–skeleta [2]

For instance, the 1–skeleton contains all the 0–simplices and 1–simplices of X.

It is always possible to associate to an abstract simplicial complex X a topological space which is called **geometric realization** $|X|$ that can be embedded in an Euclidean space.

Theorem Every abstract simplicial complex of dimension d has a geometric

realization in $\mathbb{R}^{2d+1}$.

In order to construct the topology of simplicial complexes we need to define the *standard k–simplex*.

Definition The **standard k–simplex** is the subset of $\mathbb{R}^n$:

$$\Delta^k := \left\{ x \in [0,\infty)^{k+1} : \sum_{i=0}^{k} x_i = 1 \right\} \tag{4.1}$$

The $(k-1)$–faces of Δ^k are copies of Δ^j with $j < k$. To obtain the geometric realization of an abstract simplicial complex X, for each k–simplex of X a copy is produced and then they are attached together connecting faces.

We can define the k–skeleton of X as the quotient space:

$$X^{(k)} := (X^{(k-1)} \bigcup \coprod_{\sigma : dim(\sigma)=k} \Delta^k) / \sim \tag{4.2}$$

where we have that $\coprod$ represents the disjoint union and $\sim$ represents the equivalence relation that identifies the faces of Δ^k with the corresponding faces in the lower-dimensional skeleton. The simplicial complex is therefore defined by the union $X = \bigcup_{k=0}^{\infty} X^{(k)}$.

4.4 Simplicial homology

Homology is a mathematical method that allows to associate an algebraic object or a sequence of algebraic objects to other mathematical objects such as topological spaces. It is the theory of holes which provides invariants for shape description and characterization: it is based on the observations that two shapes can be topologically distinguished by detecting their holes.

In this context we apply homology theory to simplicial complexes which is a branch called *simplicial homology*. To define simplicial homology, we need to consider some relevant definitions and some algebra concepts first.

4.4.1 Algebra background

In this section we explain some concepts that will be useful for a better understanding of the subsequent definitions.

Definition A **group** $(G, \cdot)$ is a set G endowed with a binary operation $\cdot$ that combines any two elements a and b to form an element of G, $a \cdot b$, that satisfy the *group axioms*:

- **Associativity:** $\forall a, b, c \in G, \quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- **Identity:** $\exists e \in G$ such that $\forall a \in G \quad e \cdot a = a$ and $a \cdot e = a$. This element is called *identity element* of the group.
- **Inverse:** $\forall a \in G, \exists b \in G$ such that $a \cdot b = e$ and $b \cdot a = e$ where e is the *identity element*.

Moreover, if $\forall a, b \in G, \quad a \cdot b = b \cdot a$, then the operation is commutative and the group is called **abelian group**.

Definition A subgroup of $(G, \cdot)$ is a subset $H \subset G$ such that $\forall a, b \in H, \quad a \cdot b \in H$. So the elements of H, equipped with a group operation on G restricted to H, form a group.

Definition A group homomorphism from $(G, *)$ to $(H, \cdot)$ is a function $h : G \to H$ such that for all $u, v \in G$ it holds that $h(u * v) = h(u) \cdot h(v)$ where the group operation on the left side is the one of G and on the right side is the one of H. If f is a bijection, it is called *isomorphism*. A homomorphism is a map that preserves the structure between two algebraic structures of the same type.

Definition The group $(\mathbb{Z}/2\mathbb{Z}, +)$ is a quotient of the group of integers $\mathbb{Z}$ and the subgroup $2\mathbb{Z} = \{2n, n \in \mathbb{Z}\}$ which contains the elements of $\mathbb{Z}$ that are divisible by 2 (all even integers). This quotient group has only two elements {0,1} where 0 represents the class of even numbers and 1 the class of odd numbers. It can be given a *field* structure by defining the multiplicative identity element [1] besides the additive identity [0]. The field operations on $\mathbb{Z}/2\mathbb{Z}$ are defined as the addition and multiplication modulo 2, as can be seen in Table 4.1.

+	**0**	**1**
0	0	1
1	1	0

×	**0**	**1**
0	0	0
1	0	1

Table 4.1: Addition and multiplication modulo 2

Any commutative group V such that $\forall v \in V, v + v = 0$ can be turned into a vector space over $\mathbb{Z}/2\mathbb{Z}$ by defining $0v = 0$ and $1v = v$.

4.4.2 Chains, cycles and boundaries

In order to obtain the homology groups from a simplicial complex, the first step is to orient the simplices.

Orientation Given the set S of vertices of a simplex, we define an orientation on the simplex by selecting some particular ordering of $S = (v_0, ..., v_k)$. Two orderings of the vertices determine the same orientation of the simplex if and only if an even permutation transforms an ordering into the other; if the permutation is odd, the orientations are said to be opposite. Any simplex has only two possible orientations and the orientation on k–simplex *induces* orientation on its $(k-1)$–faces (Figure 4.3).

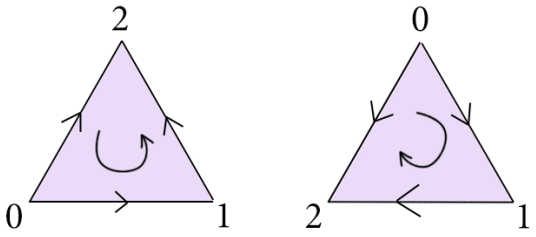


Figure 4.3: Examples of two possible orientations of a 2–simplex [9]

We can define an oriented simplex as a pair (simplex, orientation).

Chain complex The second step is to take all the formal sums of the oriented simplices in the complex. Chains are simplicial analogs of paths in the continuous domain.

Given an oriented simplicial complex X, a simplicial k–**chain** is the set $C_k(K)$ whose elements are the formal sums

$$c^k = \sum_{i=1}^{n_k} a_i \sigma_i^k, \quad a_i \in \mathbb{Z}/2\mathbb{Z} \tag{4.3}$$

where σ_i is an oriented k–simplex and n_k denotes the number of k–simplices in X. For the sake of simplicity, we consider the *modulo* 2 *coefficients* belonging to the field $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$. The set of all k–chains together with the addition operation forms a group. Moreover, we can also give $C_k(K)$ a $\mathbb{Z}/2\mathbb{Z}$–vector space structure, whose basis is the set of elementary chains $\{\sigma_i | \dim(\sigma_i) = k\}$.

- The addition of two k–chains is componentwise: if we have $c = \sum a_i \sigma_i$ and $c' = \sum b_i \sigma_i$, then $c + c' = \sum (a_i + b_i)\sigma_i$ and the coefficients satisfy $1 + 1 = 0$

- In set notation, the sum of two k–chains is their symmetric difference
- The inverse of c is $-c = c$ since $c + c = 0$
- The group of k–chains is abelian because addition modulo 2 is abelian

Example For instance, if we consider the simplicial complex $K = \{[0], [1], [2], [0,1], [0,2]\}$

- the 0–chains $C_0(K)$ contain the following 8 elements:

Figure 4.4: Examples of 0–chains for the K simplex defined above [21]
$C_0(K) = \{0, [0], [1], [2], [0] + [1], [0] + [2], [1] + [2], [0] + [1] + [2]\}$.

- the 1–chains $C_1(K)$ consist of 4 elements:

Figure 4.5: Examples of 1–chains for the K simplex defined above [21]
$C_1(K) = \{0, [0,1], [0,2], [0,1] + [0,2]\}$.

Boundary operator To relate the groups of k–chains we define a boundary operator. Given an oriented k–simplex $\sigma = (v_0, ..., v_k)$, the boundary operator

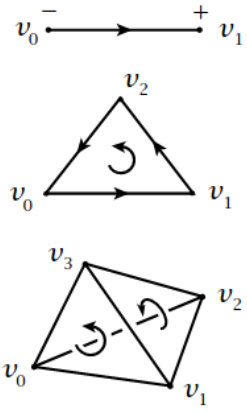
$$\delta_k : C_k \to C_{k-1} \tag{4.4}$$

is the homomorphism defined by

$$\delta_k(\sigma) = \sum_{i=0}^{k} (-1)^i [v_0, ..., \hat{v_i}, ..., v_k] \tag{4.5}$$

where $\hat{v_i}$ means that the i–th vertex has been deleted.

The boundary operator associates a k–chain to a $(k-1)$–chain by taking the linear combination of the boundaries of the simplices in the chain. If we have $c = \sum a_i \sigma_i$, then $\delta_k c = \sum a_i \delta_k \sigma_i$. The boundary of a k–simplex is the sum of its oriented $(k-1)$–dimensional faces.



$$\partial[v_0, v_1] = [v_1] - [v_0]$$

$$\partial[v_0, v_1, v_2] = [v_1, v_2] - [v_0, v_2] + [v_0, v_1]$$

$$\partial[v_0, v_1, v_2, v_3] = [v_1, v_2, v_3] - [v_0, v_2, v_3] + [v_0, v_1, v_3] - [v_0, v_1, v_2]$$

Figure 4.6: Examples of the boundary operator [8]

Example In Figure 4.6 we can see that we use the signs so that the faces are coherently oriented. For instance, if we consider the triangle example, first we cut out v_0 and we obtain $[v_1, v_2]$ which is coherent with the orientation in the picture that goes from v_1 to v_2; then we cut out v_1 and we obtain $[v_0, v_2]$ with a $-$ sign because in the picture we see that the simplex is oriented from v_2 to v_0 and not viceversa.

When we consider the simplest case of $k-$chains with modulo 2 coefficients, then the boundary operator is defined as follows:

$$\delta_k(\sigma) = \sum_{i=0}^{k} [v_0, ..., \hat{v_i}, ..., v_k] \tag{4.6}$$

because in $\mathbb{Z}/2\mathbb{Z}$ the operations of sum and subtraction coincide.

Definition A **chain complex** is the sequence of chain groups connected by boundary homomorphisms:

$$C_{k+1} \xrightarrow{\delta_{k+1}} C_k \xrightarrow{\delta_k} C_{k-1} \to ... \to C_1 \xrightarrow{\delta_1} C_0 \xrightarrow{\delta_0} 0 \tag{4.7}$$

At this point, we distinguish two subgroups of the chain group that are the keys elements to define the concept of homology.

Taking a $1-$*chain* corresponding to a path in the simplicial complex and sending it through the operator δ_1 gives as result the two boundary nodes of the path, since

the intermediate nodes will cancel. In particular, if we deal with a closed path, we can notice that its boundary is equal to 0.

Definition A $k-cycle$ is a $k-$chain with empty boundary $\delta_k c = 0$. We can define the group of $k-$cycles, which is a subgroup of C_k, as Z_k.

We can deduce that the **kernel** of the boundary operator coincides with the group of $k-$cycles: $Z_k = \ker\delta_k$.

On the other hand, also boundaries form a subgroup of the $k-$chains. A $k-$boundary is a $k-$chain that is the boundary of a $(k+1)-$chain. So, the group of $k-$boundaries is the **image** of the $(k+1)-$st boundary homomorphism: $B_k = Im\delta_{k+1}$.

We can notice that if we take the simplex $[a,b,c,d]$ we obtain:

$$\delta[a,b,c,d] = [b,c,d] - [a,c,d] + [a,b,d] - [a,b,c]$$

$$\delta(\delta[a,b,c,d]) = [c,d] - [b,d] + [b,c] - [c,d] + [a,d] - [a,c] + [b,d] - [a,d] + [a,b] - [b,c] + [a,c] - [a,b] = 0$$

It can be proven that this result is true in general.

Fundamental lemma of homology $\delta_k\delta_{k+1}d = 0$ for every k and every $(k+1)-$chain d.

From this fundamental lemma, it follows that $\mathrm{Im}\delta_{k+1} \subseteq \ker\delta_k$.

Proof Let $b \in B_k$ be a boundary. By definition, there exists $c \in C_{k+1}$ such that $b = \delta_{k+1}c$.

Using $\delta_k\delta_{k+1} = 0$ we get:

$$\delta_k b = \delta_k\delta_{k+1}c = 0 \tag{4.8}$$

Hence $b \in Z_k$.

So we have that:

$$B_k \subseteq Z_k \subseteq C_k \tag{4.9}$$

4.4.3 Homology groups

We now know that if we want to detect the topological features such as cavities in our simplicial complex, we need to look at the cycles. In fact, if there are cavities and holes, then we have some cycles of dimension k belonging to Z_k that will

sorround them, but also some other cycles belonging to B_k that are boundaries of higher dimensional simplices.

Geometrically we have that holes are cycles that are not also boundaries. Moreover, we can consider Figure 4.7. We detect three green cycles, each of them

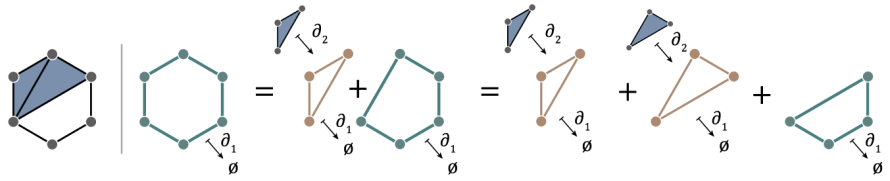


Figure 4.7: Examples of cycles (green) and boundary cycles (gold) [2]

surrounds the hole and they only differ from one another by the addition of a boundary cycle. The insight we get is that if we add a boundary cycle b to a cycle c the resulting cycle $b + c$ will surround the same hole as c. Then we want to find the equivalence classes of a cycles c, so the set of equivalent elements, where two cycles $c1$ and $c2$ are *equivalent* if $c1 = c2 + b$ for some $b \in B_k$.

From an algebraic point of view, this amounts to taking the quotient:

$$H_k := \frac{ker\delta_k}{Im\delta_{k+1}} = \frac{Z_K}{B_k} \tag{4.10}$$

The idea is that every cycle that is a boundary of a higher dimensional simplex or that can be built as a linear combination of borders of simplices acts like a 0 in this space.

H_k is called k**–th homology group** and its dimension is equal to the number of topological holes of dimension k:

$$\beta_k = dim(H_k) = dim(Z_k) - dim(B_k) \tag{4.11}$$

β_k is called the k**–th Betti number**.

β_0, β_1 and β_2 count respectively the number of connected components, the number of holes and the number of voids in the simplicial complex.

The elements of the homology groups are called *homology classes* and each homology class is an equivalence class of cycles: the elements are obtained by adding all k–boundaries to a given k–cycle. Two cycles in the same homology class are **homologous** (they can be continuously deformed into each other).

Example In the example in Figure 4.8, $\{ab, ac, bc\}$ and $\{ac, ad, cd\}$ are two

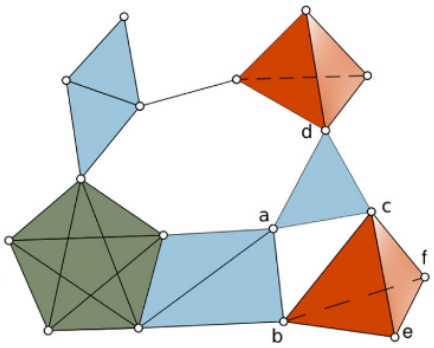


Figure 4.8: Examples of homological cycle $H1$ [17]

1–chains, of which the latter is the boundary of the triangle acd while the former is not in the boundary of any higher dimensional chain. So, $\{ab, ac, bc\}$ is a 1–dimensional homological cycle $\in H1$. Moreover, the 2–chain $\{bce, bcf, bef, cef\}$ is the boundary of the tetrahedon $bcef$ so it is not a 2–dimensional cycle.

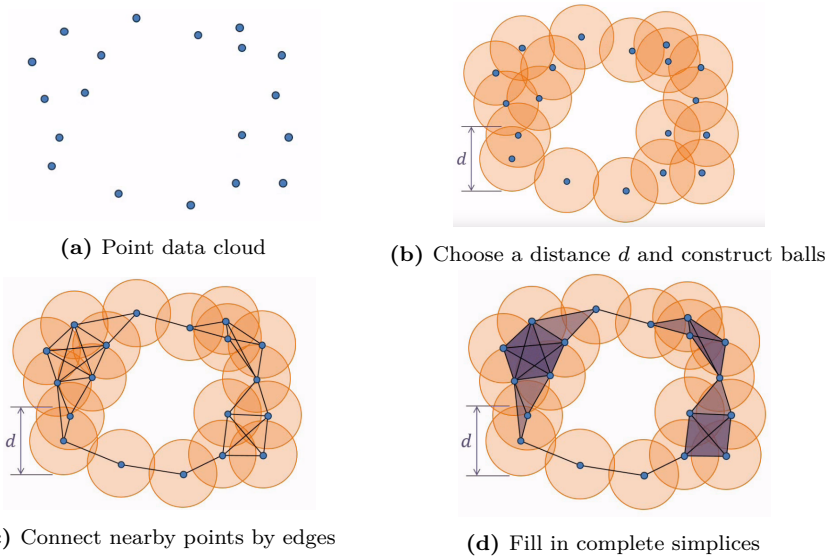
4.5 Persistent homology

4.5.1 Building simplicial complexes from data

There are many ways to construct simplicial complexes from a dataset or a topological or metric space. A metric space is defined by the couple (X, d) where X is a set equipped with a *metric* on the set. The metric is a function that defines the distance between any two points of the set.

We can define a simplicial complex that is constructed from a metric space starting from point clouds, which are sets of points in a normed vector space. The general procedure can be summarized by the following points:

- Given the data point cloud, set all the points to vertices (0–simplices)
- Choose a distance $\delta > 0$
- Draw balls of diameter δ around each point
- Connect two vertices if the distance between them is smaller or equal to δ (the corresponding balls intersect)
- We obtain a graph that captures the clusters formed by the points but does not give information about higher-order features such as holes. We need to fill the graph with simplices.



(a) Point data cloud

(b) Choose a distance d and construct balls

(c) Connect nearby points by edges

(d) Fill in complete simplices

Figure 4.9: Example of the construction of the Vietoris-Rips complex

To fill the graph with simplices we can use different strategies that give as a result different simplicial complexes.

Definition Given a data point cloud X and a distance $\delta > 0$, the **Čech complex** has a k–simplex when the intersection of the corresponding δ–balls around the points x_i is not empty:

$$\check{C}_\delta(X) := \{[p1, p2, ..., p_k] | \{p_1, p_2, ..., p_k\} \subseteq X, \cap_i B(p_i, \delta) \neq 0\} \tag{4.12}$$

Definition Given a data point cloud X and a distance $\delta > 0$, the **Vietoris-Rips complex** has a k–simplex when the pairwise distance between every pair of vertices in the simplex is at most δ:

$$VR_\delta(X) := \left\{[p1, p2, ..., p_k] | \{p_1, p_2, ..., p_k\} \subseteq X, max_{p_i, p_j \epsilon \sigma}(dist(p_i, p_j)) \leq \delta\right\} \tag{4.13}$$

We can see that the main difference between these two complexes is that in the Čech complex you need to have a triple intersection of the three balls to have the triangle. The three points do not form a 2–simplex but they are pairwise intersected so they form three 1–simplex. On the other side, in the Rips complex the three points are pairwise less than δ apart, so they form a 2–simplex.

At this point, the big deal is: how to choose the right value of δ that best captures the topological features of our data? We can see that if δ is too small, we get multiple distinct connected components and small holes that are simply a consequence of noise. If δ is too big, we get a giant simplex which has a trivial homology.

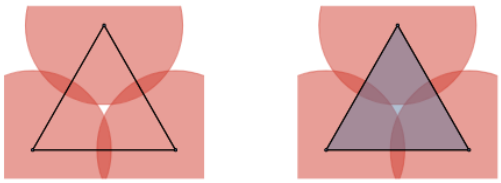


Figure 4.10: Difference between the Čech complex (left) and the Vietoris-Rips complex (right)

Persistent homology is the solution to his problem, in that it allows to consider all the distances $\delta > 0$ and provides a powerful tool to encode the evolution of homology groups and eliminate noise from data.

4.5.2 Filtration

The first element needed to capture the idea of analyzing a dataset for all the different values of the distance between points is the concept of filtration.

Given a simplicial complex X, Y is a *subcomplex* of X if it is a subset of X that is still a simplicial complex.

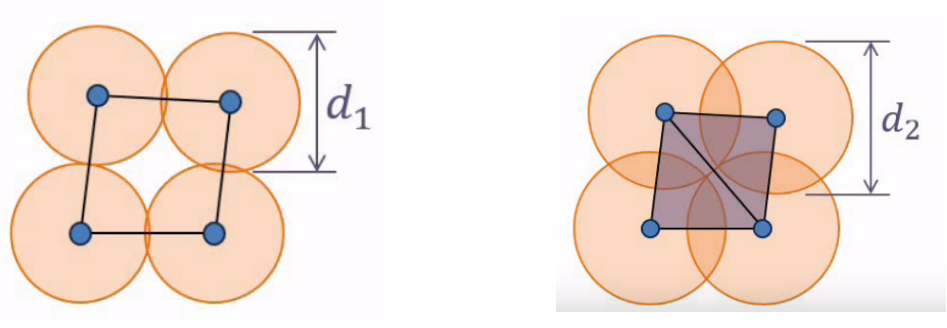
Definition Given X, a **filtration** of X is a family of subcomplexes X_i such that

$$\varnothing = X^0 \subseteq X^1 \subseteq ... \subseteq X^n = X \tag{4.14}$$

T he goal of persistence homology is to study how the topological features vary in a simplicial complex when simplices are added and when the value of the metric changes.
As we can see in the figure below, each hole appears at a particular value of δ (d_1) and disappears at another value of δ (d_2). The concept of filtration allows to describe the evolution of the simplicial complex: we start from the point cloud and, increasing the value of the distance parameter, simplices are added to the network to obtain a nested sequence of simplicial complexes.

If we consider a sequence of Vietoris-Rips complexes associated to a fixed data point cloud and we analyze increasing values of the parameter $\delta, 0 \leq \delta \leq N$, the sequence

$$VR_0(X) = VR_{\delta_0}(X) \xrightarrow{\iota} VR_{\delta_1}(X) \xrightarrow{\iota} ... \xrightarrow{\iota} VR_{\delta_M}(X) = VR_N(X) \tag{4.15}$$



(a) Point data cloud

(b) Choose a distance d and construct balls

Figure 4.11: Example of the persistence of a hole

is a filtration of simplicial complexes.

Actually, we are not interested in the sequence of complexes, but mostly in the topological evolution which is described by the sequence of homology groups. Hence, the filtration induces a sequence of homology groups connected by homomorphisms:

$$H_k(K_0) \xrightarrow{\iota^\star} H_k(K_1) \xrightarrow{\iota^\star} ... \xrightarrow{\iota^\star} H_k(K_n) = H_k(K) \tag{4.16}$$

Definition The $p-$persistent $k-$th homology group of X^i is defined as:

$$H_k^{i,p} = \frac{Z_k^i}{B_k^{i+p} \cap Z_k^i} \tag{4.17}$$

where Z_k^i and B_k^i represent the $k-$th cycle group and the $k-$th boundary group of the $i-$th complex in the filtration sequence.

The $p-$persistent Betti number of X^i is defined as:

$$B_k^{i,p} = dim(H_k^{i,p}) \tag{4.18}$$

and represents the number of $k-$holes at the filtration index i that persist at the filtration index $p + i$.

In the figure below we can see in [A] the filtered simplicial complex with complex $X^0 \subseteq ... \subseteq X^T$. Moreover, we can associate to each simplicial complex a chain complex as depicted in [E]. In this way, we can get maps from $C_k(K^i)$ to $C_k(K^{i+1})$: these maps are called *chain maps* l and are shown in the horizontal direction in [F]. Hence, we can move horizontally through the chain maps and vertically through the boundary operator as described in the previous chapter.

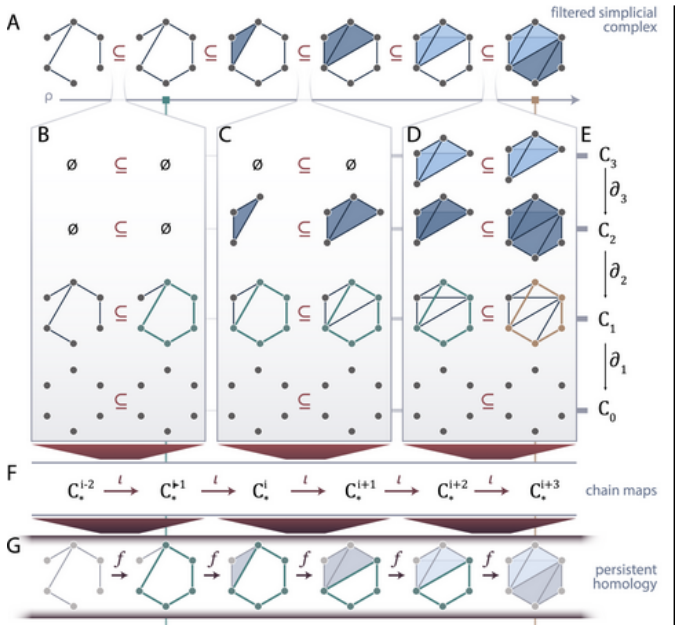


Figure 4.12: Example of persistent homology [2]

We can notice that when we move across the filtered simplicial complex sequence, the topological features change: in [B] there is a new cycle in green, which persists in [C] and is mapped to a boundary cycle in [D] and turns from green to gold.

From the chain maps we get also induced maps on the homology groups $f_k : H_k(K^i) \to H_k(K^{i+1})$, as depicted in [G]. These maps are the key to identifying the evolution of topological features from the *birth* which represents the first appearance to the *death* that is the value at which the cavity is filled.

4.5.3 Representations of Persistent Homology

There are two common ways to visualize persistent homology: via barcode or persistence diagrams.

We can represent the persistence of a hole as a pair (b, d) where the first element stands for *birth* and we say that a hole is born at step t if it appears for the first time in the corresponding step of the filtration and d represents *death* which is the step t at which the hole disappears.

The couple birth-death can be visualize through a bar that goes from b to d. A collection of such bars is a *persistence barcode* which contains a bar for each

homological feature, as in the example below. An alternative representation is

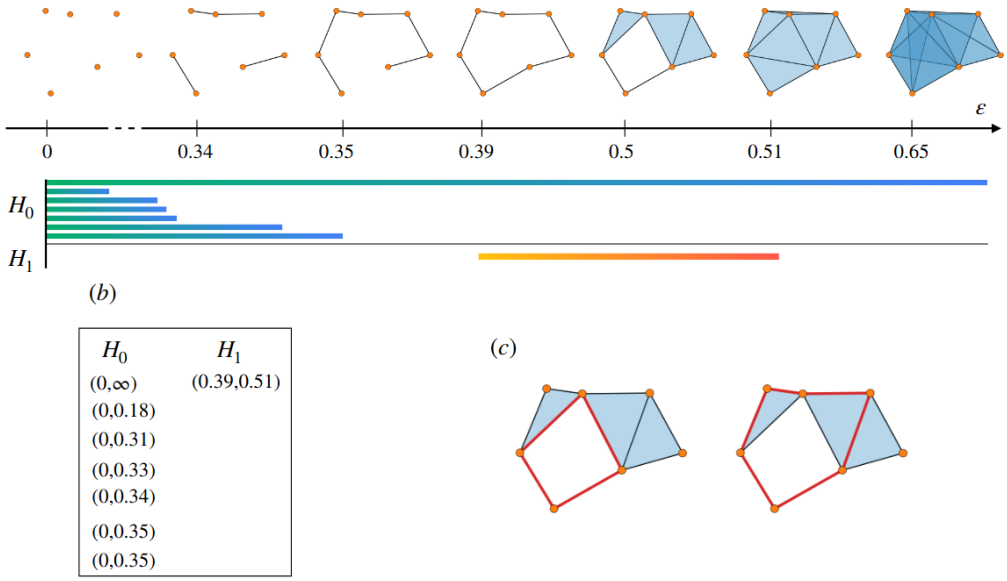


Figure 4.13: Example of a persistence barcode [12]

given by the *persistence diagram* which is obtained by plotting the points (b, d) in $\mathbb{R}^2$. Each point (i, j) represents a class whose persistence is given by its vertical distance $j - i$ from the diagonal.

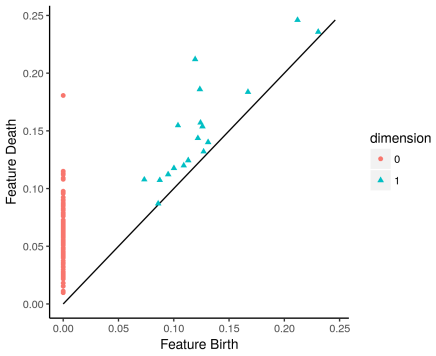


Figure 4.14: Example of a persistence diagram [12]

These two types of representations are very useful because short barcode or points that lie close to the diagonal represent noise because they refer to classes with very low persistence; on the contrary, long bars or points distant from the diagonal are very informative about the topology of data.
Moreover, barcodes are stable with respect to perturbations of data.

Chapter 5

Applications and results

5.1 Data

Structural connectivity
The first dataset used for the analysis is the one exploited in [14]. The complete dataset is obtained by joining three different datasets: the Maastricht GROUP, the Dublin and the Cobre datasets.

These data are obtained from both T1w and DWI images of 412 subjects, divided as follows:

- **Maastricht GROUP**: 68 control subjects and 83 patients affected by different pathologies: 57 patients with schizofrenia, 11 patients with psychotic disorder, 2 patients with brief psychotic disorder, 9 patients with schizoaffective disorder and 4 patients with schizophreniform disorder;
- **Dublin**: 82 control subjects and 33 patients, of whom 3 were diagnosed with schizoaffective disorder and 30 with schizophrenia;
- **Cobre**: 77 control subjects and 69 patients of whom 60 diagnosed with schizophrenia and 9 with schizoaffective disorder.

The data used in this work are the results of previous pre-processing steps performed to obtain, for each subject, a parcellation of the human brain into 308 cortical regions with the same surface area according to the Desikan-Killiany atlas. In the end we get 308 regional values of 7 morphometric features: cortical thickness, grey matter volume, surface area, mean curvature, mean diffusivity and fractional anisotropy.

All the subjects are divided in two groups: 1 = control, 2 = patient. In this case, we have a categorical response variable that we can use to perform a classification task.

The second dataset, taken from [22], is used to perform a regression task as the response variable is continuous. It contains a sample of 297 young healthy people in the age range 14-24 years stratified into 5 contiguous age-related data: 14-15 years inclusive, 16-17 years, 18-19 years, 20-21 years and 22-24 years.

All scans were acquired using 3T whole-body MRI systems, two located in Cambridge and one located in London. A backtracking algorithm was used to parcellate 66 regions defined by the Desikan–Killiany atlas into 308 contiguous parcels of approximately equal area across both hemispheres. For each parcel, 8 morphometric features are computed.

Functional connectivity
The third dataset is a pharmacological MRI dataset used to replicate the results obtained in [7]. It contains images obtained from both T1w and BOLD fMRI scans. Also in this case, the data used in this work are the results of pre-processing techniques that lead as a result to a segmentation of the structural MRI images into 194 cortical and subcortical regions according to the Destrieux anatomical atlas.

The scans refer to fifteen healthy, hallucinogen-experienced subjects. Each of them was scanned two times, the second time after 14 days: subjects received placebo (10-mL saline) the first time and psilocybin (2 mg in 10-mL saline) the second. Each scan consisted of an anatomical scan (T1w) followed by a 12 min eyes-close resting-state blood oxygen-level-dependent (BOLD) fMRI scan: 6 minutes after this last scan, injections began and continued for 60 seconds.

5.2 Machine Learning

The first step of the analysis consists in applying Machine Learning algorithms on the first two datasets to predict control vs patient in the first case and brain age in the second case. Our goal is to compare the predictive ability of the original morphometric features with respect to the one of the features obtained from the connectivity matrices computed from them. To do so, we proceed in three ways:

- apply the learning algorithms on the original features
- apply the learning algorithms on the connectivity features
- apply the learning algorithms on the stacked features

In the end, we compare all the results to understand the contribution of the different features on the prediction task.

5.2.1 Classification task

We start our analysis with the psychosis dataset to perform a classification task that consists in predicting the label of the subjects: 1 if the subject is a patient, 0 if he/she is control.

Original features

Starting with the original morphometric features, the pipeline consists in acquiring and processing all the data to obtain a three dimensional dataset composed as follows:

- 412 subjects
- 308 brain regions
- 7 morphometric features

The dataset is split into a training and a test set using a stratification sampling technique to preserve the distribution of data. In particular, in neuroimaging data there tend to be large effects of scanner site. Standardisation might correct these to some extent, but as an additional step we use the parameter referred to the scanner site as an additional parameter for stratification. This means that in this case we stratify both on the scanner site and the true label. The size of the test set is 25% of the whole dataset and before applying the split the data are shuffled.

Before feeding the learning algorithms with our data, we first need to standardize them. The standardization is done across all the subjects and regions, within each feature. To avoid any data leakage, the standardization is performed after splitting the data into two sets: we do not want our model to be biased and to know any information about the distribution of the test data, so we standardize the training data and then we use the normalization parameters previously obtained to transform the test set.

In this part, we use the original morphometric features as predictors. We fed different learning algorithms that are trained through `GridSearchCV` with 20 folds and a different set of hyperparameters for each classifier:

- **Decision Tree**:
  - *criterion*: gini, entropy;
  - *max features*: auto, sqrt, log2;
  - *splitter*: best, random.
- **Random Forest**:

  - *criterion*: gini, entropy;
  - *max features*: auto, sqrt, log2;
  - *bootstrap*: True, False.

- **Logistic Regression**:
  - *penalty*: l1, l2;
  - *C*: 0.0001, 0.01, 0.1, 1, 10, 100, 1000.

- **SVM with linear kernel**:
  - *penalty*: l1, l2;
  - *C*: 1e-4, 1e-3, 1e-2, 1e-1, 0.5, 10, 25, 100, 1000

- **Ridge classifier**:
  - *alpha*: 1, 0.1, 0.01, 0.001, 0.0001, 0;
  - *fit intercept*: True, False;
  - *solver*: svd, cholesky, lsqr, sparse_cg, sag, saga.

- **Multi-Layer Perceptron**:
  - *alpha*: 1e-4, 1e-3, 1e-2, 1e-1;
  - *learning rate init*: 1e-3, 1e-2, 1e-1, 0.5, 1.

- **Stochastic Gradient Descent**:
  - *loss*: log, hinge, modified_huber, squared_hinge, perceptron.

- **SVM with RBF kernel**:
  - *C*: 0.0001, 0.01, 1, 10, 100;
  - *gamma*: 0.001, 0.01, 1, 10, 100.

To perform cross validation the `Stratified K-Fold` cross-validator is used, that returns folds that preserve the percentage of samples for each class.

We visualize the performances of the classifier using *boxplots*. A boxplot is a type of chart which represents a five-number summary of data:

- **minimum score**: the lowest data, excluding outliers;
- **maximum score**: the largest data, excluding any outliers;

- **first quartile**: the twenty-fifth percentile;
- **median**: the middle value of a dataset;
- **third quartile**: the seventy-fifth percentile.

The interquartile range is the distance between the upper and the lower quartile and it contains the middle 50% of data points. Boxplots are useful because they also show outliers, as data that are outside the whiskers of the box (points above the maximum and below the minimum score).

For each classifier, we take the accuracy scores obtained at each iteration using the *best model*, that is the one that gives the highest mean cross-validated score.

If we consider the performances reached using the original morphometric features, we can see in Figure 5.1 that most of the algorithms suffer from high variance. The accuracy scores vary considerably according to the folds and the median is generally quite low. Also, the performances on the test set are pretty bad, meaning

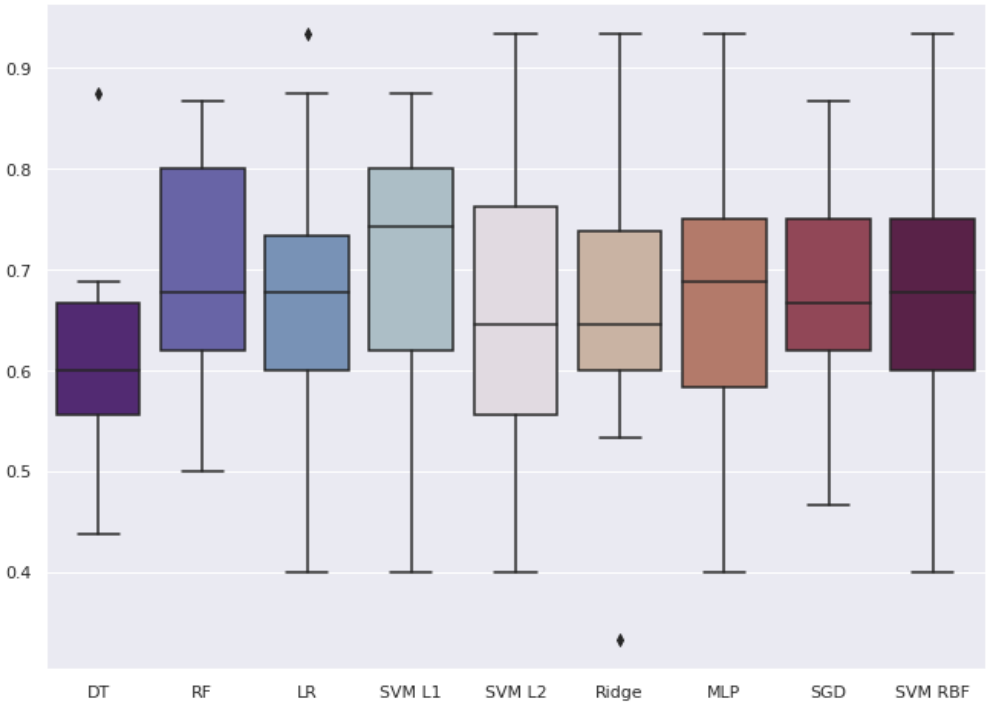
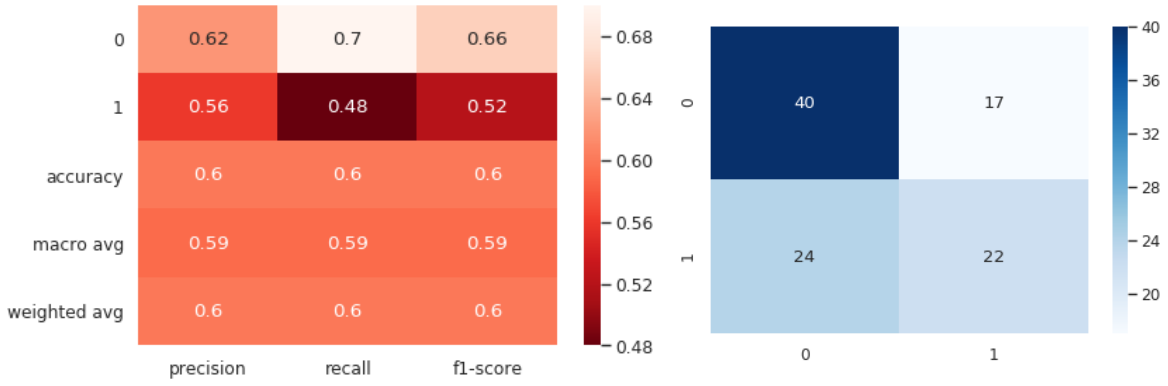


Figure 5.1: Comparison of different classification algorithms using the original morphometric features

that all the models tend to overfit. For instance, if we take the Linear SVM that

has a mean value of accuracy across all the folds of 0.71 and we analyze the results on the test set we obtain the report and the confusion matrix showed in Figure 5.3.

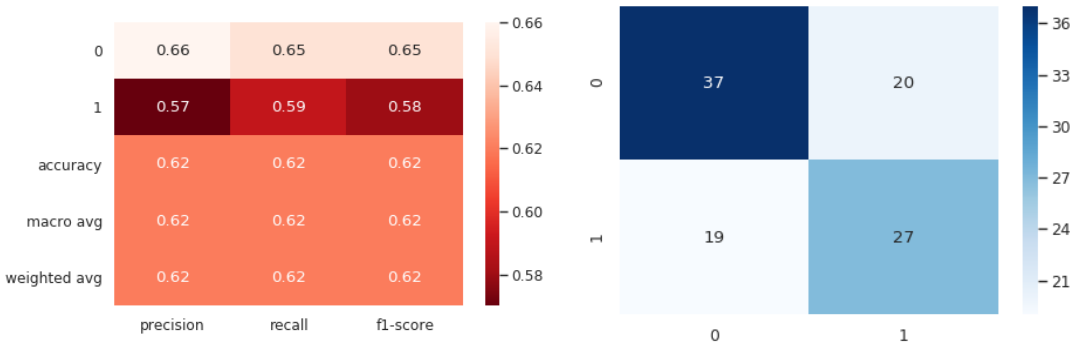


(a) Classification report

(b) Confusion matrix

Figure 5.2: Results related to Linear SVM applied on the original morphometric features

As already underlined previously, when dealing with healthcare data we should give a different weight to the kind of errors. In this context, we consider it worse if a patient is classified as control than if control is classified as a patient (imagining that in the last case the diagnosis is double-checked by human beings). For this reason, we aim at maximizing the recall. The classification report shows that the positive class (the one of patients) has a very low recall score with respect to the negative class. Even if the data are not extremely imbalanced, we try to deeper explore this problem considering some techniques to balance the dataset to see if the performances improve.



(a) Classification report

(b) Confusion matrix

Figure 5.3: Results related to Linear SVM with SMOTE applied on the original morphometric features

We try both undersampling and oversampling with SMOTE and we apply grid search using recall as score. Observing the results in Figure 5.3, we can infer that the experiment using SMOTE improves the model both in terms of recall and accuracy.

Considering that we are dealing with many features, we also try to reduce the dimensionality of our dataset applying PCA. We require to preserve at least the 90% of the variance explained by the original dataset and, as we can see in Figure 5.4, we select 162 components.

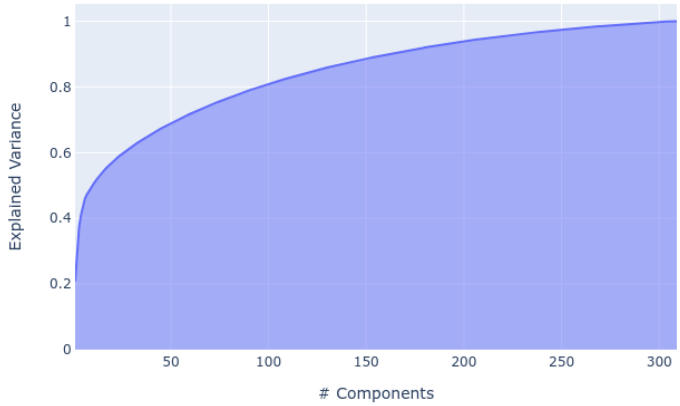
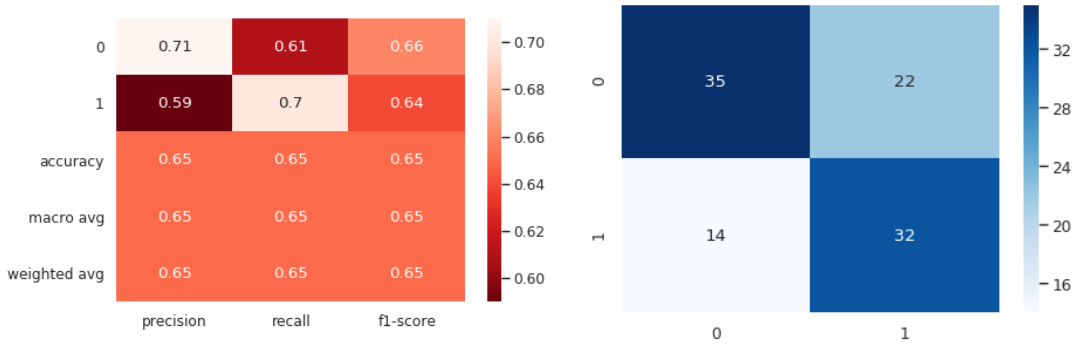


Figure 5.4: Proportion of variance explained plot

In terms of a tradeoff between accuracy and recall, the algorithms that performs better in this case is again Linear SVM that improves the results obtained so far - even if they are still quite low, as we can see in Figure 5.5.



(a) Classification report

(b) Confusion matrix

Figure 5.5: Results related to Linear SVM with PCA applied on the original morphometric features

Morphometric similarity matrices' features

At this point, we try to construct morphometric similarity matrices starting from the morphometric features and use them as predictors. We consider different kinds of correlation matrices:

- correlation with Pearson's r
- correlation with Spearman's ρ
- partial correlation
- tangent space parametrization

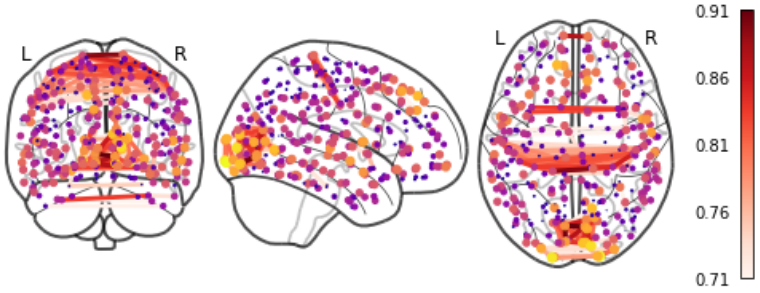
A preliminary step to construct these matrices consists in standardizing the data. The standardization is done across all the regions, within each feature and subject. We generally use Z-score standardization, but when we compute the non-parametric Spearman's ρ we also exploit a non-parametric way to standardize the data which is the median/MAD normalization. Then we use the morphometric feature vectors to form a 308×308 morphometric similarity matrix for each subject.

We can obtain a network representation of the connectome using these matrices that are thresholded to preserve only the most significant connections. In Figure 5.6 we can see three plots:

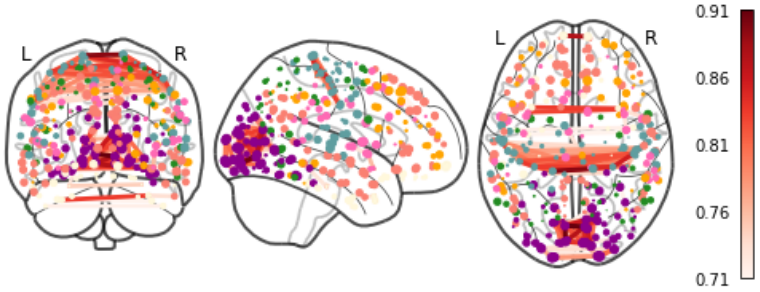
- the first plot that displays only the positive edges;
- the second plot that displays only the positive edges coloured according to the Yeo 7 networks parcellation of the human cerebral cortex (visual in purple,

somatomotor in blue, dorsal attention in green, ventral attention in pink, limbic in cream, frontoparietal in orange and default in red);

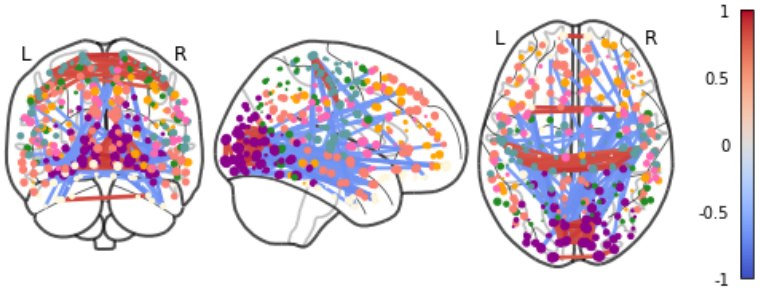
- the third plot that displays both positive and negative edges.



(a) Positive edges only



(b) Positive edges, coloured by functional Yeo subnetworks



(c) Both positive and negative edges

Figure 5.6: Plot of the connectome matrices

These matrices are processed in different ways: as a first step, we consider the same pipeline used before to divide the data into training and test sets and, since our matrices are symmetric, we extract the lower triangular matrices and we vectorize them to feed the learning algorithms. In this case, for computational reasons we reduce the number of folds to 10.

First, in Figure 5.7 we consider the overall performances of the different kinds of matrices taking into account the scores of all the learning algorithms (with the best hyperparameters) on the 10 folds. As we can see, there are no considerable differences among them and with respect to the original morphometric features, even if the variance is a little bit reduced. Moreover, the results show that the tangent space parametrization better preserves the structure of the matrices giving slightly higher scores than the other kinds of matrices.

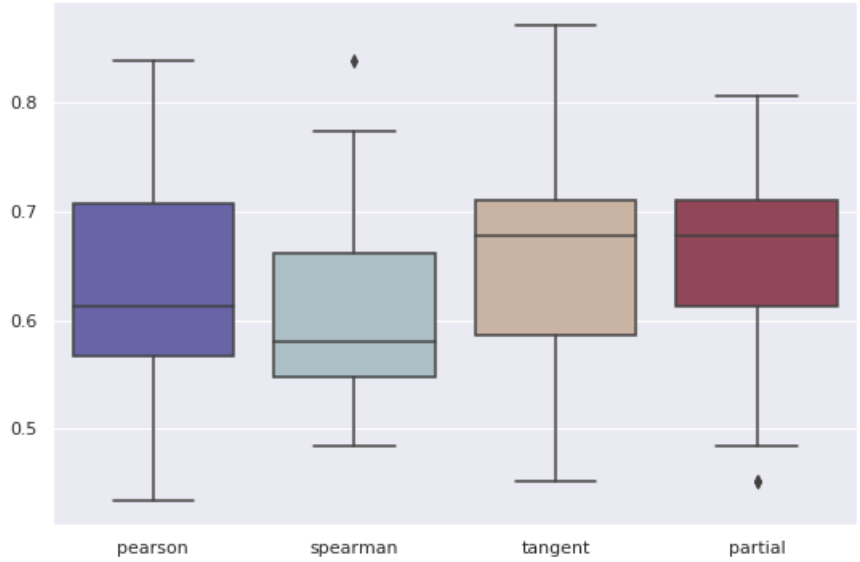


Figure 5.7: Comparison of different structural connectivity matrices for classification task

We can do further consideration analyzing the results more precisely in Table 5.2. For each kind of matrix we register the best algorithms, that are the ones that give the highest results as the mean of the 10 scores, that we will use to get insights on the test scores related to accuracy and recall.

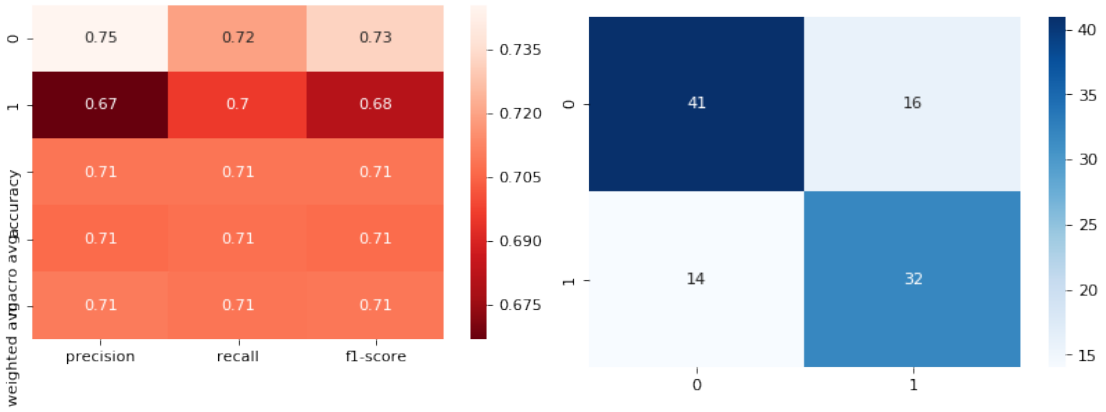
Logistic Regression and Ridge Classifier seem to be the algorithms that best model our data and the tangent space parametrization gives the predictors that perform better in most cases. Now we observe the scores on the test set to check if we deal with overfitting or not and, as before, we try to apply undersampling and oversampling to see if we can improve our results.

These are the algorithms that perform better for each kind of matrix:

Algorithm	*Pearson*	*Spearman*	*Tangent*	*Partial*
Decision Tree	0.60	0.57	0.57	0.56
Random Forest	0.64	0.60	0.60	0.62
Logistic Regression	**0.66**	**0.64**	**0.70**	0.68
Linear SVM	**0.66**	0.60	**0.69**	**0.69**
RBF SVM	0.55	0.62	**0.70**	0.68
Ridge	**0.66**	**0.63**	**0.69**	**0.69**
MLP	0.64	**0.63**	**0.70**	**0.69**
SGD	**0.65**	**0.63**	0.67	**0.70**

Table 5.1: Algorithm comparison in terms of accuracy score

- **Pearson's r**: Ridge Classifier with parameters alpha=1, fit_intercept=False, solver='svd'. Results showed in Figure 5.8.



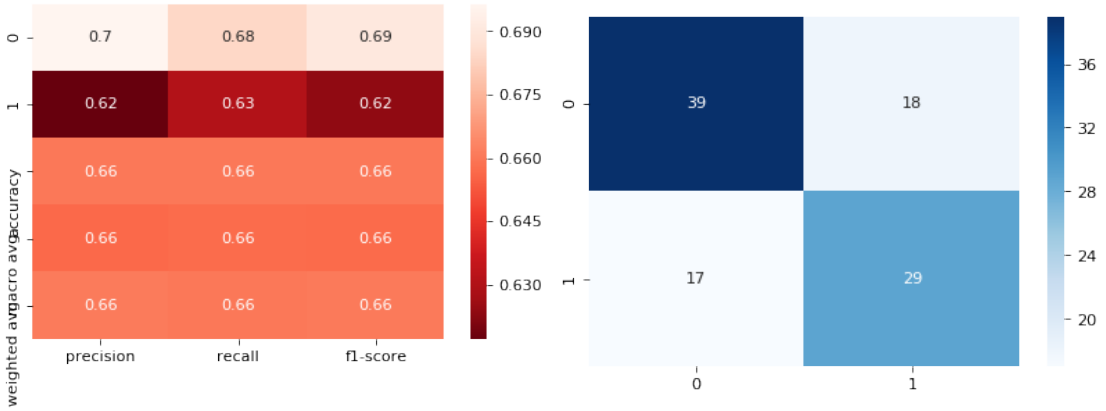
(a) Classification report

(b) Confusion matrix

Figure 5.8: Results related to Ridge Classifier applied on the features obtained from the matrices constructed with Pearson

- **Spearman's** ρ: Ridge Classifier with parameters alpha=0, fit_intercept=True, solver='auto' and undersampling applied. Results showed in Figure 5.9.

- **Tangent parametrization**: in this case, we provide two alternatives, even if in all the cases the scores don't reach considerably high values even if we try all the "best" algorithms with the application undersampling or oversampling techniques. In Figure 5.10 we show the results obtained with Ridge Classifier with parameters alpha=0.1, fit_intercept= True and solver= 'svd' that show a slight imbalance between the positive and negative class in terms of recall

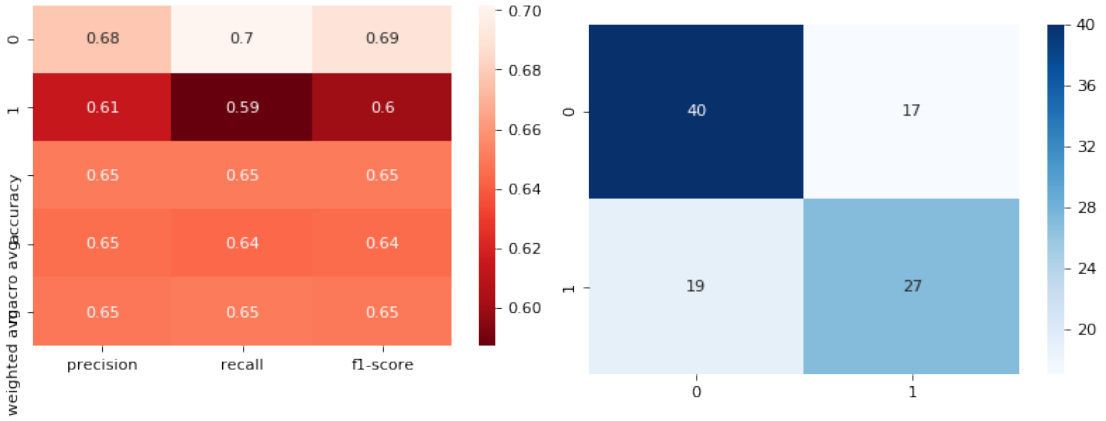


(a) Classification report

(b) Confusion matrix

Figure 5.9: Results related to Ridge Classifier with undersampling applied on the features obtained from the matrices constructed with Spearman

scores and in Figure 5.11 the results obtained with MLP with parameters alpha=0.01 and learning_rate_init=0.001 that give a very low number of false negatives still achieving a good accuracy score.

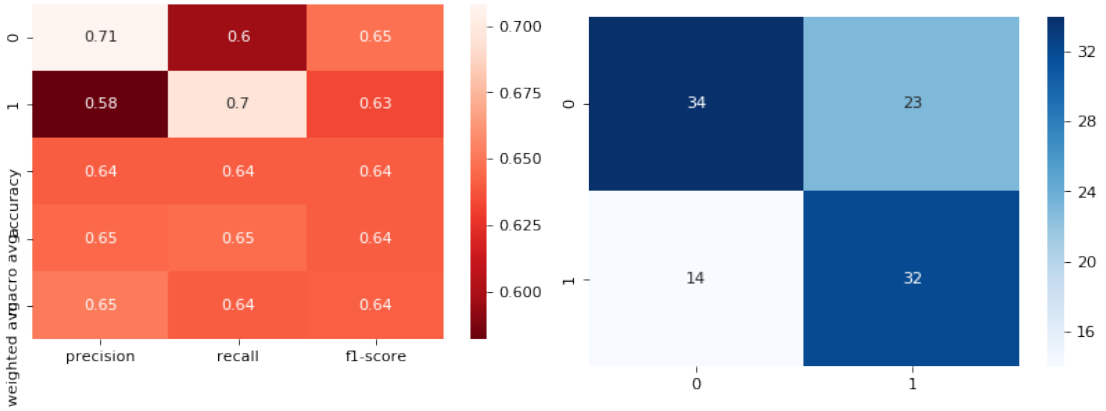


(a) Classification report

(b) Confusion matrix

Figure 5.10: Results related to Ridge Classifier applied on the features obtained from the matrices constructed with partial correlation

- **Partial correlation**: Ridge classifier with parameters alpha=0, fit_intercept=True, solver='auto' and undersampling applied. Results showed in Figure 5.12. On the other hand, SGD classifier performs very well on the positive class, reaching a recall of 0.74 and a number of false negatives equal to 11, but performs badly

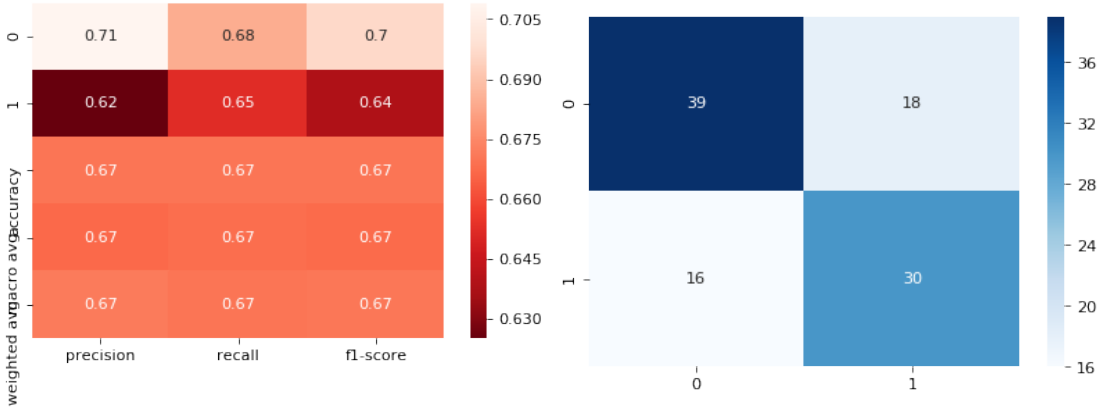


(a) Classification report

(b) Confusion matrix

Figure 5.11: Results related to MLP applied on the features obtained from the matrices constructed with the tangent parametrization

on the negative class (recall: 0.51) giving in the end an accuracy score of 0.62.



(a) Classification report

(b) Confusion matrix

Figure 5.12: Results related to Ridge Classifier with undersampling applied on the features obtained from the matrices constructed with partial correlation

Comparing the scores obtained applying the previous models on the test set, the construction of the correlation matrices with Pearson's r seems to reach the best results, both in terms of recall and accuracy. We can also notice that averagely all the scores are slightly better than the ones obtained using the original morphometric features, suggesting that the use of these matrices' features could be a good idea.

One problem could be the fact that, in these cases, the dimensionality of the

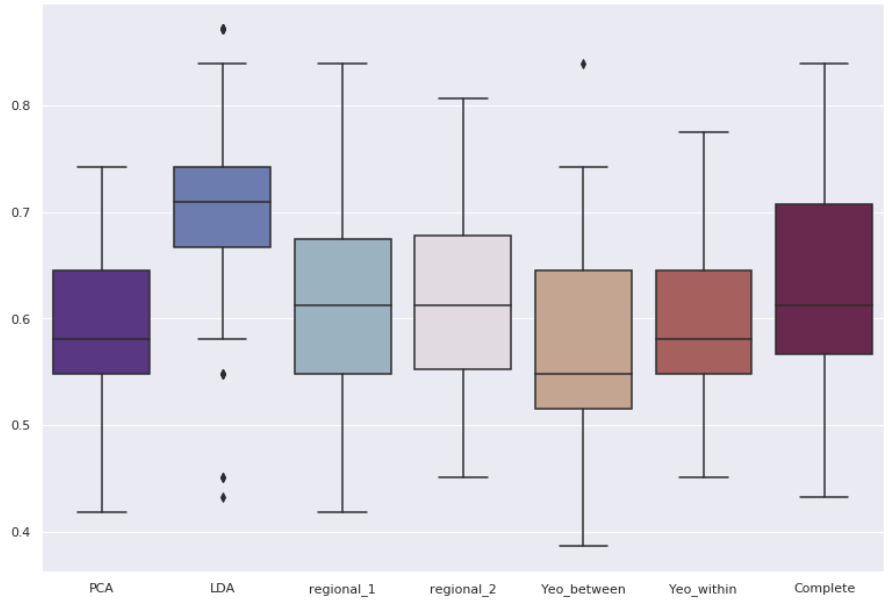
data is very high, since we are vectorizing our 308x308 matrices obtaining as a result 47586 features. For this reason, we try to deal with different dimensionality reduction techniques:

- PCA;
- LDA;
- regional mean at each region $j = 1, ...,308$ as the mean of the j-th row (or column). These values are equivalent to the weighted degree of each regional node, connected by signed and weighted edges of pair-wise similarity to all other nodes in the whole brain connectome represented by the morphometric similarity matrix;
- regional mean as the mean of the positive and negative values separately. In this context, we don't have a clear idea of the neurological and structural meaning of the negative correlations; for this reason, we try to consider these values separately and compute two different means for each regional node;
- mean according to the Yeo 7-subnetworks. We average blocks of the matrices corresponding to similarity *within* and *between* Yeo subnetworks, since we have a correspondance between the 308 regions and the 7 networks. For within-subnetwork similarity we exclude self-correlations and only take into accounts one of the triangulars of that blocks; for similarity *between* pairs of subnetworks we average the whole block. In both cases we obtain in the end a 7×7 matrix for each subject.

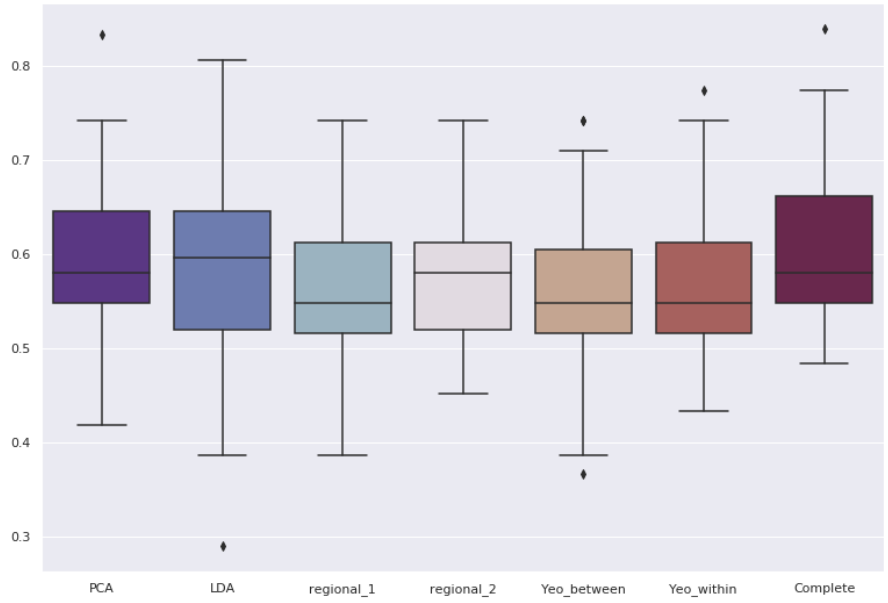
Figure 5.13 and Figure 5.14 show the results of our analysis. The last boxplot with label "Complete" refers to the previous scores obtained considering all the features. As we can see, in most cases the dimensionality reduction techniques do not improve the results and the models sometimes tend to overfit: if we consider the matrices obtained with Pearson's r and we apply LDA the results on the test set are always worse than the ones on the validation sets. For this reason, we consider as *best* results the ones obtained before.

Combination of features

We tried to concatenate and stack the original morphometric features and the vectorized correlation matrices (the lower/upper triangulars) but this worsened our results without showing signs of improvement, probably due to the high dimensionality of the data.

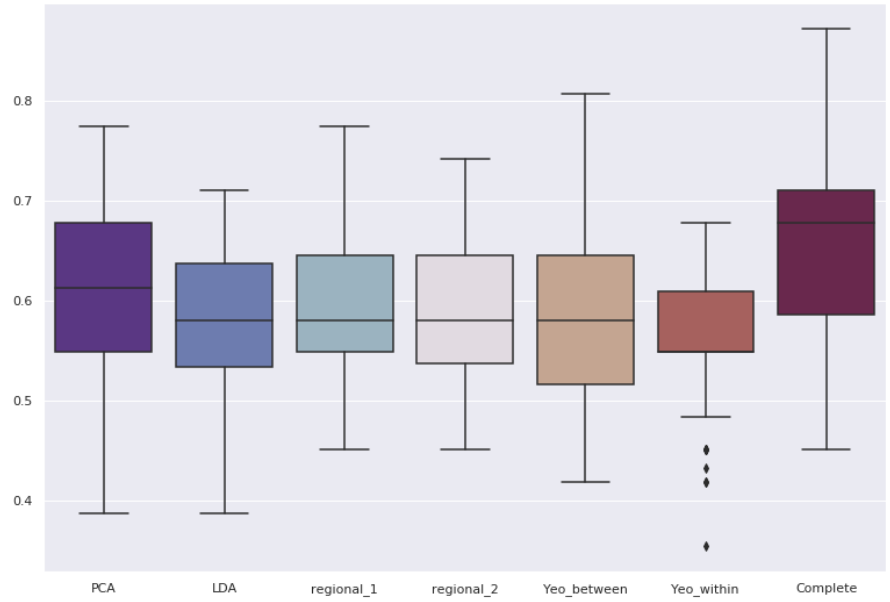


(a) Results of dimensionality reduction using Pearson's r

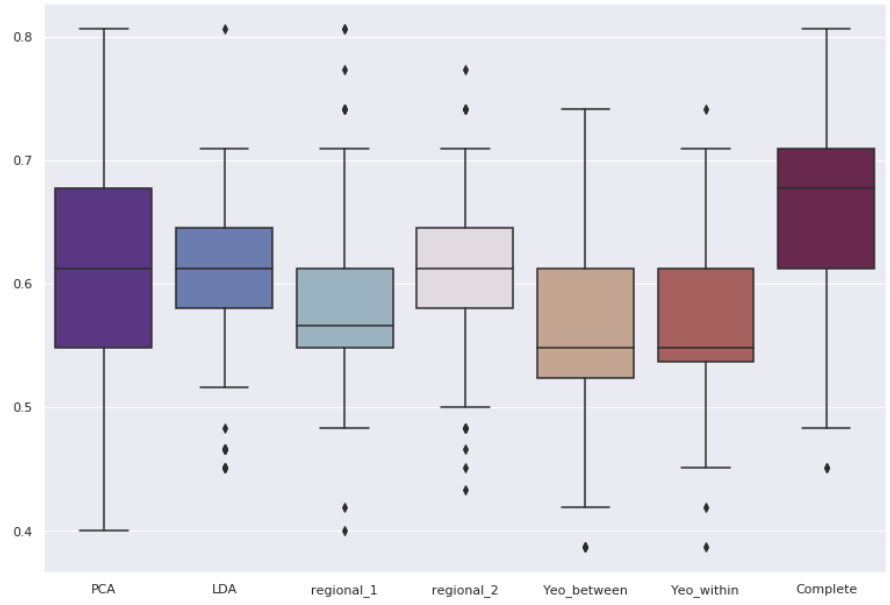


(b) Results of dimensionality reduction using Spearman's ρ

Figure 5.13: Comparison of the scores obtained with different dimensionality reduction techniques - Pearson and Spearman



(a) Results of dimensionality reduction using the tangent parametrization



(b) Results of dimensionality reduction using partial correlation

Figure 5.14: Comparison of the scores obtained with different dimensionality reduction techniques - Tangent and partial correlation

5.2.2 Regression task

We proceed our analysis performing a regression task to predict the brain ages on the second dataset.

Original features

As done before, we start using the original morphometric features as predictors, so we obtain a dataset structured as follows:

- 297 subjects
- 308 brain regions
- 8 morphometric features

We divide the dataset into a training and a test set and, in this case, we stratify on the scanner site and on the age bin to guarantee that the distribution of ages is preserved across the two samples. Then the data are standardized using `StandardScaler` and the regressor algorithms are trained through `GridSearchCV` with 10 folds and the following set of hyperparameters:

- **Linear Regression**:
  - *fit intercept*: True, False;
  - *normalize*: True, False
- **Random Forest Regressor**:
  - *n_estimators*: 10, 50, 100;
  - *max_features*: "auto", "log2", "sqrt";
  - *bootstrap*: True, False
- **Extra Trees Regressor**:
  - *n_estimators*: 10, 50, 100;
  - *max_features*: "auto", "log2", "sqrt";
  - *bootstrap*: True, False
- **AdaBoost Regressor**:
  - *loss*: "linear", "square", "exponential";
  - *learning rate*: 1e-3, 1e-2, 1e-1, 0.5, 1
- **LassoLarsCV**:
  - *normalize*: True, False;
- **RidgeCV**:

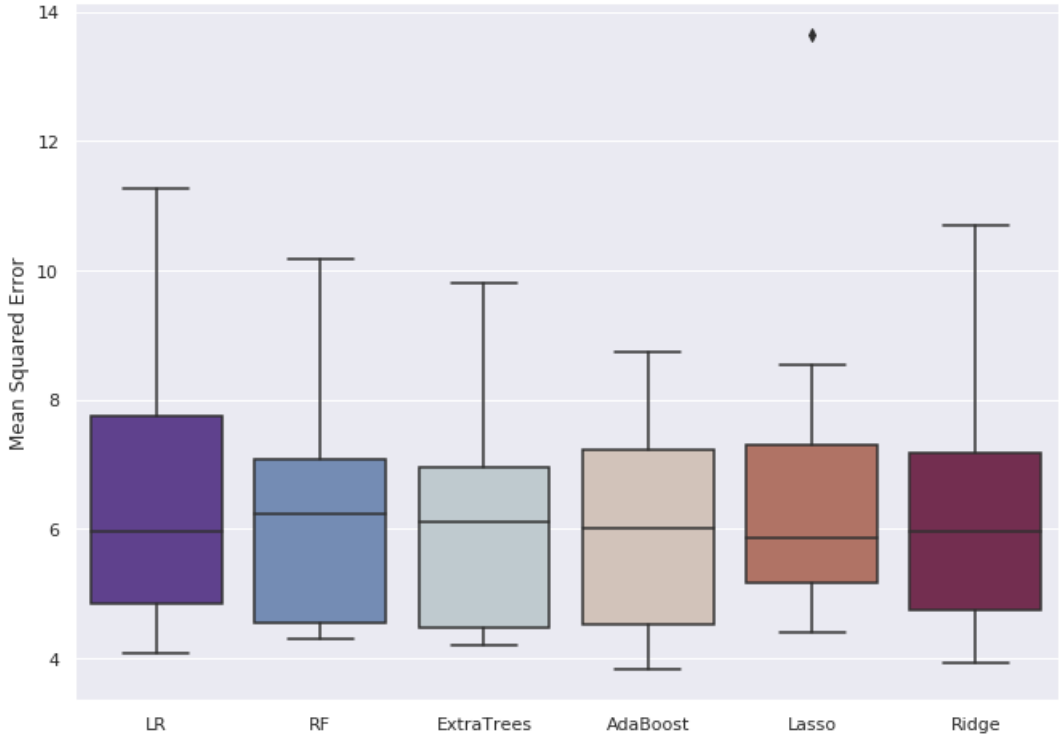


Figure 5.15: Comparison of different regression algorithms using the original morphometric features

- *fit intercept*: True, False;
- *normalize*: True, False

In Figure 5.15 we can see that all the median values are very similar and there is not much difference among the regression algorithms' performances. Also the results on the test set are always between 6 and 7. The drawback is that considering the small range values of the input brain ages (14-24) the mean squared error can't be considered very good.

If we try to apply some dimensionality reduction techniques such as PCA, we notice that the results are always pretty similar, except for the Linear Regression algorithm whose performance worse considerably.

Morphometric similarity matrices

We follow the same pipeline used before to construct and analyze different kinds of morphometric similarity matrices and in Figure 5.16 we can see the results obtained using the different learning algorithms on the 10 folds. The lower median values are

those obtained using the tangent parametrization and the partial correlation, but in the former case we have different scores considerably distant from the median and max ones that are considered as outliers, while in the latter all the values are concentrated in the same region. To further analyze the results and inspect the

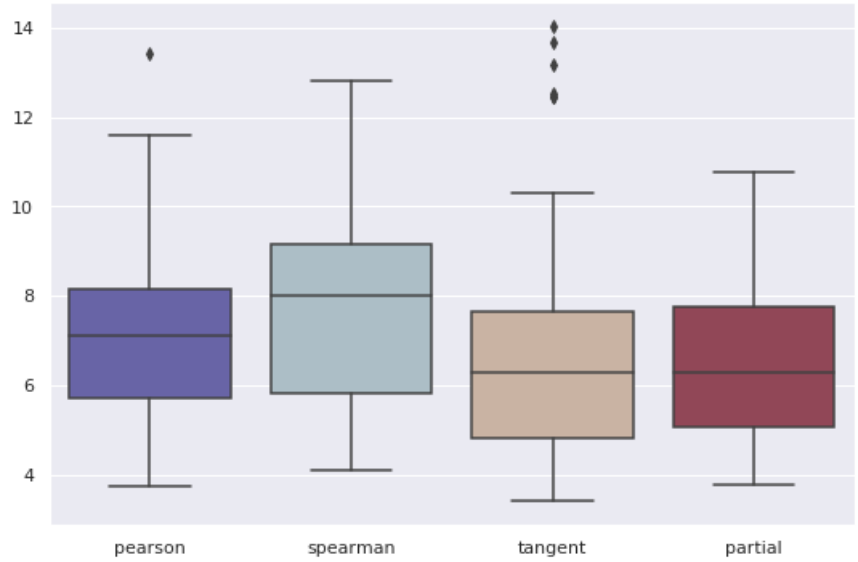


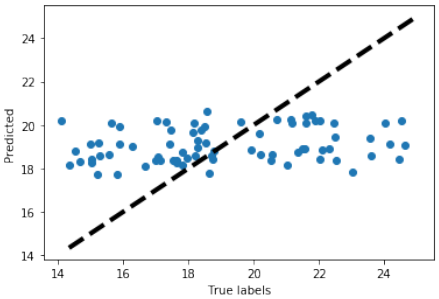
Figure 5.16: Comparison of different structural connectivity matrices for regression task

differences between the regression algorithms, the scores obtained for each one of them are registered.

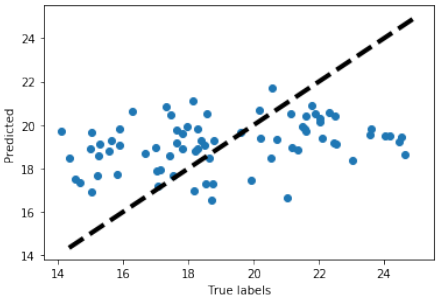
Algorithm	***Pearson***	***Spearman***	***Tangent***	***Partial***
Linear Regression	7.76	8	**6.26**	**6**
Random Forest	7.26	7.96	6.95	6.78
Extra Trees	6.97	7.91	6.95	6.63
AdaBoost	**6.90**	**7.54**	6.71	**6**
Lasso	7.70	7.81	7.94	7.35
Ridge	**6.79**	**7.55**	**6.25**	**5.96**

Table 5.2: Algorithm comparison in terms of MSE

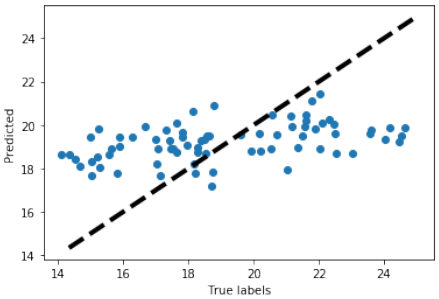
Comparing the scores on the test set and plotting the actual vs fitted graph we obtain the results in Figure 5.17.



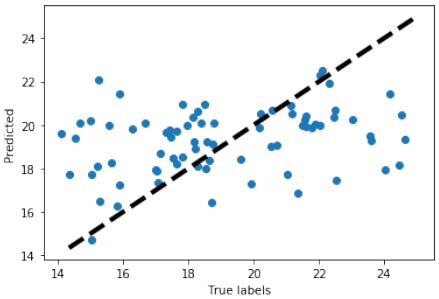
(a) Pearson's r: AdaBoost. **MAE**: 2.36; **MSE**: 7.96



(b) Spearman's ρ: RidgeCV. **MAE**: 2.88; **MSE**: 7.58



(c) Tangent parametrization: RidgeCV. **MAE**: 2.23; **MSE**: 6.92



(d) Partial Correlation: RidgeCV. **MAE**: 2.25; **MSE**: 8.08

Figure 5.17: Actual vs fitted plot for the different kinds of connectivity matrices

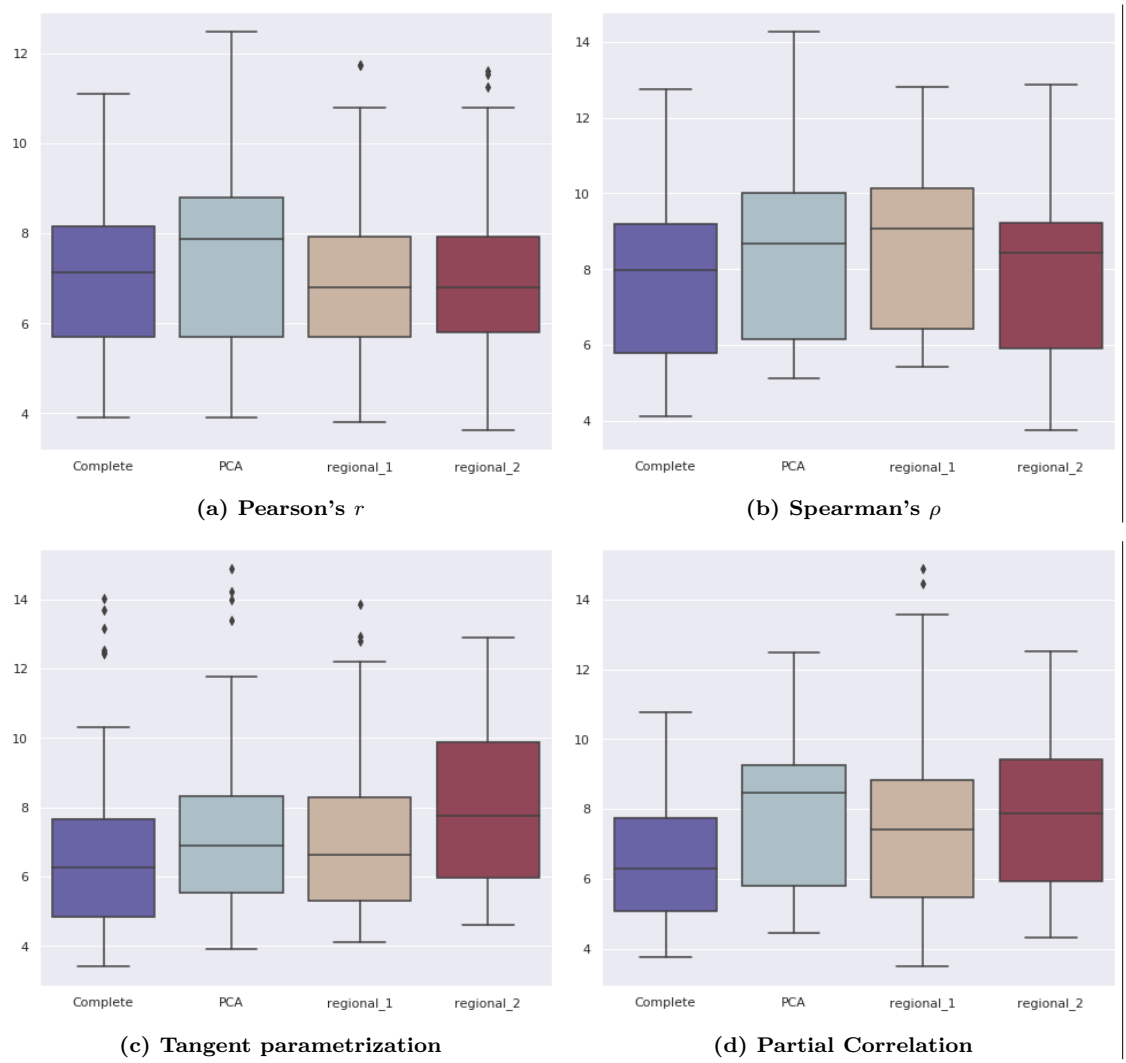
Especially in the first and last section of the plots (ages < 16 and ages > 22) the model does not perform good and it overestimates or underestimates the true ages; in the middle it seems to perform better.

As seen in Figure 5.18 we also try to apply different kinds of dimensionality reduction techniques, as did previously on the psychosis dataset. Even in this case, they do not give great improvements to the results and the best scores are reached taking into account the complete set of features.

Combination of features

We try a last attempt to check if integrating different kinds of predictors can boost the prediction capabilities of our models. To do so, we consider both the original morphometric features and the ones obtained from the morphometric similarity matrices - in this case, just the ones' computed with the Pearson correlation coefficients are taken into account. Instead of simply concatenating or stacking the features together, we exploit a multi-level approach proposed in [Rasero] and depicted in Figure 5.19.

This stacking methodology consists in a two-stage training that receives as input



(a) Pearson's r **(b) Spearman's ρ**

(c) Tangent parametrization **(d) Partial Correlation**

Figure 5.18: Comparison between different types of dimensionality reduction techniques for each kind of connectivity matrix

different groups of features, called *channels*, that in our case are the 8 morphometric features and the flattened connectivity matrices.

- After splitting the data into two sets, a 5–fold cross validation is performed on the training set using a LASSO-PCR, that is a L1-constrained variant of principal component regression applied on each channel (on each set of features separately). The cross-validation loop consists in the optimization of the L1 penalty term λ and in the generation of predictions that are further used as training data during the second stage.

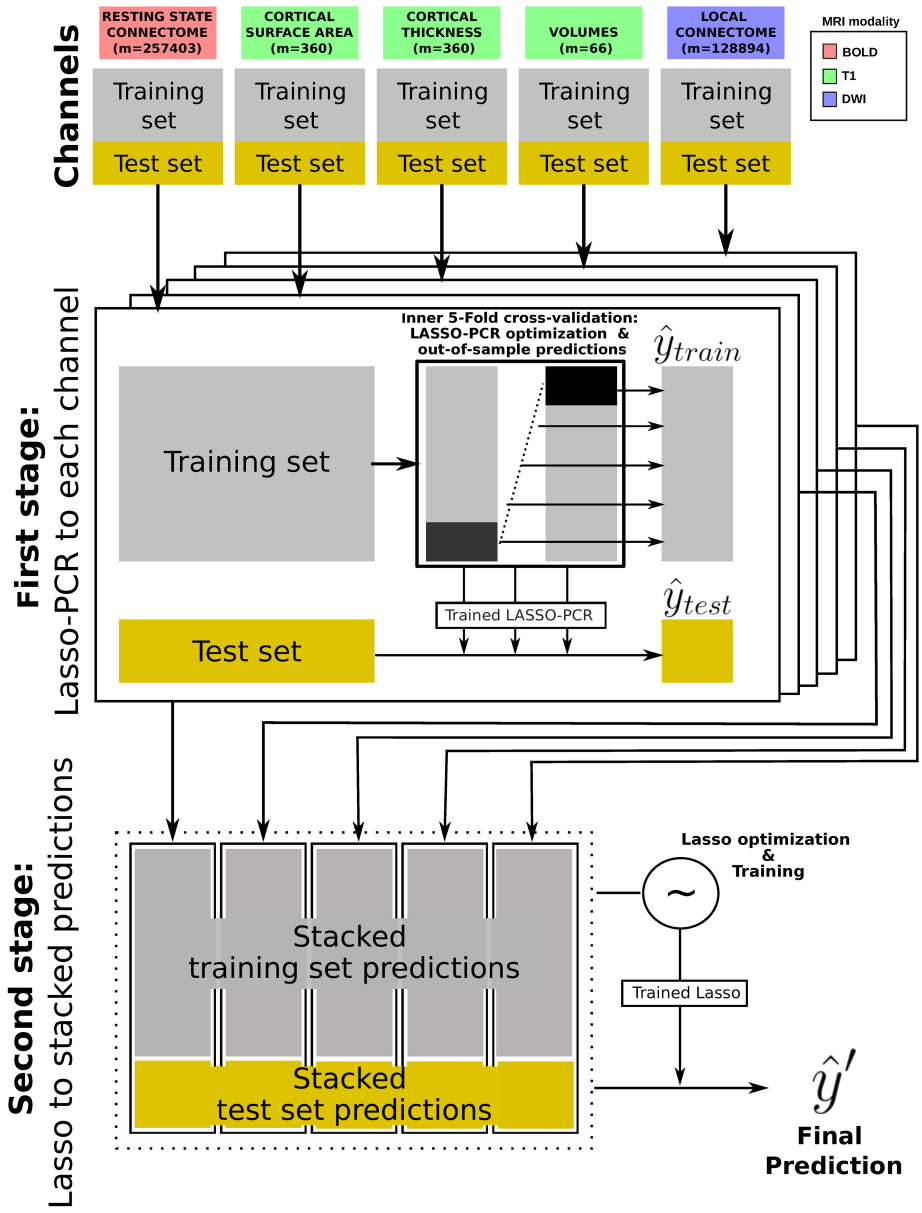


Figure 5.19: Rasero's methodology to stack predictors for regression

- At this point, the predictions on the training and the test sets are stacked across the channels to form a new training and test set and to fed a new LASSO model that performs a weighted feature selection across channels to see how much each set of feature contributes to the final prediction.

To guarantee generalization and prevent overfitting, a 100–folds cross validation is performed. The overall performance of the single-channel and stacked models

can be seen in Figure 5.20.

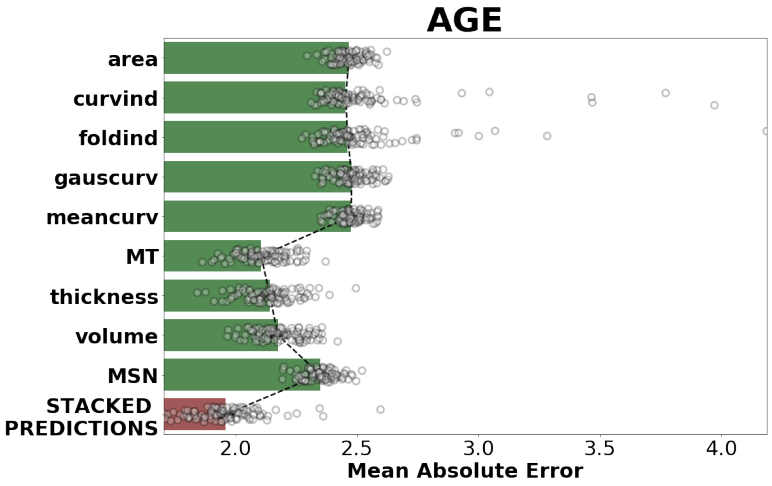


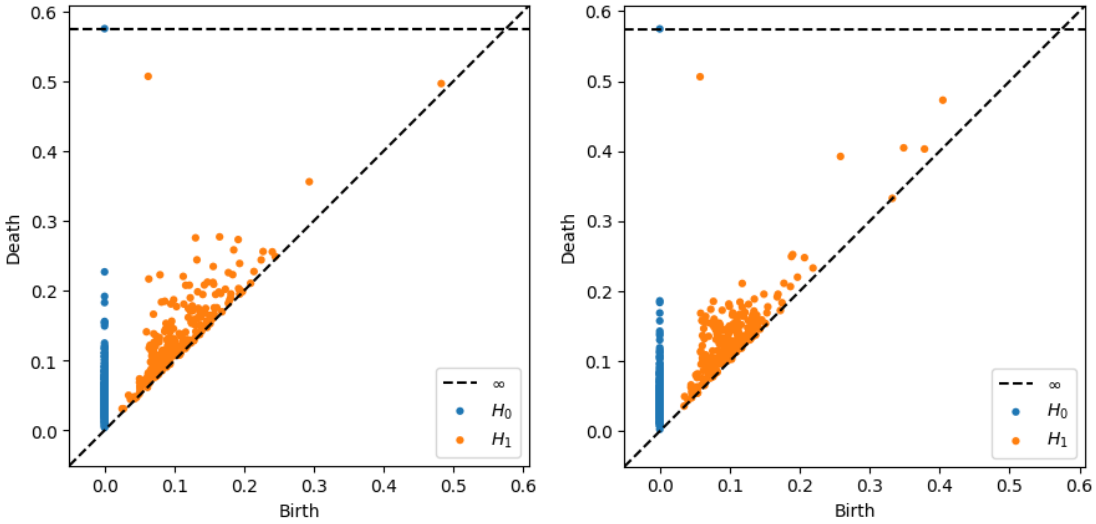
Figure 5.20: Rasero's methodology to stack predictors for regression

As we can see, stacking all the predictions together considerably reduce the mean absolute error, even though the single-channel model related to the morphometric similarity networks do not perform very well and the channel that gives the best result on its own is the one of MT (magnetization transfer). In the end, the stacked predictions outperform all the results obtained so far.

5.3 Topological Data Analysis

Topological Data Analysis is a type of analysis broadly diffused in clinical network neuroscience used to obtain a better understanding of brain connectivity and functions. After our different attempts with Machine Learning whose techniques did not lead to great results, we try to inspect the geometrical aspects of our data more using persistent homology.

As said before, persistent homology tracks the birth and death of holes across the simplicial complexes during filtration to recognize the homology classes that *persist* during time. To compute persistent homology we use the library `Ripser`, but first we need to compute distance matrices starting from our correlation matrices. To do so, we have to define a *metric*: given the r_{ij} element that represents the correlation between region i and region j, we compute $1 - |r_{ij}|, \forall i, j \in \{0, 307\}$. In the end, for each subject we obtain a persistence diagram as the ones plotted in Figure 5.21. The further a point is from the diagonal, the more persistent the generator of that homology group is.



(a) Persistence diagram associated to a control subject **(b)** Persistence diagram associated to a patient subject

Figure 5.21: Examples of two persistence diagrams obtained using `Ripser`

To inspect the differences between the two sets, the one of controls and the one of patients, we can extract different kinds of information contained in our persistence diagrams. We perform topological data analysis on both the previous datasets, plus the third one presented before, that is used as a benchmark to validate the informativeness of our results, since the dataset has been already exploited for a persistent homology study in [7]. Moreover, in order to apply this study to the second dataset we need to binarize the problem and divide the dataset into two groups; to do so, we choose a threshold value that allows to create two balanced sets and in this case the threshold corresponds to a value of age equals to 19.

The first measure that we want to study is the so called **bottleneck distance**, which measures the distance between two persistence diagrams as the cost of the optimal matching between points of the two diagrams. We compute the distances among each pair of persistence diagrams, separately for the homology groups H_0 and H_1 and we plot the results in two heatmaps, where the first block belongs to the first set and the second block to the second set. We hope to see two well-defined blocks in the heatmaps that show a great inter-class distance and a low intra-class distance. In Figure 5.22 we show the results obtained using the correlation matrices computed with the Pearson's coefficient as distance matrices:

- In the first case, the first 227 subjects are controls and the others 185 are

patients;

- In the second case, the first 144 subjects are under 19 years old, while the others 153 are older;
- In the third case, the first 15 persistence diagrams are associated to the subjects that received placebo, the others 15 refer to the scan in which subjects had taken psylocibin.

However we can see that the plots don't give us any meaningful information, so we need to explore further statistics to extract useful results that allow to differentiate our two sets.

As a second attempt, we try to look at the probability distribution functions for the persistence, the birth and death of generators of H_0 and H_1 and then compute some distances between these distributions.

To obtain the distribution of persistences, for each subject we compute the differences between the death and the birth times, we aggregate these differences and look at the distributions. We repeat the same computation for H_0 and H_1 separately. With the same process, we compute the distributions for the birth times and the death times, for each of the subgroup (patients/controls, < 19 or ≥ 19, placebo/psylocibin). All of these statistics for our two datasets are computed using as input each of the four kinds of correlation matrices analyzed before, all of them transformed into distance matrices using the metric $1 - |r|$. For simplicity, we show only the plots obtained starting from the Pearson's correlation matrices for both of the datasets.

As we can see in Figure 5.23 and Figure 5.24 that show the probability distribution functions for the first and the second dataset respectively, we are not able to detect meaningful differences between the two distributions by looking at the plots. We compare the results obtained on our datasets with the plots of the third dataset, whose PDFs are shown in Figure 5.25, and we can see that also in this case it is difficult to say something just looking at the graphs.

For this reason, we decide to rely on further statistics to compute the distances between the distributions: for each pair of probability distribution functions, we perform the Kolmogorov-Smirnov test and we store the p-value results for each kind of correlation matrices. Furthermore, we notice that the results vary considerably according to the width of the bins of the histograms, that's why we decide to compare the results considering a variable number of bins.

As we can see from the tables, in most of the cases the Kolmogorov-Smirnov test show that the two probability distributions strongly differ, since the p-values

are often very small and less than 10^{-10}. We notice that the results strongly vary according to the width of bins and, in particular, the more the dataset size grows and so the points in H_0 and H_1, the higher the width should be to detect differences and information. In fact, the psychosis dataset, whose results are shown in Table 5.3, is the largest one and the best results in this case are reached considering a width of bins equals 5000; on the other hand, for what concerns the psylocibin dataset, as shown in Table 5.5, being very small it requires a small width to show great results.

Bins	***PDFpersistence***	***PDFbirths***	***PDFdeaths***
Pearson			
1000	$H_0 : 0.5 \mid H_1 : 0.5$	$H_1 : 10^{-8}$	$H_1 : 10^{-5}$
2000	$H_0 : 10^{-37} \mid H_1 : 0.02$	$H_1 : 10^{-15}$	$H_1 : 10^{-8}$
5000	$H_0 : 10^{-76} \mid H_1 : 10^{-4}$	$H_1 : 10^{-32}$	$H_1 : 10^{-21}$
Spearman			
1000	$H_0 : 1 \mid H_1 : 0.99$	$H_1 : 1$	$H_1 : 1$
2000	$H_0 : 1 \mid H_1 : 0.99$	$H_1 : 0.99$	$H_1 : 1$
5000	$H_0 : 0.99 \mid H_1 : 0.99$	$H_1 : 1$	$H_1 : .99$
Tangent			
1000	$H_0 : 0.2 \mid H_1 : 10^{-12}$	$H_1 : 0.09$	$H_1 : 10^{-3}$
2000	$H_0 : 0.2 \mid H_1 : 10^{-19}$	$H_1 : 0.018$	$H_1 : 10^{-5}$
5000	$H_0 : 10^{-4} \mid H_1 : 10^{-37}$	$H_1 : 10^{-7}$	$H_1 : 10^{-11}$
Partial correlation			
1000	$H_0 : 10^{-18} \mid H_1 : 10^{-5}$	$H_1 : 10^{-6}$	$H_1 : 10^{-3}$
2000	$H_0 : 10^{-26} \mid H_1 : 10^{-10}$	$H_1 : 10^{-11}$	$H_1 : 10^{-5}$
5000	$H_0 : 10^{-44} \mid H_1 : 10^{-21}$	$H_1 : 10^{-23}$	$H_1 : 10^{-11}$

Table 5.3: Results of the Kolmogorov-Smirnov test for the first dataset

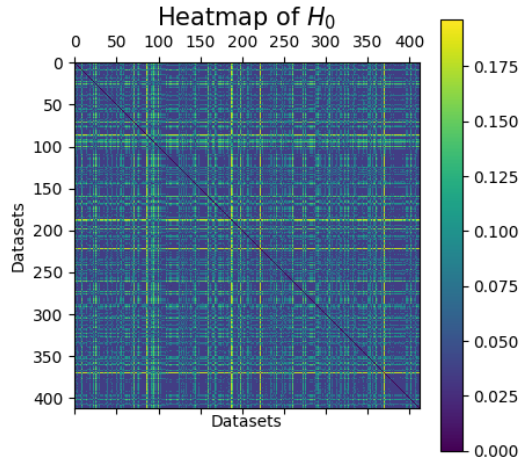
This confirms that using Topological Data Analysis can be useful even in cases in which Machine Learning does not reach great results and it demonstrates to be a powerful tool that helps in understanding the differences among data just by inspecting their geometric properties.

Bins	***PDFpersistence***	***PDFbirths***	***PDFdeaths***
Pearson			
1000	$H_0 : 10^{-10} \mid H_1 : 0.04$	$H_1 : 10^{-4}$	$H_1 : 10^{-5}$
2000	$H_0 : 10^{-20} \mid H_1 : 10^{-4}$	$H_1 : 10^{-9}$	$H_1 : 10^{-10}$
5000	$H_0 : 10^{-46} \mid H_1 : 10^{-10}$	$H_1 : 10^{-26}$	$H_1 : 10^{-37}$
Spearman			
1000	$H_0 : 1 \mid H_1 : 0.99$	$H_1 : 1$	$H_1 : 1$
2000	$H_0 : 1 \mid H_1 : 0.99$	$H_1 : 0.99$	$H_1 : 1$
5000	$H_0 : 0.99 \mid H_1 : 0.99$	$H_1 : 1$	$H_1 : .99$
Tangent			
1000	$H_0 : 0.6 \mid H_1 : 0.16$	$H_1 : 10^{-4}$	$H_1 : 10^{-22}$
2000	$H_0 : 0.5 \mid H_1 : 10^{-3}$	$H_1 : 10^{-6}$	$H_1 : 10^{-33}$
5000	$H_0 : 0.4 \mid H_1 : 10^{-5}$	$H_1 : 10^{-12}$	$H_1 : 10^{-66}$
Partial correlation			
1000	$H_0 : 10^{-3} \mid H_1 : 10^{-9}$	$H_1 : 10^{-23}$	$H_1 : 10^{-12}$
2000	$H_0 : 10^{-5} \mid H_1 : 10^{-14}$	$H_1 : 10^{-31}$	$H_1 : 10^{-19}$
5000	$H_0 : 10^{-7} \mid H_1 : 10^{-26}$	$H_1 : 10^{-56}$	$H_1 : 10^{-35}$

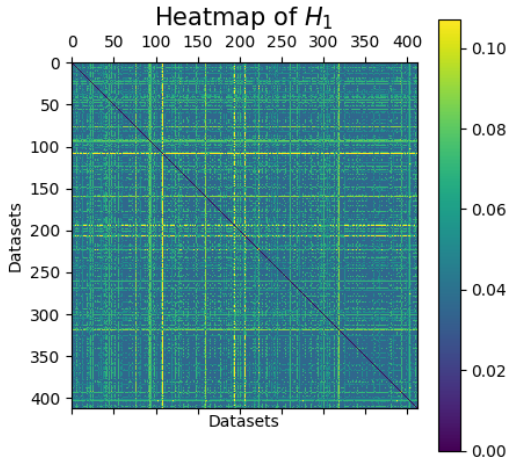
Table 5.4: Results of the Kolmogorov-Smirnov test for the second dataset

Bins	***PDFpersistence***	***PDFbirths***	***PDFdeaths***
1000	$H_0 : 10^{-18} \mid H_1 : 10^{-8}$	$H_1 : 10^{-12}$	$H_1 : 10^{-14}$
2000	$H_0 : 10^{-46} \mid H_1 : 10^{-15}$	$H_1 : 10^{-62}$	$H_1 : 10^{-60}$
5000	$H_0 : 10^{-110} \mid H_1 : 10^{-41}$	$H_1 : 10^{-131}$	$H_1 : 10^{-124}$

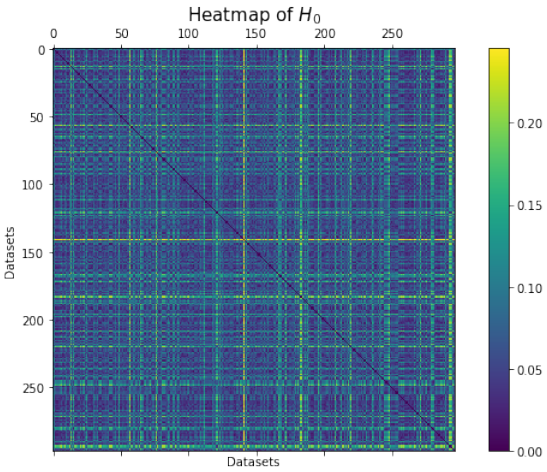
Table 5.5: Results of the Kolmogorov-Smirnov test for the third dataset



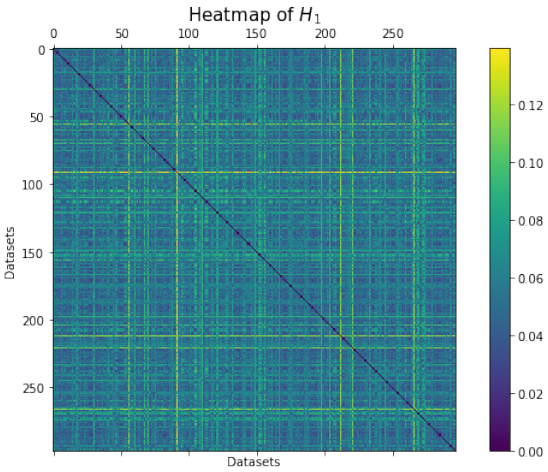
(a) First dataset: H_0 bottleneck distance



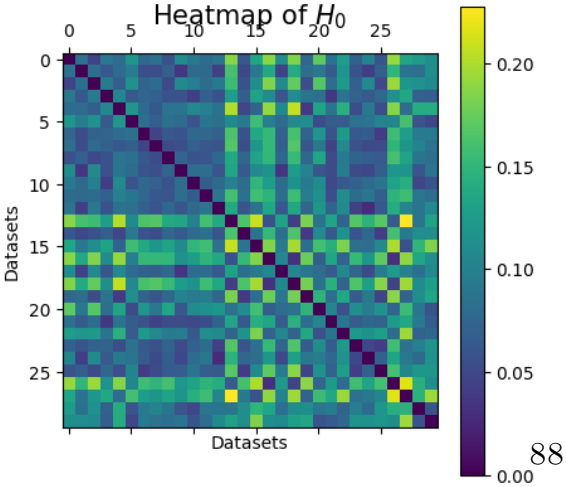
(b) First dataset: H_1 bottleneck distance



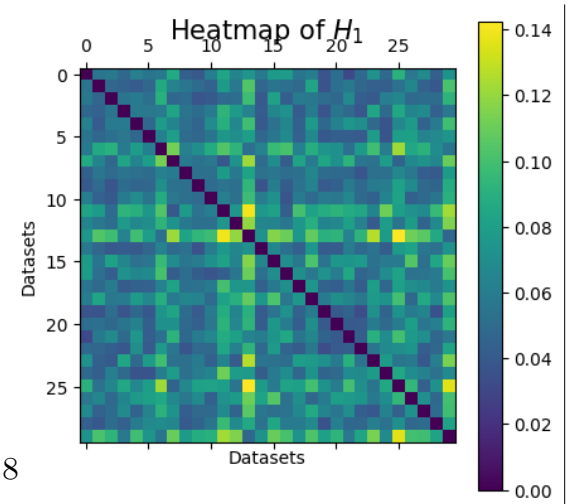
(c) Second dataset: H_0 bottleneck distance



(d) Second dataset: H_1 bottleneck distance



(e) Third dataset: H_0 bottleneck distance



(f) Third dataset: H_1 bottleneck distance

Figure 5.22: Heatmaps of the bottleneck distance between persistence diagrams

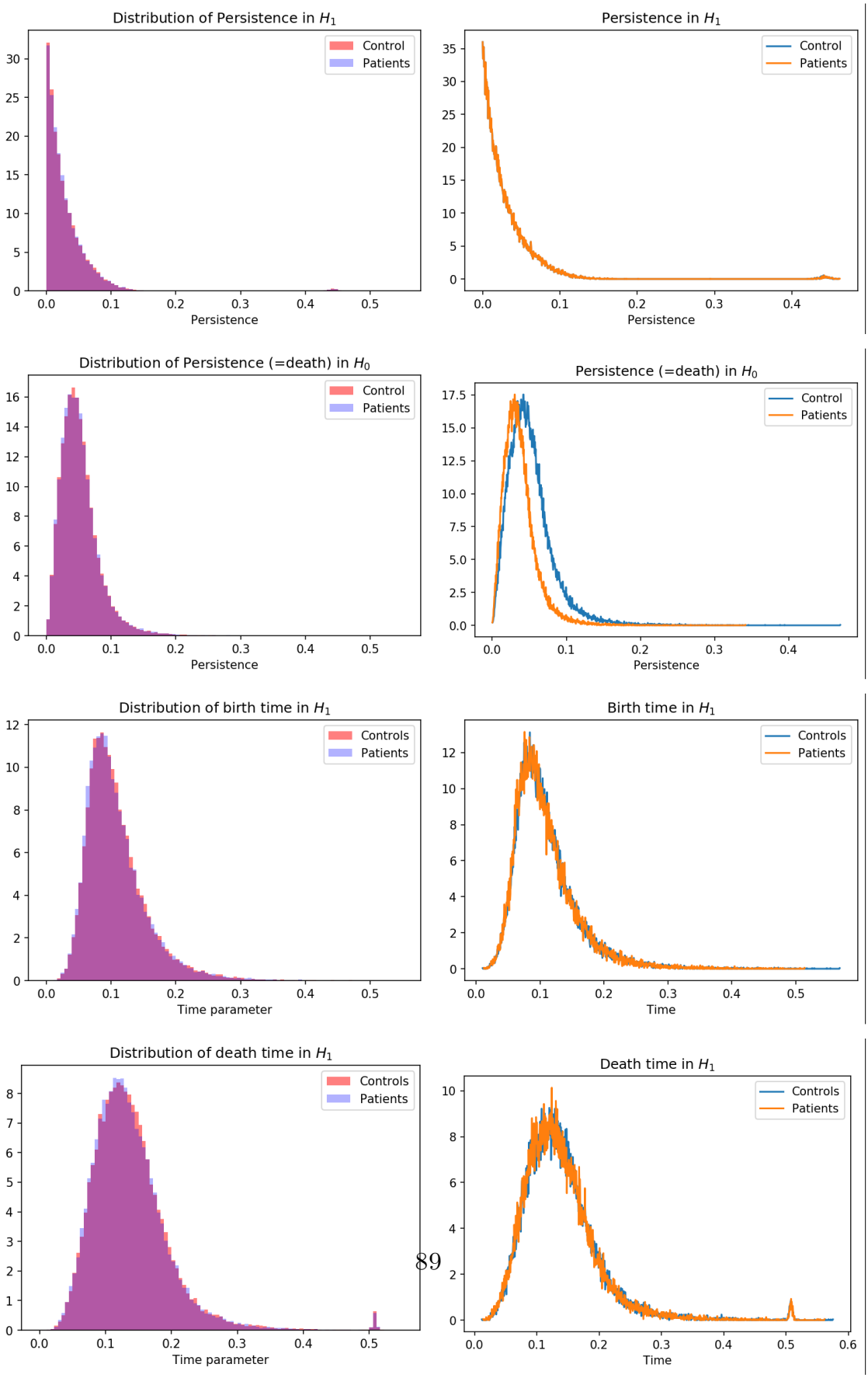


Figure 5.23: Probability distribution functions for the first dataset

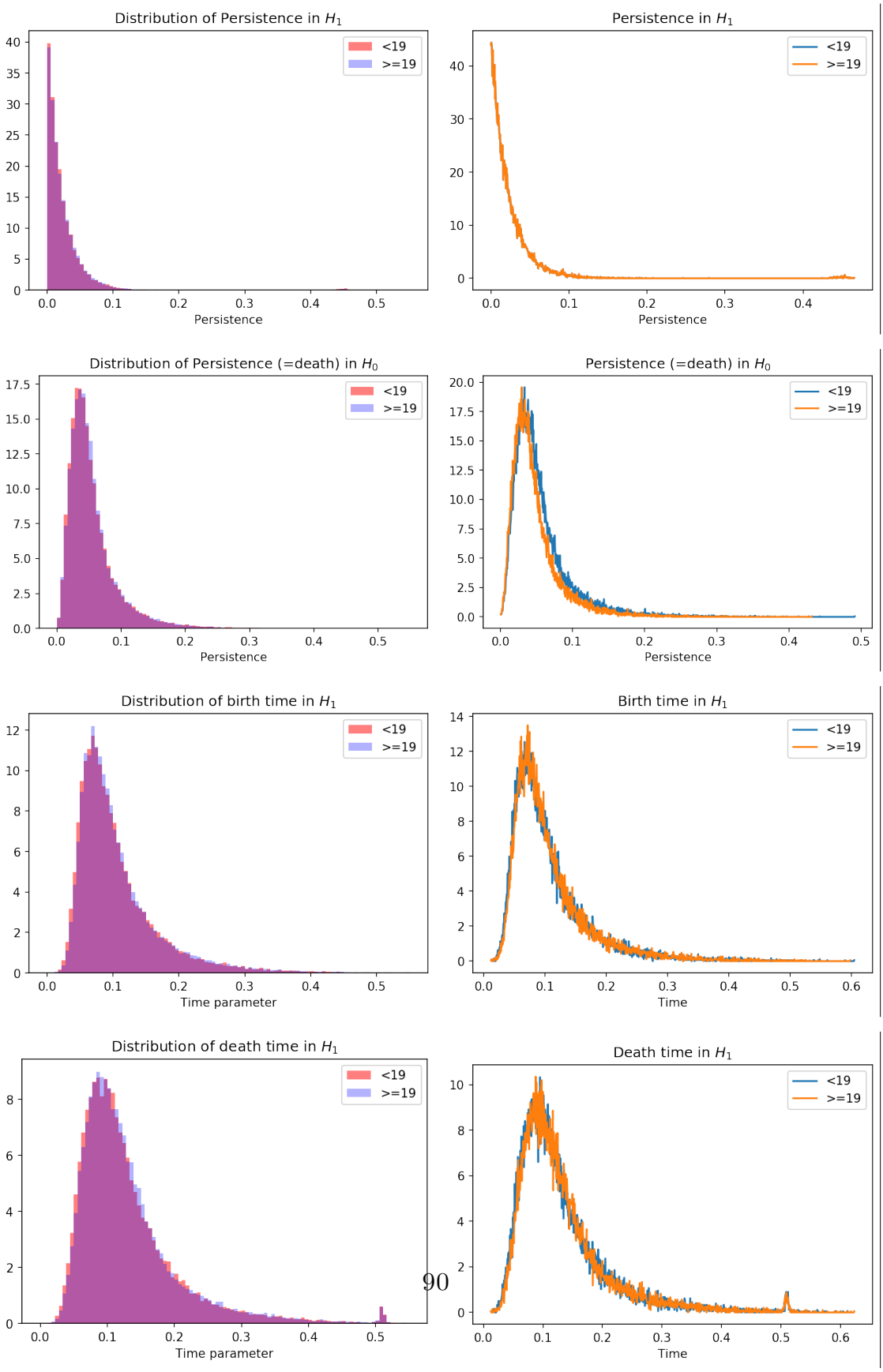


Figure 5.24: Probability distribution functions for the second dataset

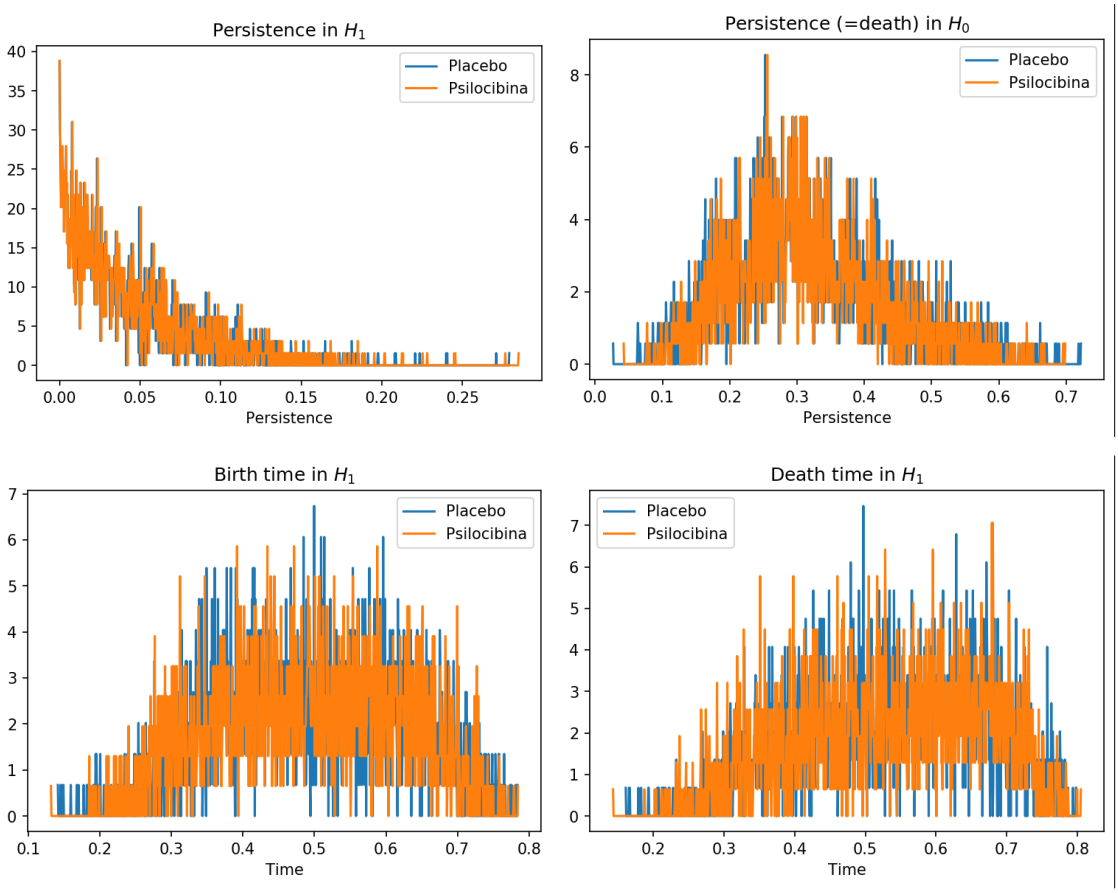


Figure 5.25: Probability distribution functions for the third dataset

Chapter 6

Conclusions

In this work we have applied advanced analytical techniques to the field of computational neuroscience. In particular, we have analyzed two different structural connectivity datasets whose morphometric features have been used to construct morphometric similarity matrices. Morphometric similarity quantifies the correspondence of brain regions in terms of multiple macrostructural (e.g. cortical thickness) and microstructural features (e.g. fractional anisotropy) measured by MRI. From a neurological perspective, a high correlation value between a pair of cortical regions indicates that there is a high degree of correspondence between them in terms of cytoarchitectonic and myeloarchitectonic features and also that the two regions are more likely to be axonally connected to each other.

Connectivity matrices are widely used in neuroscience as predictors and our goal was to compare their predictive utility with respect to the one of the original features. The common procedure for designing predictive models based on connectivity consists in defining brain regions through a parcellation of the brain, estimating the similarity between pairs of regions based on different measures using Pearson correlation and using the resulting matrices to feed a classifier for predicting non-imaging variables. However, there are different ways to estimate similarity between brain regions and analyzing correlation matrices requires an inspection of their geometrical properties. To correctly apply mathematical formulations on them, given that they are positive definite matrices and they do not naturally form a Euclidean space, Riemannian geometry should be taken into account. For this reason, we also applied the tangent space parametrization to obtain a tangent representation of the Riemannian manifold and preserve the geometry of these matrices.

The first part of our analysis consisted in the application of Machine Learning predictive models on the two datasets to predict psychosis in one case and brain age in the other. We used as input the original morphometric features and those obtained from the matrices. Different pipelines of preprocessing of the data and

different learning algorithms were used. Despite that, the classification task was difficult to solve and the final performance was rather poor. The regression task gave us better results, even if also in this case it was difficult to infer a significant improvement given by the use of MSMs. Comparing the predictive performance of the different kinds of similarity matrices, we noticed slightly better scores when using the tangent space parametrization. However, also in this case this was not significant enough to infer substantial improvement.

Given the bad results obtained using Machine Learning algorithms, we tried to analyze our data from a different perspective using Topological Data Analysis and, more specifically, persistent homology. We computed persistence diagrams for each subject, for both of the datasets, and from those we were able to obtain some statistics distributions for each group. We also used a third dataset of functional connectivity data as a benchmark and the Kolmogorov Smirnov statistic test gave us enough confidence to rule out that the two distributions (of patients and controls in the first case and of younger vs older in the second) were the same. We consider this result satisfactory with respect to the previous analysis because we have reasons to believe that the topological features allow us to distinguish the different groups. Given that, we would like to expand our analysis in the future, for example by conducting further experiments to validate the correctness of our conclusions and trying to look at the network properties of the morphometric similarity matrices. Lastly, we want to try the application of clustering and community detection algorithms on the matrices to detect similarity or differences between groups of regions.

Bibliography

[1] Aaron Alexander-Bloch, Jay N. Giedd, and Ed Bullmore. «Imaging structural co-variance between human brain regions». In: *Nature Reviews Neuroscience* 14.5 (2013), pp. 322–336. DOI: 10.1038/nrn3465. URL: http://dx.doi.org/10.1038/nrn3465.

[2] Robert Ghrist Ann E. Sizemore Jennifer E. Phillips-Cremins and Danielle S. Bassett. «The importance of the whole: Topological data analysis for the network neuroscientist». In: 1.3 (2018), pp. 222–241. DOI: 10.1162/NETN. URL: http://dx.doi.org/10.1162/netn_a_00083.

[3] Denis Blackmore and Thomas J. Peters. *Computational Topology*. 2007, pp. 493–545. ISBN: 9780444522085. DOI: 10.1016/B978-044452208-5/50049-1.

[4] Nitesh V. Chawla et al. «snopes.com: Two-Striped Telamonia Spider». In: *Journal of Artificial Intelligence Research* 16.Sept. 28 (2002), pp. 321–357. ISSN: 10769757. eprint: 1106.1813. URL: https://arxiv.org/pdf/1106.1813.pdf%0Ahttp://www.snopes.com/horrors/insects/telamonia.asp.

[5] Kamalaker Dadi et al. «Benchmarking functional connectome-based predictive models for resting-state fMRI». In: *NeuroImage* 192 (2019), pp. 115–134. ISSN: 10959572. DOI: 10.1016/j.neuroimage.2019.02.062. URL: http://dx.doi.org/10.1016/j.neuroimage.2019.02.062.

[6] T Fletcher and S Joshi. «Riemannian Geometry for the Statistical Analysis». In: *Signal Processing* 87 (2007), pp. 250–262.

[7] Petri G et al. «Homological scaffolds of brain functional networks». In: *J. R. Soc. Interface* (2014). ISSN: 20140873. DOI: http://dx.doi.org/10.1098/rsif.2014.0873.

[8] Allen Hatcher. «Algebraic Topology». In: January (2003).

[9] *Homology Theory*. https://jeremykun.com/2013/04/03/homology-theory-a-primer/.

[10] Olivier Ledoit and Michael Wolf. «A well-conditioned estimator for large-dimensional covariance matrices». In: *Journal of Multivariate Analysis* 88.2 (2004), pp. 365–411. ISSN: 0047259X. DOI: 10.1016/S0047-259X(03)00096-4.

[11] Christophe Leys et al. «Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median». In: *Journal of Experimental Social Psychology* 49.4 (2013), pp. 764–766. ISSN: 00221031. DOI: 10.1016/j.jesp.2013.03.013. URL: http://dx.doi.org/10.1016/j.jesp.2013.03.013.

[12] Guerra Marco et al. «Homological scaffold via minimal homology bases». In: *Scientific reports* 11.1 (2021), p. 5355. ISSN: 20452322. DOI: 10.1038/s41598-021-84486-1. arXiv: 2004.11606.

[13] Afshin Rostamizadeh Mehryar Mohri and Ameet Talkwalkar. *Foundations of Machine Learning*. ISBN: 9780262039406.

[14] Sarah E. Morgan et al. «Cortical patterning of abnormal morphometric similarity in psychosis is associated with brain expression of schizophrenia-related genes». In: *Proceedings of the National Academy of Sciences of the United States of America* 116.19 (2019), pp. 9604–9609. ISSN: 10916490. DOI: 10.1073/pnas.1820754116.

[15] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. «HAL Id: inria-00070743 https://hal.inria.fr/inria-00070743 A Riemannian Framework for Tensor Computing A Riemannian Framework for Tensor Computing». In: (2006). URL: https://hal.inria.fr/inria-00070743.

[16] Usama Pervaiz et al. «Optimising network modelling methods for fMRI». In: *NeuroImage* 211 (2020), p. 116604. ISSN: 10959572. DOI: 10.1016/j.neuroimage.2020.116604. URL: https://doi.org/10.1016/j.neuroimage.2020.116604.

[17] Vsevolod Salnikov et al. «Co-occurrence simplicial complexes in mathematics: identifying the holes of knowledge». In: *Applied Network Science* 3.1 (2018). ISSN: 23648228. DOI: 10.1007/s41109-018-0074-3. arXiv: 1803.04410.

[18] Jakob Seidlitz et al. «Morphometric Similarity Networks Detect Microscale Cortical Organization and Predict Inter-Individual Cognitive Variation». In: *Neuron* 97.1 (2018), 231–247.e7. DOI: 10.1016/j.neuron.2017.11.039.

[19] Jakob Seidlitz et al. «Transcriptomic and cellular decoding of regional brain vulnerability to neurogenetic disorders». In: *Nature Communications* 11.1 (2020), pp. 1–14. ISSN: 20411723. DOI: 10.1038/s41467-020-17051-5.

[20] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Vol. 9781107057135. 2013, pp. 1–397. ISBN: 9781107298019. DOI: 10.1017/CBO9781107298019.

[21] Raphael Tinarrage. *Homology Theory*. https://raphaeltinarrage.github.io/files/EMAp/SummerCourseTDA.pdf. 2021.

[22] František Váša et al. «Adolescent tuning of association cortex in human structural brain networks». In: *Cerebral Cortex* 28.1 (2018), pp. 281–294. ISSN: 14602199. DOI: 10.1093/cercor/bhx249.

[23] Manasij Venkatesh, Joseph Jaja, and Luiz Pessoa. «Comparing functional connectivity matrices: A geometry-aware approach applied to participant identification». In: *NeuroImage* 207.November (2020), p. 116398. ISSN: 10959572. DOI: 10.1016/j.neuroimage.2019.116398. URL: https://doi.org/10.1016/j.neuroimage.2019.116398.